DATA SCIENCE USING PYTHON

B.A / B.Com.(Computer Applications)

THIRD YEAR, SEMESTER-V

Lesson Writers:

Dr. K. Lavanya
Asst. Professor,
Dept. of Comp. Science & Engg.,
Acharya Nagarjuna University,

Mr. G.V. Suresh,
Assoc. Professor
Dept. of Comp. Science & Engg.,
Lakireddy Balireddy College of Engineering, Mylavaram

Mrs. A. Sravani
Assoc. Professor,
Dept. of Information Technology,
Lakireddy Balireddy College of
Engineering, Mylavaram

Editor & Lesson Writer

Dr. U. Surya Kameswari
Asst. Professor,
Dept. of Comp. Science & Engg.,
Acharya Nagarjuna University

Director
Prof. V.VENKATESWARLU
M.A., (Soc), M.S.W.,M.Phil., Ph.D.
Centre for Distance Education
Acharya Nagarjuna University
Nagarjuna Nagar 522 510

Ph: 0863-2346222, 2346208
0863- 2346259 (Study Material)
Website www.anucde.info
E-mail: anucdedirector@gmail.com

B.A/ B.Com. (C.A.)

First Edition : 2021

No. of Copies :

Published by:

Prof. V.VENKATESWARLU,
Director
Centre for Distance Education,
Acharya Nagarjuna University

Printed at:

FOREWORD

Since its establishment in 1976, Acharya Nagarjuna University has been forging ahead in the path of progress and dynamism, offering a variety of courses and research contributions. I am extremely happy that by gaining 'A' grade from the NAAC in the year 2016, Acharya Nagarjuna University is offering educational opportunities at the UG, PG levels apart from research degrees to students from over 443 affiliated colleges spread over the two districts of Guntur and Prakasam.

The University has also started the Centre for Distance Education in 2003-04 with the aim of taking higher education to the door step of all the sectors of the society. The centre will be a great help to those who cannot join in colleges, those who cannot afford the exorbitant fees as regular students, and even to housewives desirous of pursuing higher studies. Acharya Nagarjuna University has started offering B.A., and B.Com courses at the Degree level and M.A., M.Com., M.Sc., M.B.A., and L.L.M., courses at the PG level from the academic year 2003-2004 onwards.

To facilitate easier understanding by students studying through the distance mode, these self-instruction materials have been prepared by eminent and experienced teachers. The lessons have been drafted with great care and expertise in the stipulated time by these teachers. Constructive ideas and scholarly suggestions are welcome from students and teachers involved respectively. Such ideas will be incorporated for the greater efficacy of this distance mode of education. For clarification of doubts and feedback, weekly classes and contact classes will be arranged at the UG and PG levels respectively.

It is my aim that students getting higher education through the Centre for Distance Education should improve their qualification, have better employment opportunities and in turn be part of country's progress. It is my fond desire that in the years to come, the Centre for Distance Education will go from strength to strength in the form of new courses and by catering to larger number of people. My congratulations to all the Directors, Academic Coordinators, Editors and Lesson- writers of the Centre who have helped in these endeavours.

Prof. K. Gangadhara Rao
Vice-Chancellor, I/c.
Acharya Nagarjuna University

B.A/B.Com. (Computer Applications)
Third Year, Semester-V
509BCE21- DATA SCIENCE USING PYTHON

Syllabus

Learning Outcomes:

Upon successful completion of the course, a student will be able to:

1. Understand basic concepts of data science
2. Understand why python is a useful scripting language for developers.
3. Use standard programming constructs like selection and repetition.
4. Use aggregated data (list, tuple, and dictionary).
5. Implement functions and modules.

II. Syllabus :(Total hours: 75 including Theory, Practical, Training, Unit tests etc.) **Unit — 1: Introduction to data science (12h)**

Data science and its importance, advantages of data science, the process of data science,

Responsibilities of a data scientist, qualifications of data scientists, would you be a good data scientist, why to use python for data science.

Unit — 2: Introduction to python (14h)

What is python , features of python, history of python, writing and executing the python program, basic syntax, variables, keywords, data types ,operators ,indentation, Conditional statements-if, if-else, nested if-else, looping statements-for, while, break, continue, pass

Unit — 3: Control structures and strings (10h)

Strings - definition, accessing, slicing and basic operations

Lists - introduction, accessing list, operations, functions and methods,

Tuples - introduction, accessing tuple

Dictionaries - introduction, accessing values in dictionaries

Unit — 4: Functions and modules (13h)

Functions - defining a function, calling a function, types of functions, function arguments, local and global variables, lambda and recursive functions, Modules-math and random

Unit-5: Classes & Objects (11h)

Classes and Objects, Class method and self-argument, class variables and object variables, public and private data members, private methods, built-in class attributes, static methods.

Reference Books:

1. Steven cooper--- Data Science from Scratch, Kindle edition
2. Reemathareja—Python Programming using problem solving approach, Oxford Publication.

(509BCE21)

MODEL QUESTION PAPER
BA &B.Com. (C.A) DEGREE EXAMINATION
Third Year - Fifth Semester

DATA SCIENCE USING PYTHON

Time : Three hours Max. Marks: 70

SECTION A-(5 x 4 = 20 marks)
Answer any FIVE of the following.
Each question carries 4 marks.

1. What are the primary responsibilities of a data scientist in an organisation?
2. Explain the importance of data science.
3. Explain about arithmetic operators in python.
4. Explain about break and pass statements in python.
5. Explain about tuple data structure.
6. Explain about dictionaries in Python.
7. Discuss the concepts of lambda functions and their use in Python.
8. Explain the concept of recursive functions in Python.
9. Discuss the built-in class attributes in Python.
10. Explore the concept of static methods in Python classes.

SECTION B – (5 x 10 = 50 marks)
Answer any FIVE of the following questions.
Each question carries 10 marks.

11. Describe the key steps involved in the process of data science.
12. Justify why Python is a preferred programming language for data science.
13. Discuss the different types of decision-making statements in Python.
14. Elaborate on the concept of looping statements in Python.
15. Discuss the various methods available for manipulating lists in Python.
16. Explain how string accessing, slicing operations work in Python. Also explain about basic operations on strings.
17. Discuss the modules "math" and "random" in Python.
18. Discuss the steps involved in defining and calling a function in Python. Explain types of functions. Explain with examples.
19. Explain the concepts of classes and objects in object-oriented programming. Provide an example.
20. (a) Define private methods in Python classes.
    (b) Explain the significance of the "self" argument.

CONTENTS

LESSON- 01

INTRODUCTION TO DATA SCIENCE

AIMS AND OBJECTIVES

The primary goal of this chapter is to understand the role of data scientist in the field of data science. The chapter began with understanding what data science, importance of data science is and so on. After completing this chapter, the student will understand the complete knowledge about data science and its process in business.

STRUCTURE

1.1. INTRODUCTION

Quantitative and statistical analysis, specialized programming, advanced analytics, artificial intelligence (AI), and machine learning are all components of data science. These components are combined with specialized subject matter expertise to unearth meaningful insights that are concealed inside an organization's data. The decision-making process and the planning of strategic actions can both benefit from these insights.

To identify trends, organizations can examine massive amounts of organized and unstructured big data thanks to the application of data science. The result of this is that businesses can improve their efficiency, better control their costs, discover new opportunities in the market, and strengthen their competitive advantage. This chapter explores what is data science, the importance of data science, and business processes steps etc.

1.2 WHAT IS DATA SCIENCE?

It is a deep study of the huge quantity of data that is known as data science. Data science involves the collection, analysis, and interpretation of relevant insights, patterns, and trends from large volumes of data that are either raw, structured, or unstructured. These insights and trends can be utilized to make educated decisions and address problems that occur in the real world.

1.2.1 Pillars of Data Science

The analysis of huge volumes of data is accomplished through the utilization of a multidisciplinary approach that integrates concepts and procedures from a variety of domains, including mathematics, statistics, artificial intelligence, computer engineering, and others. Data scientists can ask and answer questions such as what occurred, why it occurred, what will occur, and what can be done with the results with the assistance of this analysis. The pillars of Data Science is followed and described in Figure 1.1.

❖ **Domain Knowledge**

Data science relies most on domain expertise. Understand the problem domain, which can be medicine, finance, or retail. Domain knowledge includes knowing how many customers bought something at an online store in each month, which can help your business decide whether to hire more people or buy more inventory during busy periods like Black Friday sales season.

❖ **Math & Statistic Skills**

The problem determines the math and statistics needed. Math and statistics are varied, but you only need the essentials. Probabilities may be analysed using algebra.

❖ **Computer Science**

Data scientists need computer science expertise. No programming experience is required, but computer knowledge is. Every aspiring data scientist should know about data storage and processing concepts in addition to computer science fundamentals like algorithms and software engineering principles like object-oriented programming (OOP).

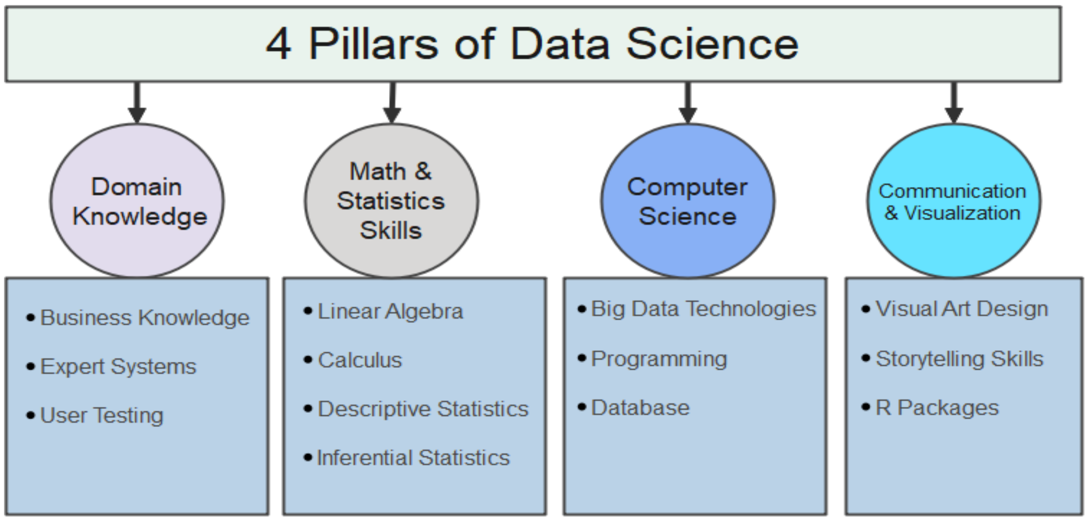


Fig 1.1. Pillars of Data Science

❖ **Communication & Visualization**

Data science relies on visualization to share ideas and explain data. Improve data comprehension. Visualizing a dataset can help you understand and uncover new information.

In summary, data science involves:

- ➢ Analysing raw data and asking the right questions.
- ➢ Using efficient algorithms to model data.
- ➢ Increased data visualization for better understanding.
- ➢ Using data for informed decision-making and outcome analysis.

1.2.2 Real-time Example

Data Science uses data from e-commerce sites, surveys, social media, and internet searches. This data access is feasible thanks to improved data collection tools.

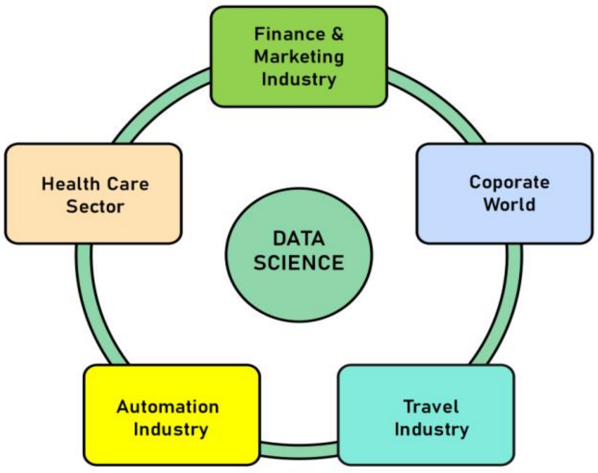


Fig 1.2. Data Science Sectors

This data helps businesses predict and profit. Data can be used to enhance sales funnels, determine which client segments favor products, and discover which discounts customers like best. Data science can boost company efficiency internally. Data science is the most talked about topic nowadays and a popular career choice due to its many sectors and are shown in Figure 1.2.

Example:

Amazon can provide suggestions based on product browsing history and customer ratings and complete idea shown in Figure 1.3. Users can receive product recommendations based on their preferences. This keeps consumers active on such sites and boosts corporation profitability.

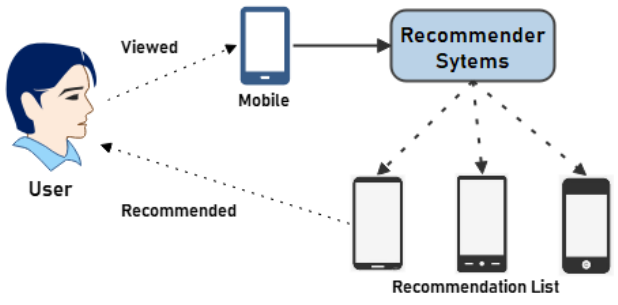


Fig 1.3. Recommendation System

Data science in healthcare is medical imaging. For diagnosis, doctors use X-rays, MRIs, and CAT scans to visualize inside body parts. Microscopic characteristics that indicate an injury, disease, or illness might be difficult to detect even with substantial training. X-rays may not show a hairline fracture in a bone, therefore doctors may use data science to evaluate them. These applications segment and detect anomalies in images. In addition to the above examples, the number of other tasks done by the data science in health care the complete details given in Figure 1.4.

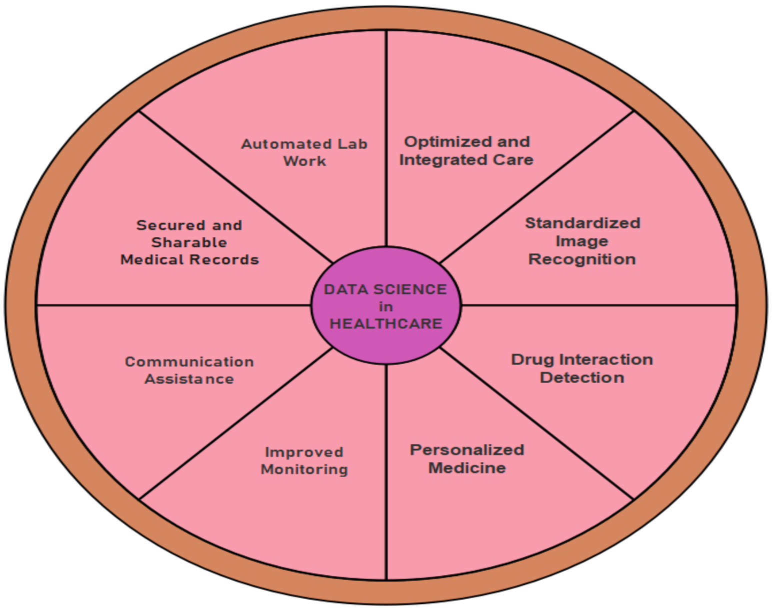

Automated Lab Work
Optimized and Integrated Care
Secured and Sharable Medical Records
Standardized Image Recognition
DATA SCIENCE in HEALTHCARE
Communication Assistance
Drug Interaction Detection
Improved Monitoring
Personalized Medicine

Fig 1.4. Data Science in Health Care

1.3 IMPORTANCE OF DATA SCIENCE

Companies are flooded with data. Data Science will combine methodology, technology, and tools to gain insights. Data science is used for predictive analytics, machine learning, data visualization, recommendation systems, fraud detection, sentiment analysis, and decision-making in healthcare, finance, marketing, and technology. The importance of data science in mentioned sectors are included in Figure 1.5.

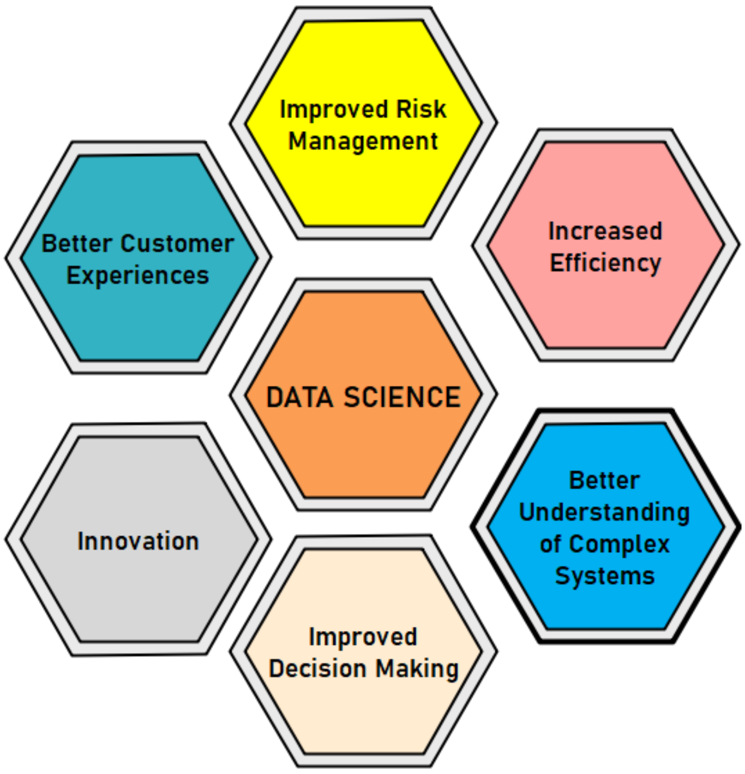


Fig 1.5. Importance of Data Science in Real World

❖ Improved Decision Making

In today's data-driven environment, data science powers decision-making. Data science can improve decision-making by providing analytical approaches and models to overcome psychological barriers to risk perception and belief creation []. Its data-driven approach helps firms realize data science's potential and align themselves. Data science helps firms make better decisions and calculate cost-benefit ratios. Decision support systems can optimize marketing and brand management with data science technologies like regression analysis and machine learning. Data science may also examine huge data and quickly interpret outcomes in decision support systems to help decision-makers develop better strategic strategies.

❖ Better understanding of complex systems

Complex systems are studied because they emerge, depend on beginning conditions, and have chaotic behaviour. Even though they follow simple principles and interact, they're unpredictable. Feature engineering in data science and machine learning identifies additional factors or simplifies the mathematics of the model's variables. Complex systems and data science also overlap in graph analysis.

❖ **Increased efficiency**

Data science finds operational inefficiencies and bottlenecks. Optimizing workflows and resource allocation streamlines operations, cuts costs, and boosts efficiency. A company may utilize data science to assess its supply chain and find bottlenecks causing delays. Based on this data, the company can improve delivery times and efficiency by changing their supply network.

❖ **Better customer experiences**

Data science helps companies understand customer behaviour, preferences, and needs. This insight helps adjust products and services to client needs, improving happiness and loyalty. A company can analyse client transactions and recommend products using data science. This may increase repeat business.

❖ **Improved risk management**

Decision-makers can make educated decisions without guesswork by using data-driven insights. Data science underpins evidence-based decision making, eliminating bias and subjectivity.

❖ **Descriptive analysis**

Data is analysed descriptively to understand what happened or is happening. Data visualizations include pie charts, bar charts, line graphs, tables, and produced narratives. Flight booking services may track daily ticket sales. This service's booking spikes, slumps, and best months will be shown using descriptive analysis.

❖ **Predictive Analytics**

Based on prior data, data science can predict future results. Analysing vast information using machine learning algorithms helps businesses spot trends and predict future events. Data science can help doctors anticipate who is most likely to get an illness and provide preventive care.

❖ **Prescriptive analysis**

Prescriptive analytics enhances predictive data. It anticipates the expected outcome and advises the best solution. It can evaluate options and suggest the best one. Graph analysis, simulation, complicated event processing, neural networks, and machine learning recommendation engines are used.

❖ **Personalized Marketing and Customer Segmentation**

Data science helps companies' segment and personalize marketing. By evaluating consumer data and behaviour, businesses may send targeted communications that boost customer engagement and conversion. This helps them grasp individual preferences and needs. For instance, a retail organization can use data science to identify high-value customers and create customized marketing campaigns or loyalty programs to retain them. Customer segmentation lets an e-commerce platform propose products based on a user's browsing history and buying patterns.

❖ **Efficient Resource Allocation**

Data science helps firms allocate resources by analysing demand, supply chain, and resource utilization. Waste is eliminated, operating efficiency is increased, and inventory, personnel, and equipment are properly allocated.

❖ **Continuous Improvement**

Data science helps organizations with a development culture. Data analysis helps organizations evaluate performance, track progress, and identify opportunities for improvement. Data-driven strategies promote continuous development and innovation.

❖ **Innovation and New Opportunities**

Finally, data science may help companies innovate and find new opportunities. Data science is driving innovation by giving companies new perspectives and untapped potential. Data science may also identify new company opportunities by analysing competition, market, and consumer behaviour. Data science-driven innovation goes beyond product production. In process innovation, firms use data analysis to identify inefficiencies, bottlenecks, and automation or optimization opportunities.

1.4 ADVANTAGES OF DATA SCIENCE

Today's world, data is becoming so vast, i.e., approximately 2.5 quintals bytes of data is generating on every day, which led to data explosion. It is estimated as per research, that by 2020, 1.7 MB of data will be created every single second, by a single person on earth. Every Company requires data to work, grow, and improve their businesses. Now, handling such a huge amount of data is a challenging task for every organization. So, to handle, process, and analysis of this, we required some complex, powerful, and efficient algorithms and technology, and that technology came into existence as data Science. Data science is revolutionizing the way companies operate. Many businesses, regardless of size, need a robust data science strategy to drive growth and maintain a competitive edge. And because of this huge amount of data the value of the field of Data Science has several advantages and disadvantages which are shown in Table 1. 1.

Table 1.1. Advantages and disadvantages of Data Science

Advantages	Disadvantages
Discover unknown transformative patterns	Data Privacy
Improving business decisions	Cost
Innovate new products and solutions.	Complete Understanding is not Possible
Innovate new products and solutions	
Giving internal financial information	
Mitigate Fraud and Risks	
Real-time optimization	
Multiple Job Options	
Business and Hiring benefits	

❖ **Discover unknown transformative patterns.**

Data science allows businesses to uncover new patterns and relationships that have the potential to transform the organization. It can reveal low-cost changes to resource management for maximum impact on profit margins. For example, an e-commerce company uses data science to discover that too many customer queries are being generated after business hours. Investigations reveal that customers are more likely to purchase if they receive a prompt response instead of an answer the next business day. By implementing 24/7 customer service, the business grows its revenue by 30%.

❖ **Improving business decisions**

Data and risk analysis techniques help in making wise business decisions. Higher-ups can make use of the data analysis and render unbiased support for complex business decisions.

❖ **Innovate new products and solutions.**

Data science can reveal gaps and problems that would otherwise go unnoticed. Greater insight into purchase decisions, customer feedback, and business processes can drive innovation in internal operations and external solutions. For example, an online payment solution uses data science to collate and analyze customer comments about the company on social media. Analysis reveals that customers forget passwords during peak purchase periods and are unhappy with the current password retrieval system. The company can innovate a better solution and see a significant increase in customer satisfaction.

❖ **Giving internal financial information**

Businesses can make the most of the benefits of data science in developing financial reports and examining economic patterns. and making projections for more-informed decisions on spending budgets. This ultimately results in generating income that is fully optimized and gives a clear picture of internal financial conditions.

❖ **Mitigate Fraud and Risks**

Your company may be able to strengthen security and safeguard potentially sensitive information by using data science. Machine learning algorithms can detect fraud by employing analysis of user behavior. Machine learning may be able to capture these events with high accuracy if big clumps of data are generated from these cases.

❖ **Real-time optimization**

It's very challenging for businesses, especially large-scale enterprises, to respond to changing conditions in real-time. This can cause significant losses or disruptions in business activity. Data science can help companies predict change and react optimally to different circumstances. For example, a truck-based shipping company uses data science to reduce downtime when trucks break down. They identify the routes and shift patterns that lead to faster breakdowns and tweak truck schedules. They also set up an inventory of common spare parts that need frequent replacement so trucks can be repaired faster.

❖ **Multiple Job Options**

Being in demand, it has given rise to many career opportunities in its various fields. Some of them are Data Scientist, Data Analyst, Research Analyst, Business Analyst, Analytics Manager, Big Data Engineer, etc.

❖ **Business benefits**

Data Science helps organizations know how and when their products sell best and that's why the products are delivered always to the right place and right time. Faster and better decisions are taken by the organization to improve efficiency and earn higher profits.

❖ **Highly Paid jobs & career opportunities**

As Data Scientist continues to be the sexiest job and the salaries for this position are also grand. According to a Dice Salary Survey, the annual average salary of a Data Scientist is $106,000 per year.

❖ **Hiring benefits**

It has made it comparatively easier to sort data and look for the best candidates for an organization. Big Data and data mining have made processing and selection of CVs, aptitude tests and games easier for the recruitment teams.

1.5. DISADVANTAGES OF DATA SCIENCE

Everything that comes with several benefits also has some consequences. So, let's have a look at some of the disadvantages of Data Science: -

❖ **Data Privacy**

Data is the core component that can increase productivity and the revenue of industry by making game-changing business decisions. But the information or the insights obtained from the data can be misused against any organization or a group of people or any committee etc. Extracted information from the structured as well as unstructured data for further use can also be misused against a group of people of a country or some committee.

❖ **Cost**

The tools used for data science and analytics can cost a lot to an organization as some of the tools are complex and require the people to undergo training in order to use them. Also, it is very difficult to select the right tools according to the circumstances because their selection is based on the proper knowledge of the tools as well as their accuracy in analyzing the data and extracting information.

❖ **Complete Understanding is not Possible.**

Data science is vast. It is utilized in mathematics, statistics as well as computer science. Thus, being perfect in all fields is not a simple task.

1. 6 THE PROCESS OF DATA SCIENCE

Data science is a methodical process that turns data into useful insights. From identifying the problem and analyzing the data to designing models, evaluating outcomes, and deploying solutions, each step is critical to data value extraction. The process involves the following phases and are shown in Figure 1.6.

1.6.1 Discovery

First comes discovery, which includes asking the proper questions. Before starting a data science project, you must identify its requirements, priorities, and budget. In this phase, we determine all project needs, such as the amount of people, technology, time, data, and end goal, and then frame the business challenge on a first hypothesis level.

1.6.2 Data preparation

Data preparation cleans, standardizes, and enriches raw data for analytics and data science. Data analysts struggle to get relevant data before analyzing. It takes data scientists about 80% of their work to prepare data for machine learning (ML) models. The 80/20 rule: Business analysis takes only 20% of data analysts' and scientists' time. The rest is spent gathering, cleaning, and organizing data.

This phase requires the following tasks:

- Cleaning of data
- Data reduction
- Integration and manipulation of data

❖ **Data cleansing**

It handles missing values, outliers, and discrepancies. Format data for analysis. Additional actions include removing duplicate values, irrelevant observations, missing values, reformatting data types, filtering undesired outliers, reformatting strings, validating, and more. To efficiently store huge amounts of originally sourced data as reduced data, data reduction techniques eliminate redundancy. More correctly, data reduction uses data deduplication and data consolidation to achieve its goals.

❖ **Data integration**

This involves merging and harmonizing data from numerous sources into a coherent format for analytical, operational, and decision-making reasons. Steps in data integration include:

1.6.3 Model Planning

We must determine the strategies and procedures to construct input variable relationships in this phase. Exploratory data analytics (EDA) will use statistical formulas and visualization tools to identify variable relationships and learn from data.

1.6.4 Model-building

This phase begins model building. Data analytics requires model building to gain insights and inform company decisions and strategy. The data science team must create training, testing, and production data sets during this phase. These data sets let data scientists train an analytical approach and save some data for model testing. Data analytics model creation aims for high accuracy on training data and generalization and performance on fresh data. Instead of memorizing the training data, the goal is to create a model that captures data patterns and correlations.

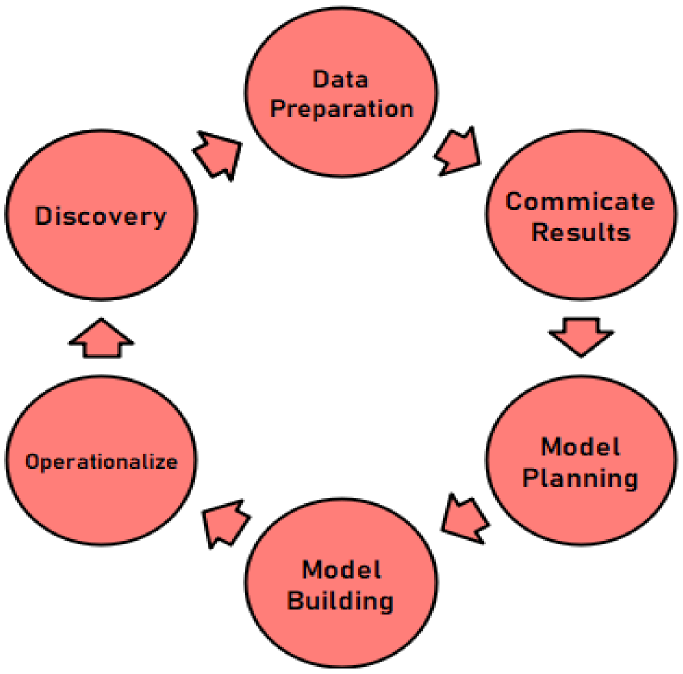


Fig 1.6. Key Steps in the Data Science Process

1.6.5 Operationalize

The project's final reports, briefings, code, and technical documents will be delivered in this phase. Before deployment, this phase gives you a small-scale view of project performance and other components. If your model performed better than predicted, you can start measuring real-world efficacy with a small-scale pilot project in a real-time production setting. This will show any unexpected limits that must be considered before using your model. Processing model outputs online outside the sandbox requires an appropriate API.

1.6.6 Communicate results:

In this phase, we'll see if we met our first target. Our results and conclusion will be shared with the business team.

If your procedure must be improved to improve results, start over at phase one with a more specific problem. Each refinement brings your model closer to real-time deployment.

1.7 WELL-DEFINED DATA SCIENCE PROCESS

By following a well-defined process and using proper methodologies, organizations may use data science to make informed decisions, acquire a competitive edge, and uncover new opportunities in the data-driven era.

A clear Data Science approach has many benefits and are described below and shown in Figure 1.7. :

- **Efficiency**: A planned methodology ensures project efficiency and effectiveness. This aids commercial decision-making.
- **Collaboration**: A defined procedure helps stakeholders collaborate with the data science team.
- **Reproducibility**: A clear process helps other data scientists understand and replicate it.
- **Domain Agnostic**: Data Science is domain neutral, meaning it may be used to any industry having data available]. This makes Data Science useful for tackling challenges across fields.
- **Reduces errors:** This technique minimizes analysis errors and biases. Data scientists can spot and fix mistakes early by breaking down the task into smaller pieces.
- **Increases transparency:** Data science transparency increases by explaining how results were obtained. Establishing stakeholder trust and credibility is crucial.
- **Enables continuous improvement:** Data scientists improve performance by evaluating and refining models. This helps companies outperform competitors and make better judgments.

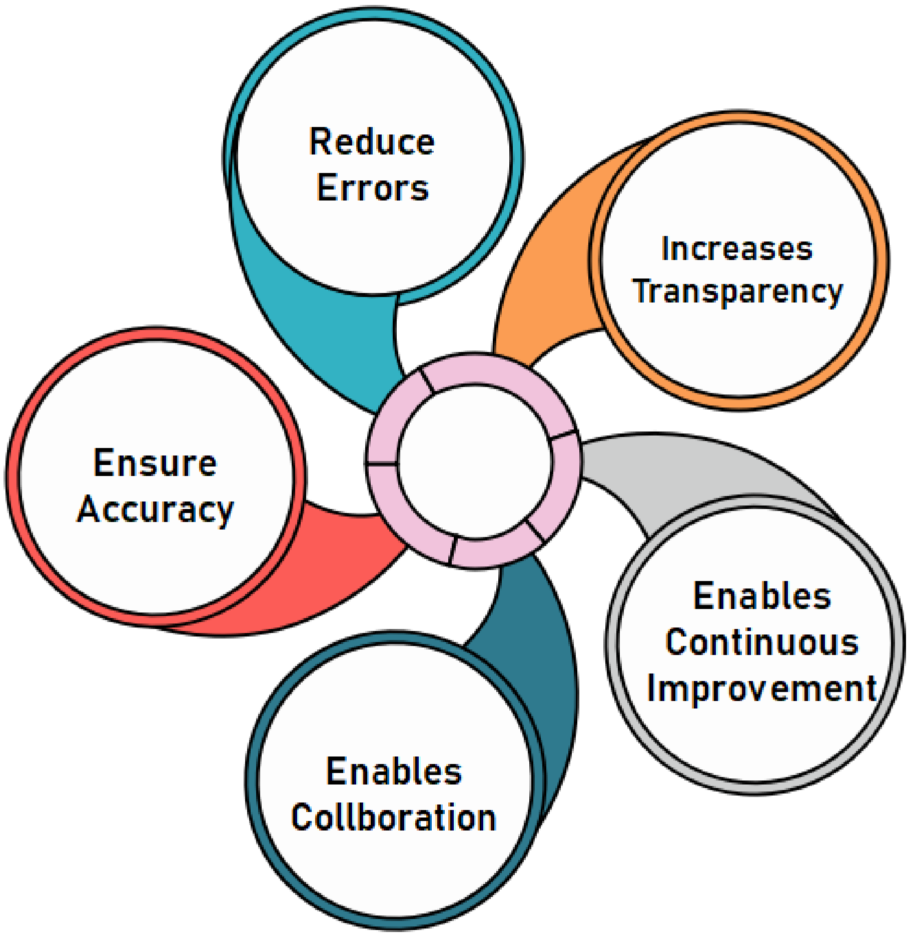


Fig 1.7. Benefits of Well-Defined Data Science

1.8 SUMMARY

Our world is digital and will become increasingly so. Data science, which helps organizations obtain insight and intelligence from their own records, will increase and become even more vital, helping them grow and compete. Data scientists are and will be essential to every IT organization, making data science one of the most promising fields in research and enabling the development of new technologies like AI and ML.Data science has several benefits for companies in various industries. Data science uses advanced statistical analysis, machine learning, and computer science to help organizations enhance decision-making, efficiency, customer experience, competitiveness, and new opportunities. Data-driven insights can improve decision-making, identify inefficiencies, and optimize operations to save money. Data science helps firms customize marketing strategies, forecast results, improve healthcare service, spend resources efficiently, and foster a continuous improvement culture.

1.9 TECHNICAL TERMS

Data Science, Machine Learning, Deep Learning, Artificial Intelligence, Health Care, Business Process, Efficiency and Collaboration.

1.10 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Illustrate about data science process.
2. Describe about importance of data science.
3. Explain about advantages and disadvantages of data science/

Short Notes:

1. Write is data science.
2. Discuss about applications of data science.
3. List out benefits of well-defined data science process.

1.11 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.

2. Reemathareja – Python Programming using problem solving approach, Oxford Publication

3. "Python Data Science Handbook" by Jake VanderPlas - This comprehensive book covers essential tools and techniques for data science in Python, including NumPy, pandas, matplotlib, scikit-learn, and more.

4. "Data Science from Scratch: First Principles with Python" by Joel Grus - This book teaches data science concepts from the ground up using Python, covering topics like statistics, machine learning, and data manipulation.

Dr. KAMPA LAVANYA

LESSON- 2

THE ROLE OF DATA SCIENTIST IN DATA SCIENCE

AIMS AND OBJECTIVES

The primary goal of this chapter is to understand the role of data scientist in the field of data science. The chapter began with understanding who is data scientist, responsibilities of data scientist, qualifications to become a data scientist and so on. After completing this chapter, the student will understand how to be a data scientist how to become a good data scientist.

2.1. INTRODUCTION

A field of study known as Data Science integrates several disciplines, including statistics, data analysis, and machine learning, to extract insights and knowledge from data. Data science is a multidisciplinary discipline that extracts insights and knowledge from data using computational and statistical methods. It requires domain expertise in addition to skills and knowledge from diverse disciplines, including computer science, statistics, and mathematics. Data science encompasses a series of sequential stages, which comprise data collection, cleansing, investigation, analysis, and interpretation.

The ability to create code is the most fundamental and ubiquitous talent that data scientists possess. This may be less true in five years, when many more people will have the title "data scientist" on their business cards. More persistent will be the requirement for data scientists to communicate in language that all their stakeholders understand, as well as to exhibit the specialized abilities required for storytelling with data, whether verbally, visually, or—ideally—both.

This chapter will cover the basic responsibilities of data scientist. Also provided qualifications and tips to become a good data scientist.

2.2 DATA SCIENTIST RESPONSIBILITY

Data scientists work differently based on the organization's size and needs. They usually follow the data science process, although details vary. A data scientist may collaborate with analysts, engineers, machine learning experts, and statisticians to complete the data science process and meet business goals in bigger data science teams. Data scientists use a variety of methods, tools, and technologies. They choose the ideal combinations for faster, more accurate outcomes based on the problem. Data scientists use machine learning to model and interpret data, unlike data analysts. Synthesize and convey results to key stakeholders to promote organizational strategic decision-making. Some of the data scientist in present market shown in Figure 2.1.

2.2.1. Types of Data Scientists

We can distinguish three main types of data scientists:

1. Traditional data scientists
2. Research scientists.
3. Applied scientists.
4. Engineering data scientist
5. Operational data scientists
6. Product-focused data scientists

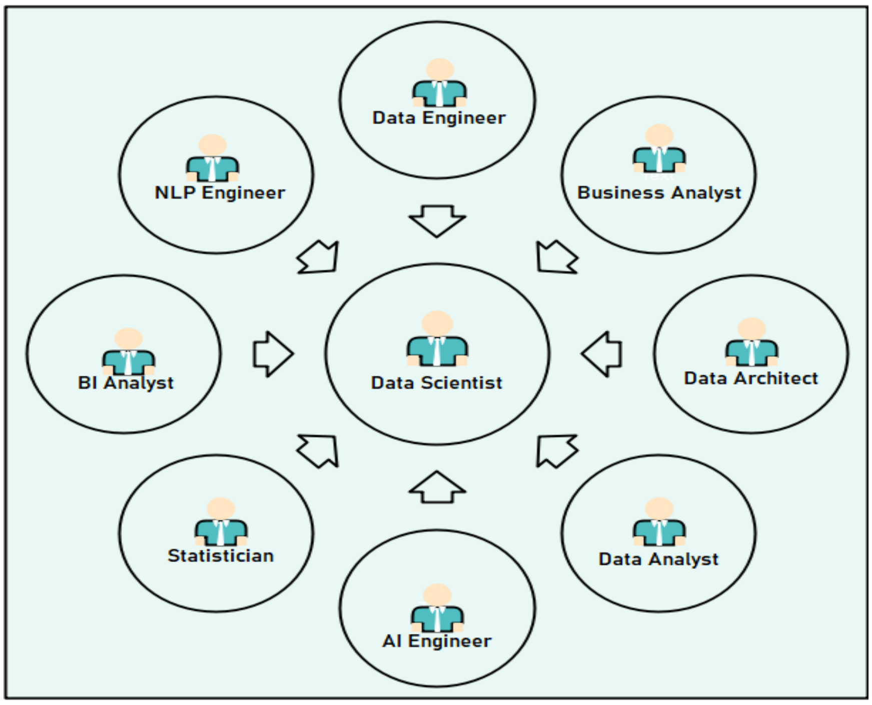


Fig 2.1. Data Scientists in Market

❖ **Traditional data scientists**

Traditional data scientists explore data, perform complex statistical modelling, A/B test, and construct and tune machine learning models.

- **Research scientists.**

Focus on creating new machine learning models for huge enterprises. This describes a data scientist who seeks innovative methods and algorithms. The typical data scientist has a PhD in machine learning and works for Google, Deep Mind, or IBM. This is the most glamorous data scientist since they understand data science and have strong academic credentials. However, most firms don't need their whole skillset or may not have the infrastructure to support them. Companies often hire someone with these talents when they need an operational or product-focused data scientist.

- **Applied scientists.**

Big tech and larger organizations hire data scientists for one of the highest-paying occupations. These experts produce models using data science and software engineering. Applied scientists can model data for machine learning, choose an algorithm, train the model, fine-tune hyperparameters, and deploy the model.

- **Engineering data scientist**

This job is very technical. Data scientists like this are similar to data engineers (some say they are the same). Engineering data scientists ensure system stability and construct scalable pipelines.

- **Operational data scientists**

These data scientists know the business and its systems. Technical issues become business goals, and data science helps a company enhance efficiency or accomplish goals. An operational data scientist may utilize more advanced modelling than a business analyst, who may use dashboards.

- **Product-focused data scientists**

These data scientists develop or improve products using data science. They may be on a company's product team. This type of data scientist develops a company's website recommendation system.

2.2.2. Differences between Data Scientist and Data Analytics

The comparison of Data Scientist and Data Analytics is shown in Table 2.1.

Table 2.1. Difference between Data Scientist and Data Analytics

Data Scientist	Data Analytics
Conduct research on both historical and contemporary patterns.	Investigate and retrieve the information.
Create reports on both the operations and the finances.	An examination of the data collected statistically.
Utilize applications like Excel to carry out forecasting.	Support for the training and development of deep learning frameworks.

Make use of infographics.	Build an architecture that can manage massive amounts of data.
Understand the data and explain in a clear manner.	Create automation that makes the process of data collection and processing more efficient.
Through the examination of documents and the correction of data corruption, carry out data screening.	Make your observations known to the leadership team and help with making decisions based on the data.

2.2.3 Business Process with a Data Scientist

A data scientist uses several techniques to solve business problems, such as:

- ❖ By asking the proper questions and acquiring insight, the data scientist determines the problem before beginning the process of collecting and analyzing data.
- ❖ The appropriate combination of factors and data sets is then identified by the data scientist.
- ❖ The data scientist collects both organized and unstructured data from a variety of sources, including public and enterprise data.
- ❖ After gathering the data, the data scientist prepares it for analysis by processing and formatting the raw data. To ensure consistency, accuracy, and completeness, the data must be cleaned and validated.
- ❖ The data is input into the analytical tool once it has been transformed into a useable format.
- ❖ The data scientist analyzes the data to identify possibilities and solutions after it has been fully produced.
- ❖ The data scientists complete the work by explaining the findings and producing the insights to be shared with the relevant parties.

2.2.4 Responsibilities of Data Scientist

Data scientists are crucial to ensuring that businesses make well-informed decisions. They collaborate closely with company executives to pinpoint particular goals, like determining client segmentation and promoting enhancements to goods and services. Data scientists may analyze huge datasets to find trends and insights that support businesses in making wise decisions. They do this by using sophisticated machine learning algorithms and statistical models. A mix of technical expertise and understanding of data interpretation and visualization is typically possessed by data scientists. They need to be knowledgeable with database management systems, machine learning techniques, programming languages, and statistical analysis.

Let's examine the duties performed by a qualified data scientist in summary form.

- Compiling, sanitizing, and arranging data for prescriptive and predictive models.
- Examining enormous volumes of data to find patterns and trends.
- Constructing the data and turning it into information that can be used by using programming languages.
- Collaborating with stakeholders to comprehend business issues and create solutions based on facts.
- Creating predictive models to predict future trends using statistical methods.
- Creating, preserving, and keeping an eye on machine learning models. Creating data-driven solutions by developing and applying cutting-edge machine learning algorithms and other analytical techniques.
- Use a range of data mining technologies to find hidden patterns and trends in large datasets.
- Creating and approving data solutions using dashboards, reports, presentations, and data visualizations.

2.3 QUALIFICATION OF DATA SCIENTIST

Degrees in quantitative disciplines like math, statistics, computer science, physics, or information technology are ideal for anyone who wants to work in data science. By completing classes in college, you can frequently specialize in a particular field of data science. If you want to work as a marketing analyst, for example, business and marketing courses can be helpful. If you want to go into data reporting, a foundation in journalism and writing-intensive disciplines can be helpful. Since data science is used in all industries, you can have a strong foundation to build from if you have a particular understanding of a field you enjoy. For instance, if you want to work in data science in an investment bank, having a background in finance is ideal.

When transitioning into more business-focused professions (as opposed to focusing on analysis or engineering), advanced degrees can be helpful for upward mobility. To have a deeper understanding of the business aspect of their profession, data scientists frequently seek Master of Business Administration (MBA) degrees.

If you're interested in pursuing a career in data science, review the requirements listed below:

- Proficiency in statistics, mathematics, computer science, or information technology.
- Strong problem-solving abilities
- Capable of working in a team.
- Enjoy manipulating data.
- Possess strong communication abilities.
- Willing to pick up the newest tech.

2.3.1 Data Scientist Skills

Major competencies are required for employment as a data scientist. Candidates for various data science jobs must be well-versed in the most recent developments in technology. The following are some of the key competencies required to succeed in this department:

- **Statistics**

Academic and professional statistics collects, analyzes, and interprets data. Statistics professionals must also share their findings. Thus, data scientists need statistics to collect, evaluate, and report on vast amounts of organized and unstructured data.Data scientists must master statistical methods to identify hidden patterns and correlations between data features.

- **Descriptive Statistics**

A data set's basic properties are analysed and identified using descriptive statistics. Descriptive statistics summarize and visualize data. Many raw data sets are hard to summarize and communicate. Descriptive statistics provide effective data presentation.

- **Probability Theory**

According to Encyclopaedia Britannica, open_in_new mathematics measures random event probability. A random experiment is a physical condition with an unpredicted outcome. Like coin flipping. Probability is a number between zero and one that indicates the possibility of an event. Higher probabilities (closer to one) increase likelihood. Flipping a coin has 0.5 probability since heads and tails are equally likely.

- **Machine Learning**

Data and algorithms are used in machine learning (ML) to teach AI to learn like humans, enhancing its accuracy. Machine learning algorithms usually predict or classify. Your algorithm will estimate a pattern from labeled or unlabeled input data. Data scientists must understand model-building methods to train machines.

- **Computer Science**

A Data Scientist must employ software engineering, database system, AI, and numerical analysis concepts.

- **Programming**

Data Scientists need at least one programming language to use the proper algorithms. They must be comfortable writing Python, R, and SQL code.

- **Analytical and Critical Thinking**

Business problems require analytical thinking from a Data Scientist. Data Scientists must think critically before drawing conclusions.

- **Interpersonal Skills**

Data Scientists must communicate well with various company audiences.

2.3.2 Data Science in Latest Domains

Here, data science is investigated in a wide range of sectors and offers some of the most recent developments based on data science's daily activities.

The domains include the following: business, education, healthcare, and the IT industry and complete details is shown in Table 2.2.

Table 2.2. Data Science in Latest Domains

Education	**Business**	**Healthcare**	**IT industry**
• New Learning Methods	• Predictive Analytics for product outcomes	• Managed and analyzed effectively to obtain true findings.	• analysing data
• Finding the Issues of Students	• Improve decision-making across the organization	• managed demograph ics, treatment plans, outcomes of medical exams, insurance data of health care	• Developing ne w strategies.
• Improve the Educational Organization	• improve consumer engagement, corporate performance, and boost revenue	• Decision-making for the healthcare system by offering useful insights.	• Preparing data for analysis, visualization of data.
• Provide Opportunities for Students	• Identifying and refining target audiences	• creating a complete picture of clients, patients, and professionals	• building model s with data using different programming languages,
• Continuous Monitoring and Updating	• Managing business efficiency	• improve healthcare quality	• Deploying mo dels into application.
• Observing and Evaluating Teacher Performance	• Automating Recruiting Process	• medical improvements	• Testing models

2.3.3 Data Scientist Job Role

Data Scientists address business challenges with statistics and arithmetic. Data scientists should be able to write business proposals, design predictive models, solve business challenges, and tell stories to visualize data for clients. Data Scientists with computer programming skills can improve corporate decisions, address real-world problems, and apply their knowledge. Statisticians develop models by applying statistical approaches to data.

Computer programming, statistics, and mathematics are needed for a Data Scientist. The complete job role related to Data Scientist is described in Table 2.3.

Table 2.3. Data Scientist Job Role

Job Role	Data Scientist
Eligibility	• Bachelor's degree in computer science or relevant field • Certification course in Data science with relevant project
Responsibilities	• Analyzing data and extract important insights from it. • Decision making using the data. • Using various tools and techniques to ensure maximum use of data.
Average salary	• 5 to 6 Lacs per annum (for freshers)
Experience	• 0-1 years for freshers

2.4 WOULD YOU BE A GOOD DATA SCIENTIST

Although data scientists have distinct skills or jobs, they all need a few things to succeed. Their business partners must help them integrate into the core business and product line. Data partners include software application and data infrastructure engineers. These professionals assure accurate, full, and accessible foundational data instrumentation and feeds. They require leaders who will invest in data quality, management, visualization and access platforms, and a culture that values data in business and product development. This requires allocating time for data and measurement during development, which is often overlooked. The advantages and disadvantages of a good data scientist is shown in Table 2.4.

Table 2.4. Advantages and Disadvantages of a Good Data Scientist.

Advantages	Disadvantages
❖ Discover unknown transformative patterns	❖ Data Privacy
❖ Improving business decisions	❖ Cost
❖ Innovate new products and solutions.	❖ Complete Understanding is not Possible
❖ Innovate new products and solutions	
❖ Giving internal financial information	
❖ Mitigate Fraud and Risks	
❖ Real-time optimization	
❖ Multiple Job Options	
❖ Business and Hiring benefits	

Finally, when employing data scientists, look for people that love solving issues, not specific answers, or approaches, and who are very collaborative. Whatever type of data scientist you hire, they must be able to collaborate with engineers, product managers, marketers, and executive teams. Finally, seek honest people. We must appreciate and use data for good as a society. Data scientists are responsible for data governance inside and outside their company.

2.5 SUMMARY

The position of a Data Scientist is necessary for companies that want to make decisions based on the data they collect. Data scientists are tasked with the responsibility of gathering, organizing, analyzing, and interpreting data to uncover patterns through correlations and trends. The development of data processing pipelines, the design of reports and dashboards, and the creation of models to forecast future trends are all among their responsibilities. To be successful in the industry, they need to understand the business context as well as the requirements of the client. In this chapter, we have seen complete responsibilities, role, qualifications, and other details regard data scientist.

2.6 TECHNICAL TERMS

Data Scientist, Machine Learning, Deep Learning, Data Analyst, Job Role, Responsibility, Good Scientist.

2.7 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Illustrate about responsibilities of data scientist.
2. Describe about qualifications of data scientist.
3. Explain about how to become a good data scientist.

Short Notes:

Write about skills required for data scientist.

1. Discuss about business process followed by data scientist.
2. List out advantages and disadvantages of good data scientist.

2.8 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication

3. "Python for Data Analysis" by Wes McKinney - This book focuses on practical data analysis using Python's tools and libraries, particularly pandas.

4. "Data Science Handbook" by Jake VanderPlas - This book covers various aspects of data science, including data manipulation, visualization, and machine learning using Python.
5. "Introduction to Machine Learning with Python: A Guide for Data Scientists" by Andreas C. Müller and Sarah Guido - This book provides an introduction to machine learning concepts and their implementation in Python using libraries like scikit-learn.
6. "Python Data Science Handbook" by Jake VanderPlas - This comprehensive book covers essential tools and techniques for data science in Python, including NumPy, pandas, matplotlib, scikit-learn, and more.

Dr. KAMPA LAVANYA

LESSON- 3

INTRODUCTION TO PYTHON

AIMS AND OBJECTIVES

The primary goal of this chapter is to understand the introductory concept of Python programming. The chapter began with an understanding of what is python, feature of python, History of python and so on. After completing this chapter, the student will understand the complete basic concepts of python in detail with suitable examples.

STRUCTURE

3.1. INTRODUCTION

Python is a high-level computer language that is interpreted and object-oriented. Its semantics change over time. Its high-level built-in data structures, along with dynamic typing and dynamic binding, make it a great choice for Rapid Application Development. Python's grammar is simple and easy to learn. It focuses on readability, which lowers the cost of maintaining programs.

Python lets you use modules and packages, which makes it easier to break up programs into smaller pieces and reuse code. For all major systems, you can get the Python interpreter and the large standard library for free in source or binary form, and you can share them with anyone else.

This chapter will cover the major basic concept of python programming includes what is python, history of python, advantages and disadvantages of python, applications of python etc.

3.2. WHAT IS PYTHON?

Python is a high-level, interpreted, object-oriented language with dynamic semantics. Its dynamic typing and dynamic binding, along with its high-level built-in data structures, make it an appealing language for Rapid Application Development and for usage as a scripting or glue language to join existing components. Because of its straightforward, basic syntax, Python promotes readability, which lowers software maintenance costs.

Python's support for packages and modules promotes code reuse and program modularity. The large standard library and the Python interpreter are freely distributable and accessible for free on all major platforms in source or binary form.

3.1.1. Advantages and Disadvantages of Python

Python language is holds number of advantages and disadvantages which are shown in Table 3.1.

Table 3.1. Advantages and Disadvantages of Python

Advantages	Disadvantages
1. Easy to learn, read, and understand.	1. Restrictions in design
2. Versatile and open source	2. Memory inefficient
3. Improves productivity.	3. Weak mobile computing
4. Supports libraries.	4. Runtime errors
5. Huge library	5. Slow execution speed
6. Strong community	
7. Interpreted language.	

3.1.2. Companies used by Python.

This is a list of the best companies that use Python on a regular basis. Some of the names on the list provided below may surprise you which are shown in Figure 3.1.

- ➢ Facebook
- ➢ Instagram
- ➢ Spotify
- ➢ Reddit
- ➢ Uber
- ➢ Netflix
- ➢ Google
- ➢ Dropbox

Fig.3.1. Companies Use Python

3.1.3. Applications of Python

Python is emerging language and is used in wide range applications and are described detailed in below and is shown in Figure 3.2.

- **Web Development**

Python's simplicity and features make it popular for web development. Python frameworks allow them to build user-friendly dynamic websites. The frameworks include Django for backend development and Flask for frontend. Because Python is easy to deploy, scalable, and efficient, most online companies utilize it as their primary technology. Top Python applications include web development, which is used across the business to build effective websites.

- **Data Science**

Python snippets help data scientists develop effective AI models. Its simplicity lets developers design complicated algorithms. Data science creates models and neural networks that learn like human brains but are faster. It helps organizations make decisions by extracting patterns from prior data. This field helps organizations invest in the future.

- **Artificial Intelligence and Machine Learning**

Data analysis and machine learning specialists can use Pandas and TensorFlow for statistical analysis, data manipulation, etc. One of the most popular programming languages is Python. The language of AI and ML is Python. Python has helped this field with its many libraries and community support. Python use will rise as artificial intelligence and machine learning evolve significantly.

- **Game Development**

Python developers can use Pygame to create 2D and 3D games. Pirates of the Caribbean, Battlefield 2, and others are popular Python games. Pygame is a Python library for making fun games. Since the gaming industry is growing, these types of development have become more popular. This package makes game development easy, so you can try building some simple games.

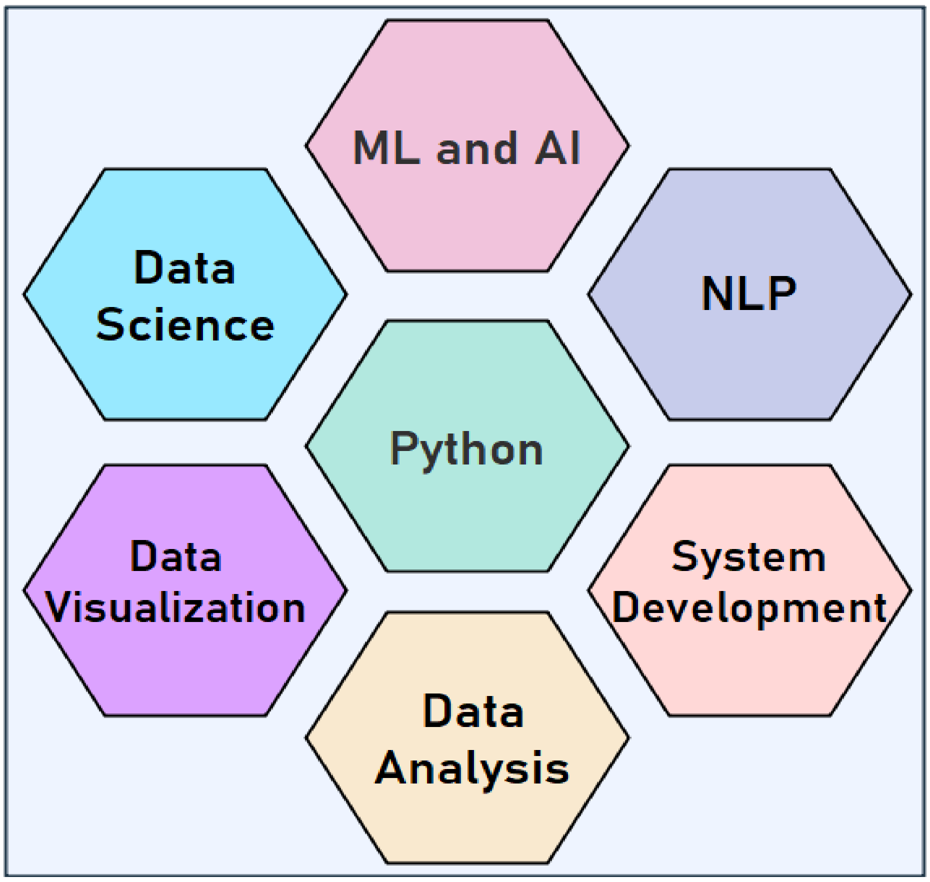


Fig 3.2. Applications of Python

3.3 FEATURES OF PYTHON

There are several characteristics that distinguish the Python programming language from others the main reason is its features and described below and shown in Figure 3.3.

❖ **Popularity**

Python is the fourth most popular and fastest-growing programming language, according to the Stack Overflow Developer Survey 2022. Businesses including Google, Instagram, Netflix, and Spotify use it.

❖ **Interpretation**

Python is an interpreted language; unlike compilers, which need the creation of machine code from the source code before it can be executed, Python passes directly to the interpreter, simplifying and speeding up the execution process.

❖ **Open Source**

The fact that Python is a free language created under an open-source license certified by OSI is among its strongest features.

❖ **Portability**

Major trouble comes in transferring a code from one platform to another without making blunders in the command. Python programming language, being a portable code can easily be transferred without making any errors.

❖ **Simplicity**

The only programming language that is similar to English is Python. It's so simple to read and comprehend. The Python programming language utilizes fewer keywords than C++ or Java. As a result, developers everywhere now favor the Python language above all others.

❖ **A high-level language**

Compared to several other programming languages, Python is more similar to human languages. As a result, its core features, such memory management and architecture, are unimportant to programmers.

❖ **An object-oriented language**

Python is a programming language that supports a variety of programming styles, including structured and functional programming, in addition to the standard object-oriented programming paradigm.

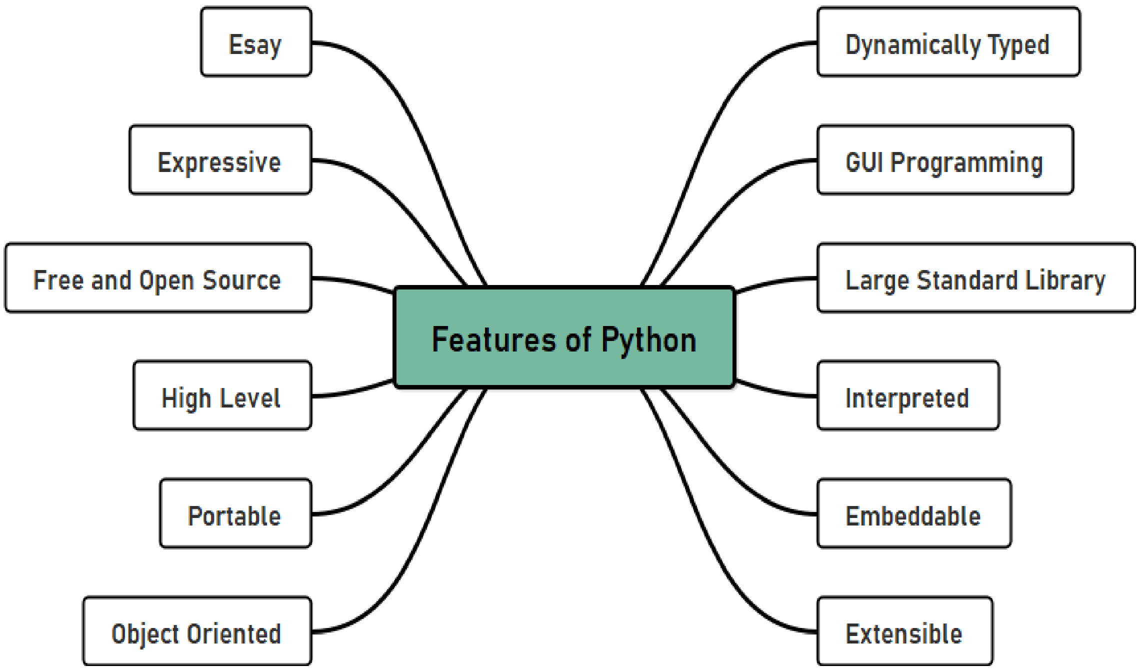


Fig 3.3. Features of Python

3.4 HISTORY OF PYTHON

Python was created by Guido van Rossum in 1980s. While in the Netherlands' National Research Institute for Mathematics and Computer Science, he created Python, an easy-to-read and use programming language. This programming language was called after the Pythons from Monty Python's Flying Circus, the founder's favorite comedians.

The first version, launched in 1991, contained few built-in data types and rudimentary capabilities. Python 1.0 was introduced in 1994 with map, lambda, and filter functions after scientists adopted it for numerical computations and data analysis. After that, adding features and releasing updated Python versions became popular. Python 1.0 introduced map, filter, and reduce methods in 1994 to process lists. Unicode support and a shorter list loop were added to Python 2.0 on October 16, 2000. Python 3.0 debuted December 3, 2008. It added print and number division support and error handling.

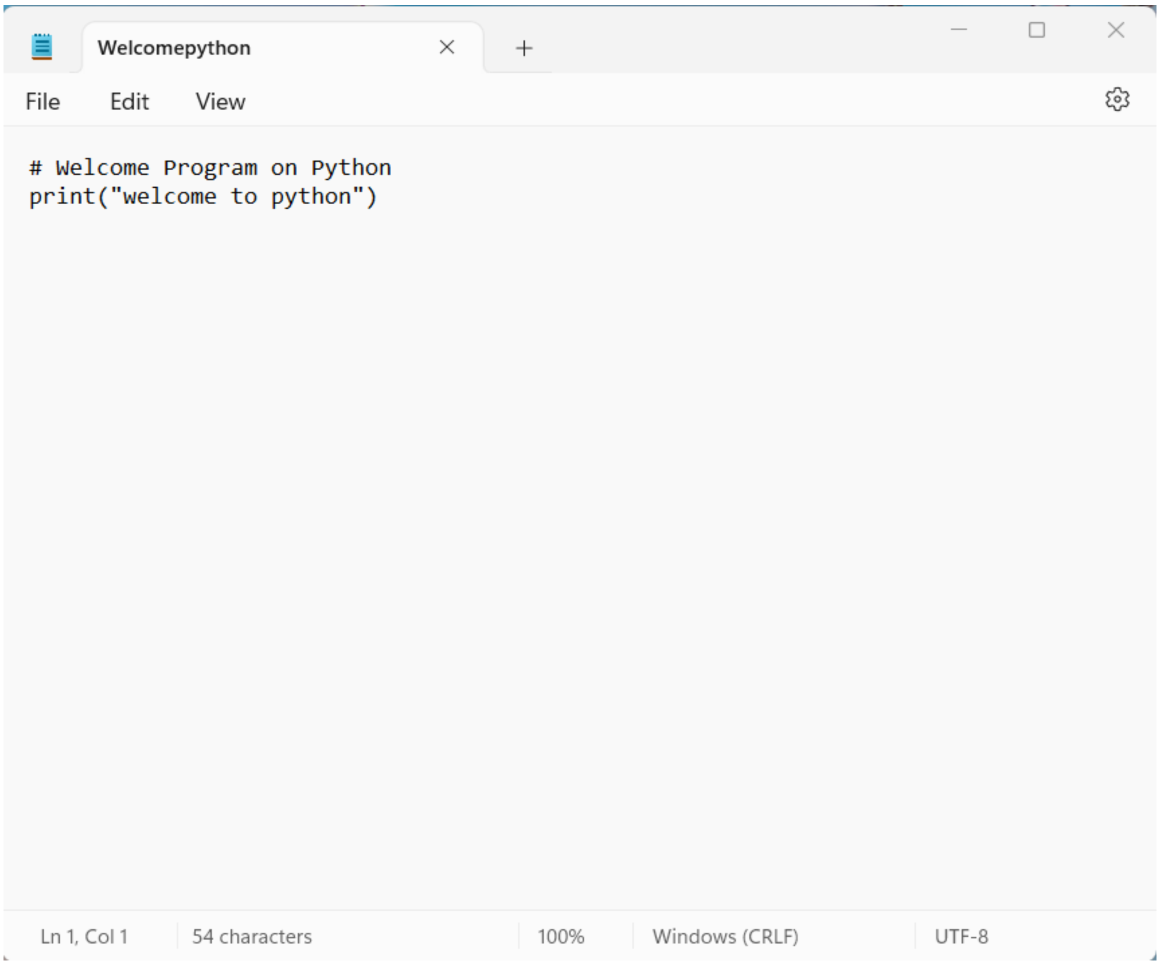
Python's new features benefit developers and boost performance. Python has grown in popularity and is a challenging programming language. It's in demand in machine learning, AI, data analysis, web development, and more, offering high-paying jobs. Python became the major programming language for many programmers and developers worldwide.

3.5 HOW TO WRITE AND RUN A PYTHON SCRIPT

Python programmers need to be familiar with all possible script and code execution scenarios. There is no other way to confirm that the code is functioning as intended. The Python programs are executed by the Python interpreter. A Python interpreter is a software that functions as a bridge between computer hardware and Python programs.

Here, we'll go over the various methods for executing Python programs. The simple program is created using notepad is shown in Figure 3.4.

- The operating system command-line or terminal.
- The Python interactive mode.
- The IDE

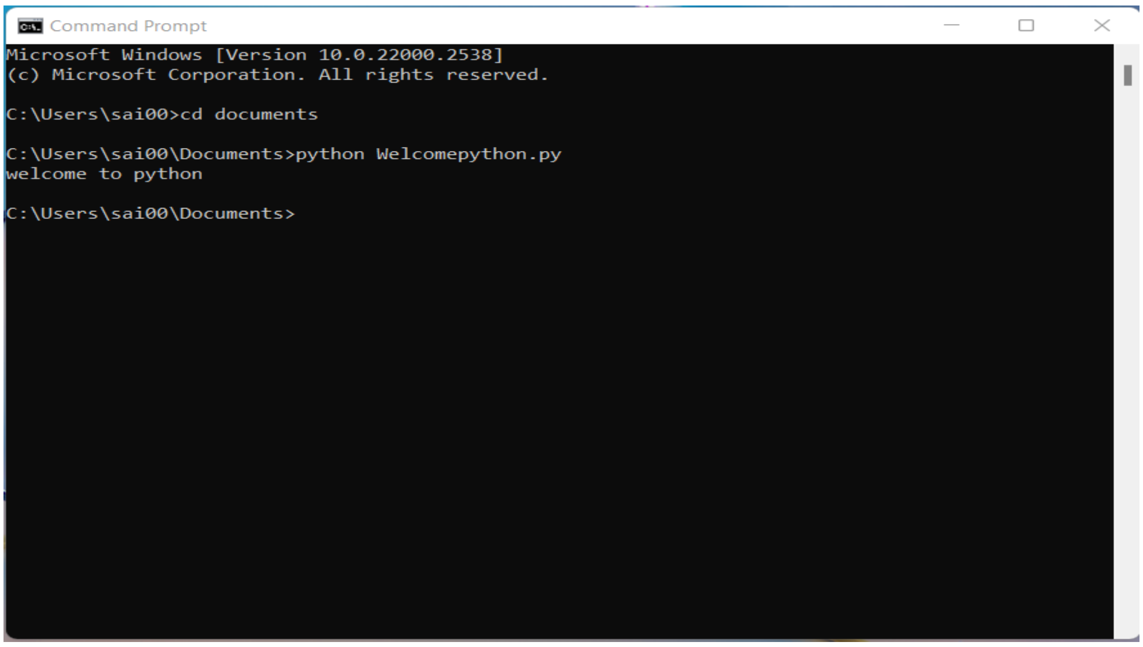


```
# Welcome Program on Python
print("welcome to python")
```

Fig 3.4. A sample python program created and saved on notepad.

3.5.1 The operating system command-line or Terminal

Since the Python shell loses all the code we write when the session is closed, we can run the Python code using a command line. Thus, using plain text files to write Python code is an excellent idea. The text file needs to be saved with the.py suffix. The Python print statement is written and saved in the working directory as welcomepython.py. We are going to use the command-line to execute this file now. To run a Python script, open a command line. To run the file, we must input the file name and then Python. Once you press the enter key, the result will look like this if there are no errors in the file and is shown in Figure 3.5.



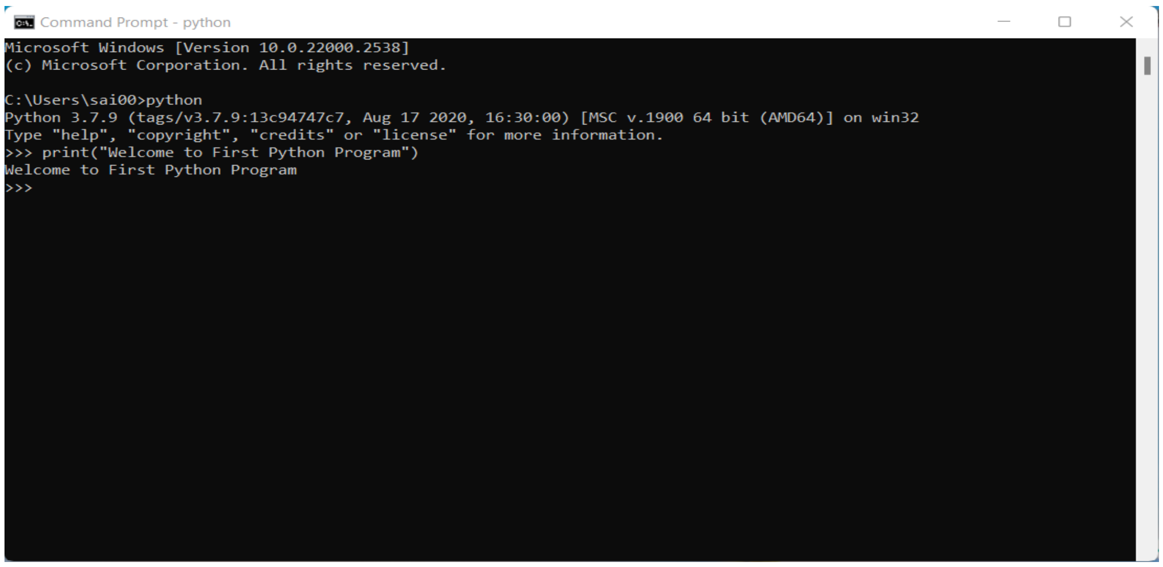
```
Microsoft Windows [Version 10.0.22000.2538]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sai00>cd documents

C:\Users\sai00\Documents>python Welcomepython.py
welcome to python

C:\Users\sai00\Documents>
```

Fig 3.5. Command Line to Run python program.

3.5.2 The Python Program Create and Run on Interactive Shell

We can utilize the Python interactive session to write and execute the Python code. To launch an interactive Python session, simply select a command-line or terminal from the Start menu, type python, and hit the Enter key. It is a fantastic development tool because it enables us to review every line of code. However, all of our written code will be lost when the session ends. To exit the interactive shell, type quit(), exit(), or press the Ctrl+Z key.

This is an illustration of how to use an interactive shell to run Python code is shown in Figure 3.6.



```
Microsoft Windows [Version 10.0.22000.2538]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sai00>python
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Welcome to First Python Program")
Welcome to First Python Program
>>>
```

Fig 3.6. Create and Run python program in Interactive Shell.

3.5.3 How to Run Python Program on IDLE?

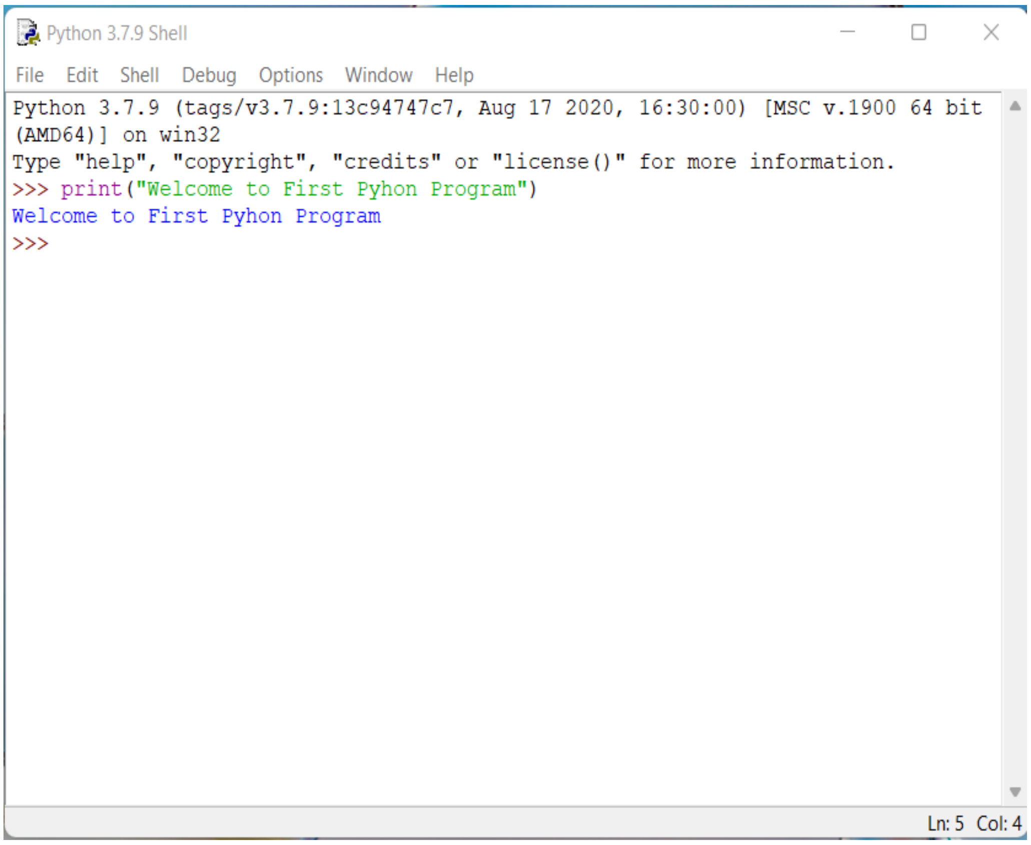
Windows and Mac Python installations contain Python IDLE. If you use Linux, you should be able to utilize your package manager to locate and download Python IDLE. After installation, Python IDLE can be used as a file editor or as an interactive interpreter.

Python is included with DLE, an Integrated Development and Learning Environment. A complete development environment for authoring, debugging, and testing code is offered by IDLE.

You must do the following actions to launch a Python application on IDLE:

Step 1: Launch the Python IDLE first. Since IDLE operates in the shell by default, this window will appear on your screen.

Step 2: Using the IDLE, we can create and run Python scripts and see the results directly on the screen, and is shown in Figure 3.7.



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Welcome to First Pyhon Program")
Welcome to First Pyhon Program
>>>
```

Fig 3.7. Create Python scripts in IDE.

Step 3: Open a new file by selecting File → New File in order to run a whole Python program on IDLE.

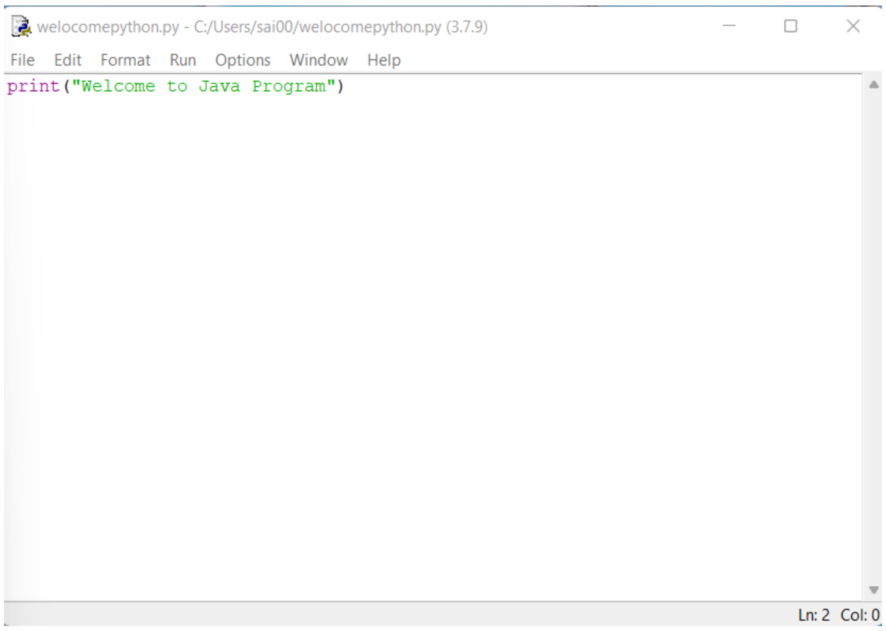
Step 4: Write your Python program in the "New File" that appears when the previous step is completed shown in Figure 3.8.

Step5: Save your file in this step. It is saved here under the filename welcomepython.py.

Step 6: Click RUN → Run Module to start the process shown in Figure 3.9.

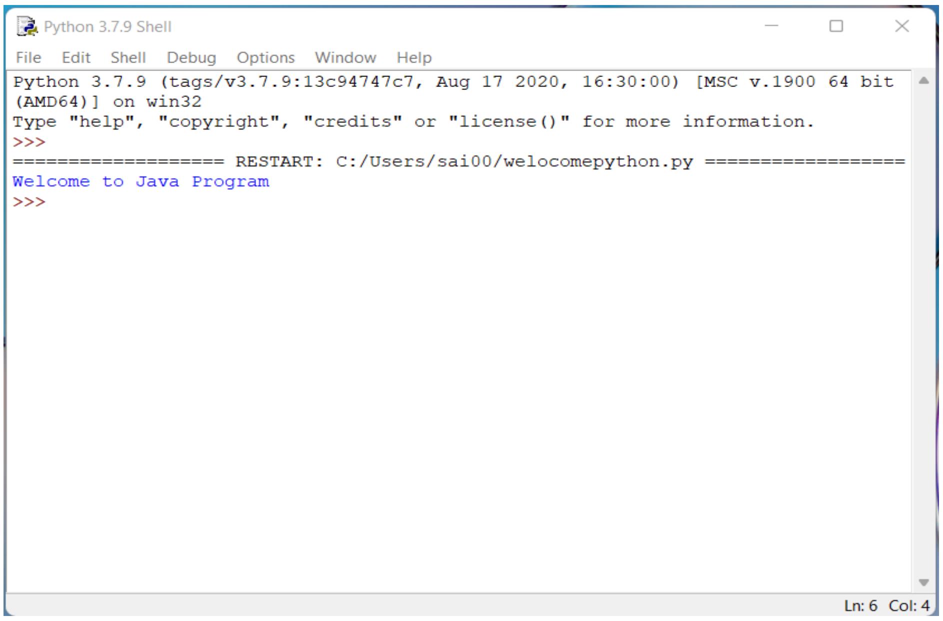
Step 7: The IDLE Shell will display the output.

You can run Python applications with ease by following the instructions in the description above, which include utilizing text editors, IDEs, or the command line. You can become more adept at executing Python code and utilizing its features to take on a variety of tasks and challenges with practice and experimentation.


welocomepython.py - C:/Users/sai00/welocomepython.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 2 Col: 0

```
print("Welcome to Java Program")
```

Fig 3.8. Create and Save Python Scripts in new file of IDE.



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/welocomepython.py ==================
Welcome to Java Program
>>>
```

Fig 3.9. Run Python Scripts in IDE.

One of the most important skills for anyone studying or using Python is the ability to run programs. Knowing how to run Python code is essential, regardless of your level of experience—whether you're a novice learning the fundamentals of the language or an expert in creating complex apps.

3.6 VARIABLES, KEYWORDS, DATATYPES, OPERATORS IN PYTHON

The next section goes over Python's operators, datatypes, variables, and keywords.

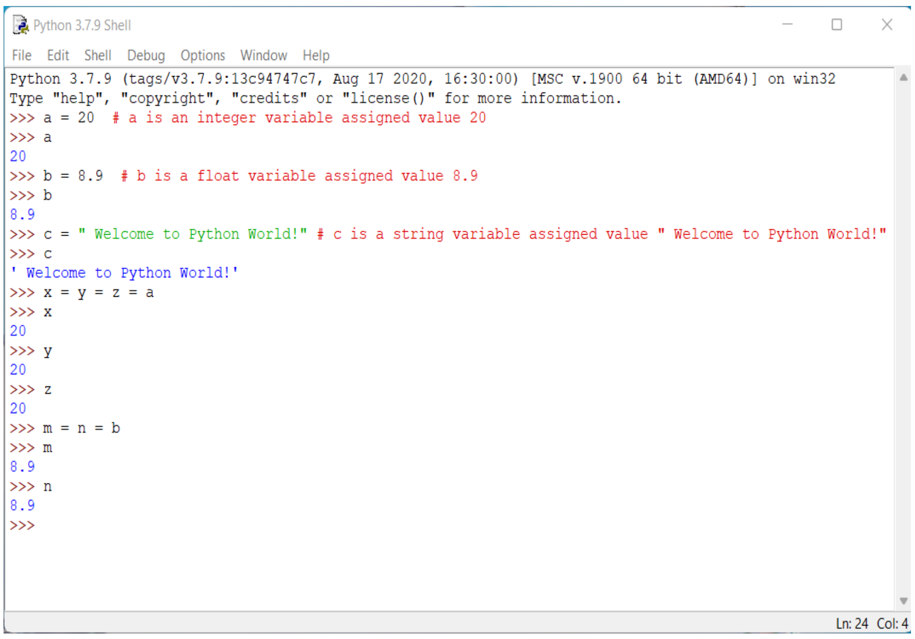
3.6.1 Python Variables

A memory location set aside to hold a value is called a variable. Python's variable type is determined by the values provided to it, unlike other programming languages where variables need to have their types explicitly stated. Python does not require an explicit declaration in order to reserve memory.

Assigning value to variables:

The type of variable is automatically determined by the interpreter based on the data it holds or is assigned. The equal sign(=), also referred to as the assignment operator, is used to set a value for the variable.

The following example demonstrates how to declare variables and give them values and is shown in Figure 3.10:

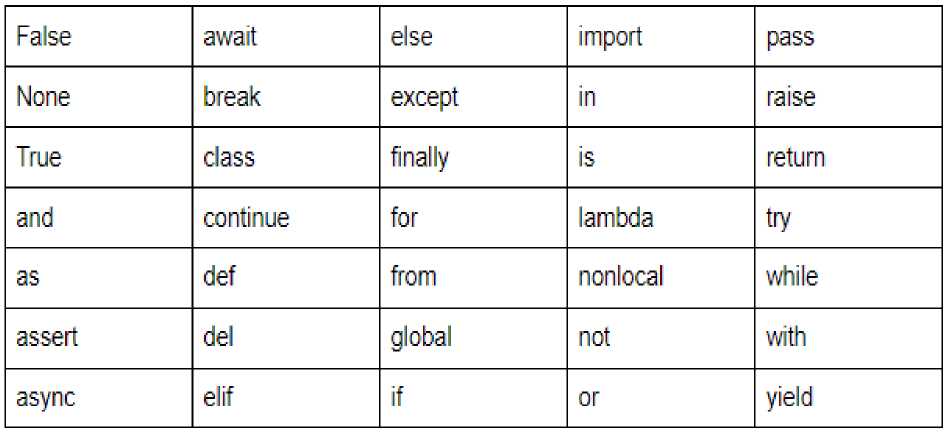


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a = 20  # a is an integer variable assigned value 20
>>> a
20
>>> b = 8.9  # b is a float variable assigned value 8.9
>>> b
8.9
>>> c = " Welcome to Python World!" # c is a string variable assigned value " Welcome to Python World!"
>>> c
' Welcome to Python World!'
>>> x = y = z = a
>>> x
20
>>> y
20
>>> z
20
>>> m = n = b
>>> m
8.9
>>> n
8.9
>>>
```

Fig 3.10. Example of Variable Declaration and Assignment

3.6.2 Python Keywords

Each language has words and rules that make sense when put together in a sentence. Also, the computer language Python has a set of predefined words that are called Keywords. You can't use these words anywhere else in Python because they have special meanings. Keywords set the rules for how the code is written. That word can't be used as a variable, function, or symbol name. The only words in Python that are written in capital letters are True and False. Python 3.11 has 35 keywords and are'shown in Figure 3.11.



False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Fig 3.11. Python Kewords

3.6.3 Python Data Types

The variable sets aside a stored in memory to store a value, and when it is given a value, that value is stored in that stored. Now, what kind of data that variable is linked to determines how much memory it takes up. In other words, the data types tell you how much memory you need to store the value. The data types in python is shown in Table 3.2.

Table 3.2. Python Data Types

Data Type	Category	Description
Numbers	int, float, complex	numeric values
String	Str	sequence of characters
Segquence	List,tuple	sequence of items
Mapping	Dict	data in key-value pair
Set	Set	collection of unique items

Python has 5 standard data types:

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary
6. Set

❖ Python Numbers

Based on their names, these are the types of data that store numbers: integer, float, and complex. It can be either an int or a long int.

There are three numbers in Python:

Example:

```
A =  20   # Assing 20 to A

B = 4.67 # Assign 4.67 to B

print(A) # prints 20 on screen

print(B) # prints 4.65 on screen
```

Output:

```
20

4.65
```

Python Strings

Strings in Python are groups of characters that are kept in memory together, like an array of characters. Either a single quote or two double quotes are used to show these characters.

Example:

```
S =  “  Happy “ #prints Happy to console

print S[1]  # prints first character to console

print S + “ Morning “  # concatenates Morning to Happy and prints on console
```

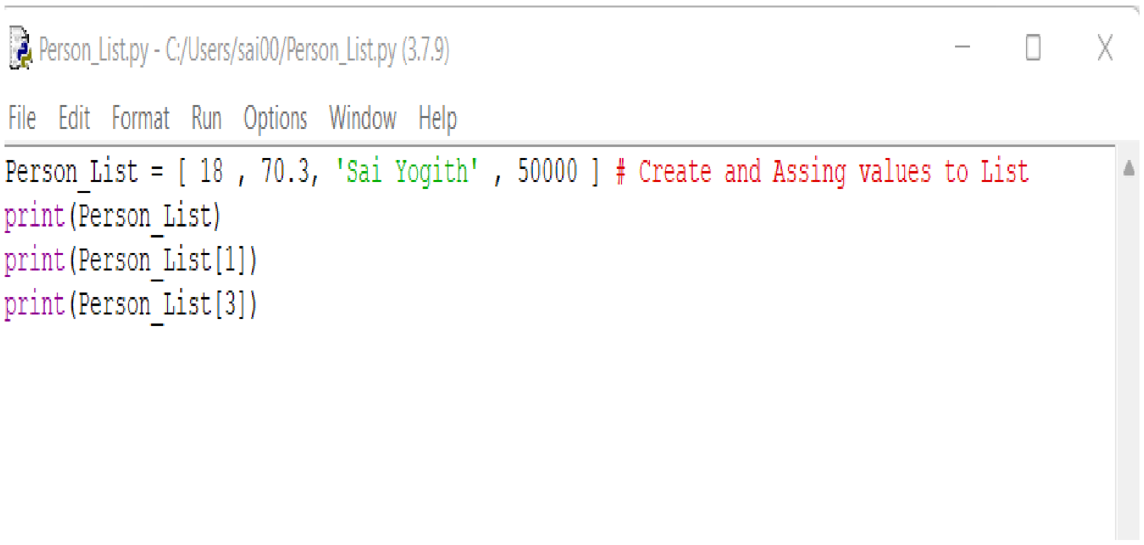
Output:

```
Happy

H

Happy Morning
```

❖ Python List

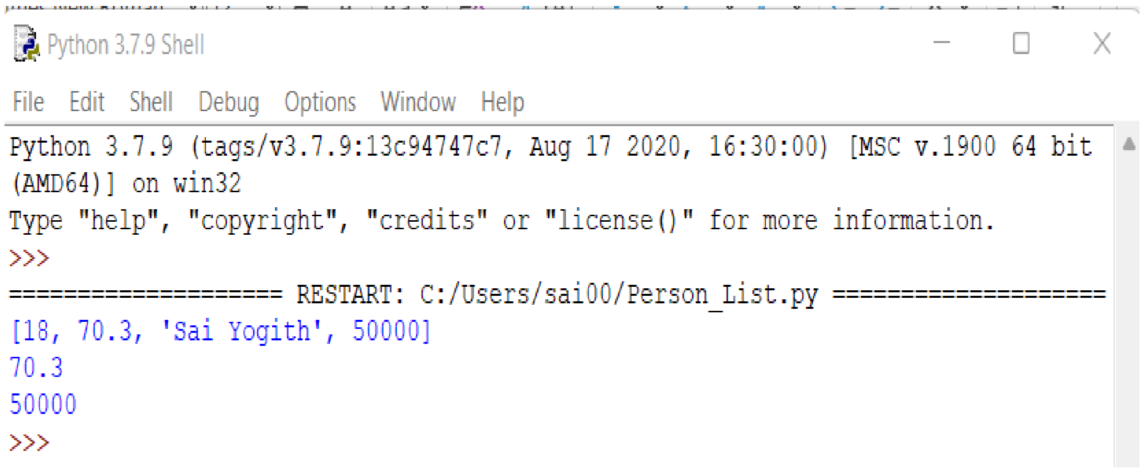
In Python, a list is a sorted list of things separated by commas (,) and enclosed in square brackets ([]). If you access a Python list using the slicing operator [], you can change the value of any item in it. A list in Python is like a collection. The main difference is that an array is a collection of items that are all of the same type, while a list is a collection of items that can be of different kinds. The Python list can be changed.

Example:


Person_List.py - C:/Users/sai00/Person_List.py (3.7.9)
File Edit Format Run Options Window Help

```
Person_List = [ 18 , 70.3, 'Sai Yogith' , 50000 ] # Create and Assing values to List
print(Person_List)
print(Person_List[1])
print(Person_List[3])
```

Output:



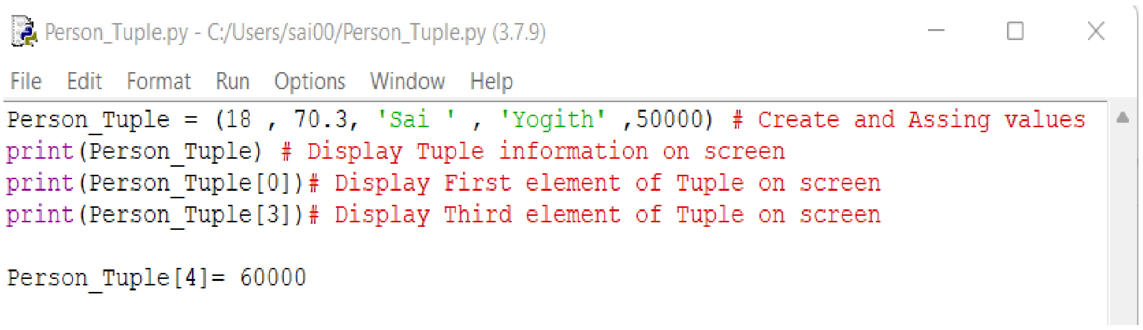
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/Person_List.py ====================
[18, 70.3, 'Sai Yogith', 50000]
70.3
50000
>>>
```

The code above shows that Person_List has items that are numbers, floats, strings, and long ints. The result shows that the whole Person_List was shown first. Python Tuple

Python tuples are the same as Python lists. The only difference is that Python tuples are immutable, which means that you can access the things in them but not change their values.

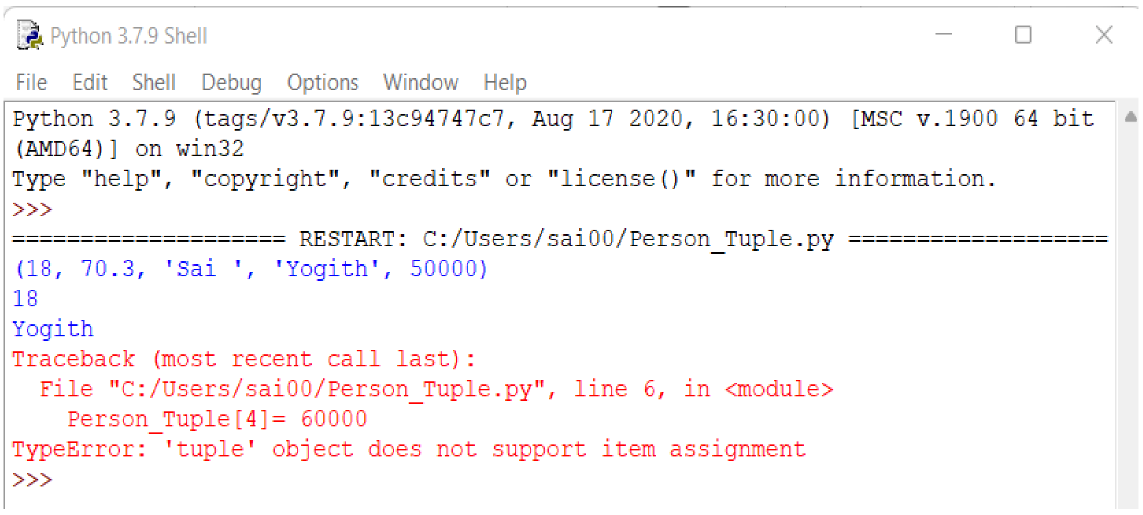
Besides being able to change, another big difference between tuples and lists is that lists are defined inside square braces [], while tuples are defined inside parentheses ().

Example:


Person_Tuple.py - C:/Users/sai00/Person_Tuple.py (3.7.9)
File Edit Format Run Options Window Help

```
Person_Tuple = (18 , 70.3, 'Sai ' , 'Yogith' ,50000) # Create and Assing values
print(Person_Tuple) # Display Tuple information on screen
print(Person_Tuple[0])# Display First element of Tuple on screen
print(Person_Tuple[3])# Display Third element of Tuple on screen

Person_Tuple[4]= 60000
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/Person_Tuple.py ===================
(18, 70.3, 'Sai ', 'Yogith', 50000)
18
Yogith
Traceback (most recent call last):
  File "C:/Users/sai00/Person_Tuple.py", line 6, in <module>
    Person_Tuple[4]= 60000
TypeError: 'tuple' object does not support item assignment
>>>
```

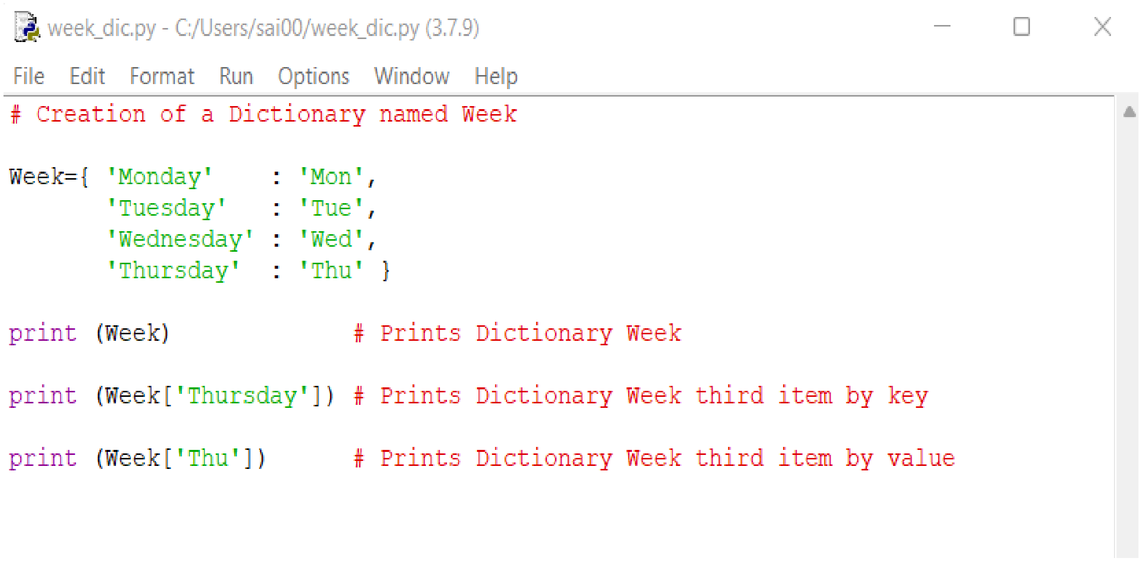
The code above shows that the items in Person_Tuple are integers, floats, strings, long integers, and strings. The result shows the full Person_Tuple as the first item. After that The first and fourth items were printed.

But at the end of the last line, an error is made because the fourth member of the tuple is being changed. Based on the finding, we can say that tuple items can't be changed, but List data types can.

Python Dictionary

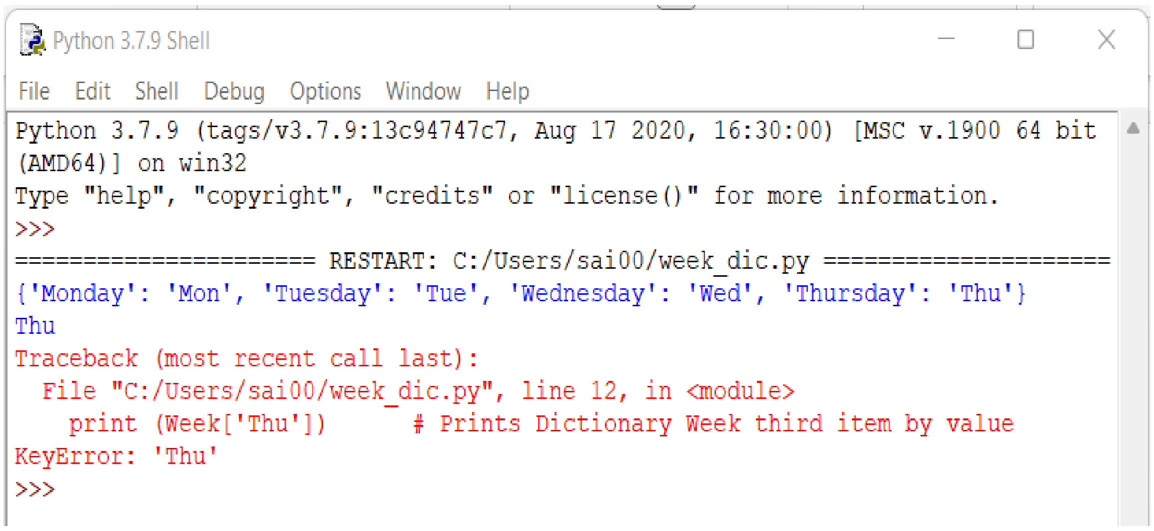
A sorted list of key-value pairs is called a dictionary in Python. The dictionary's entries are key-value pairs separated by commas. The value can always be retrieved if we know the key, but the opposite is not true. Python dictionaries are therefore designed for data retrieval. Python dictionaries are defined inside curly braces ({}), and the slicing operator ([ ]) is used to access and assign values.

Example:


week_dic.py - C:/Users/sai00/week_dic.py (3.7.9)
File Edit Format Run Options Window Help

```
# Creation of a Dictionary named Week

Week={ 'Monday'    : 'Mon',
       'Tuesday'   : 'Tue',
       'Wednesday' : 'Wed',
       'Thursday'  : 'Thu' }

print (Week)              # Prints Dictionary Week

print (Week['Thursday']) # Prints Dictionary Week third item by key

print (Week['Thu'])       # Prints Dictionary Week third item by value
```

Output:



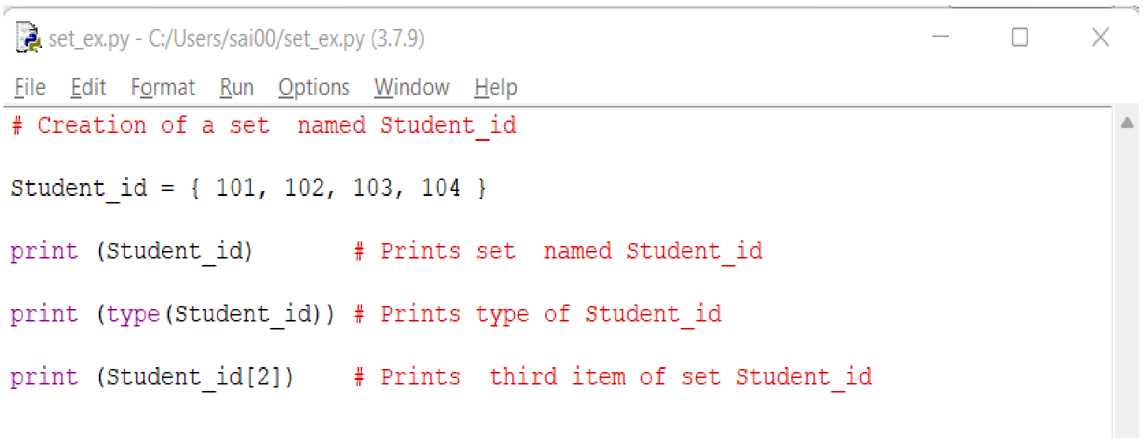
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/week_dic.py =====================
{'Monday': 'Mon', 'Tuesday': 'Tue', 'Wednesday': 'Wed', 'Thursday': 'Thu'}
Thu
Traceback (most recent call last):
  File "C:/Users/sai00/week_dic.py", line 12, in <module>
    print (Week['Thu'])       # Prints Dictionary Week third item by value
KeyError: 'Thu'
>>>
```

We have created a dictionary called week in the example above. In this case, the keys are Monday, Tuesday, Wednesday, and Thursday, and the values are Monday, Tuesday, Wednesday, and Thursday. To get the appropriate value, we employ keys. not the other way around, though. Here, we've used the week dictionary's keys to obtain the data. Capital_city['Thursday'] retrieves its corresponding value, Thu, since 'Thursday' is the key. But since 'Thu' is the value assigned to the 'Thursday' key, capital_city['Thu'] raises an error.

❖ Python Set Data Type:

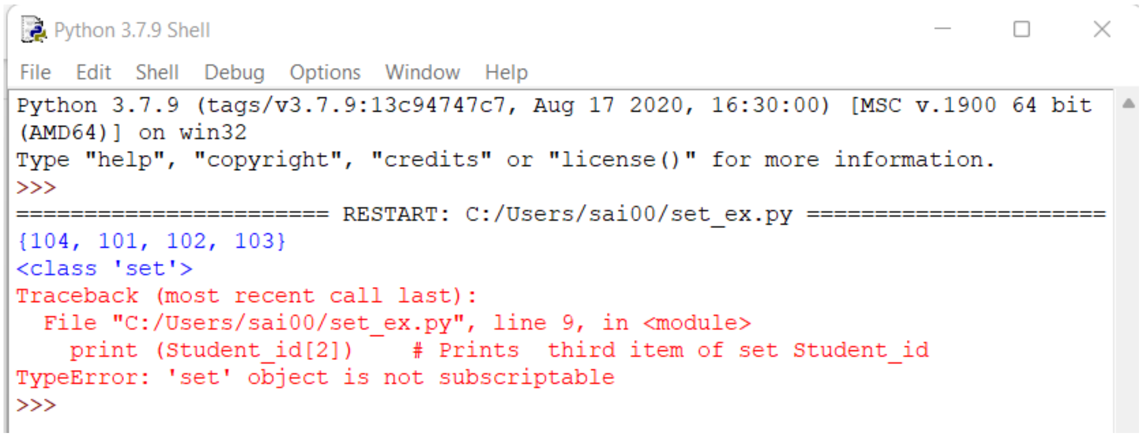
A set is an arbitrary grouping of distinct objects. Values inside braces {} and separated by commas define a set.

Example:


set_ex.py - C:/Users/sai00/set_ex.py (3.7.9)
File Edit Format Run Options Window Help

```
# Creation of a set  named Student_id

Student_id = { 101, 102, 103, 104 }

print (Student_id)       # Prints set  named Student_id

print (type(Student_id)) # Prints type of Student_id

print (Student_id[2])    # Prints  third item of set Student_id
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======================= RESTART: C:/Users/sai00/set_ex.py ======================
{104, 101, 102, 103}
<class 'set'>
Traceback (most recent call last):
  File "C:/Users/sai00/set_ex.py", line 9, in <module>
    print (Student_id[2])    # Prints  third item of set Student_id
TypeError: 'set' object is not subscriptable
>>>
```

Here, four integer values have been added to a set called student_id. As sets are collections that are not ordered, indexing is meaningless. The entire set is shown first. Afterwards, trying to access the element of the set using the slicing operator [] does not work. Similar to the output accessing the third item with the error message generated by the index.

❖ Python Boolean Data Type:

The datatype which returns only 2 values either TURE or FALSE.

Example:

A = 50 ;

Output:

>>> **A = = 40**

>>> **FALSE**

3.6.3 Python Operators

Operators are unique symbols or keywords in Python that perform operations on values and variables. They form the foundation of expressions, which are used to work with data and carry out calculations. Python has a number of operators, each having a distinct function. The Python programming language supports the following types of operators:

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Assignment Operators
4. Logical Operators
5. Bitwise Operators
6. Membership Operators

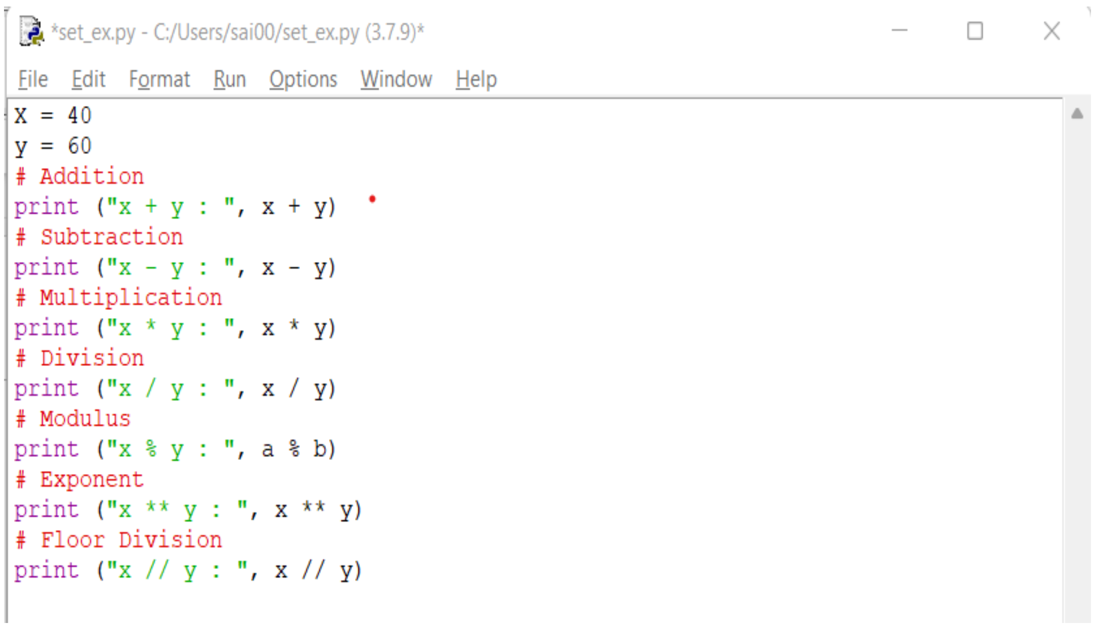
- **Python Arithmetic Operators**

Common mathematical operations in addition modules are addition, subtraction, multiplication, and division; additional arithmetic operations include exponential and floor divisions are shown in Table 3.3. Expressions, variables, and integers are supported by all.

Table 3.3. Arithmetic Operations in Python

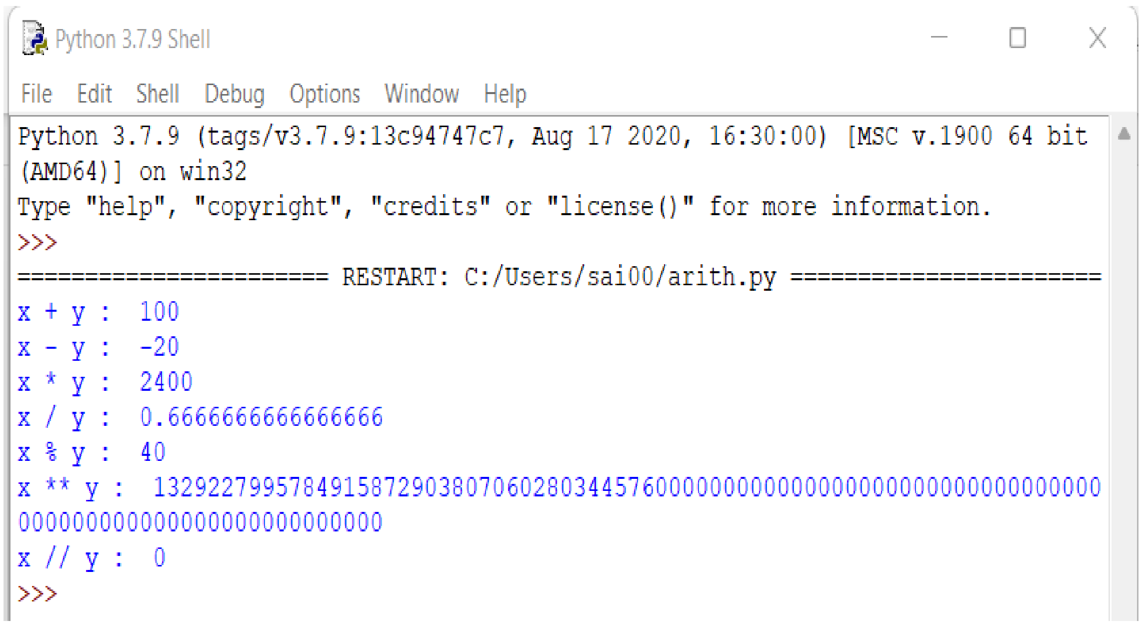
Operator	**Description**	**Python Expression**
+	Addition	**x + y**
-	Subtraction	**x - y**
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponent	x ** y
//	Floor Division	x // y

Example:


set_ex.py - C:/Users/sai00/set_ex.py (3.7.9)
File Edit Format Run Options Window Help

```
x = 40
y = 60
# Addition
print ("x + y : ", x + y)
# Subtraction
print ("x - y : ", x - y)
# Multiplication
print ("x * y : ", x * y)
# Division
print ("x / y : ", x / y)
# Modulus
print ("x % y : ", a % b)
# Exponent
print ("x ** y : ", x ** y)
# Floor Division
print ("x // y : ", x // y)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/arith.py ======================
x + y :  100
x - y :  -20
x * y :  2400
x / y :  0.6666666666666666
x % y :  40
x ** y :  13292279957849158729038070602803445760000000000000000000000000000000
000000000000000000000000000
x // y :  0
>>>
```

Once the two variables "x" and "y" are defined, this code does a number of mathematical operations, including floor division, modulus, addition, subtraction, multiplication, and division, and reports the results.

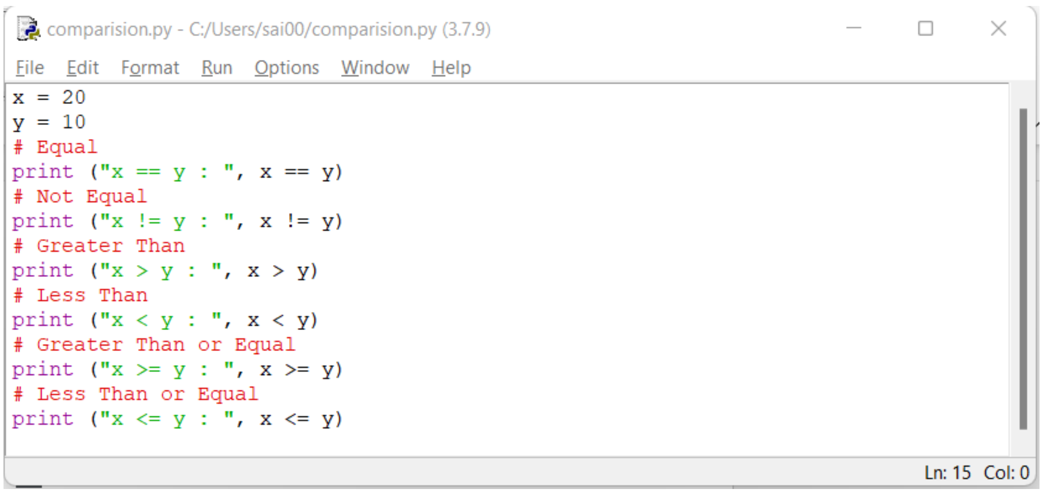
- **Python Comparison Operators**

Python comparison operators are required in order to compare two values.They produce a Boolean value (True or False) based on the comparison. The comparison operators in python is shown in Table 3.4.

Table 3.4. Comparison Operations in Python

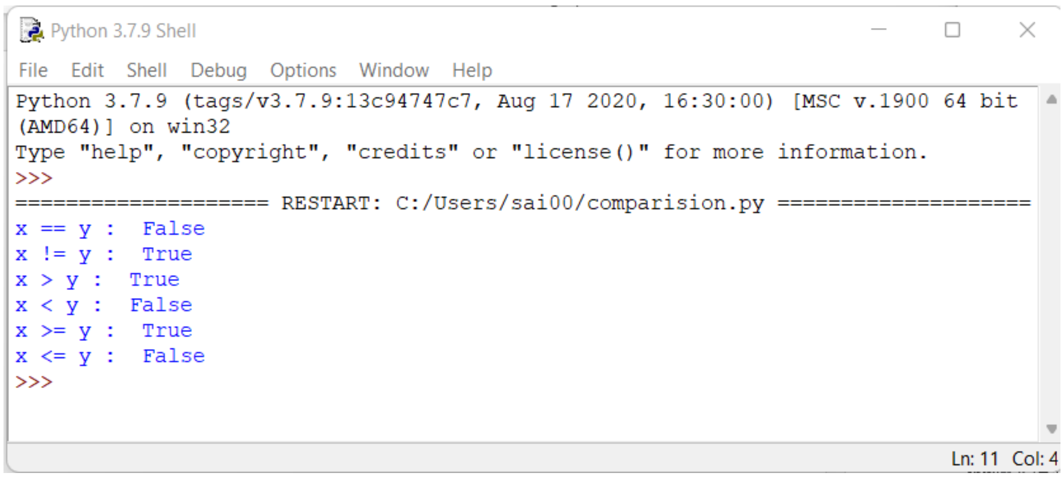
Operator	Description	Python Expression
= =	Equal	**x == y**
! =	Not Equal	**x != y**
>	Greater Than	x > y
<	Less Than	x < y
> =	Greater Than or Equal	x >= y
< =	Less Than or Equal	x <= y

Example:


comparision.py - C:/Users/sai00/comparision.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 15 Col: 0

```
x = 20
y = 10
# Equal
print ("x == y : ", x == y)
# Not Equal
print ("x != y : ", x != y)
# Greater Than
print ("x > y : ", x > y)
# Less Than
print ("x < y : ", x < y)
# Greater Than or Equal
print ("x >= y : ", x >= y)
# Less Than or Equal
print ("x <= y : ", x <= y)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/comparision.py ====================
x == y :  False
x != y :  True
x > y :  True
x < y :  False
x >= y :  True
x <= y :  False
>>>
```

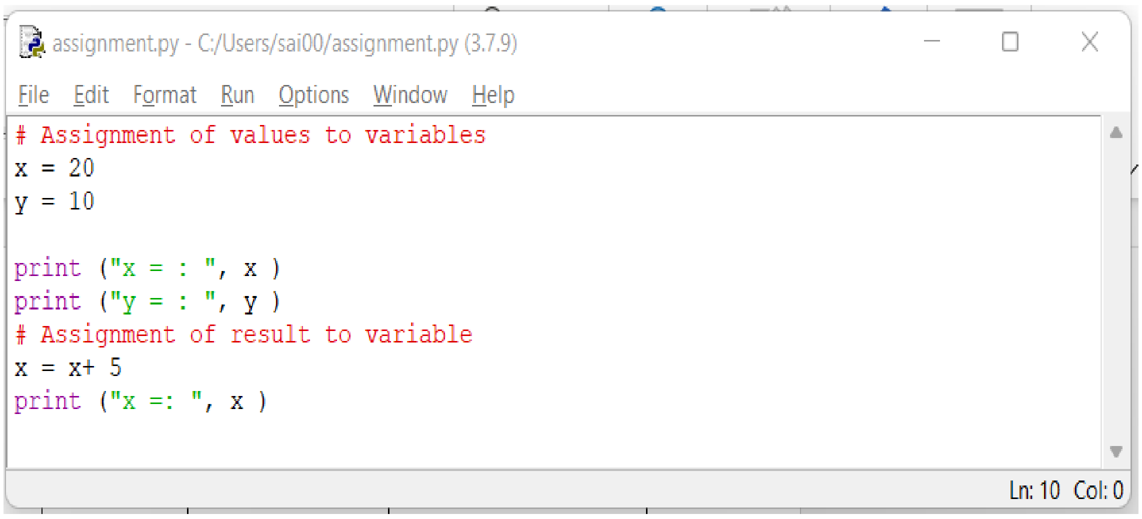
- **Python Assignment Operators**

To assign values to Python, utilize the assignment operators. The simplest assignment operator is the single equal symbol (=). The variable on the operator's left side is given the value on the operator's right side. The different approaches to use assignment operator in python is shown in Table 3.5.

Table 3.5. Assignment Operations in Python

Operator	Description	Python Expression
=	Equal	**x = y** **x=x+5** **x= 2 * x + 4 * 5 + 8**

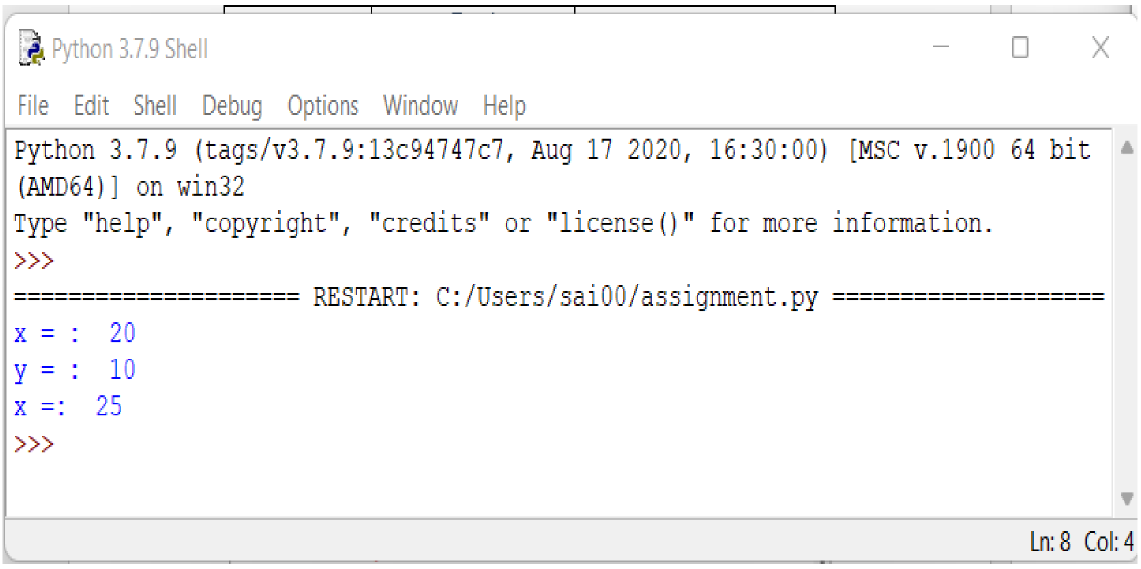
Example:



```
# Assignment of values to variables
x = 20
y = 10

print ("x = : ", x )
print ("y = : ", y )
# Assignment of result to variable
x = x+ 5
print ("x =: ", x )
```

assignment.py - C:/Users/sai00/assignment.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 10 Col: 0

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/assignment.py ====================
x = :  20
y = :  10
x =:  25
>>>
```

Above displays the assignment operators in Python. First, 'x' and 'y' have values of 20 and 10, respectively. Afterwards, x=25 is the output of applying expression x+5 to x.

Python Bitwise Operators

Bitwise operators in Python carry out actions on discrete binary integer bits.They operate on each bit location logically while working with integer binary representations.Many bitwise operations, including AND (&), OR (|), NOT (), XOR (), left shift (), and right shift (>>), are included in Python.

❖ **Python Logical Operators**

Boolean expressions are composed, and their truth values are evaluated using logical operators in Python. They are necessary for controlling the program's execution flow and for creating conditional statements. The three fundamental logical operators in Python are AND, OR, and NOT.

❖ **Python Membership Operators**

To determine whether a particular value appears in a series or not, one can utilize Python membership operators. They simplify the process of figuring out which elements belong in many types of data structures, including sets, tuples, lists, and strings. The is and is not operators are the two main membership operators in Python.

3.7 INDENTATION IN PYTHON

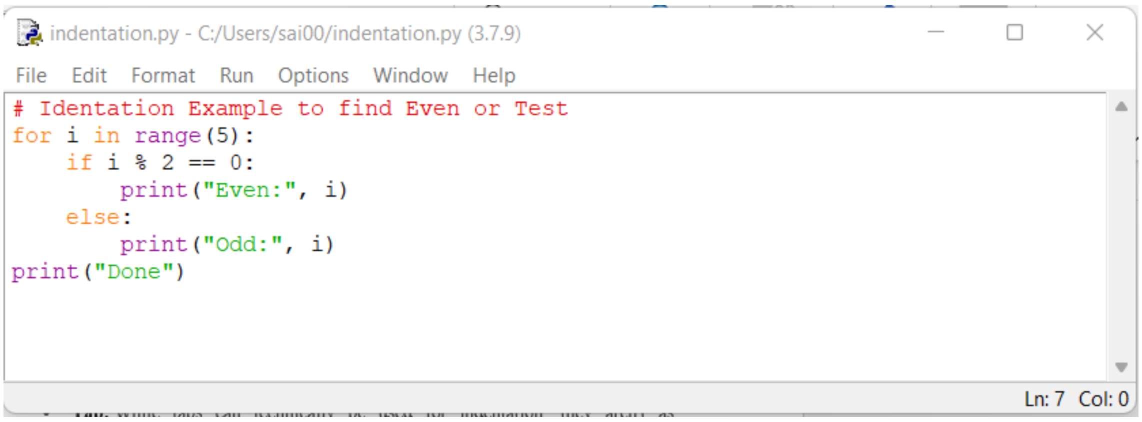
One of the main features of Python syntax is indentation, which describes the spaces or tabs at the start of a line. It is notable not only as a custom but also as a necessity. In Python, indentation is not just a matter of style; it is essential to the way the code flows.

3.7.1 Types of Indentation

• **Space:** The recommended indentation approach in Python is using spaces. The official Python style guide (PEP 8) recommends using four spaces for each indentation level, which is typical practice.

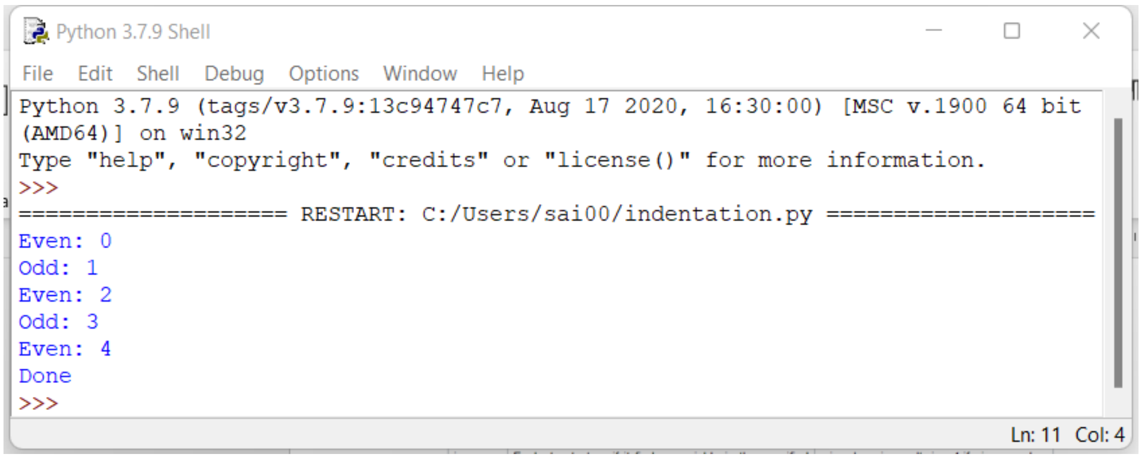
• **Tab:** Although tabs are technically acceptable for indentation, their use isn't as widespread. The primary problem is uneven code presentation caused by sharing or viewing code in multiple editors. Therefore, especially in collaborative situations, employing tabs can unintentionally result in an indentation issue in Python.

Example:


indentation.py - C:/Users/sai00/indentation.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 7 Col: 0

```python
# Identation Example to find Even or Test
for i in range(5):
    if i % 2 == 0:
        print("Even:", i)
    else:
        print("Odd:", i)
print("Done")
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/indentation.py ====================
Even:  0
Odd:  1
Even:  2
Odd:  3
Even:  4
Done
>>>
```

The even odd test for the number in python with indentation is shown in above example and output also shown respectively.

3.7.2. Common Indentation Errors

The common errors shown below:

Inconsistent Indentation: Make sure that the indentation level is the same for every line inside a block. Convert tabs to spaces if you mix them with spaces.

Mismatched Indentation: Look for invisible characters like trailing spaces or tabs at line beginnings if you notice a warning stating "indentation error" but no other evident problems.

Missing Colon: Python will raise an indentation error if you omit to include a colon (:) after a statement that should initiate an indented block (for example, after if, for, while, or a function definition).

3.8 SUMMARY

Conditional statements, which include if, else, and elif statements, are essential programming structures that enable you to control the execution of your program based on the conditions that you define. They make it possible for your program to make decisions and then execute different codes based on those decisions after they have been made. In this chapter, we have seen various examples of how to utilize these statements in Python. Some of the examples include student performance test, number test etc.

3.9 TECHNICAL TERMS

Python, History, Indentation, Variable, Operators, Data types, Arithmetic, Membership, comparison, space, and tab

3.10 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Illustrate about various data types in Python.
2. Describe about operators in python.
3. Explain about indentation in python.

Short Notes:

1. Write about features of python.
2. Discuss about how to run python script.
3. Write about membership operator with example.

3.11 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. Think Python: How to Think like a Computer Scientist
4. Brown, A.- Mastering Python Modules. Publisher.

Mr. G. V. SURESH

LESSON- 4

PYTHON CONDITIONAL STATEMENTS

AIMS AND OBJECTIVES

The primary goal of this chapter is to understand the concept of looping and statements in Python programming. The chapter began with an understanding of basic types of loop and statements in python, discuss each one detail and so on. After completing this chapter, the student will understand how to work with loop and control statements in python in terms programs.

STRUCTURE

4.1. INTRODUCTION

Every day, we analyze our current situation and make decisions. Then, we act further depending on those decisions. Therefore, every action we take in a given day depends on the choices we make. The decision process is the most important component of almost all programming languages. As the name suggests, decision-making enables us to execute a particular piece of code to reach a particular conclusion. Condition verification is the cornerstone of decision-making. Conditional statements are used in Python to make decisions. This chapter will cover the use of if, else, and elif statements in Python and provide some real-world examples of their application.

4.2 PYTHON CONDITIONAL STATEMENTS

Conditional statements are an essential part of programming in Python. They allow you to make decisions based on the values of variables or the result of comparisons. The uses of control statement in python listed below:

- A conditional statement checks to see if a specific condition exists before executing code.
- Conditional statements can help increase the performance of your code by giving you control over the flow of your code, such as when and how it is run.
- This can be quite useful for determining whether a specific condition occurs before the code begins to execute, as you may want to execute specific code lines only when criteria are satisfied.
- Conditional statements, for example, can be used to verify the existence of a specific variable or file before executing code, or to execute more code if certain criteria are met, such as a calculation yielding a specified result.

4.2.1 Advantages and Disadvantages of Python Conditional Statements

Advantages of Python Conditional Statements

- It offers flexibility in program flow by enabling conditional code execution depending on a certain condition.
- Its ability to handle numerous circumstances using else-if statements gives it another benefit and permits more intricate decision-making.
- increases the readability of the code when handling several situations.

Disadvantages of Python Conditional Statements

- Complex and layered if-else statements can make the code more difficult to read and update.
- Furthermore, if the same logic needs to be repeated several times, using if-else expressions may result in code duplication.
- If-else statements also have the drawback of being prone to mistakes, like forgetting to include an else statement or inadvertently employing the incorrect condition.

4.2.1 Application of Python Conditional Statements

- Showing an error message if the user input is invalid.
- Program flow control is the process of directing the execution of a program based on variables.
- Applying logic to menu selection.
- Categorizing data based on several criteria.
- Processing the selections made by users within a menu?
- Implementing state machines

4.3 TYPES OF CONDITIONAL STATEMENT

As is the case with other programming languages, Python has four distinct types of conditional statements, which are provided in the following order:

- if Statements
- if-Else Statements
- elif Statements
- Nested If-Else Statements

4.3.1 If Statements

The if statement in Python is one of the conditional statements that is used the most frequently in programming languages. In this way, it determines whether or not particular statements are required to be executed. It performs a check to determine whether a particular condition is satisfied; if the condition is satisfied, the set of code included within the "if" block will be run; otherwise, it will not be executed.

Syntax:

```
if ( EXPRESSION = = TRUE ) :
    if- Block of code
Next statement after Block of code is executed.
```

In the syntax presented above,

- if the expression "EXPRESSION = = TRUE" is successfully executed, then the conditional block of code will be run
- Otherwise, the statement that comes after the conditional block of code will be executed.

The flow chart of if statement is shown in Figure 4.1.

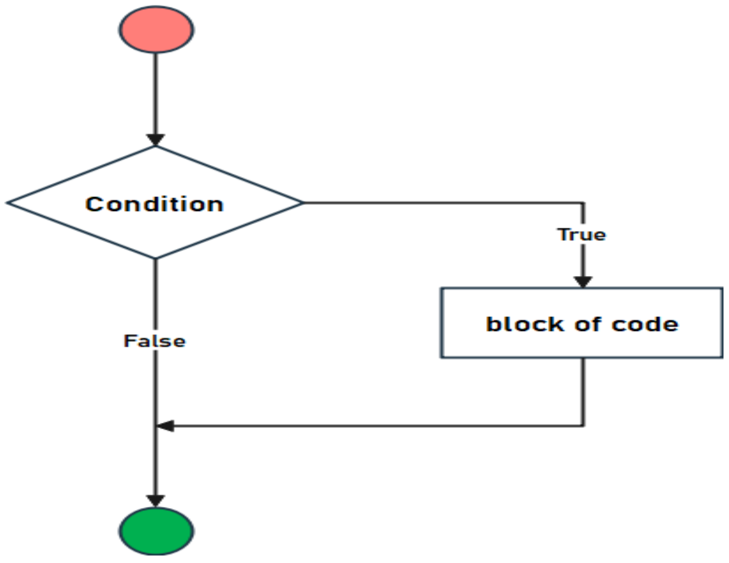
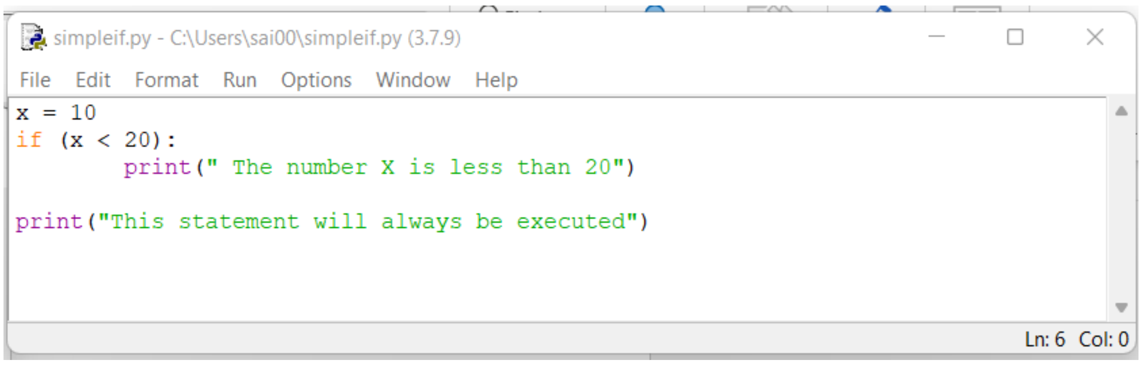


Fig 4.1. Flow Chart of if Statement

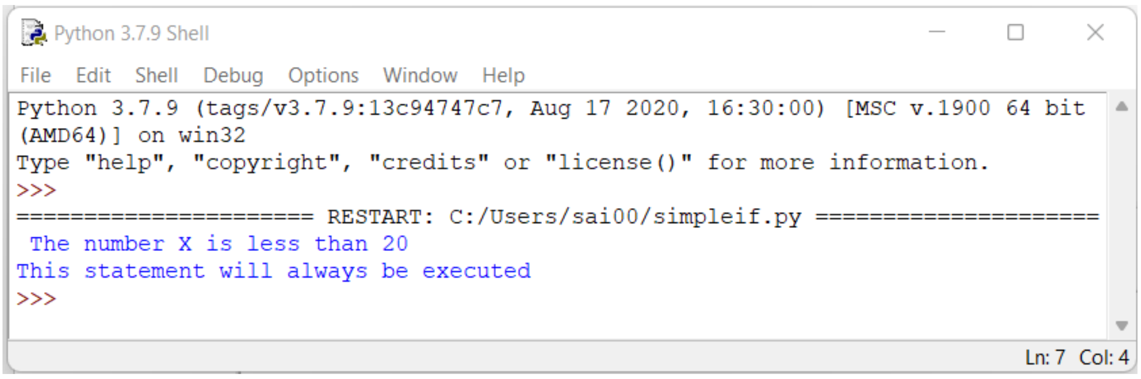
- If you look at the flowchart that was just presented, you will notice that the controller will first arrive at an if condition and then evaluate the condition.
- If the condition is true, then the statements will be executed.
- if it is not true, then the code that is present outside the block will be executed.

Example: 1


simpleif.py - C:\Users\sai00\simpleif.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 6 Col: 0

```
x = 10
if (x < 20):
        print(" The number X is less than 20")

print("This statement will always be executed")
```

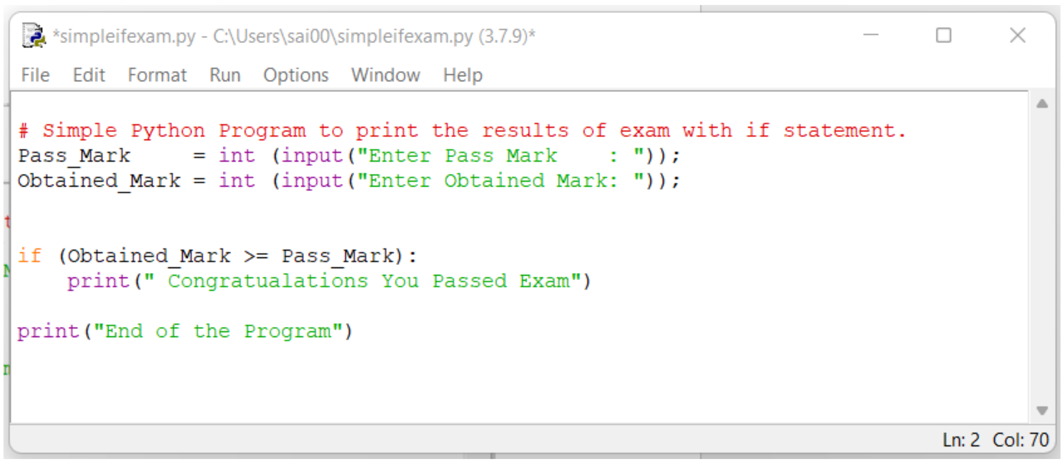
Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/simpleif.py =====================
 The number X is less than 20
This statement will always be executed
>>>
```

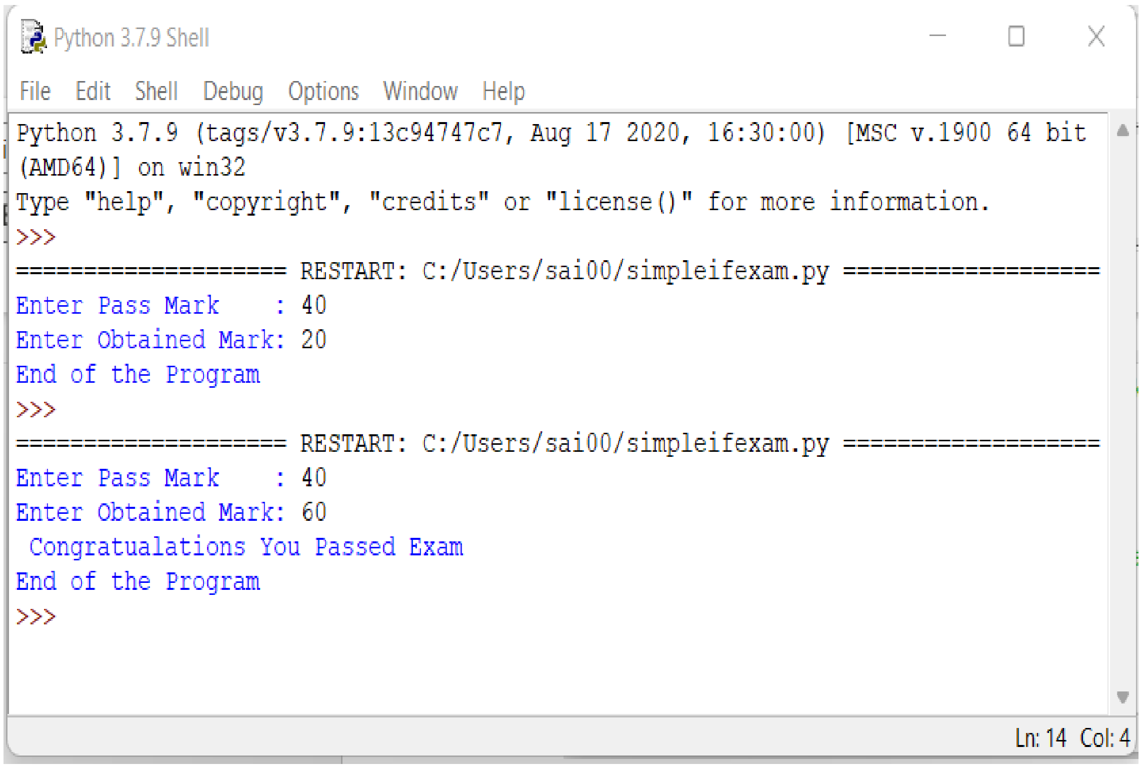
The above code tests the condition "x<20." If the test is successful, a block of code will be executed, as really seen in the output, and finally the last line, "**This statement will always be executed,**" will be executed. This statement is also clearly displayed in the output.

Example 2:


simpleifexam.py - C:\Users\sai00\simpleifexam.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 2 Col: 70

```
# Simple Python Program to print the results of exam with if statement.
Pass_Mark     = int (input("Enter Pass Mark    : "));
Obtained_Mark = int (input("Enter Obtained Mark: "));


if (Obtained_Mark >= Pass_Mark):
    print(" Congratualations You Passed Exam")

print("End of the Program")
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/simpleifexam.py ===================
Enter Pass Mark    : 40
Enter Obtained Mark: 20
End of the Program
>>>
=================== RESTART: C:/Users/sai00/simpleifexam.py ===================
Enter Pass Mark    : 40
Enter Obtained Mark: 60
 Congratulations You Passed Exam
End of the Program
>>>
```

The code condition (Obtained_Mark > = Pass_Mark) is tested in the previous example; if it passes, the if-block will be executed. The code is executed twice. The first time, the condition is not met (20 < 40), and the final message, "**End of the Program**," is shown. Nevertheless, the second attempt met the success requirement (i.e., 60 > 40), printed "**Congratulations on Passing the Exam**," and showed the final message, "**End of the Program.**"

4.3.2. if-else statements

The Boolean expression is evaluated by the if-else statement. The code in the "if" block will be executed if the condition is TRUE; otherwise, the code in the "else" block will be executed.

Syntax:

```
If (EXPRESSION == TRUE):
    If-Statement (Body of the block)
else:
    else-Statement (Body of the block)
```

When the syntax (EXPRESSION = = TRUE) in the following example is successfully executed, a block of code will be executed if it is not, otherwise it will be executed.

The flow chart of if-else statement is shown in Figure 4.2.

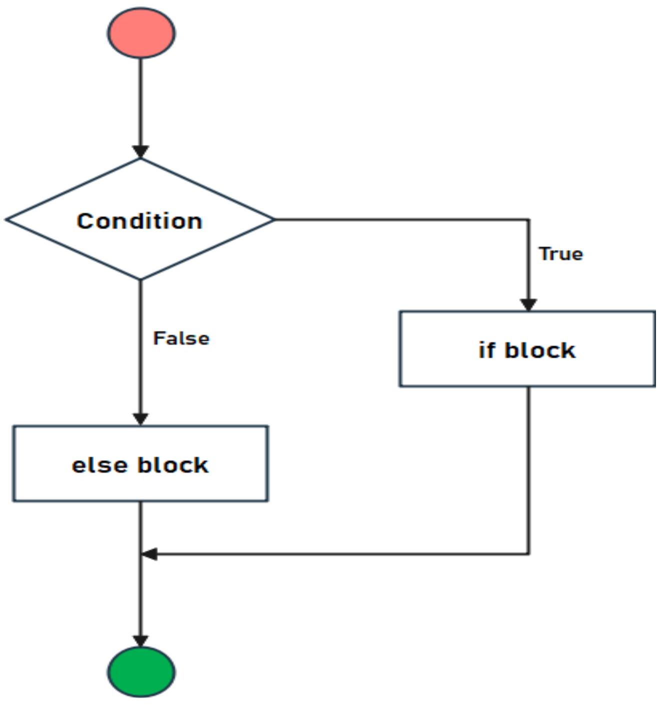
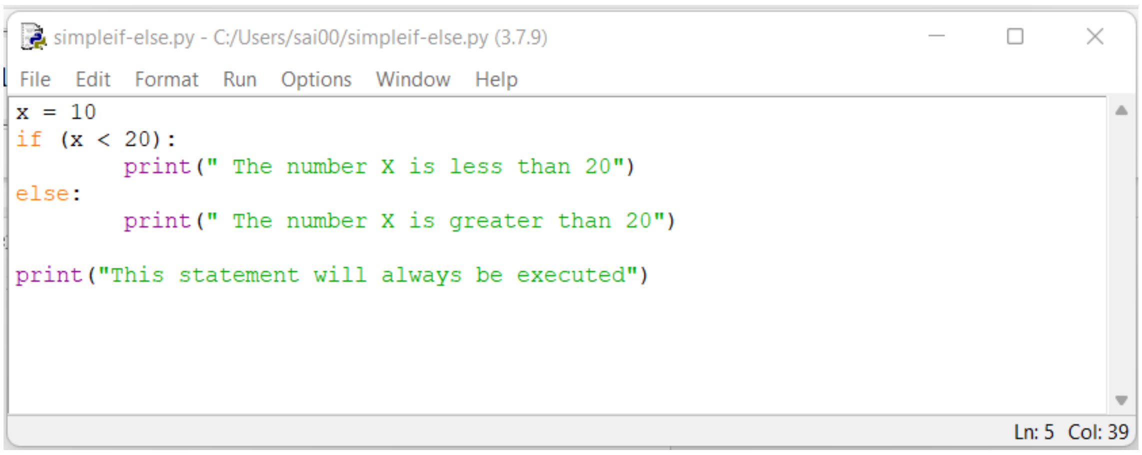


Fig 4.2. Flow Chart of if-else Statement

According to the flow chart above, the controller will first reach the if condition and determine if the condition is true. If it is, the statements in the if block will then be run; if not, the "else" block will be executed, and finally the remaining code that is included outside the "if-else" block will be executed.

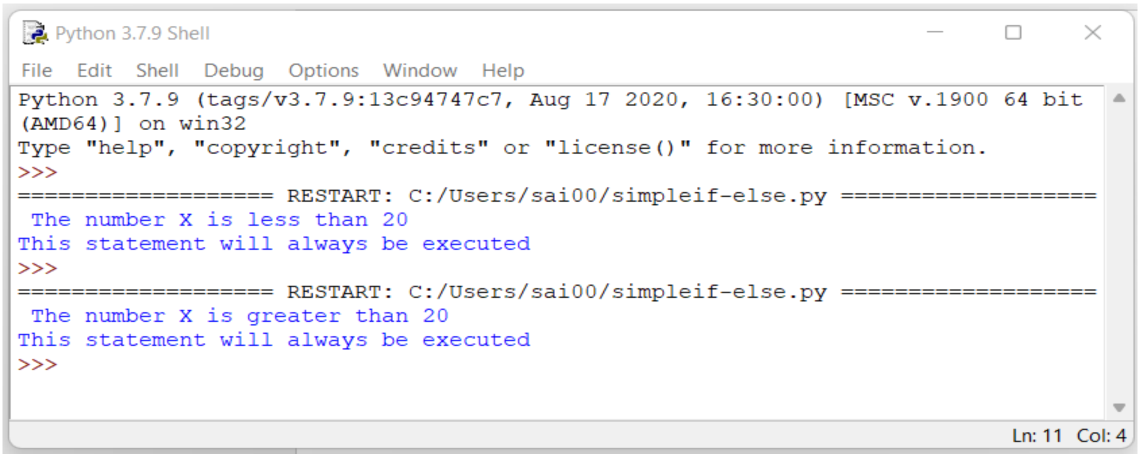
Example: 1


simpleif-else.py - C:/Users/sai00/simpleif-else.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 5 Col: 39

```
x = 10
if (x < 20):
        print(" The number X is less than 20")
else:
        print(" The number X is greater than 20")

print("This statement will always be executed")
```

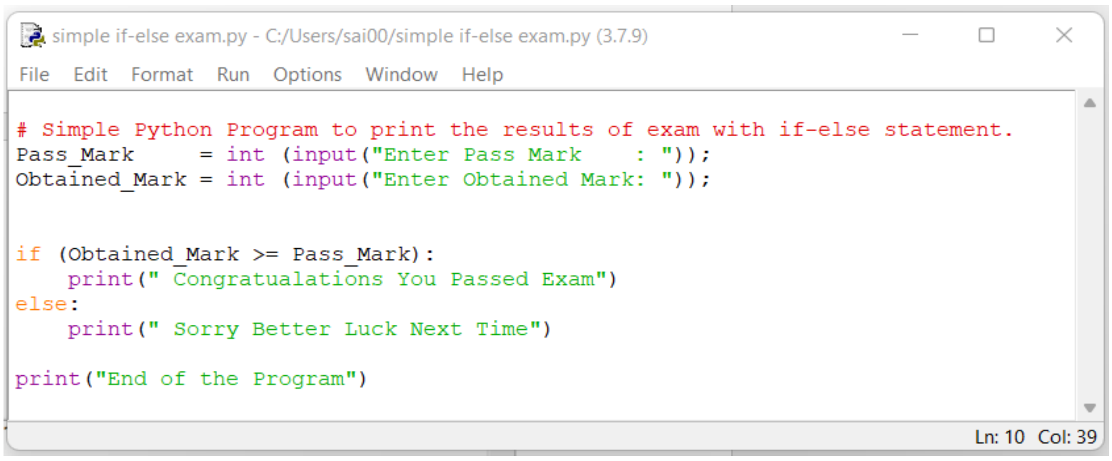
The condition (x<20) is tested twice in the code above. The first time it is run, if it is successful, a block of code will be executed, as we can see in the output. Finally, the final statement, "**This statement will always be executed**," is executed, and this is also clearly displayed in the output. Nevertheless, the second run condition failed by evaluating x=30, executing the else-Block of code, and generating the output "**X is greater than 20**." The final statement, "**This statement will always be executed**," is finally carried out and is likewise displayed in the output.

Output:



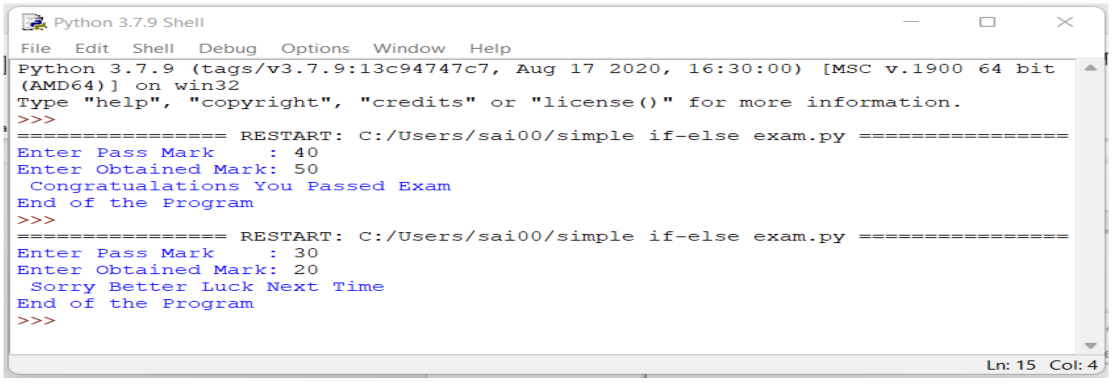
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/simpleif-else.py ===================
 The number X is less than 20
This statement will always be executed
>>>
=================== RESTART: C:/Users/sai00/simpleif-else.py ===================
 The number X is greater than 20
This statement will always be executed
>>>
```

Example2:


simple if-else exam.py - C:/Users/sai00/simple if-else exam.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 10 Col: 39

```python
# Simple Python Program to print the results of exam with if-else statement.
Pass_Mark      = int (input("Enter Pass Mark     : "));
Obtained_Mark = int (input("Enter Obtained Mark: "));


if (Obtained_Mark >= Pass_Mark):
    print(" Congratualations You Passed Exam")
else:
    print(" Sorry Better Luck Next Time")

print("End of the Program")
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/sai00/simple if-else exam.py ================
Enter Pass Mark     : 40
Enter Obtained Mark: 50
 Congratualations You Passed Exam
End of the Program
>>>
================ RESTART: C:/Users/sai00/simple if-else exam.py ================
Enter Pass Mark     : 30
Enter Obtained Mark: 20
 Sorry Better Luck Next Time
End of the Program
>>>
```

The code condition (Obtained_Mark > = Pass_Mark) is tested in the previous example; if it passes, the if-block will be executed. The code is executed twice. The first time, if the condition is met (i.e., 50 > 40), the message "**Congratulations You Passed Exam**" is displayed, and the final phrase, "**End of the Program**," is printed. Nevertheless, the second time around, the condition failed (20 < 30), printing "**Sorry, Better Luck Next Time**" and displaying the last sentence, "**End of the Program**."

4.3.3 elif statements

"elif" statements are an additional type of conditional statement in Python. The "elif" statement checks for multiple conditions only in the event that the supplied condition is false. The sole distinction between it and a "if-else" expression is that the condition will be checked in "elif" rather than "else."

Syntax:

```
   if (EXPRESSION-1 = = TRUE):
       If-Statement (Body of the block)

  elif(EXPRESSION-2 = = TRUE):
       elif-Statement (Body of the block)

 elif(EXPRESSION-3 = = TRUE):
       elif-Statement (Body of the block)
  else:
       else-Statement (Body of the block)
```

- In the above syntax (EXPRESSION-1 = = TRUE) is executed successfully then if-Block of code will be executed
- otherwise (EXPRESSION-2 = = TRUE) is tested, if it is executed successfully then elif- Block of code related EXPRESSION-2 will be executed
- otherwise (EXPRESSION-3 = = TRUE) is tested, if it is executed successfully then elif- Block of code related EXPRESSION-3 will be executed otherwise else-Block of code will be executed.

The flow chart of else-if- lader statement is shown in Figure 4.3.

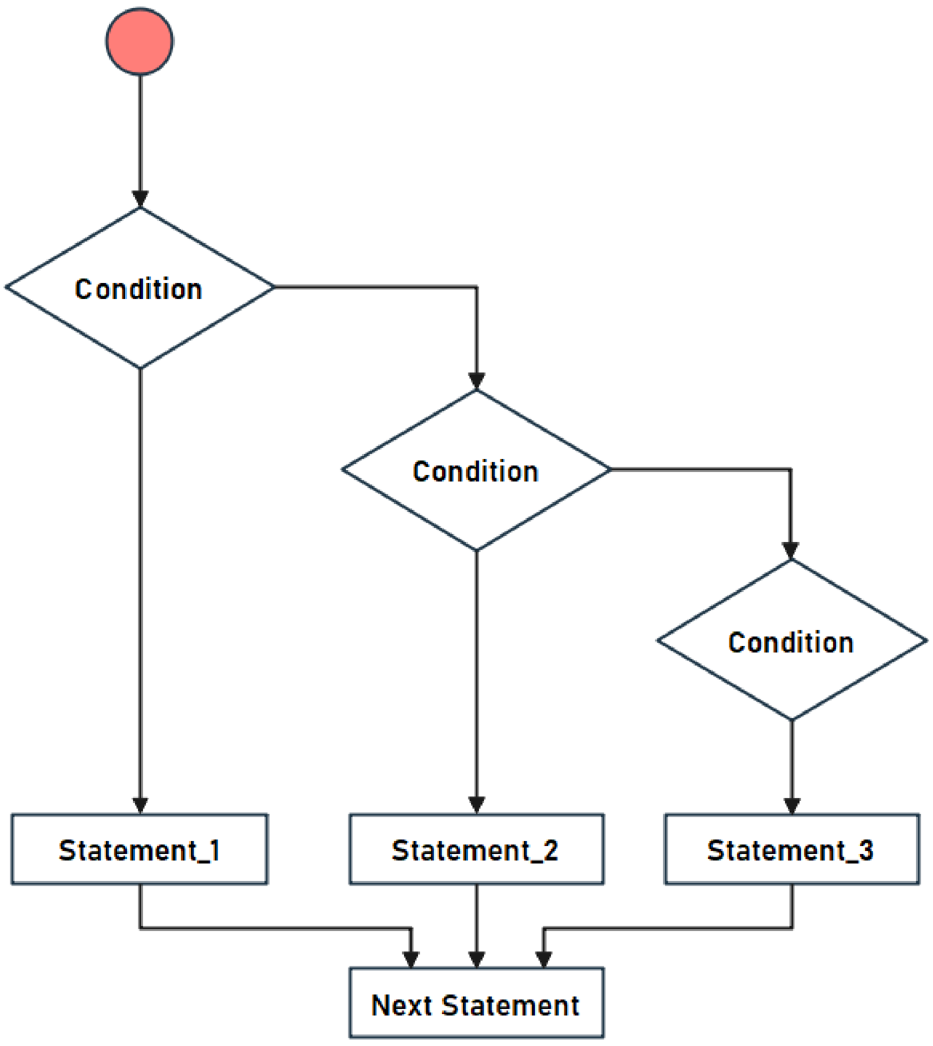
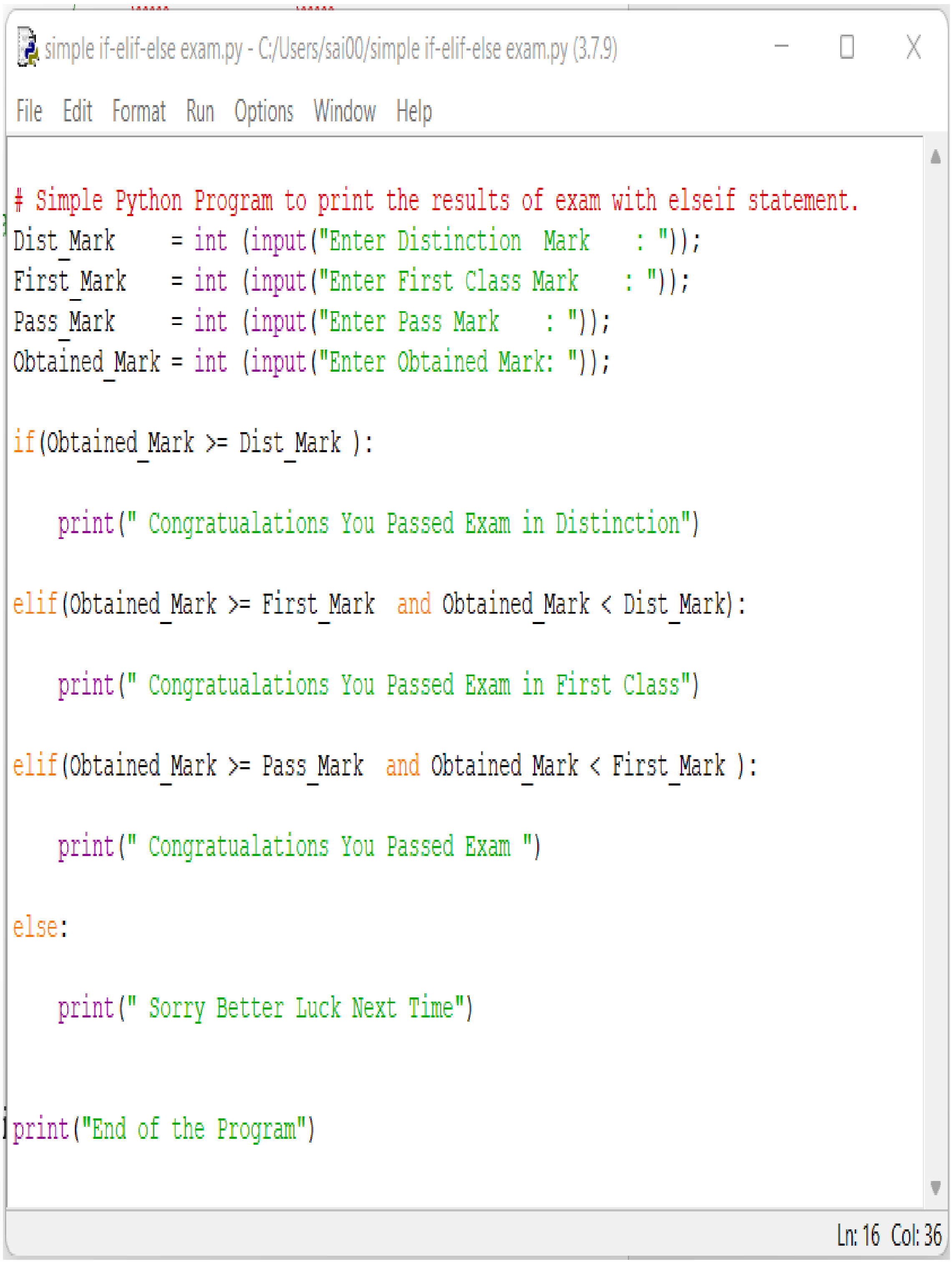


Fig 4.3. Flow Chart of else-if ladder Statement

Example:

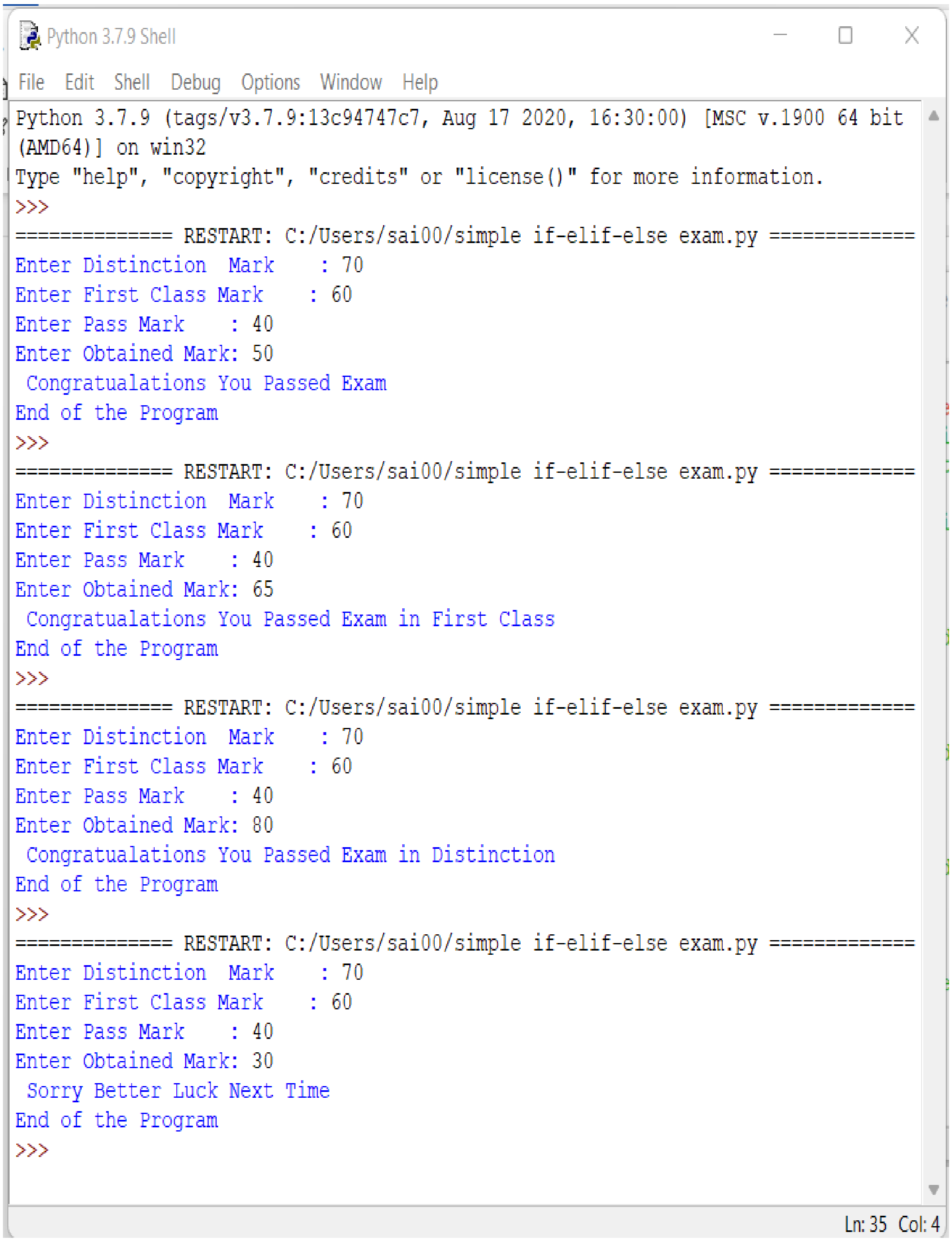
In the code below, the condition (Obtained_Mark >= Dist_Mark) is tested; if it is successful, the if-block of code is executed; otherwise, the following succeeding blocks are executed based on the criteria; otherwise, the else statement and the end statement are executed. The code is executed four times; the first time the condition is met (i.e., 50 > 40), the message "**Congratulations You Passed Exam**" is displayed, and the last statement, "**End of the Program**", is printed. However, the second time run condition (65 > 60) is successful and prints "**Congratulations You Passed Exam in First Class**" before displaying the last line, "End of the Program". Similarly, in the third run, the requirement (i.e., 80 > 70) is met, and the message "**Congratulations You Passed Exam in Distinction**" is displayed, followed by the final sentence "End of the Program". During the last run, if the condition (i.e., 30 < 40) is not met, the else block is activated and the message "**Sorry, Better Luck Next Time**" is written. The last statement displayed is "**End of the Program**".

Example:


simple if-elif-else exam.py - C:/Users/sai00/simple if-elif-else exam.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 16 Col: 36

```
# Simple Python Program to print the results of exam with elseif statement.
Dist_Mark      = int (input("Enter Distinction  Mark    : "));
First_Mark     = int (input("Enter First Class Mark    : "));
Pass_Mark      = int (input("Enter Pass Mark    : "));
Obtained_Mark = int (input("Enter Obtained Mark: "));

if(Obtained_Mark >= Dist_Mark ):

    print(" Congratualations You Passed Exam in Distinction")

elif(Obtained_Mark >= First_Mark  and Obtained_Mark < Dist_Mark):

    print(" Congratualations You Passed Exam in First Class")

elif(Obtained_Mark >= Pass_Mark  and Obtained_Mark < First_Mark ):

    print(" Congratualations You Passed Exam ")

else:

    print(" Sorry Better Luck Next Time")


print("End of the Program")
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:/Users/sai00/simple if-elif-else exam.py ============
Enter Distinction  Mark    : 70
Enter First Class Mark    : 60
Enter Pass Mark    : 40
Enter Obtained Mark: 50
 Congratualations You Passed Exam
End of the Program
>>>
============= RESTART: C:/Users/sai00/simple if-elif-else exam.py ============
Enter Distinction  Mark    : 70
Enter First Class Mark    : 60
Enter Pass Mark    : 40
Enter Obtained Mark: 65
 Congratualations You Passed Exam in First Class
End of the Program
>>>
============= RESTART: C:/Users/sai00/simple if-elif-else exam.py ============
Enter Distinction  Mark    : 70
Enter First Class Mark    : 60
Enter Pass Mark    : 40
Enter Obtained Mark: 80
 Congratualations You Passed Exam in Distinction
End of the Program
>>>
============= RESTART: C:/Users/sai00/simple if-elif-else exam.py ============
Enter Distinction  Mark    : 70
Enter First Class Mark    : 60
Enter Pass Mark    : 40
Enter Obtained Mark: 30
 Sorry Better Luck Next Time
End of the Program
>>>
```

4.3.4 Nested if-else statements

Nested "if-else" statements indicate that one "if" or "if-else" statement is contained within another if or if-else block. Python has this feature as well, which allows us to verify several conditions in a single application.

Syntax:

```
        if (EXPRESSION-1 = = TRUE):

           if (EXPRESSION-2 = = TRUE):

               Inner-If-Statement (Body of the block)

           else:

               Inner-else-Statement (Body of the block)

        else:

          Outer-else-Statement (Body of the block)
```

The syntax used above obviously shows that the if block will include another if block, and so on. If block can have 'n' number of if blocks within it.

- In the below flow chart, (EXPRESSION-1 = = TRUE) is executed successfully, then (EXPRESSION-2 = = TRUE) is tested;
- if it is executed successfully, then if- Block of code related EXPRESSION-2 will be executed;
- otherwise, Block of code related EXPRESSION-2 will be executed; otherwise, Block of code related EXPRESSION-1 will be executed.

The Flow Chart of nested if Statement is shown in Figure 4.4.

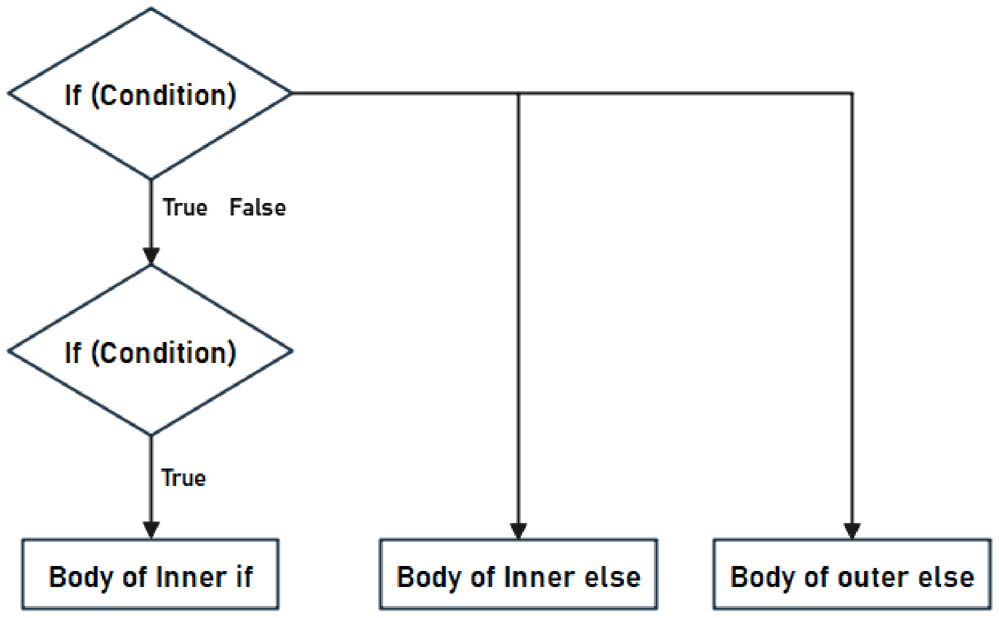
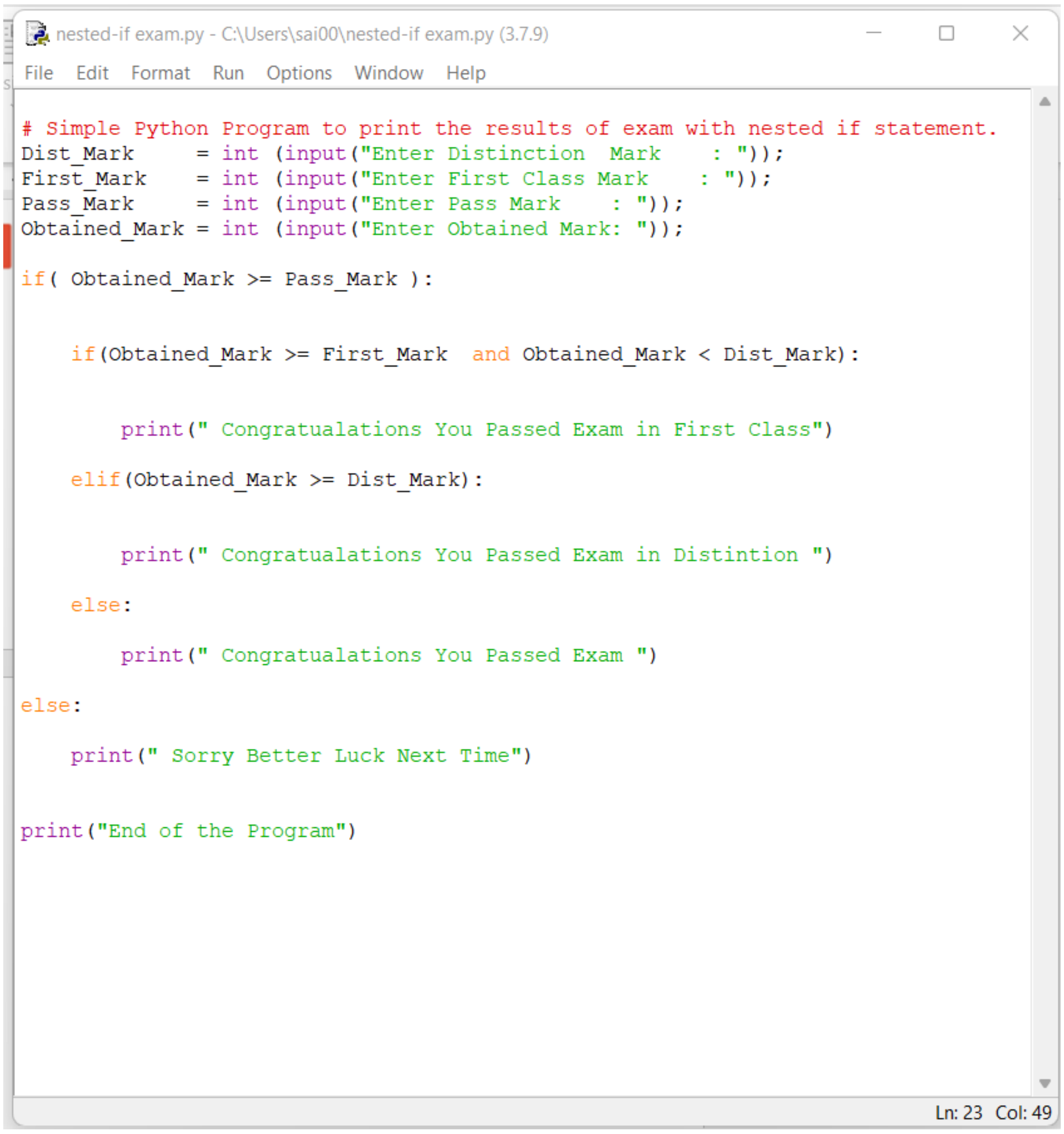


Fig 4.5. Flow Chart for nested-if Statement

Example: 1

In the code below, the condition (Obtained_Mark > = Pass_Mask) is tested. If it is successful, the inner if-statement (Obtained_Mark > = First_Mask and Obtained_Mark < Dist_Mask) is tested. If it is successful, the block-related inner condition is executed. Otherwise, the block-related inner condition is executed. Otherwise, the else block from the outer condition is executed. The ensuing blocks are run based on the circumstances; otherwise, the else statement is executed, followed by the end statement.

Example:

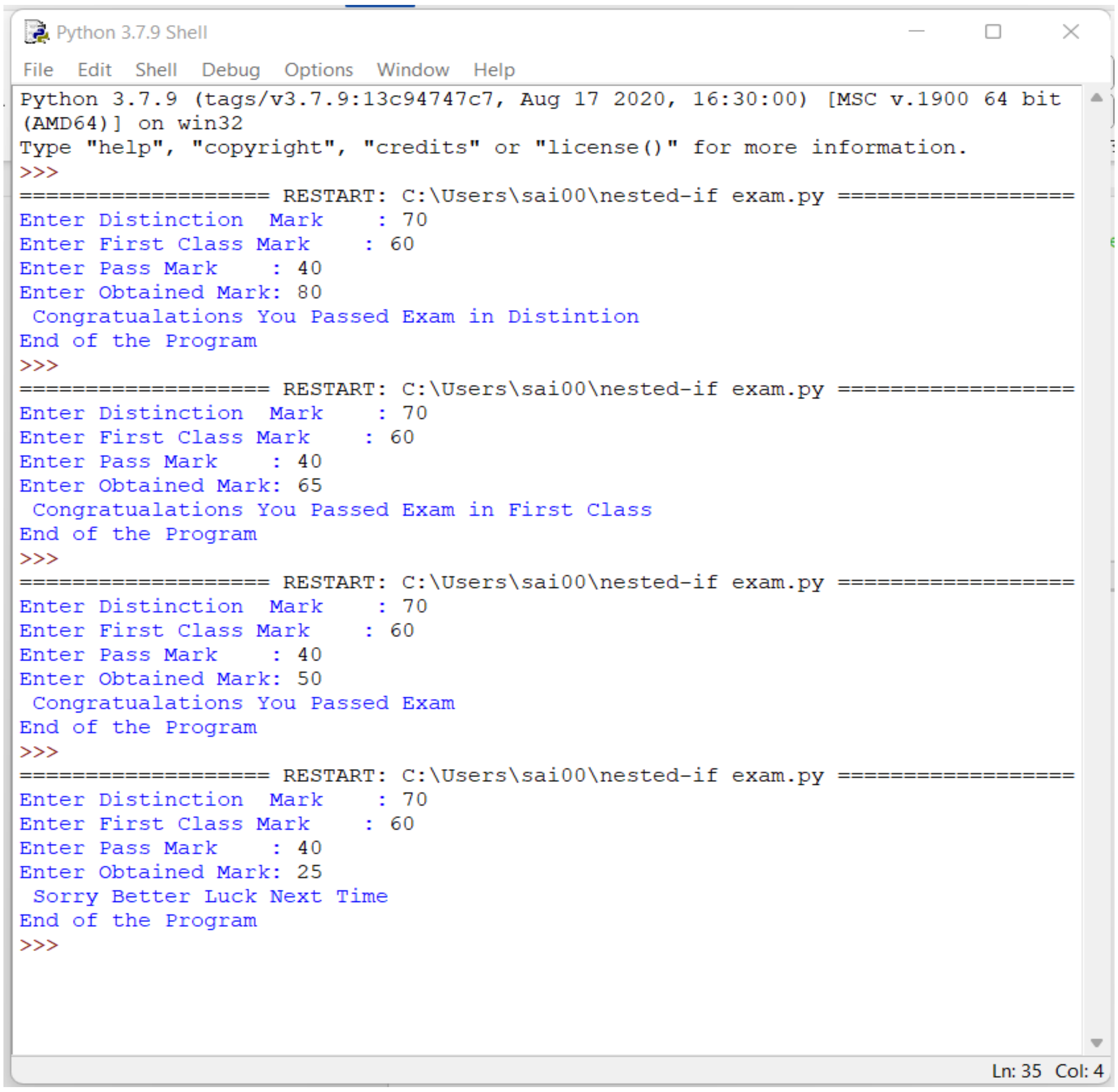

nested-if exam.py - C:\Users\sai00\nested-if exam.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 23 Col: 49

```
# Simple Python Program to print the results of exam with nested if statement.
Dist_Mark     = int (input("Enter Distinction  Mark    : "));
First_Mark    = int (input("Enter First Class Mark    : "));
Pass_Mark     = int (input("Enter Pass Mark    : "));
Obtained_Mark = int (input("Enter Obtained Mark: "));

if( Obtained_Mark >= Pass_Mark ):


    if(Obtained_Mark >= First_Mark  and Obtained_Mark < Dist_Mark):


        print(" Congratualations You Passed Exam in First Class")

    elif(Obtained_Mark >= Dist_Mark):


        print(" Congratualations You Passed Exam in Distintion ")

    else:

        print(" Congratualations You Passed Exam ")

else:

    print(" Sorry Better Luck Next Time")


print("End of the Program")
```

The code is executed four times. The first time, the condition is successful (i.e., 80 > 40), and the second time, the condition is likewise successful (i.e., 80 > 70), and the message "**Congratulations You Passed Exam in Distinction**" is displayed, followed by the last statement, "**End of the Program**". However, the condition is successful the second time

(65>40) and then tested (65>60) and printed "**Congratulations You Passed Exam in First Class**" and displayed the last statement, "End of the Program". Similarly, in the third run, the condition is successful (i.e., 50 > 40), and then tested (i.e., 50 > 40), which is successful and prints "**Congratulations You Passed Exam**" and displays the last statement, "**End of the Program**". If the condition is not met (i.e., 25 < 40), the else block is activated and the message "**Sorry, Better Luck Next Time**" is written. The last statement displayed is "**End of the Program**".

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:\Users\sai00\nested-if exam.py ==================
Enter Distinction  Mark    : 70
Enter First Class Mark    : 60
Enter Pass Mark     : 40
Enter Obtained Mark: 80
 Congratualations You Passed Exam in Distintion
End of the Program
>>>
=================== RESTART: C:\Users\sai00\nested-if exam.py ==================
Enter Distinction  Mark    : 70
Enter First Class Mark    : 60
Enter Pass Mark     : 40
Enter Obtained Mark: 65
 Congratualations You Passed Exam in First Class
End of the Program
>>>
=================== RESTART: C:\Users\sai00\nested-if exam.py ==================
Enter Distinction  Mark    : 70
Enter First Class Mark    : 60
Enter Pass Mark     : 40
Enter Obtained Mark: 50
 Congratualations You Passed Exam
End of the Program
>>>
=================== RESTART: C:\Users\sai00\nested-if exam.py ==================
Enter Distinction  Mark    : 70
Enter First Class Mark    : 60
Enter Pass Mark     : 40
Enter Obtained Mark: 25
 Sorry Better Luck Next Time
End of the Program
>>>
```

4.4 SUMMARY

Conditional statements, which include if, else, and elif statements, are essential programming structures that enable you to control the execution of your program based on the conditions that you define. They make it possible for your program to make decisions and then execute different codes based on those decisions after they have been made . In this chapter, we have seen various examples of how to utilize these statements in Python. Some of the examples include student performance test, number test and etc.

4.5 TECHNICAL TERMS

Conditional Statement, expression, if , elif , nested if, reusability, flexibility, state machine

4.6 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Illustrate the concept of elif statement in python.
2. Describe about if-else statements in python.
3. Differentiate among simple if and if-else statement.

Short Notes:

Write about nested if-statement with example.

1. Discuss about advantages and disadvantage of conditional statements.
2. Write a python program to find biggest among three numbers.

4.7 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. "Think Python: How to Think Like a Computer Scientist" by Allen Downey
4. "Python Cookbook" by David Beazley and Brian K. Jones
5. "Programming Python" by Mark Lutz

MR. G . V. SURESH

LESSON- 5

PYTHON LOOP & CONTROL STATEMENTS

AIMS AND OBJECTIVES

The primary goal of this chapter is to understand the concept of looping and statements in Python programming. The chapter began with an understanding of basic types of loop and statements in python, discuss each one detail and so on. After completing this chapter, the student will understand how to work with loop and control statements in python in terms programs.

STRUCTURE

5.1. INTRODUCTION

All programs, regardless of their programming language, automatically follow a sequential flow. In a function, the first statement is run first, then the second, and so on. A scenario can arise when the programmer needs to run a code block multiple time. To achieve this, programming languages offer a variety of loop types that can repeatedly execute a specific code. In this chapter, we will discuss Python looping statements.

As you may know, Python uses loops to repeatedly iterate over a section of code. But after a certain circumstance is satisfied, you can desire to change the direction of control. This is where Python's control statements are useful. Python control statements, their various forms, and their applications will all be covered in this chapter.

5.2. PYTHON LOOP AND CONTROL STATEMENTS

It could be necessary to repeat a block of code more than once in some circumstances. Programming languages offer a variety of loops to handle this issue, which enable a series of instructions to be repeated until a predetermined condition is satisfied. We'll talk about the many kinds of looping statements that Python offers here. The Python control statements that regulate how Looping Statements flow. Control statements are an essential aspect of any programming language, including Python.

Control statements in Python are used to manage the flow of execution of a program based on certain conditions. Control statements in Python are a powerful tool for managing the flow of execution. They allow developers to make decisions based on specific conditions and modify the normal sequential flow of a program. By using control statements effectively, developers can write more efficient and effective code.

5.2.1 Advantages and disadvantages of Loop Statements

Advantages of Loop Statements:

- Loops speed up software execution by repeating instructions. This is especially significant when processing many database records or iterating over an array of information. Loops speed up and use less computing power than writing down the same instructions for each repetition.

- Without duplicating code, programmers can design a single piece of code that can be executed several times using a loop. This simplifies program debugging and modification by reducing code writing and maintenance. Loops make it easier to vary the number of times a set of instructions is executed without modifying the code many times.

- Loops increase programming flexibility. Loops let programmers develop adaptable programs by repeating a set of instructions a certain number of times.

- Loops can do operations in multiple orders or under varied situations, giving them a great tool for developing algorithms that can handle many scenarios.

Disadvantages of Control Statements:

- If you have a look at the examples of processing collections using loops that are provided below, you will notice that the majority of the methods have logic that is very similar to cycle through the elements.

- Coding that is not related to business logic takes up more of our time.

- It leads to an excessive amount of code, which might become a problem when it comes to maintenance.

5.2.2 Advantages and disadvantages of Control Statements

Advantages of Control Statements:

- The software engineer can indicate the conditions under which parts of code should be run using control explanations.
- It is possible to use control explanations to construct intricate decision-making systems within a program.
- Mistake handling and other exception-handling tasks are among the tasks that can be accomplished with their assistance.
- The usage of control statements allows for the formation of uniform structures inside a program, which makes it easier to read and understand the program.
- The formation of settled structures, which may then be utilized to construct more sophisticated decision-making structures, can be accomplished through the utilization of control articulations.

Disadvantages of Control Statements:

- If misused, control statements can complicate and make code difficult to find.
- Control statements, especially if misused, can make code difficult to understand.
- In excess or wrong use, control statements can slow program execution. For large or time-sensitive programs, this may slow program execution.

5.3. TYPES OF LOOP STATEMENTS

As is the case with other programming languages, Python has three distinct types of looping statements, which are provided in the following order:

- For Loop Statement
- While Loop Statements
- Nested Loop Statement

5.3.1 For Loop Statement

It is possible to iterate over a series of elements in Python by using the for loop, which is one of the looping instructions contained inside the language. There are a variety of objects that can be iterated, including a list, a tuple, a text, and any other object.

Syntax:

```
for variable in sequence:
    # Code block to be executed
```

The preceding syntax,

- variable is a temporary variable that stores the value of each element in the sequence during each iteration of the loop
- The code block that comes after the for statement is carried out many times for each individual element that is included in the sequence.

Example:

for **i** in **10**:

# Code block to be executed

Total 10 time block will be repeated

The flowchart to represent for loop statement in python is shown in Figure 5.2

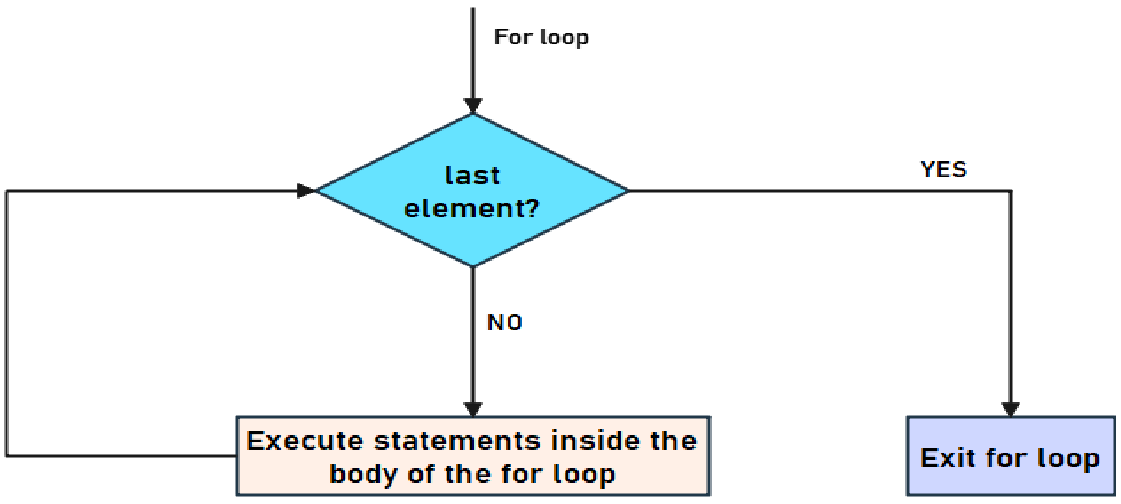
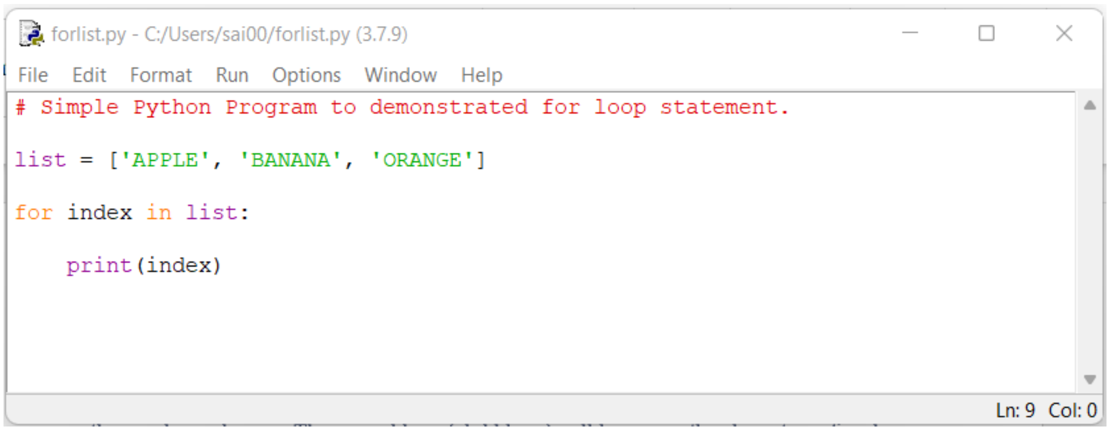


Fig 5.2. Flowchart of For-loop Statement

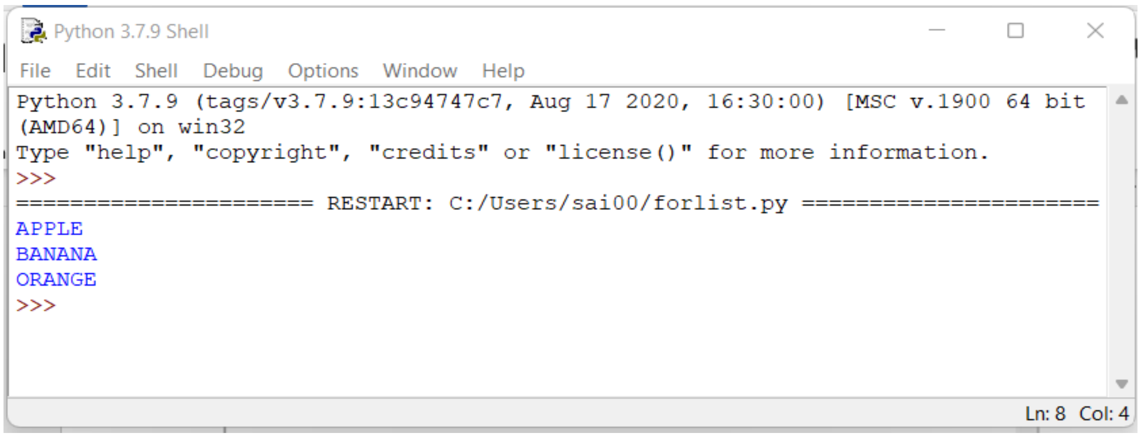
Example :


forlist.py - C:/Users/sai00/forlist.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 9 Col: 0

```
# Simple Python Program to demonstrated for loop statement.

list = ['APPLE', 'BANANA', 'ORANGE']

for index in list:

    print(index)
```

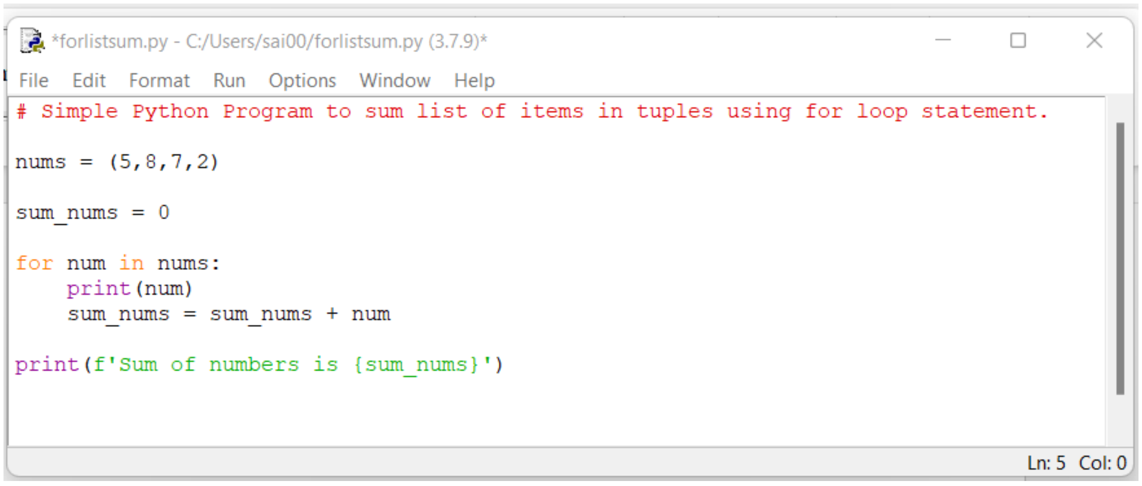
The code that you see above has a for loop that prints each element of the 'list' list on a new line after iterating over each entry in the list. The output is shown on the next page.

Output:



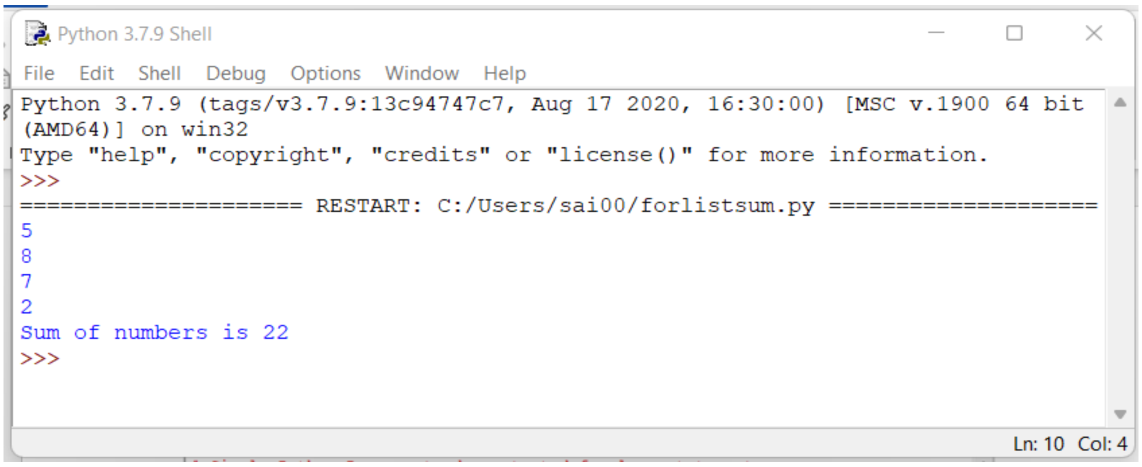
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/forlist.py ======================
APPLE
BANANA
ORANGE
>>>
```

Example 2:


forlistsum.py - C:/Users/sai00/forlistsum.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 5 Col: 0

```
# Simple Python Program to sum list of items in tuples using for loop statement.

nums = (5,8,7,2)

sum_nums = 0

for num in nums:
    print(num)
    sum_nums = sum_nums + num

print(f'Sum of numbers is {sum_nums}')
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/forlistsum.py ====================
5
8
7
2
Sum of numbers is 22
>>>
```

Using the code that was just presented, the for loop will iterate over each element in the tuple that is referred to as 'num' and then display it on a new line. In addition, the sum of each number was computed, the result was saved in the "sum_nums" variable, and the sum value was eventually printed out. In the run tuple, a sequence of distinct integers (5,8,7,2) is used, and the result is "the sum of the numbers is 22"

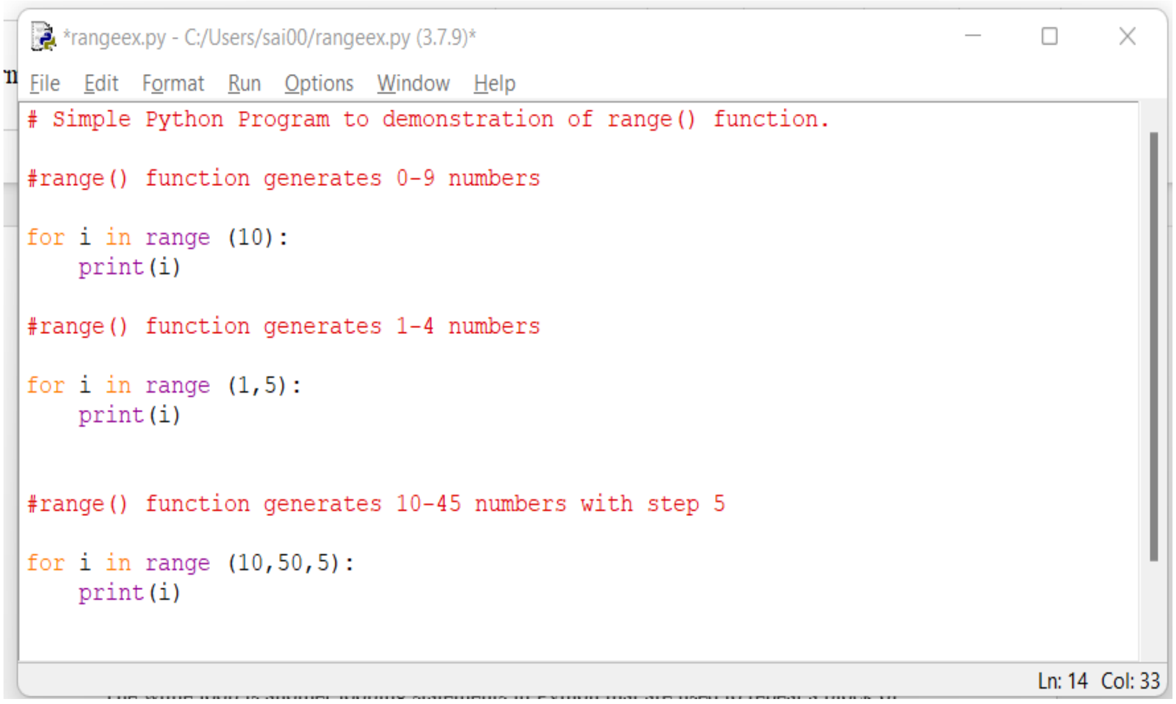
According to Python, a range object is a sequence of numbers that cannot be changed. When using a for loop, it is helpful to keep track of the number of times a block is repeated.

You can use the range() method in the following ways:

range ([start], stop, [step])

Every one of the three arguments must be an integer. The value of the [start] parameter is always set to zero, unless an alternative number is provided. The only parameter that is required for the function described above is stop. It is one less than the stop parameter that the last integer in the series is. In the intervals between, the [step] value, which is set to 1 by default, is used to increment the numbers.

Example:

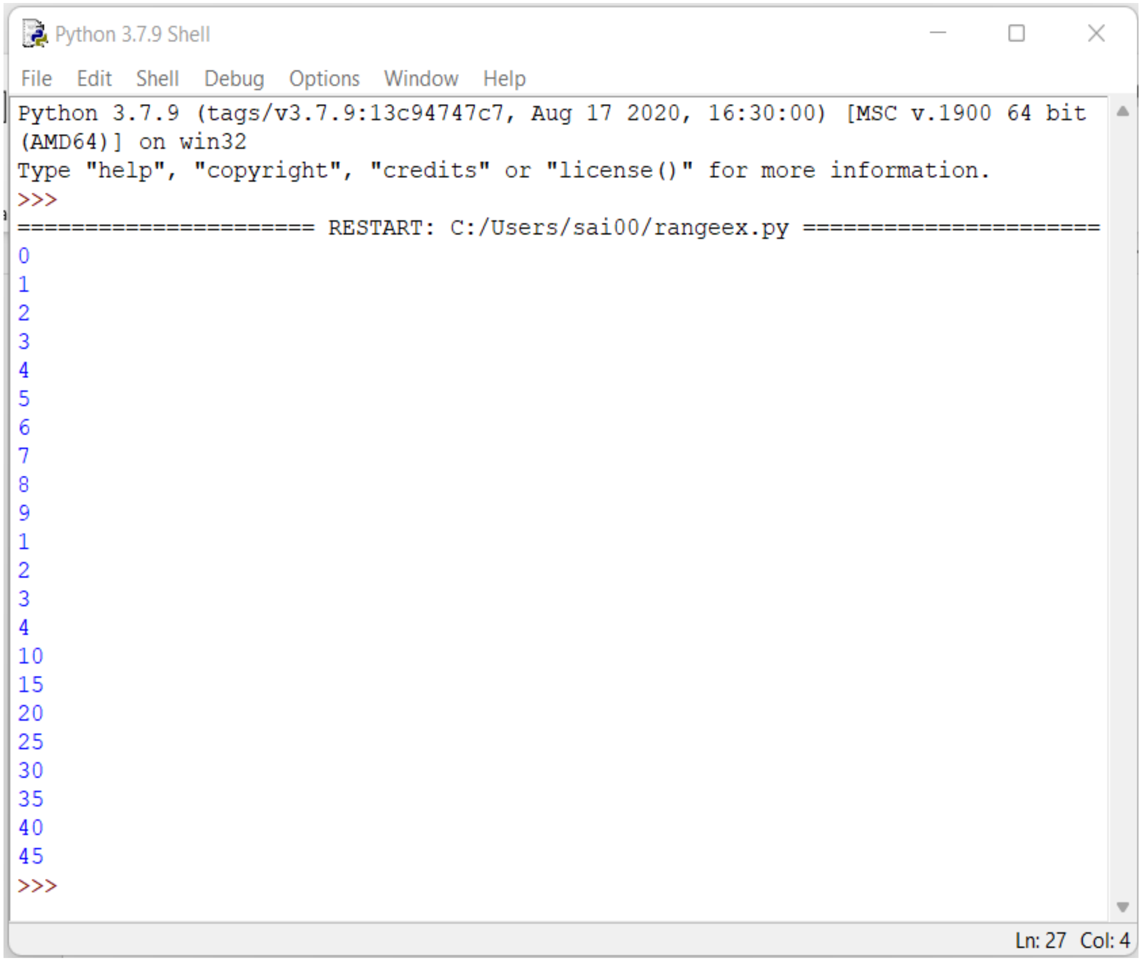

rangeex.py - C:/Users/sai00/rangeex.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 14 Col: 33

```
# Simple Python Program to demonstration of range() function.

#range() function generates 0-9 numbers

for i in range (10):
    print(i)

#range() function generates 1-4 numbers

for i in range (1,5):
    print(i)


#range() function generates 10-45 numbers with step 5

for i in range (10,50,5):
    print(i)
```

The range() method was used instead of a for loop statement in the Python code above. Three for loop statements in all, each printing a distinct range of numbers according on the inputs passed to the range () function.

When the first "10" value was entered into range (10) it produced numbers starting at 0 and ending with 10-1, or 9. A for-loop statement is then given range(1,5), and values are printed starting at 1 and ending at end 5-1, or 4. Lastly, range(10,50,5) is sent to the for-loop expression, which outputs values starting at 10 and ending at 50-4, or 45, because step=5.

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/Users/sai00/rangeex.py =====================
0
1
2
3
4
5
6
7
8
9
1
2
3
4
10
15
20
25
30
35
40
45
>>>
```

5.3.2 While Loop Statement

Another Python looping expression used to repeat a block of code until a predetermined condition is met is the while loop.

Syntax:

The syntax of the while loop in Python is given below.

```
while condition:

    # Code block to be executed
```

A boolean expression called condition in this syntax is evaluated at the beginning of each loop iteration. The while statement is followed by a code block that is periodically run until the condition evaluates to False.

The flowchart to represent while loop statement in python is shown in Figure 5.3

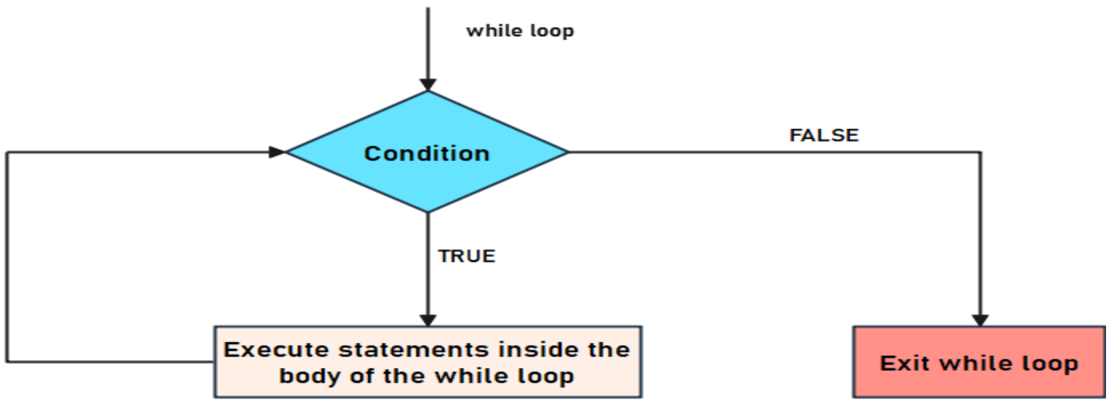
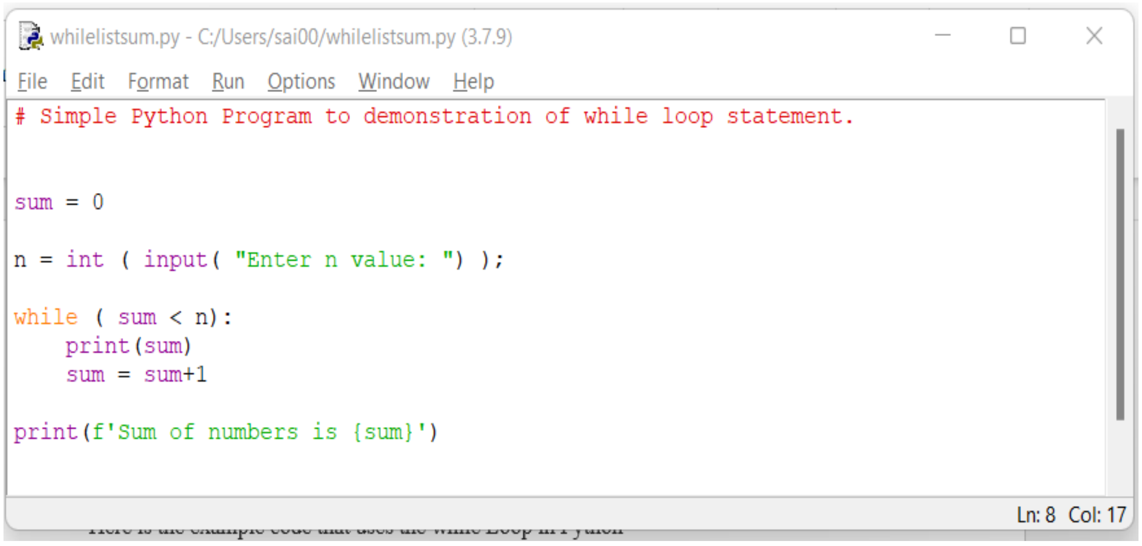


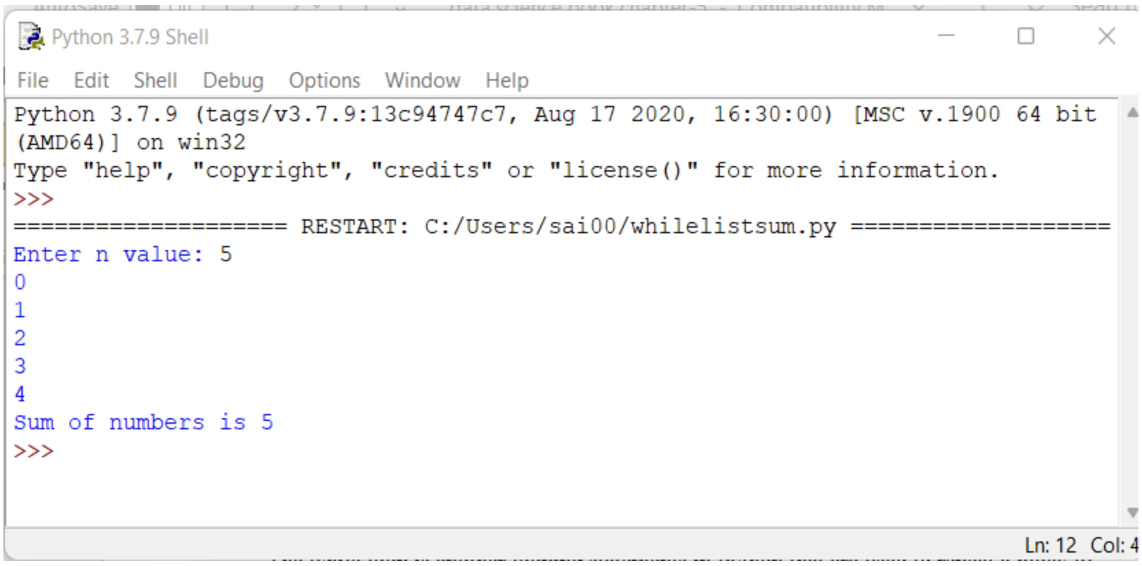
Fig 5.3. Flowchart of While-loop Statement

Example:


whilelistsum.py - C:/Users/sai00/whilelistsum.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 8 Col: 17

```
# Simple Python Program to demonstration of while loop statement.


sum = 0

n = int ( input( "Enter n value: ") );

while ( sum < n):
    print(sum)
    sum = sum+1

print(f'Sum of numbers is {sum}')
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/whilelistsum.py ===================
Enter n value: 5
0
1
2
3
4
Sum of numbers is 5
>>>
```

The code block is repeated here by the while loop until the sum variable is less than 5. As we can see in the output, the sum variable is increased by 1 at each iteration, and the current value of the sum is printed on a new line.

5.3.3 Nested Loop Statement

A loop inside another loop is known as a nested loop in Python. When we wish to loop over a series of components with several degrees of nesting, we utilize it.

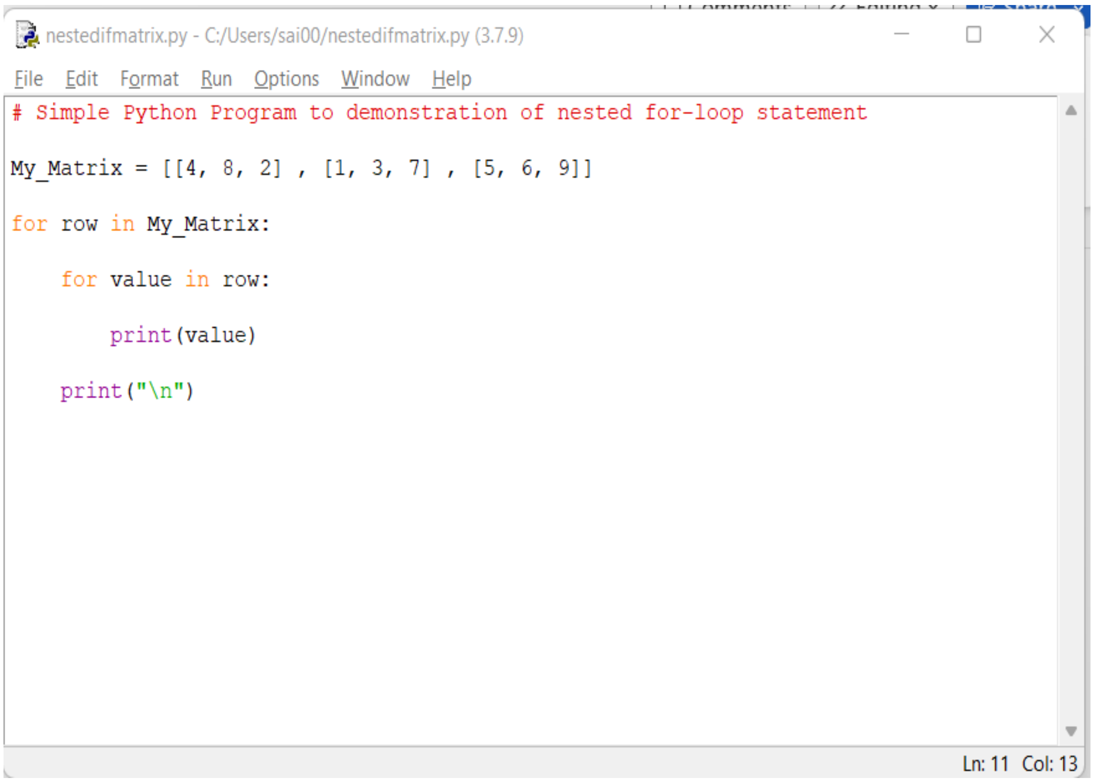
Syntax:

```
for variable in sequence:

    for i_variable in i_sequence:

        # Code block to be executed
```

Variable, as used in this syntax, is a temporary variable that, for each iteration of the outer loop, stores the value of each element in the sequence. Every time the inner loop iterates, the value of every element in the i_sequence is stored in the i_variable, a temporary variable. Every element in the inner sequence and every element in the outer sequence is subjected to several executions of the code block that follows the inner for statement.

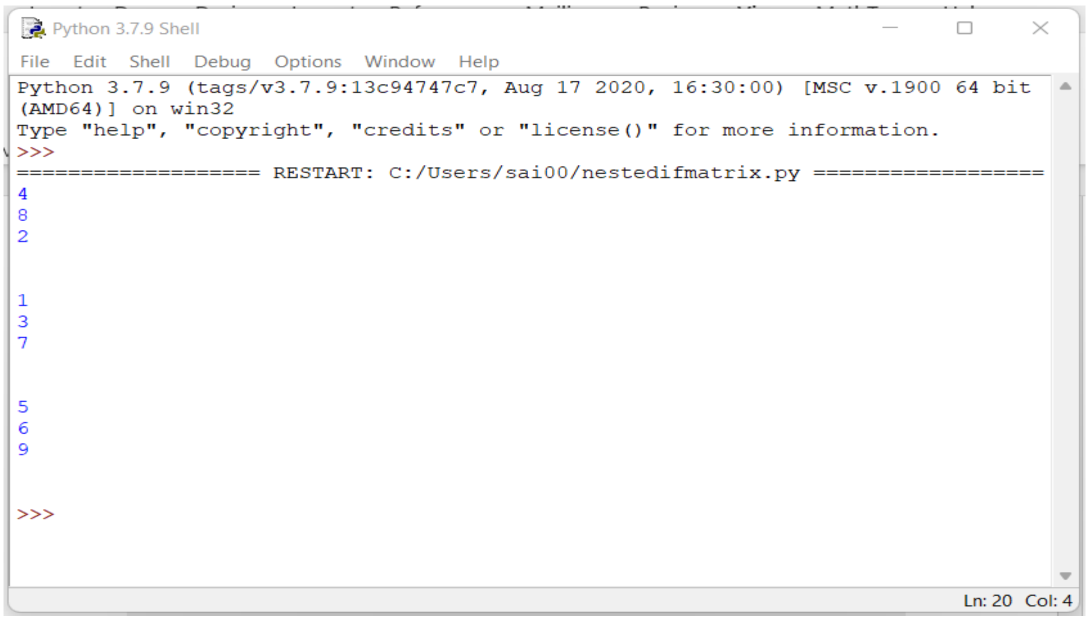
Example :

The code given below uses the Nested Loop.


nestedifmatrix.py - C:/Users/sai00/nestedifmatrix.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 11 Col: 13

```
# Simple Python Program to demonstration of nested for-loop statement

My_Matrix = [[4, 8, 2] , [1, 3, 7] , [5, 6, 9]]

for row in My_Matrix:

    for value in row:

        print(value)

    print("\n")
```

Output:

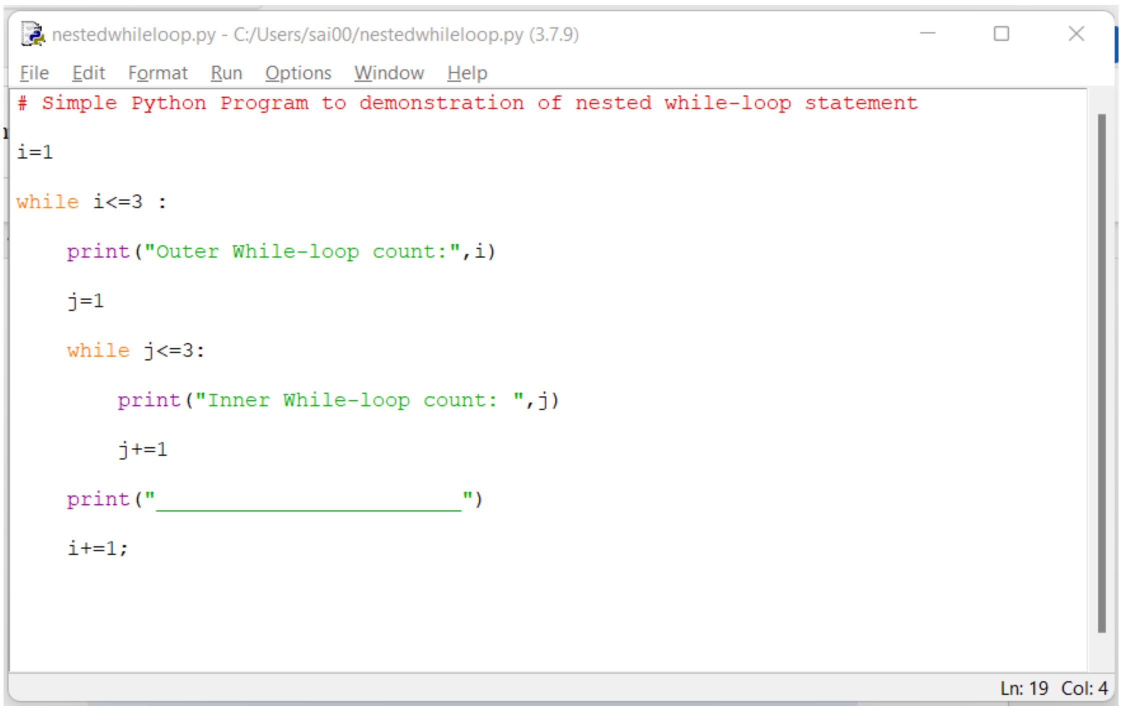


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/nestedifmatrix.py =================
4
8
2


1
3
7


5
6
9


>>>
```

In this example, the nested loop performs an iteration over each item in the ‘matrix' list and then prints the elements on a new line.

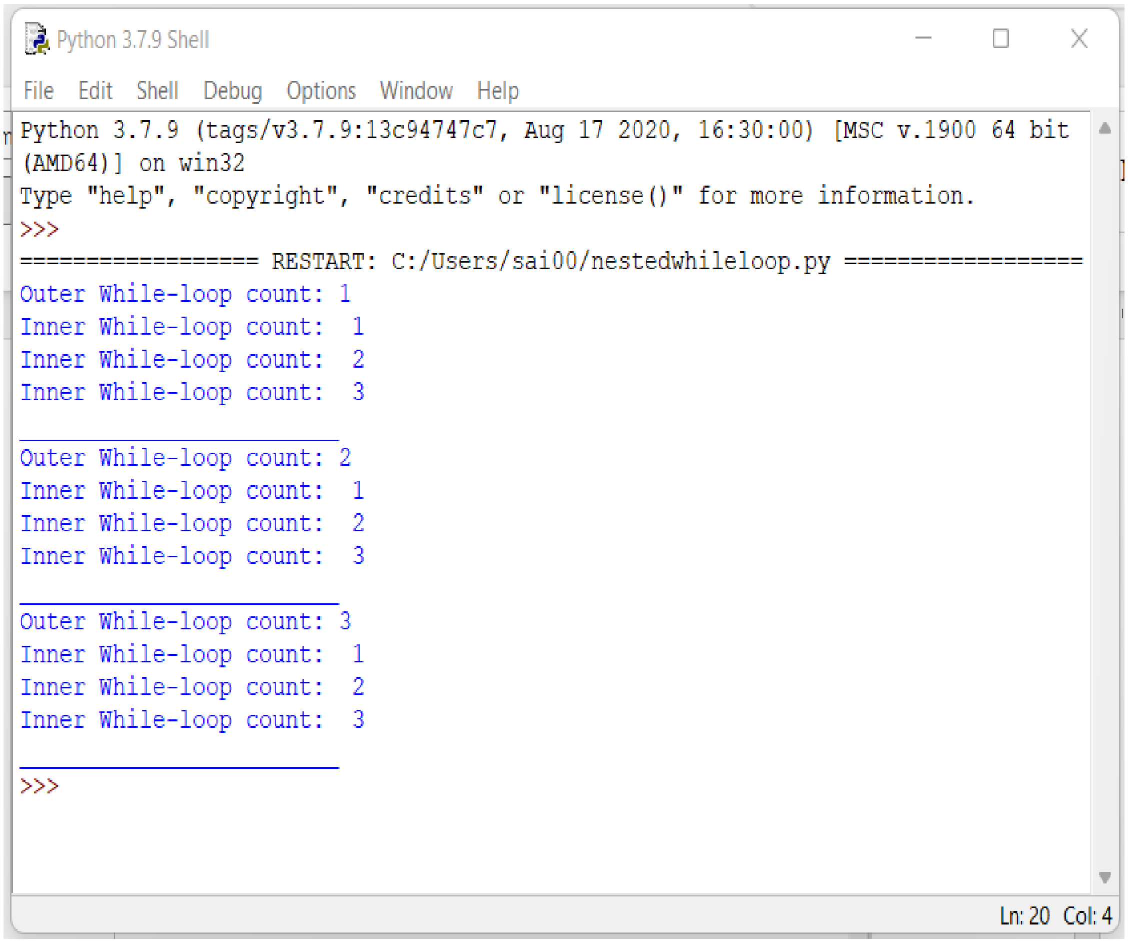
When one while loop is contained within another while loop, the resulting structure is referred to as a nested while loop. We require nested loops in most of our apps.

Example:


nestedwhileloop.py - C:/Users/sai00/nestedwhileloop.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 19 Col: 4

```
# Simple Python Program to demonstration of nested while-loop statement

i=1

while i<=3 :

    print("Outer While-loop count:",i)

    j=1

    while j<=3:

        print("Inner While-loop count: ",j)

        j+=1

    print("________________________")

    i+=1;
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/nestedwhileloop.py ==================
Outer While-loop count: 1
Inner While-loop count:  1
Inner While-loop count:  2
Inner While-loop count:  3
________________________
Outer While-loop count: 2
Inner While-loop count:  1
Inner While-loop count:  2
Inner While-loop count:  3
________________________
Outer While-loop count: 3
Inner While-loop count:  1
Inner While-loop count:  2
Inner While-loop count:  3
________________________
>>>
```

In this example, the nested loop performs an iteration over each item in the ‘matrix' list and then prints the elements on a new line.

5.4 TYPES OF CONTROL STATEMENTS

Python, in addition to loop statements, has three different sorts of control statements, which are given below. These control statements are used to govern the flow of execution.

- Break Statement
- Continue Statement
- Pass Statement

5.4.1 Break Statement

A premature termination of the loop in Python can be accomplished with the help of the break statement. It is utilized in situations in which we wish to exit the loop prior to it having finished all of its iterations.

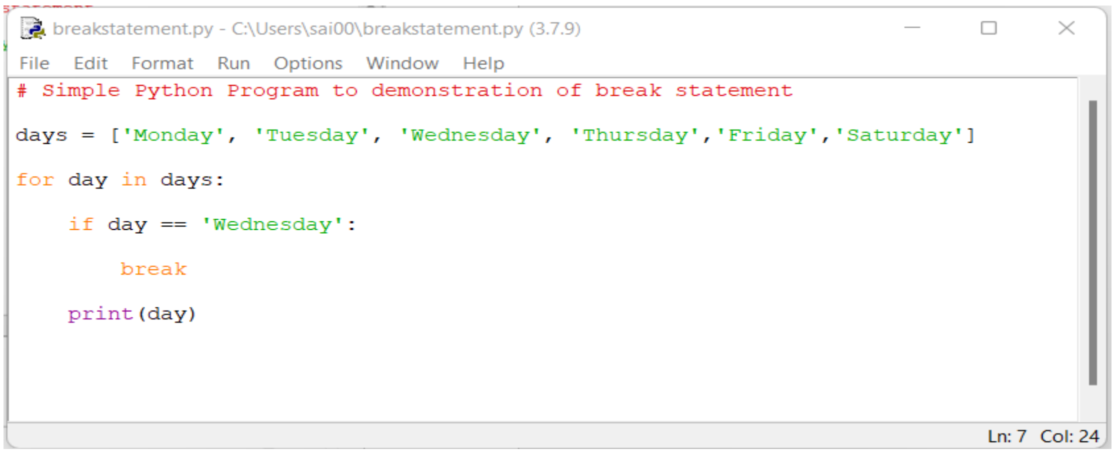
Syntax:

The syntax of the break statement in Python is as follows:

```
for variable in sequence:
        if condition:
          break
```

- The value of each element in the sequence is stored in the variable, which is a temporary variable, and it is used for each iteration of the loop to save the value.
- The condition is a statement that receives a boolean value and is evaluated at the beginning of each iteration of the loop. If the condition is found to be true, the break statement is carried out, therefore bringing an end to the loop.

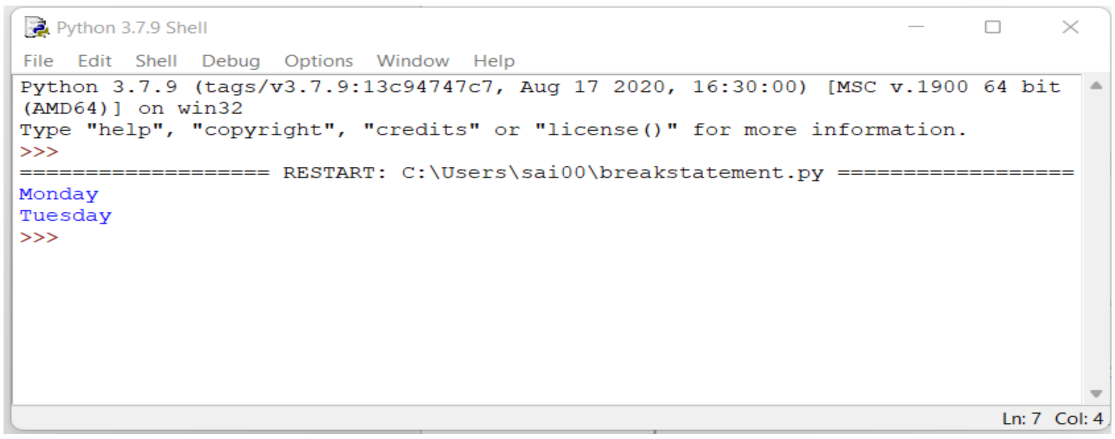
Example:


breakstatement.py - C:\Users\sai00\breakstatement.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 7 Col: 24

```
# Simple Python Program to demonstration of break statement

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday','Friday','Saturday']

for day in days:

    if day == 'Wednesday':

        break

    print(day)
```

The code that you see above has a for loop that outputs each item in the "fruits" list on a new line after iterating over each item in the list. On the other hand, the break statement is executed, and the loop is halted when the value of the "fruit" variable is equal to "banana."

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:\Users\sai00\breakstatement.py ==================
Monday
Tuesday
>>>
```

5.4.2 Continue Statement

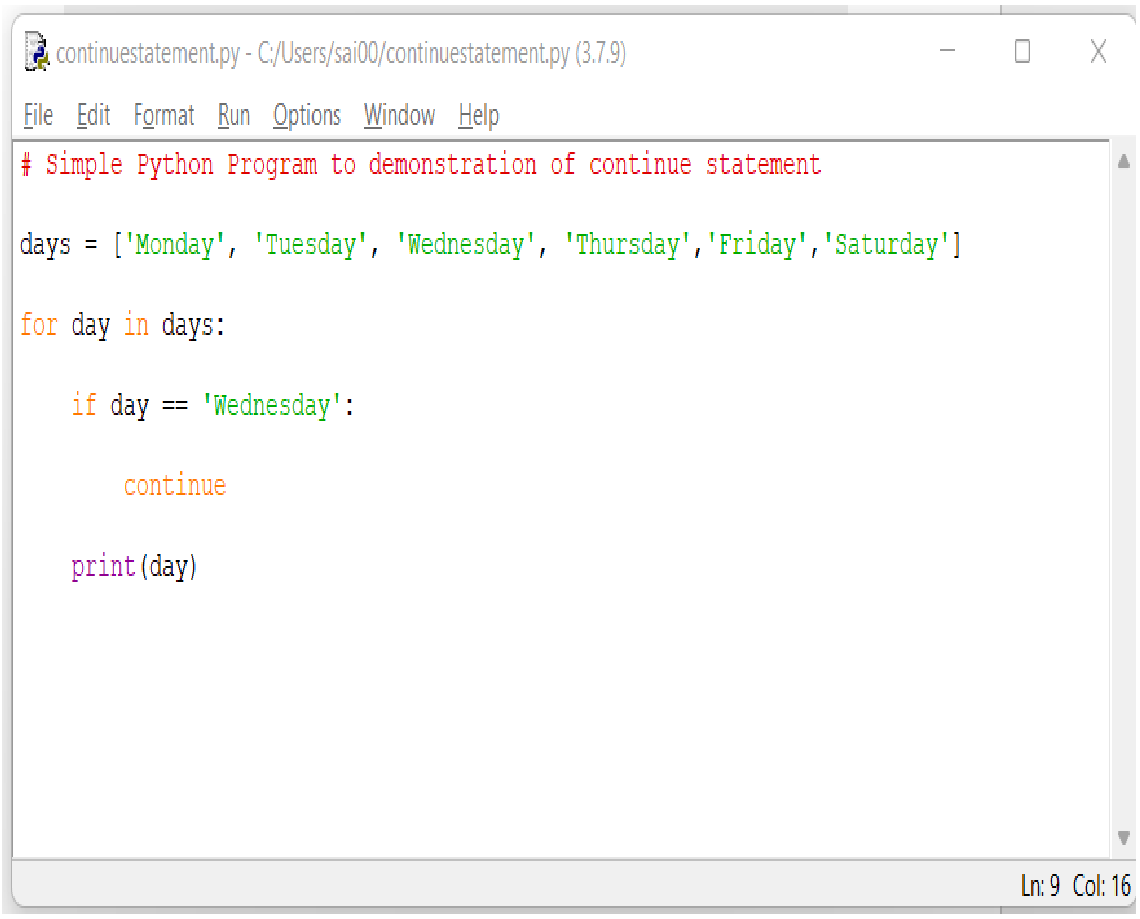
Using the continue statement in Python, one can skip the iteration of the loop that is currently being executed. It is utilized in situations in which we wish to skip a certain component of the sequence and proceed with the subsequent iteration of the loop onward.

Syntax:

```
for variable in sequence:
    if condition:
        continue
    # Code block to be executed
```

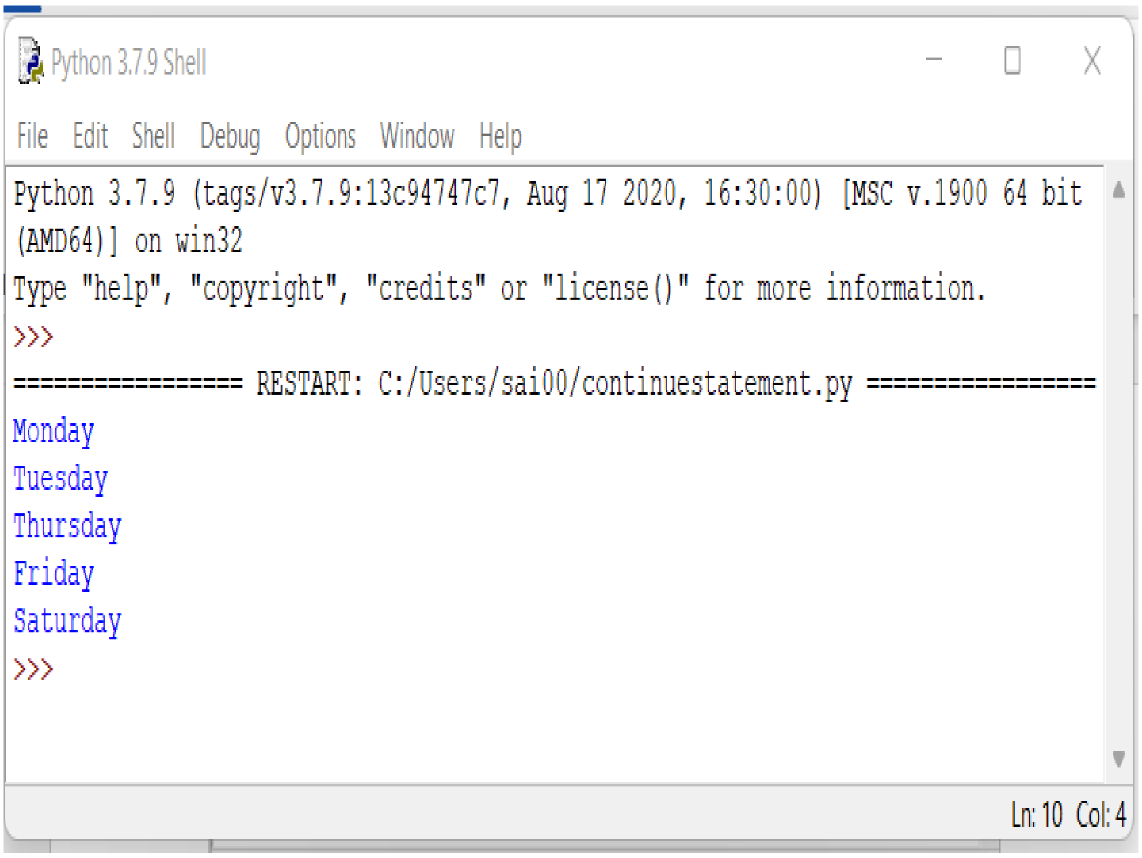
- The value of each element in the sequence is stored in the variable, which is a temporary variable, and it is used for each iteration of the loop to save the value.
- The condition is a statement that receives a boolean value and is evaluated at the beginning of each iteration of the loop.

Example


continuestatement.py - C:/Users/sai00/continuestatement.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 9 Col: 16

```
# Simple Python Program to demonstration of continue statement

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday','Friday','Saturday']

for day in days:

    if day == 'Wednesday':

        continue

    print(day)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/sai00/continuestatement.py ================
Monday
Tuesday
Thursday
Friday
Saturday
>>>
```

The for loop iterates through each item in the "fruits" list in this example, printing each one on a new line. Nevertheless, the loop's current iteration is skipped and the continue statement is executed when the value of the "fruit" variable equals "banana."

5.4.3 Pass Statement

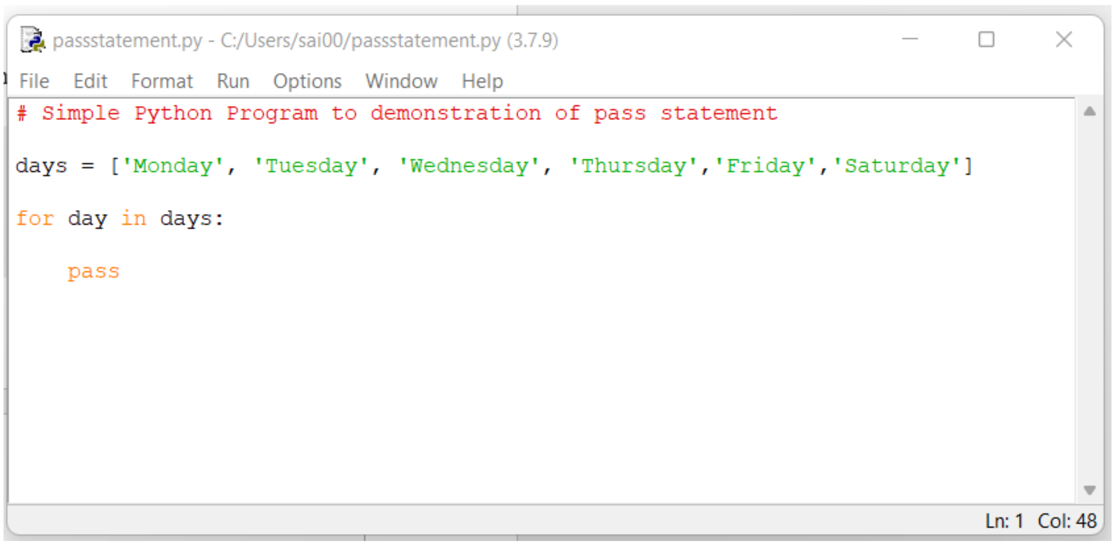
The pass statement is used as a placeholder in Python. It is used when we want to write empty code blocks and want to come back and fill them in later. The syntax of the pass statement in Python is given below.

Syntax:

```
for variable in sequence:
        pass
```

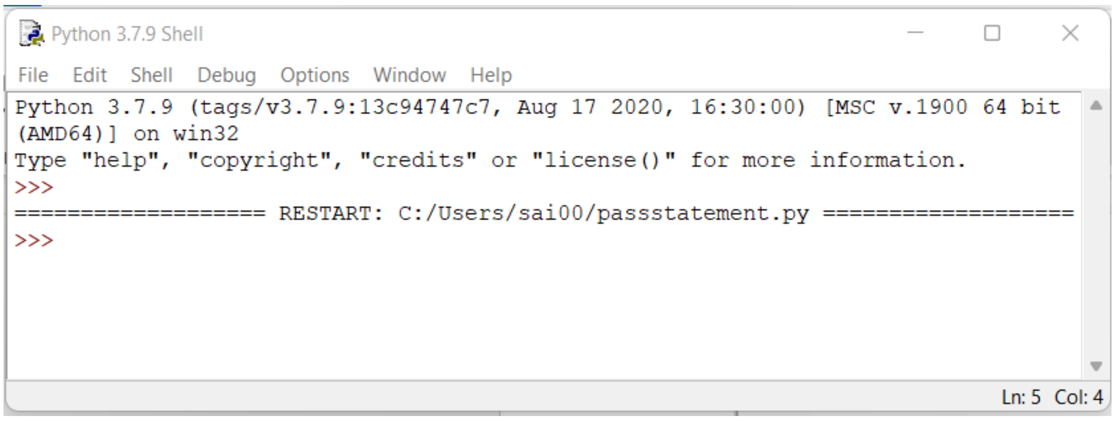
- Every time the loop iterates, the variable—which is a temporary variable—holds the value of every element in the sequence.
- An empty code block is created using the pass statement and is subsequently filled in.

Example:


passstatement.py - C:/Users/sai00/passstatement.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 1 Col: 48

```
# Simple Python Program to demonstration of pass statement

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday','Friday','Saturday']

for day in days:

    pass
```

In this example, the pass statement is used to create an empty code block while the for loop iterates over each element in the "fruits" list.



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/passstatement.py ===================
>>>
```

5.5 SUMMARY

Looping statements are an important component of Python because they allow the programmer to repeat a sequence of instructions until a specified condition is met, which simplifies difficult issues and avoids repetitive code. In Python, looping statements are classified into three types: for loops, while loops, and nested loops, each with their own set of features and applications. We reviewed all these looping statements in Python.

When nothing in a conditional loop's body affects its conditional statement, the loop has the potential to become infinite. The two most common types of conditional loops are While and For loops. You can also specify a range (sequence) of numbers to control how frequently the code executes.

5.6 TECHNICAL TERMS

Loop Statement, Control Statement, For, While, Nested loop, break, continue, pass, reusability, and redundancy

5.7 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Illustrate the concept of Loop statement in python.
2. Describe about control statements in python.
3. Differentiate among break and continue statement.

Short Notes:

1. Write about nested loop with example.
2. Discuss about advantages and disadvantage of loop statements.

5.8 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. "Python Crash Course" by Eric Matthes
4. "Automate the Boring Stuff with Python" by Al Sweigart
5. "Learning Python" by Mark Lutz

Dr. KAMPA LAVANYA

LESSON- 6

PYTHON STRING

AIMS AND OBJECTIVES

The primary goal of this chapter is to grasp the concept of string in Python programming. The chapter began with an understanding of basic definition of string, creating a string, and so on. After completing this chapter, the student will understand how to work with string in python in terms various methods, operations, and functions.

STRUCTURE

6.1. INTRODUCTION

Python strings, like those in many other well-known programming languages, are arrays of bytes that represent unicode characters. Nevertheless, a single character in Python is just a string with a length of 1. Python does not have a character data type. You can access the string's constituents by using square brackets.

Since it is an immutable data type, you are unable to alter a string after you have created it. Strings are extensively utilized in a wide range of applications, including the storing and manipulation of text data as well as the representation of names, addresses, and other text-representable data types. This chapter will cover Python strings, one of the core data types in Python programming, and will cover Python string methods, operators and functions, working with them, and more.

6.2. PYTHON STRING

A string is a sequence of alphabets, words, or other characters. It is one of the most basic data structures, serving as the foundation for data manipulation. Python includes a built-in string class called str. Python strings are "immutable," which implies they cannot be modified once formed.

6.2.1. Creating a Python String

To create a String in python there are three different types of approaches:

- With a Single quotes

  ‘Welcome to the world of "Python" keep Loving.'

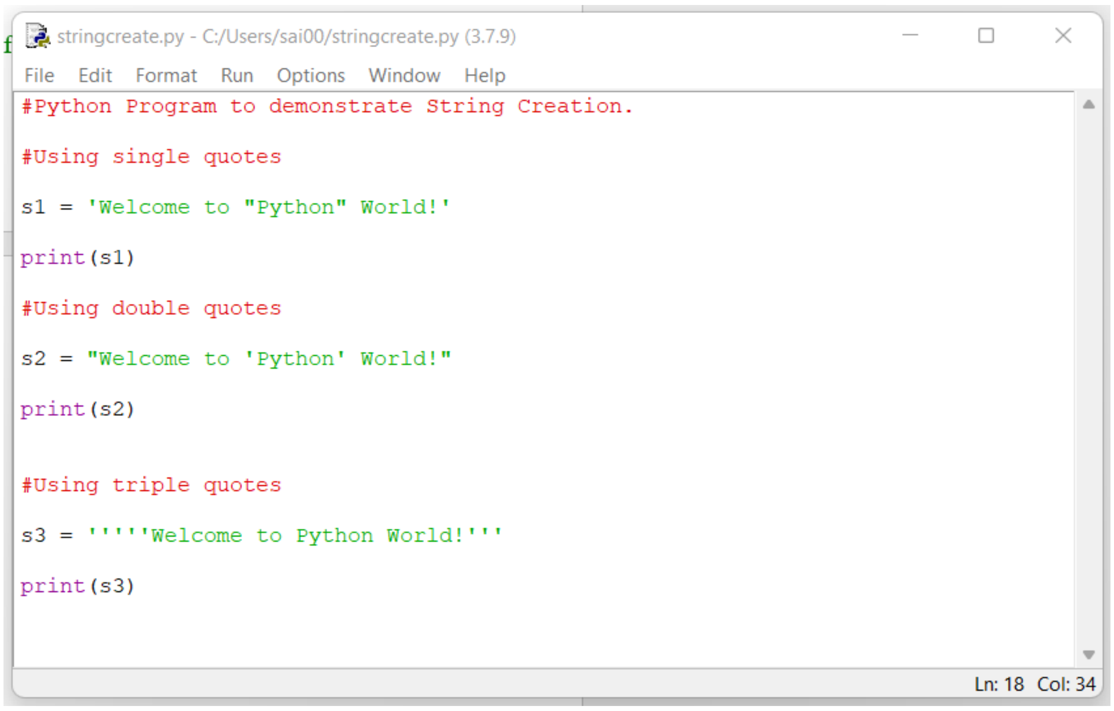
- With a Double quotes.

  “Welcome to the world of ' Python ' keep Loving."

- With a Triple quotes,

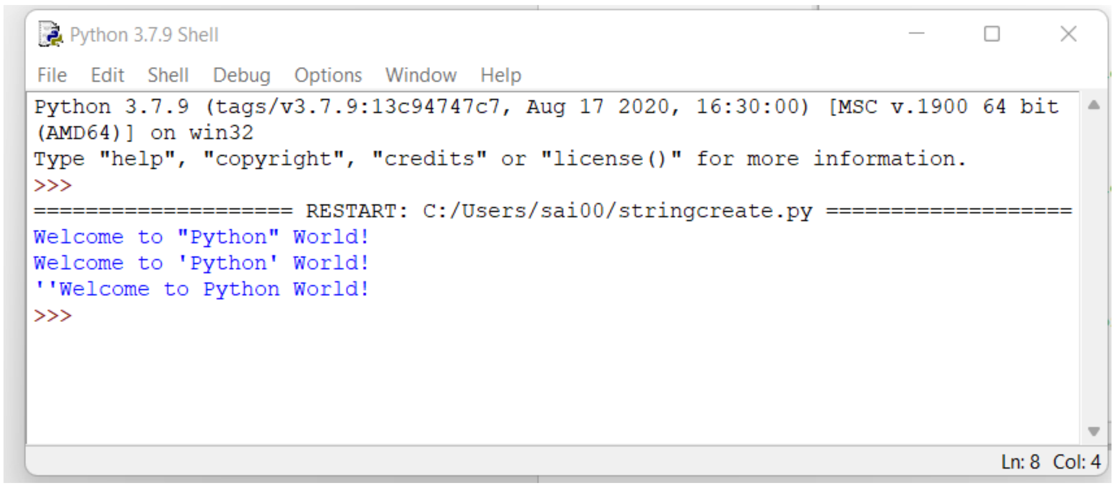
  """ Welcome to the world of Python """, '''Keep Loving.'''

Example:


stringcreate.py - C:/Users/sai00/stringcreate.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 18 Col: 34

```
#Python Program to demonstrate String Creation.

#Using single quotes

s1 = 'Welcome to "Python" World!'

print(s1)

#Using double quotes

s2 = "Welcome to 'Python' World!"

print(s2)


#Using triple quotes

s3 = '''''Welcome to Python World!'''

print(s3)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/stringcreate.py ===================
Welcome to "Python" World!
Welcome to 'Python' World!
''Welcome to Python World!
>>>
```

The above example, where three strings are created namely S1,S2 and S3 in different styles with same content. Finally displayed the three strings output is shown above.

6.2.2. Applications of Python Sting

- Use of string matching algorithms to quickly detect instances of plagiarism in both code and text.
- Strings can be utilized for encoding and decoding purposes, ensuring the secure movement of data from source to destination.
- We are able to offer better filters for the approximate suffix-prefix overlap problem by utilizing strings and the techniques associated with them.
- HTTP requests and responses, among other data exchanged over networks, are encoded and decoded using strings.
- When working with files, you'll need to know that strings are the go-to for reading and writing file names and locations.
- Applications like sentiment analysis and natural language processing make use of strings to glean useful insights from massive text datasets.

6.3 ACCESSING THE STRING

There are three various methods that we can get the characters from the individual String that was already constructed in the previous section. The information is given below:

- Indexing
- Negative Indexing
- Slicing

6.3.1 Indexing

Using index values and treating strings like a list is one method. In Python, the Indexing function can be used to retrieve specific characters from a String. The idea of indexing technique is shown in Figure 6.1.

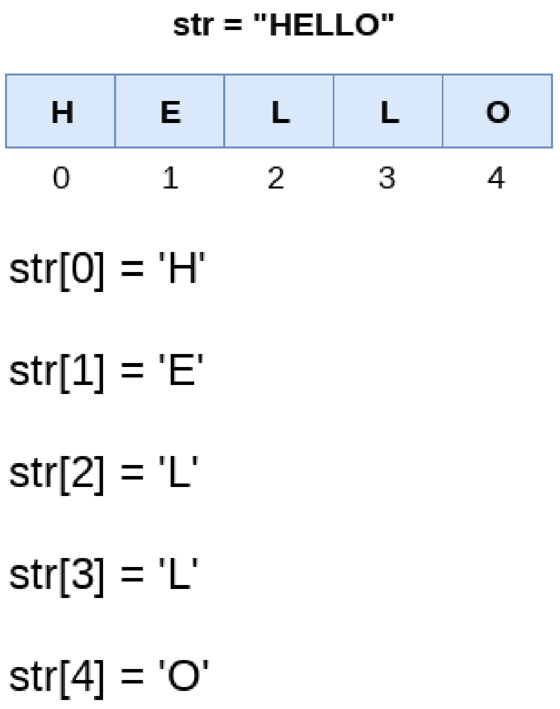
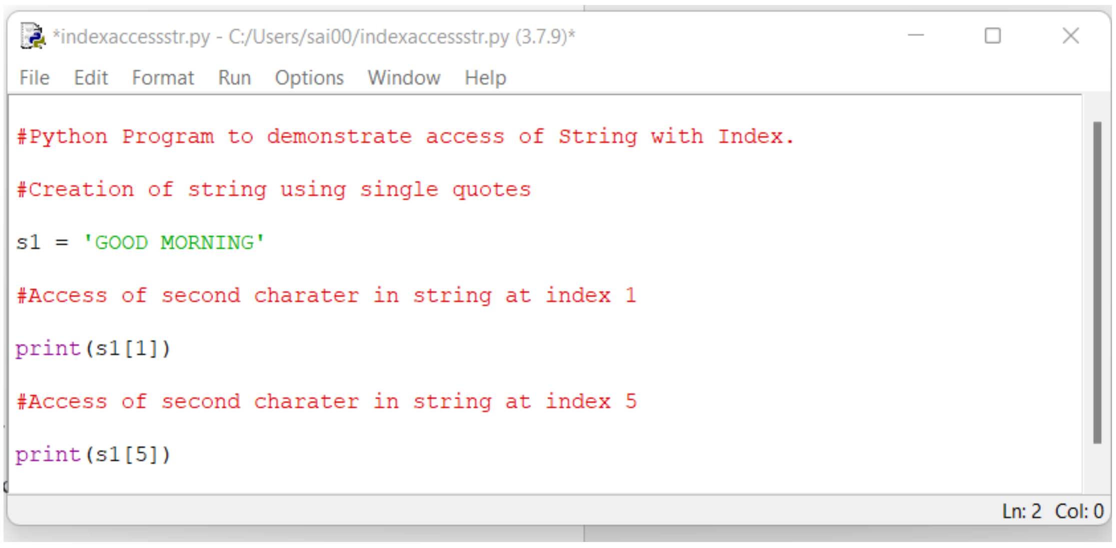


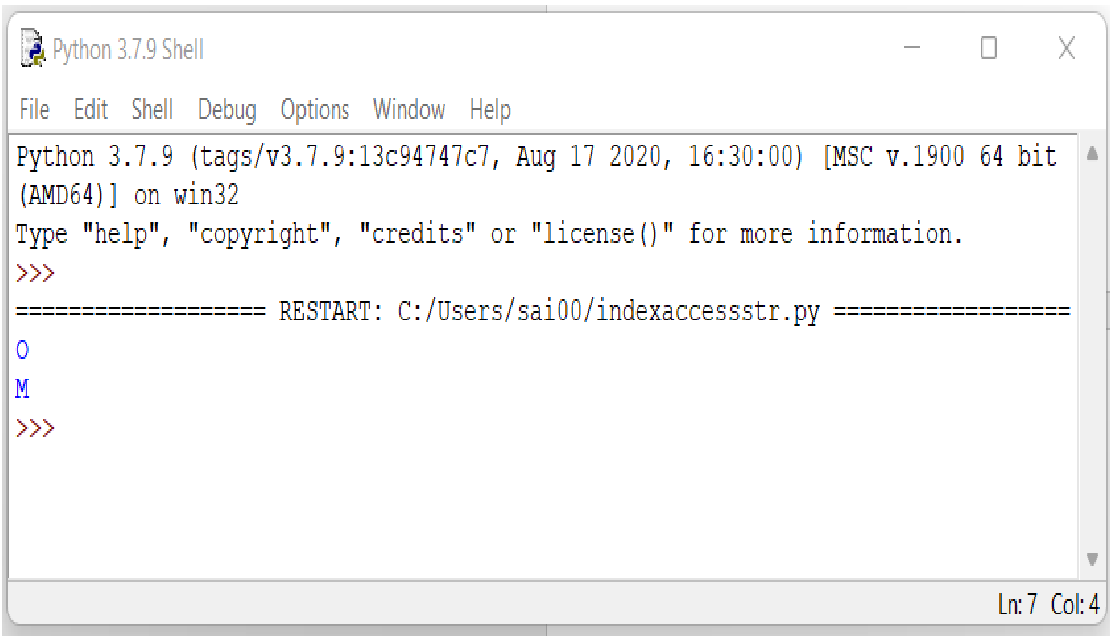
Fig 6.1. Indexing technique to access String

Example:


indexaccessstr.py - C:/Users/sai00/indexaccessstr.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 2 Col: 0

```
#Python Program to demonstrate access of String with Index.
#Creation of string using single quotes
s1 = 'GOOD MORNING'
#Access of second charater in string at index 1
print(s1[1])
#Access of second charater in string at index 5
print(s1[5])
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/indexaccessstr.py =================
O
M
>>>
```

The above example illustrate the concept of indexing method, where one strings S1 is created with the content of “GOOD MORNING” and then accessed character at index 1 and 5. Finally displayed the extracted characters output is shown above.

6.3.2 Negative Indexing

Python's string language permits negative indexing, just as that of a list. Negative address references, such as -1 for the final character, -2 for the second last character, and so forth, can access characters from the back of the String thanks to indexing. The idea of negative indexing is shown in Figure 6.1.

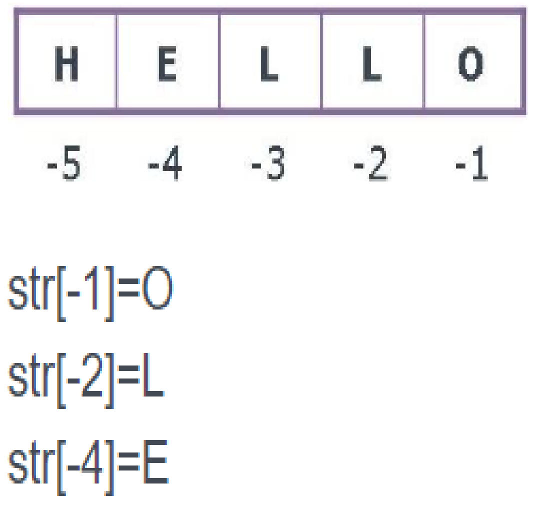
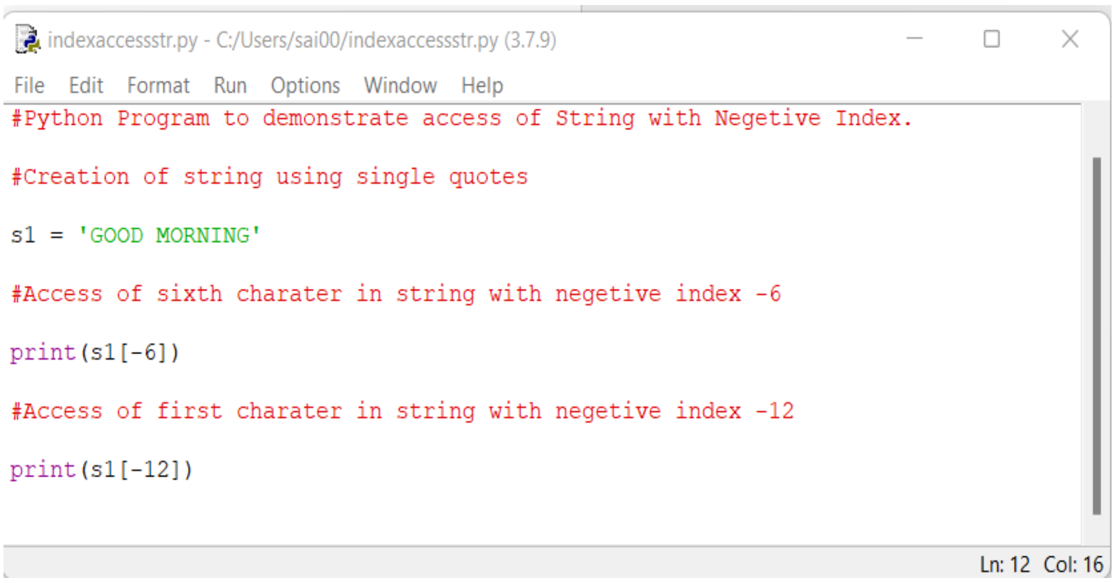


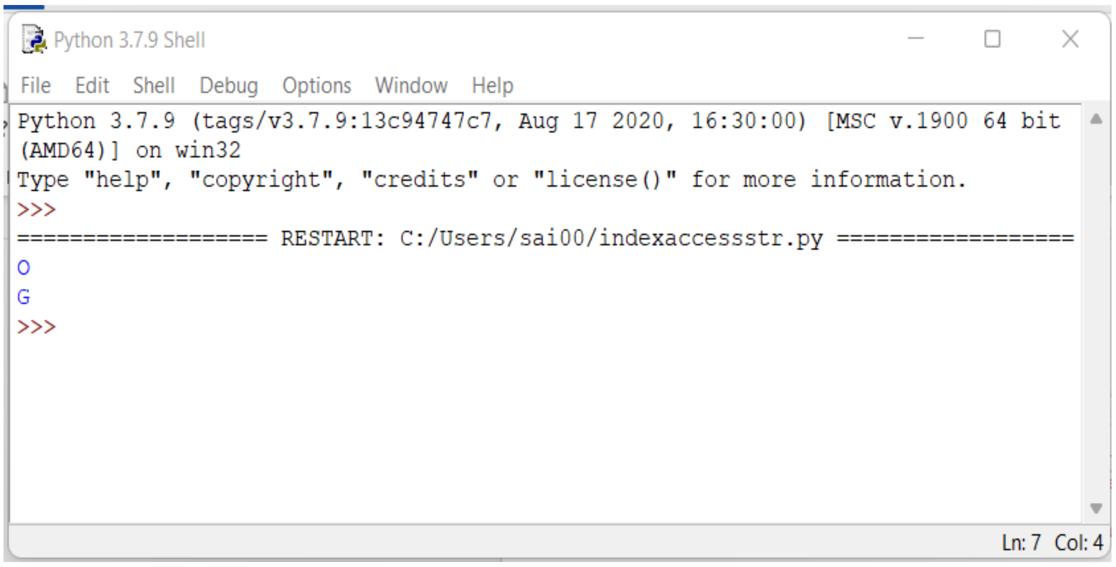
Fig 6.2. Negetive Indexing technique to access String

Example:


indexaccessstr.py - C:/Users/sai00/indexaccessstr.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 12 Col: 16

```
#Python Program to demonstrate access of String with Negetive Index.

#Creation of string using single quotes

s1 = 'GOOD MORNING'

#Access of sixth charater in string with negetive index -6

print(s1[-6])

#Access of first charater in string with negetive index -12

print(s1[-12])
```

Output:



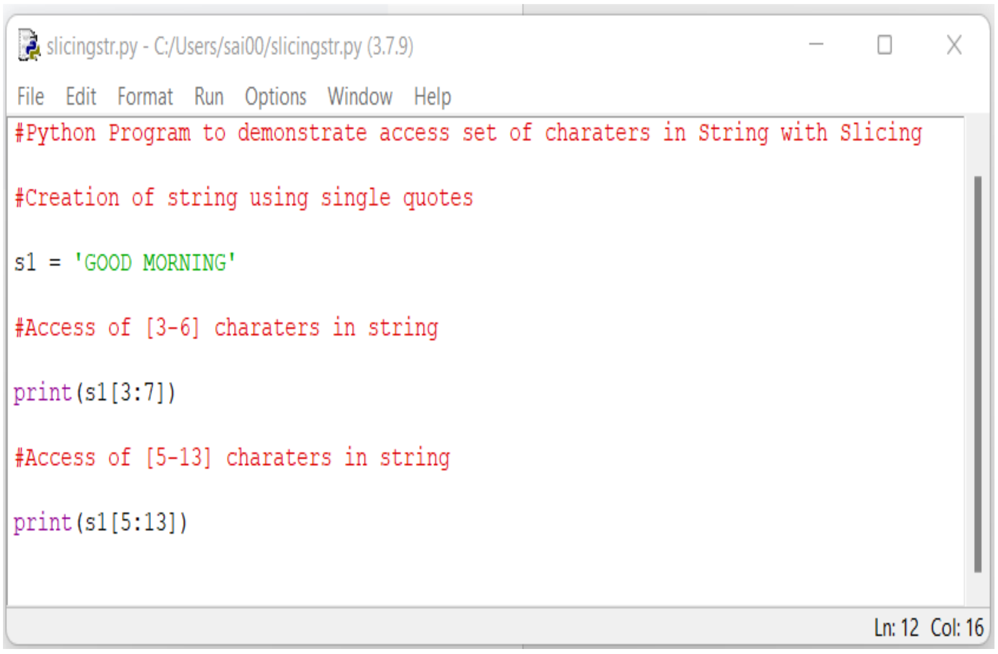
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/indexaccessstr.py =================
O
G
>>>
```

The above example illustrate the concept of negative indexing method, where one strings S1 is created with the content of “**GOOD MORNING**” and then accessed character at index-6 and -12. Finally displayed the extracted characters output is shown above.

6.2.3. Slicing

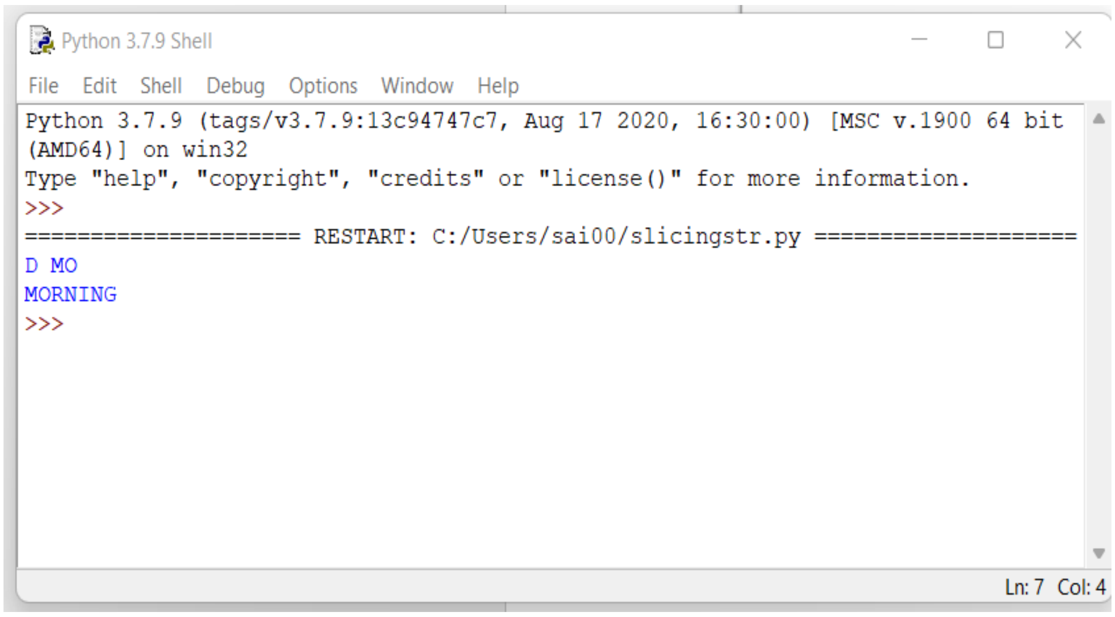
The String Slicing function in Python can be used to retrieve a range of characters from the String. To slice something in a string, use a slicing operator, such as a colon (:). When utilizing this method, bear in mind that the character at the start index is included in the string that is returned, but the character at the last index is not.

Example:



```
#Python Program to demonstrate access set of charaters in String with Slicing

#Creation of string using single quotes

s1 = 'GOOD MORNING'

#Access of [3-6] charaters in string

print(s1[3:7])

#Access of [5-13] charaters in string

print(s1[5:13])
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/slicingstr.py ===================
D MO
MORNING
>>>
```

The above example illustrate the concept of slicing method, where one strings S1 is created with the content of "**GOOD MORNING**" and then accessed character with range of [3-7] and [5-13] . Finally displayed the extracted sub string output is shown above.

6.4 PYTHON STRING OPERATIONS

Python's basic string operations include doing simple arithmetic operations, verifying the character of an existing substring, repeating a string, and much more are shown in Table 6.1.

Table 6.1. Python String Operations

Operation	Python Expression	Description
Concatenation	s1 + s2	"Concatenation operator" is the name given to this operator, which is used to unite two or more stings
Repetition	s * n	The repetition operator is the name given to this. There will be several copies of the same string created by it.
Membership	in	The membership operator is the name given to this. Whether or whether a certain character or sub string is included in the string that was supplied is returned by it.
	not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular string or character is not present in the specified. It gives a return value of true if the character or sub string is not included in the string that was supplied. Otherwise return false.
Comparison	s1 == s2	Returns True if string, s1 is the same as string, s2. Otherwise False.
	s1 != s2	Returns True if string, s1 is not the same as string, s2. Otherwise False.

6.4.1 Concatenation Operator

Concatenating or joining two or more strings is a common task while programming. To connect or concatenate two strings in this sense, use the plus operator (+) and the idea is shown in Figure 6.3. Python's concatenation operator only joins items of the same type, in contrast to other languages like JavaScript where type coercion allows us to concatenate a string and an integer.

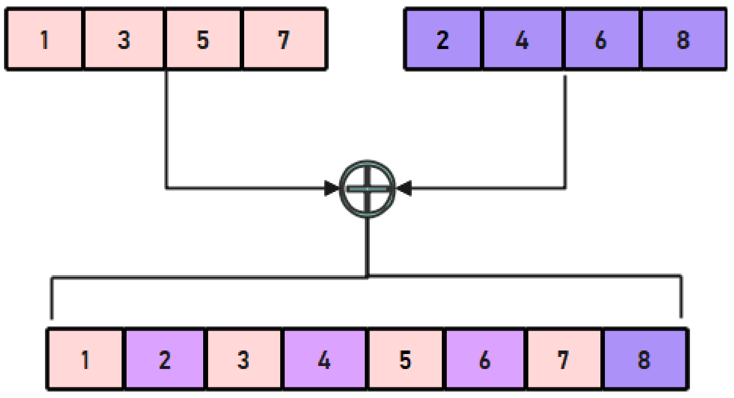
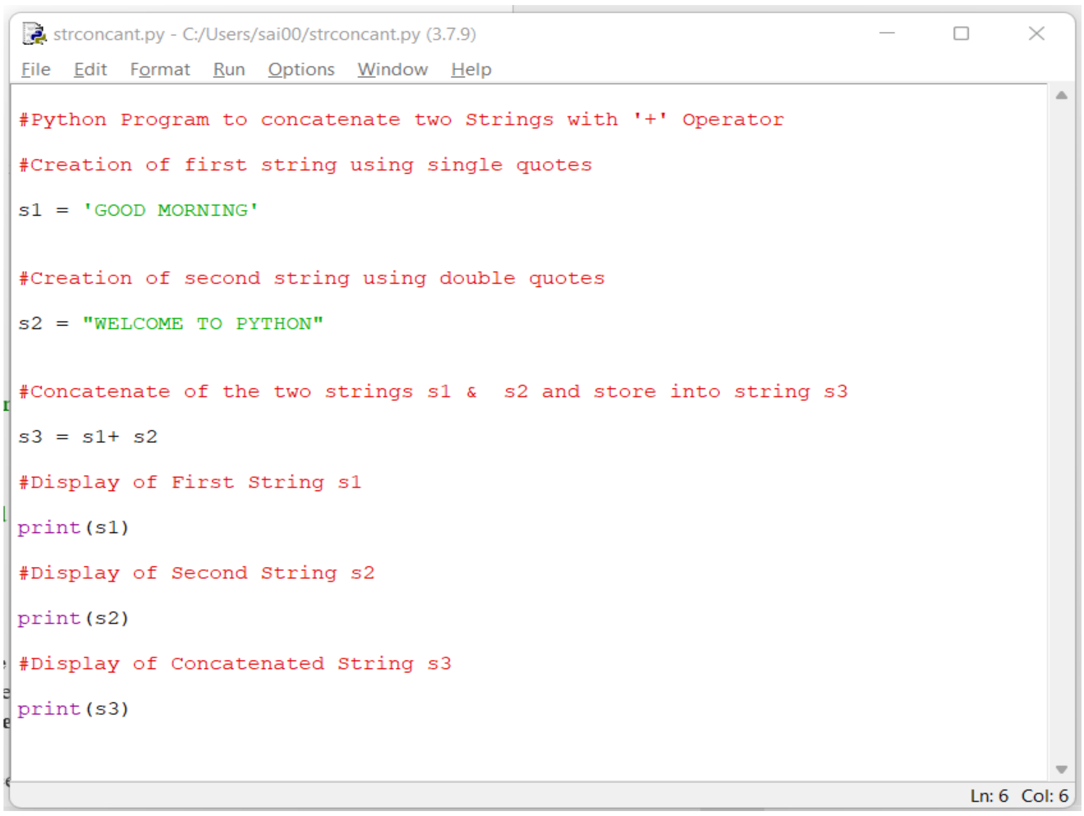


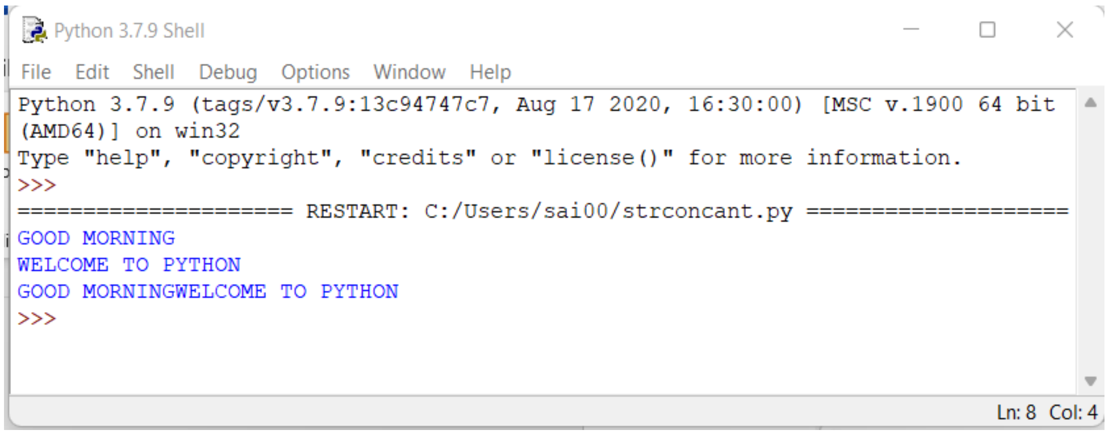
Fig 6.3. Concatenate of Two Lists with '+' Operator

Example:


strconcant.py - C:/Users/sai00/strconcant.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 6 Col: 6

```
#Python Program to concatenate two Strings with '+' Operator
#Creation of first string using single quotes
s1 = 'GOOD MORNING'

#Creation of second string using double quotes
s2 = "WELCOME TO PYTHON"

#Concatenate of the two strings s1 &  s2 and store into string s3
s3 = s1+ s2
#Display of First String s1
print(s1)
#Display of Second String s2
print(s2)
#Display of Concatenated String s3
print(s3)
```

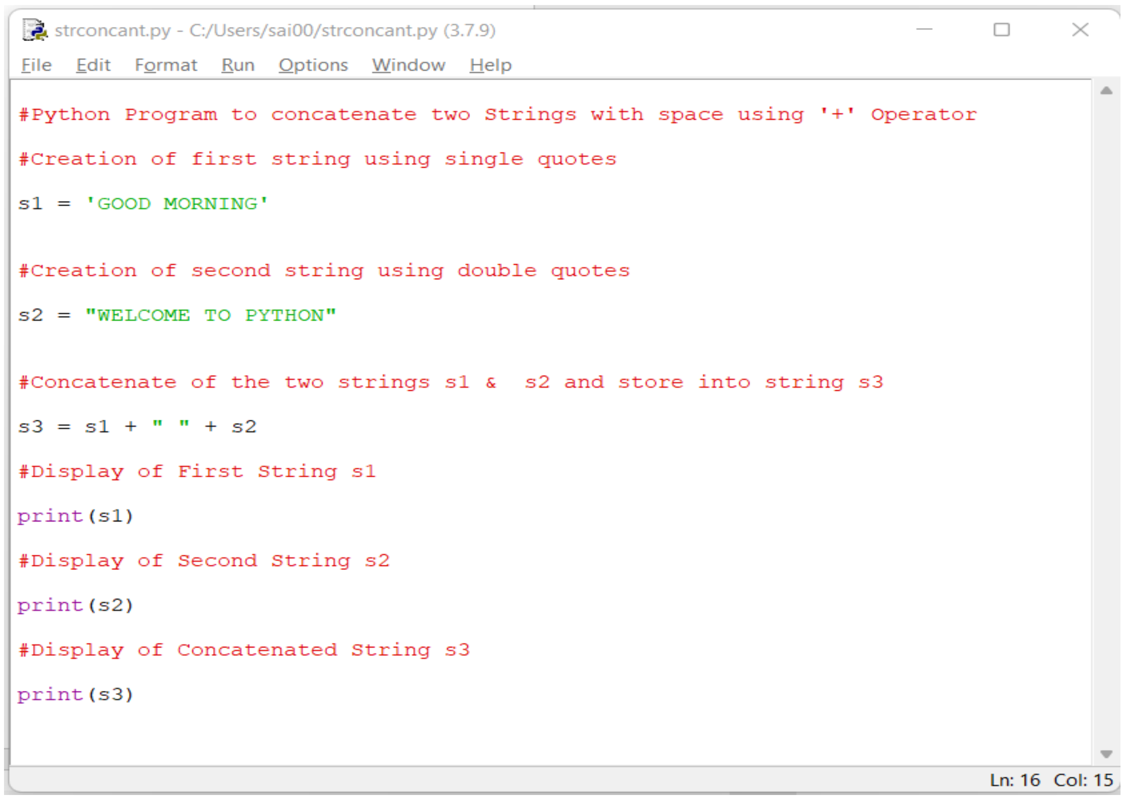
Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/strconcant.py ===================
GOOD MORNING
WELCOME TO PYTHON
GOOD MORNINGWELCOME TO PYTHON
>>>
```

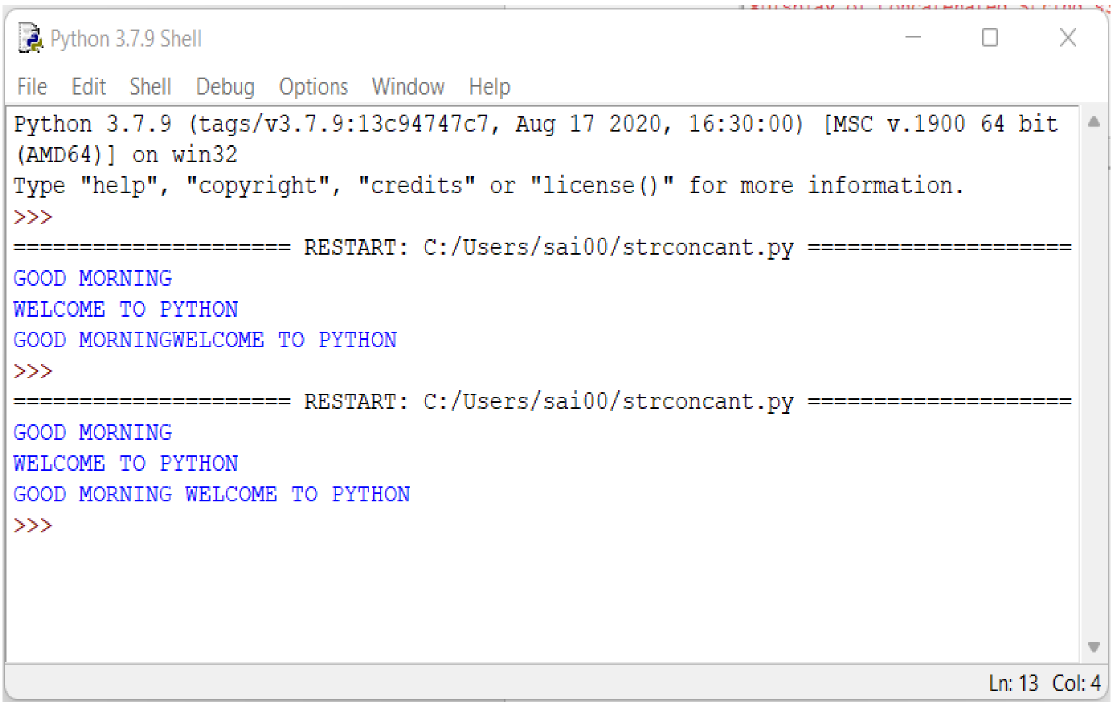
Strings are sequences that cannot be changed, as we previously stated. Concatenating the two strings in the previous example doesn't change either string. Rather, the process generates a new string called "S3" from the two strings "S1" and "S2." This operator is frequently used by beginners to add spaces between strings. This space is a string as well, but it's empty this time.

Example:


strconcant.py - C:/Users/sai00/strconcant.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 16 Col: 15

```
#Python Program to concatenate two Strings with space using '+' Operator
#Creation of first string using single quotes
s1 = 'GOOD MORNING'

#Creation of second string using double quotes
s2 = "WELCOME TO PYTHON"

#Concatenate of the two strings s1 &  s2 and store into string s3
s3 = s1 + " " + s2
#Display of First String s1
print(s1)
#Display of Second String s2
print(s2)
#Display of Concatenated String s3
print(s3)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/strconcant.py ====================
GOOD MORNING
WELCOME TO PYTHON
GOOD MORNINGWELCOME TO PYTHON
>>>
==================== RESTART: C:/Users/sai00/strconcant.py ====================
GOOD MORNING
WELCOME TO PYTHON
GOOD MORNING WELCOME TO PYTHON
>>>
```

6.4.2 Repetition Operator

The purpose of this operator is to return a string that has been repeated a predetermined number of times. This string is included in the new string, and it is repeated the number of times that was requested. This is accomplished by the utilization of the multiplication operator (*). Take for example that we have a string S and an integer N. Doing S times N or N times S will result in S being repeated N times. The idea is shown in Figure 6.4.

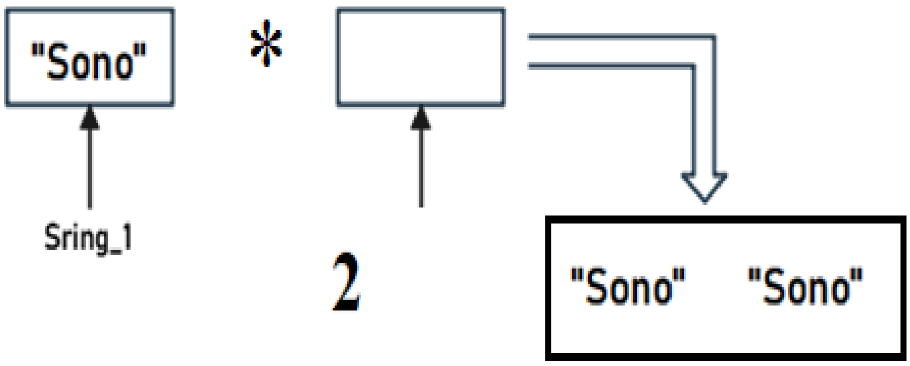
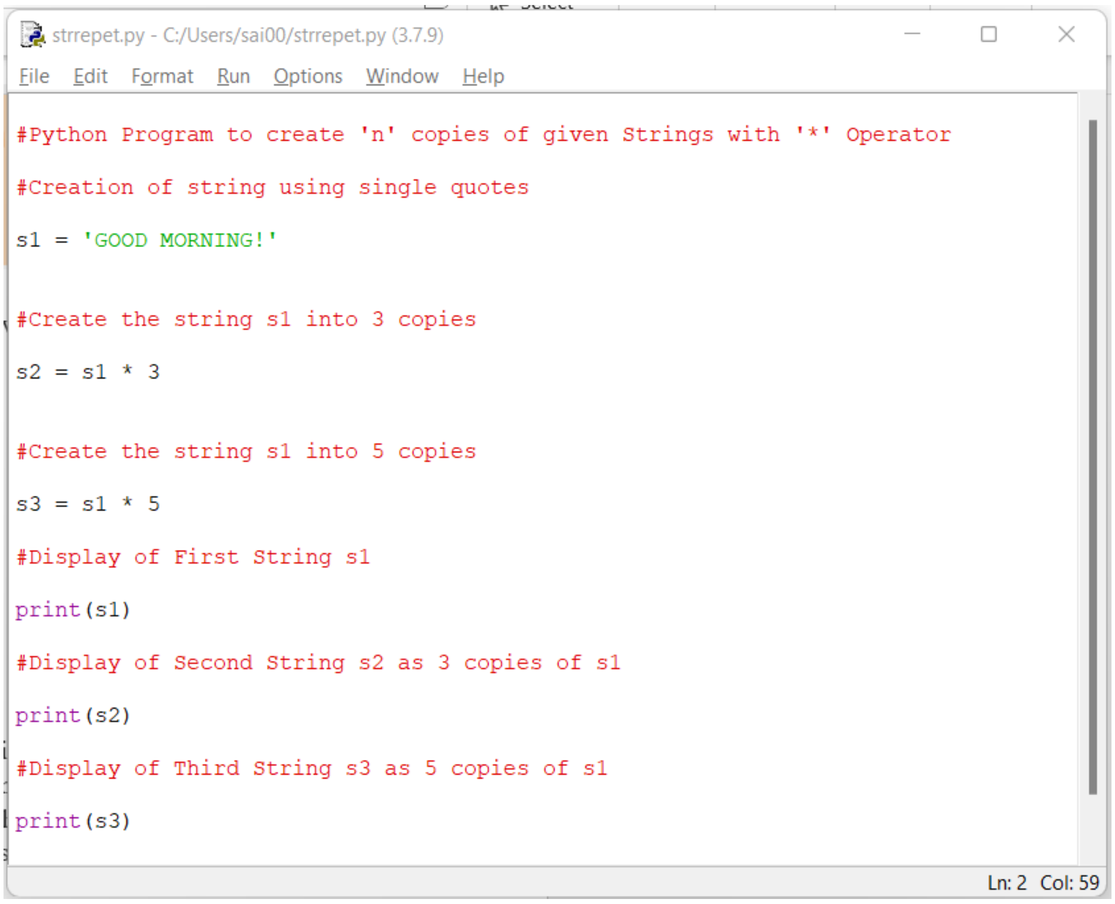


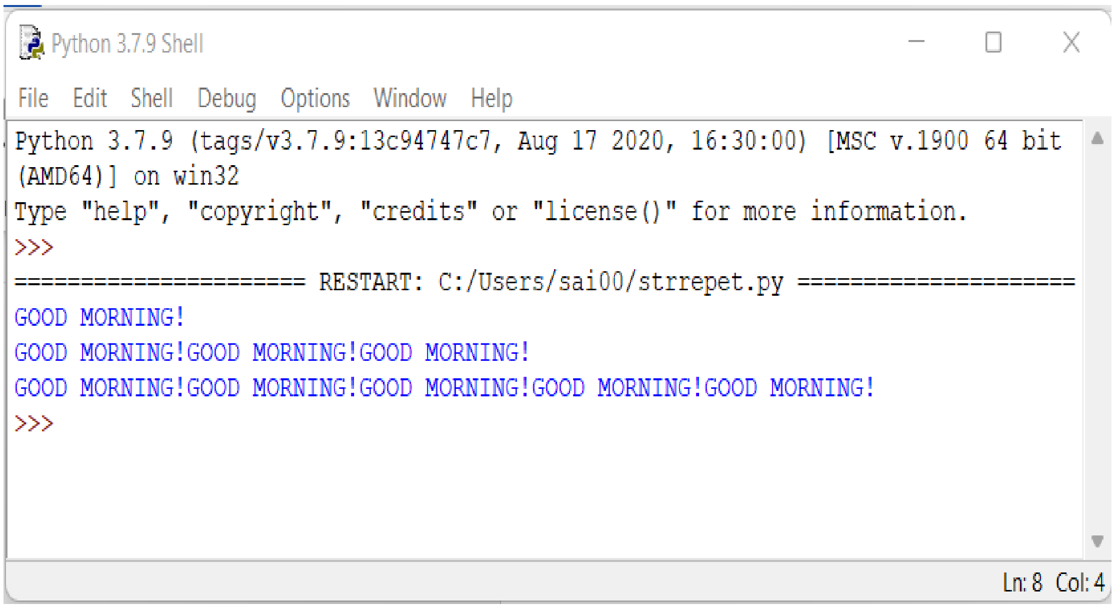
Fig 6.4. Repetition of Strings with '*' Operator

Example:


strrepet.py - C:/Users/sai00/strrepet.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 2 Col: 59

```
#Python Program to create 'n' copies of given Strings with '*' Operator
#Creation of string using single quotes
s1 = 'GOOD MORNING!'

#Create the string s1 into 3 copies
s2 = s1 * 3

#Create the string s1 into 5 copies
s3 = s1 * 5
#Display of First String s1
print(s1)
#Display of Second String s2 as 3 copies of s1
print(s2)
#Display of Third String s3 as 5 copies of s1
print(s3)
```

Output:



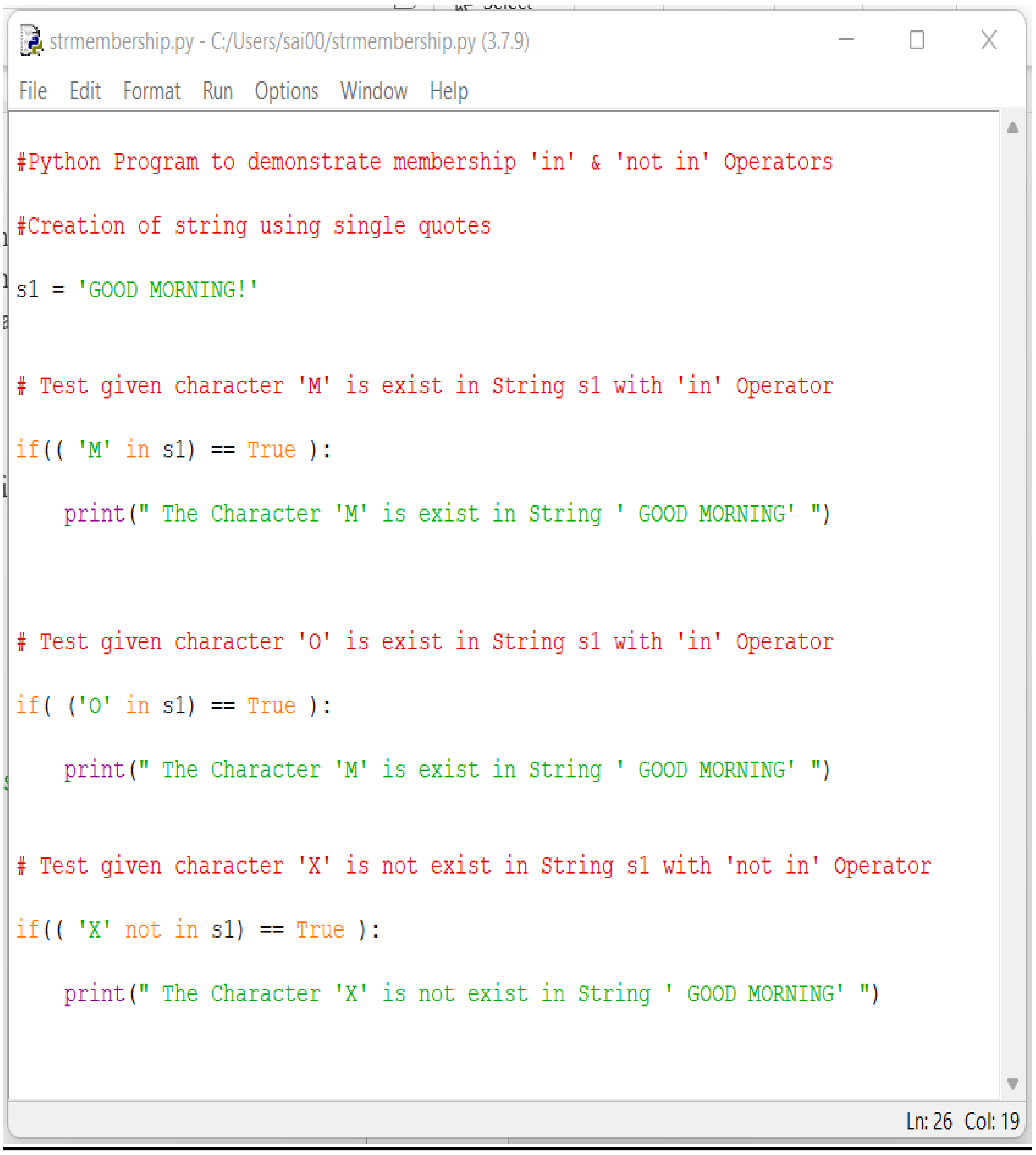
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/Users/sai00/strrepet.py ====================
GOOD MORNING!
GOOD MORNING!GOOD MORNING!GOOD MORNING!
GOOD MORNING!GOOD MORNING!GOOD MORNING!GOOD MORNING!GOOD MORNING!
>>>
```

Notice the last two print functions in the preceding example. Both actually output empty strings. The last but one step seems sense because it creates zero copies of the string, but the last operation appears strange. However, multiplying a string by a negative number yields an empty string.

6.4.3 Membership Operator

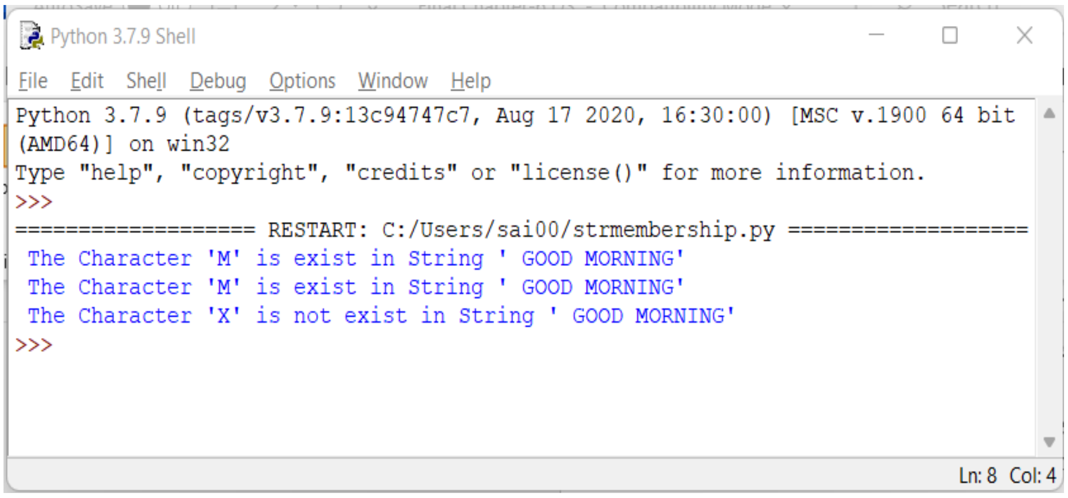
These operators are commonly used to determine whether or not an element or character occurs in a specific string. The in function returns True if a character x exists in a given string, and False otherwise. The not in function returns True if a character x does not appear in a provided string, and False otherwise.

Example:


strmembership.py - C:/Users/sai00/strmembership.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 26 Col: 19

```
#Python Program to demonstrate membership 'in' & 'not in' Operators
#Creation of string using single quotes
s1 = 'GOOD MORNING!'

# Test given character 'M' is exist in String s1 with 'in' Operator
if(( 'M' in s1) == True ):
    print(" The Character 'M' is exist in String ' GOOD MORNING' ")


# Test given character 'O' is exist in String s1 with 'in' Operator
if( ('O' in s1) == True ):
    print(" The Character 'M' is exist in String ' GOOD MORNING' ")

# Test given character 'X' is not exist in String s1 with 'not in' Operator
if(( 'X' not in s1) == True ):
    print(" The Character 'X' is not exist in String ' GOOD MORNING' ")
```

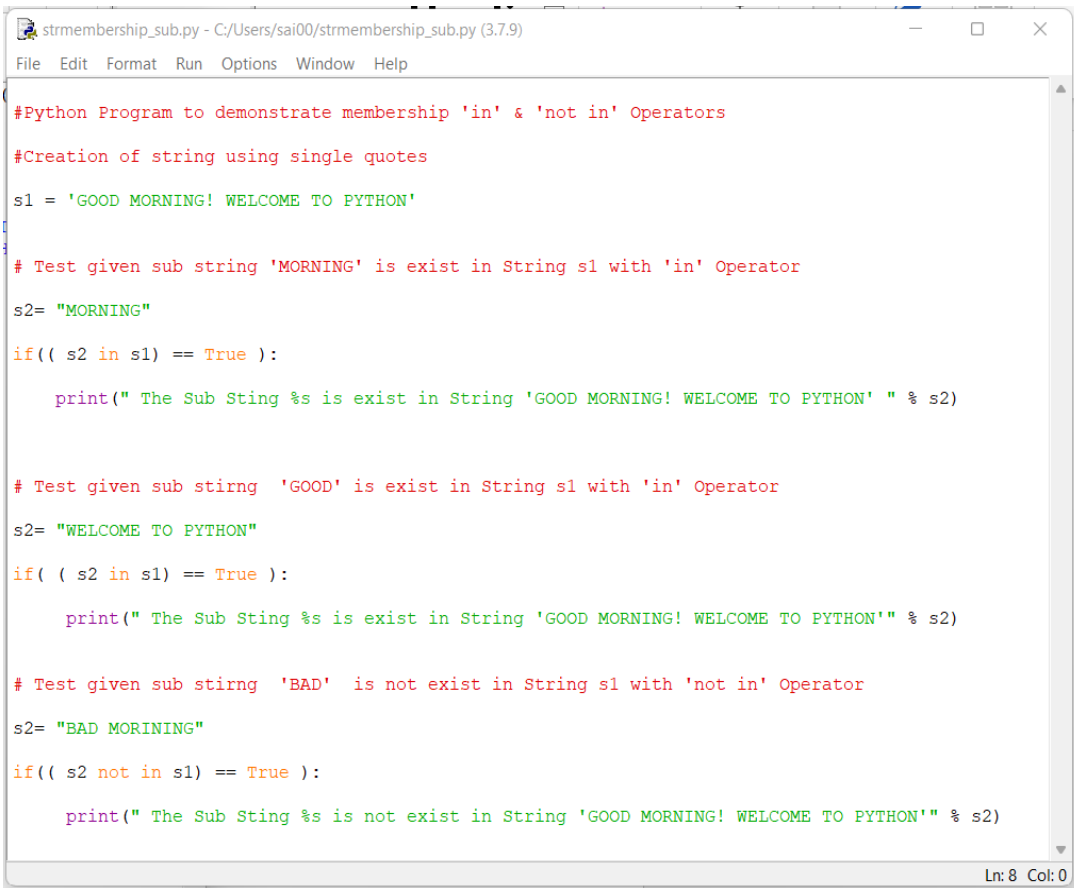
Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/strmembership.py ===================
 The Character 'M' is exist in String ' GOOD MORNING'
 The Character 'M' is exist in String ' GOOD MORNING'
 The Character 'X' is not exist in String ' GOOD MORNING'
>>>
```

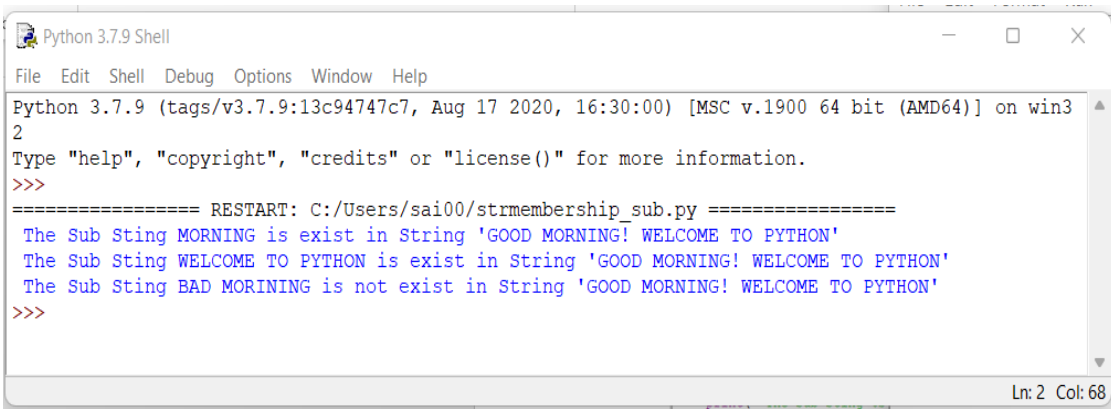
It is important to keep in mind that the membership operators are also capable of working with substrings; that is, they can determine whether or not a substring is present in a string.

Example:


strmembership_sub.py - C:/Users/sai00/strmembership_sub.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 8 Col: 0

```
#Python Program to demonstrate membership 'in' & 'not in' Operators
#Creation of string using single quotes
s1 = 'GOOD MORNING! WELCOME TO PYTHON'

# Test given sub string 'MORNING' is exist in String s1 with 'in' Operator
s2= "MORNING"
if(( s2 in s1) == True ):
    print(" The Sub Sting %s is exist in String 'GOOD MORNING! WELCOME TO PYTHON' " % s2)


# Test given sub stirng  'GOOD' is exist in String s1 with 'in' Operator
s2= "WELCOME TO PYTHON"
if( ( s2 in s1) == True ):
     print(" The Sub Sting %s is exist in String 'GOOD MORNING! WELCOME TO PYTHON'" % s2)

# Test given sub stirng  'BAD'  is not exist in String s1 with 'not in' Operator
s2= "BAD MORINING"
if(( s2 not in s1) == True ):
     print(" The Sub Sting %s is not exist in String 'GOOD MORNING! WELCOME TO PYTHON'" % s2)
```

Output:

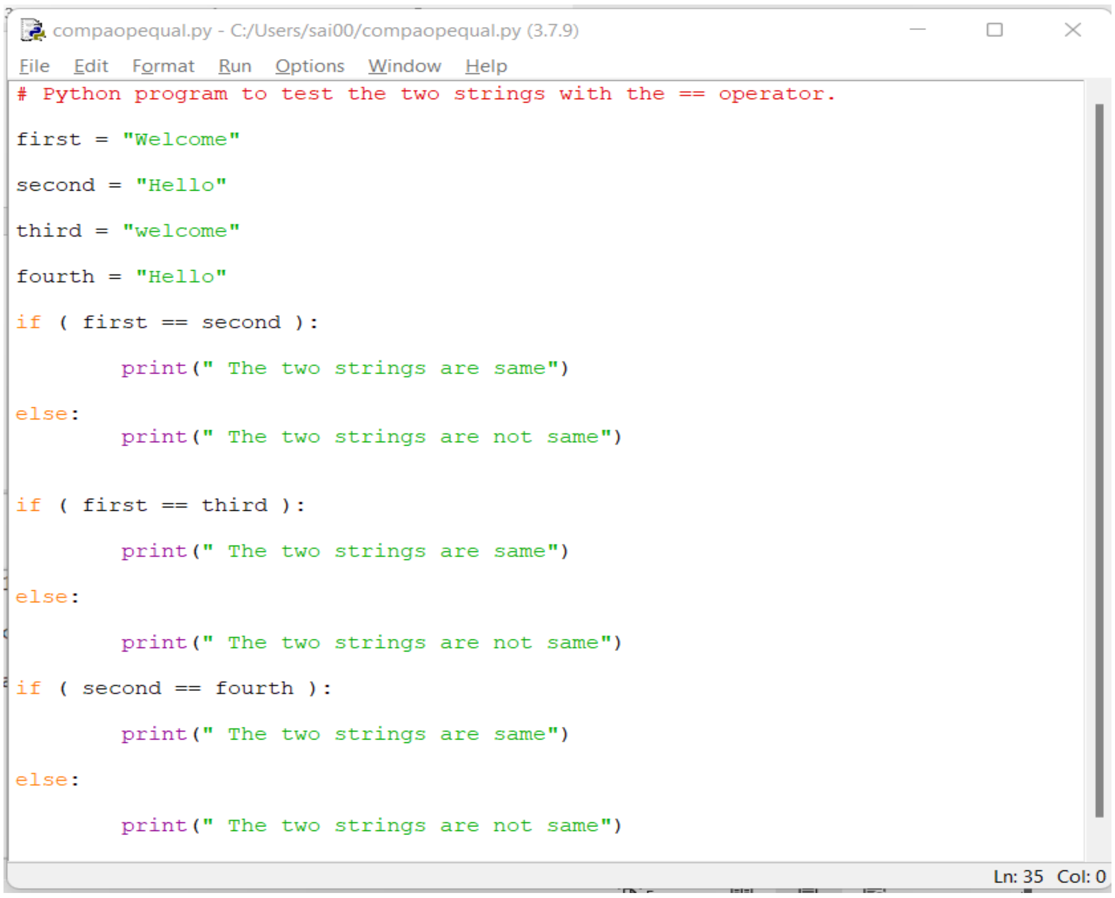


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit (AMD64)] on win3
2
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:/Users/sai00/strmembership_sub.py =================
 The Sub Sting MORNING is exist in String 'GOOD MORNING! WELCOME TO PYTHON'
 The Sub Sting WELCOME TO PYTHON is exist in String 'GOOD MORNING! WELCOME TO PYTHON'
 The Sub Sting BAD MORINING is not exist in String 'GOOD MORNING! WELCOME TO PYTHON'
>>>
```

6.4.4 Comparison Operator

The purpose of these operators in Python is to verify the equivalence of two operands, which in this case are two strings.

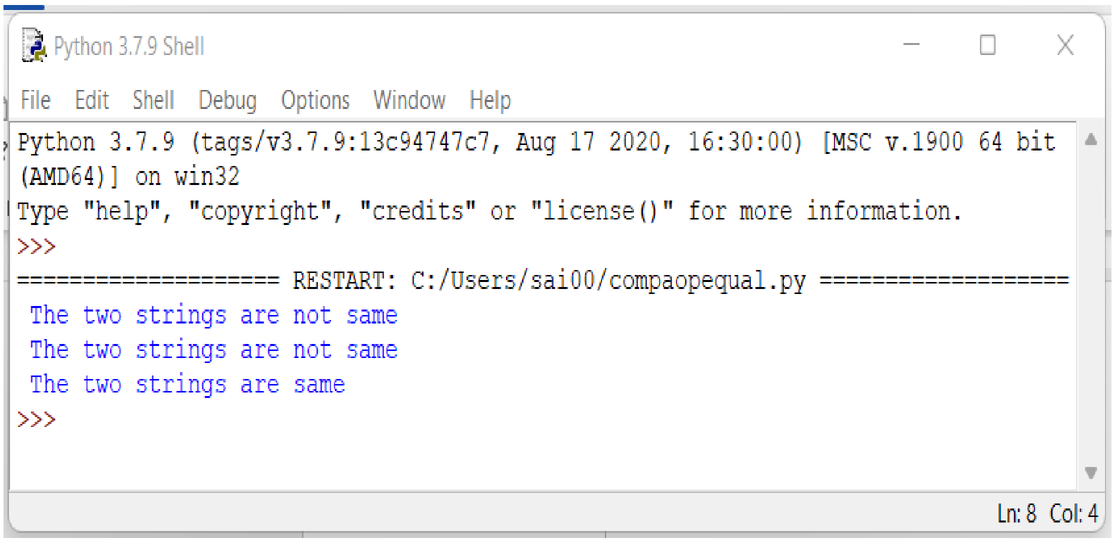
Example:



```
# Python program to test the two strings with the == operator.

first = "Welcome"

second = "Hello"

third = "welcome"

fourth = "Hello"

if ( first == second ):

        print(" The two strings are same")

else:
        print(" The two strings are not same")

if ( first == third ):

        print(" The two strings are same")

else:

        print(" The two strings are not same")

if ( second == fourth ):

        print(" The two strings are same")

else:

        print(" The two strings are not same")
```

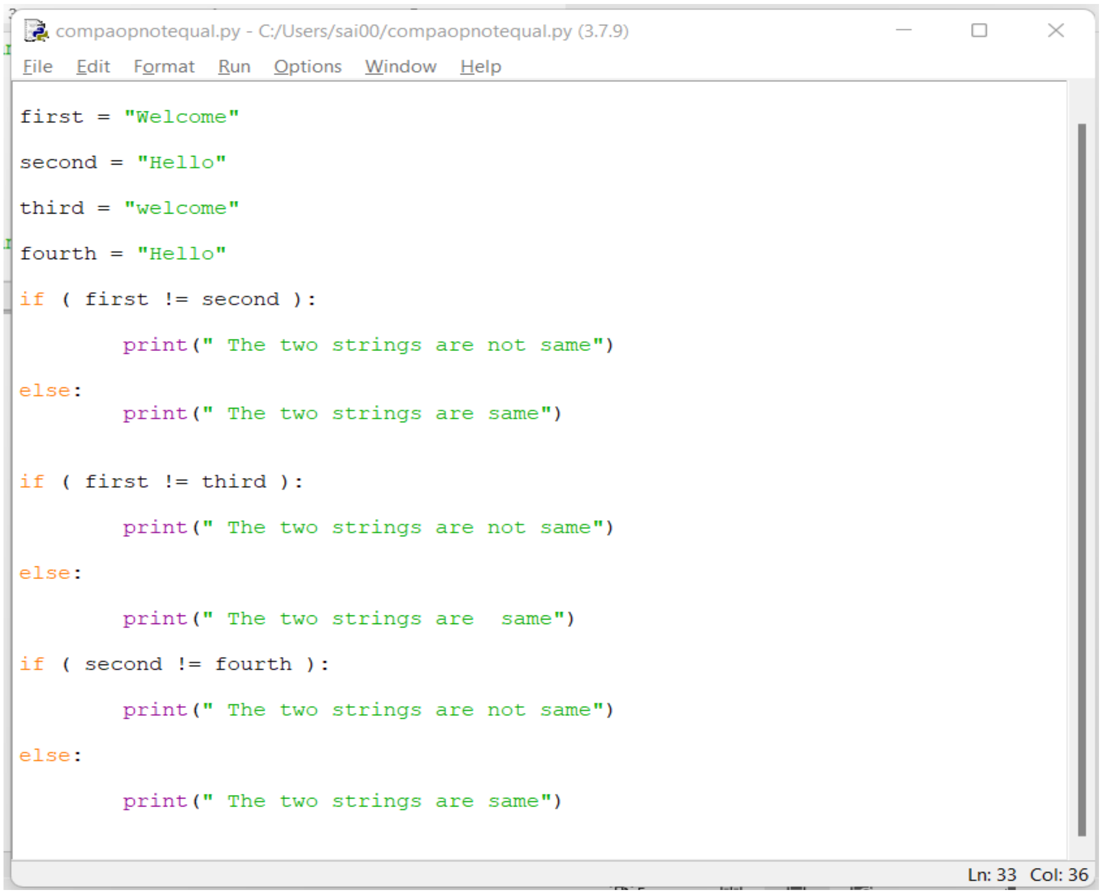
Their names also indicate that they are used for this purpose. However, because they return a boolean, they are most utilized in conditional expressions to determine whether or not two strings are identical. A True value is returned by the == operator when the two strings in question are identical, whereas a False value is returned when the strings in question are not identical.

Output



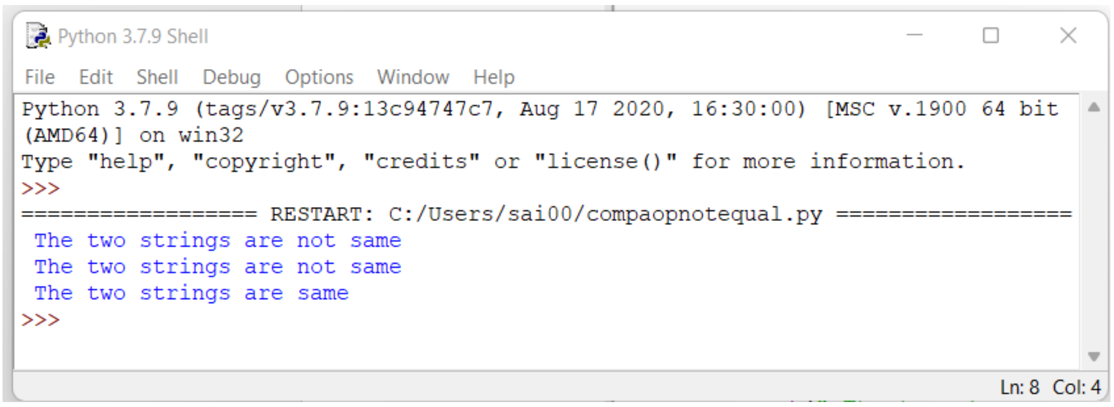
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/compaopequal.py ==================
 The two strings are not same
 The two strings are not same
 The two strings are same
>>>
```

Example:


compaopnotequal.py - C:/Users/sai00/compaopnotequal.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 33 Col: 36

```
first = "Welcome"
second = "Hello"
third = "welcome"
fourth = "Hello"
if ( first != second ):
        print(" The two strings are not same")
else:
        print(" The two strings are same")

if ( first != third ):
        print(" The two strings are not same")
else:
        print(" The two strings are  same")
if ( second != fourth ):
        print(" The two strings are not same")
else:
        print(" The two strings are same")
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/compaopnotequal.py ==================
 The two strings are not same
 The two strings are not same
 The two strings are same
>>>
```

As shown in the preceding examples, make a comparison between the first string, the second string, and the third string. Along the same lines, second with fourth. != is the operator that returns. True when the two strings are equal otherwise return false.

6.5. PYTHON STRING METHODS

Python comes with several built-in methods that can be used to execute operations and manipulations while working with strings. Listed below are several string methods that are frequently used:

6.5.1 len()

It is possible to utilize the `len()` function in order to determine the length of a string. A count of the characters contained in the string is returned by it.

Example:

```
String_new  =  “Welcome to Python!”

length = len(String_new)

print(String_new)
```

Output

18

The total number of characters includes space returned by the len() function i.e. 13

6.5.2 upper()

The string that is returned by the upper() method is one in which all of the characters are capitalized.

Example:

```
String_new = “Welcome to Python!”

String_new = String_new.upper()

print(String_new)
```

Output :

```
WELCOME TO PYTHON!
```

The upper() is called along with string object String_new.upper and it returns a string where all characters are in upper case.

6.5.3 replace()

Using the replace() method, a phrase that is supplied is replaced with another term that is also specified.

Example:

```
String_new = “Welcome to Python!”

String_new = String_new.replace(“Pyhton”, “PYTHON” )

print(String_new)
```

Output:

```
Welcome to PYTHON!
```

The reverse r() is called along with old and new string and it replaces an old string “Python” with new string “ PYHON”

6.5.4. find()

Using the find() method, one can locate the initial instance of the value that has been supplied. However, if the value cannot be located, this procedure will return -1. This method is essentially identical to the index() method; the only difference is that the index() method throws an exception if the value is not found. In addition, this method is almost identical to the index() method.

Example:

```
String_new = “Welcome to Python!”

String_new = String_new.find(“Python”)

print(String_new)
```

Output:

```
11
```

6.6 SUMMARY

Strings are an essential data type in Python, and they are utilized widely for activities that involve working with textual data. In this chapter, we covered the fundamentals of creating and manipulating strings, as well as accessing characters, string slicing, concatenation, string length, and the different string methods that are available in Python. Your ability to work effectively with strings in your Python programs and to handle text-based data in an efficient manner will be directly correlated to your level of comprehension of these ideas.

6.7 TECHNICAL TERMS

String, Indexing, Negative Indexing, Concatenation, Membership, comparison and Slicing

6.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. How is a String created and called? Explain.
2. What are the various List Operations? Explain.
3. Explain about List Methods with example.

Short Notes:

1. Write about indexing method for sting access.
2. Discuss about applications of python string.
3. Explain about Slicing method with example.

6.9 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. "Python Crash Course" by Eric Matthes
4. "Automate the Boring Stuff with Python" by Al Sweigart
5. "Learning Python" by Mark Lutz

Dr. KAMPA LAVANYA

LESSON- 7

PYTHON LIST

AIMS AND OBJECTIVES

The primary goal of this chapter is to grasp the concept of lists in Python programming. The talk began with an understanding of what a list is, its attributes, applications, and so on. After completing this chapter, the student will understand what a list is and how it differs from other data types. Also knows how to access List using various methods, operations, functions, and methods.

STRUCTURE

7.1. INTRODUCTION

Python is a popular high-level, general-purpose programming language that excels at creating graphical user interfaces and web applications. It is also a popular choice for application development due to its dynamic type and binding features. In this chapter we'll learn about List, an important data structure in Python programming.

Python Lists are a data structure that is quite like array of elements. The primary advantage of List is mutable, which means they can be modified once generated. This allows modify the data at any time. A List can contain any number of objects of various types, including strings, integers, floats, lists, and so on. Let's look at how to generate and use a List in python.

7.2. PYTHON LIST

Lists written in Python are identical to dynamically scaled arrays defined in other languages, such as Array List in Java and Vector in C++. Python lists are one of the most used and versatile built-in types. They allow us to store multiple items in a single variable. In Python, lists are usually stored in a type of object called a list. A list is a sequence of objects.

A list is a collection of objects separated by commas and denoted by the symbol []. The objects can be of any type: numbers, strings, even other lists. Lists are mutable, meaning you can change their content by adding, removing, or modifying objects. This chapter was able to give you a clear understanding about Python Lists.

7.2.1. Creating a List

When using Python, the process of creating a list is simple. To define a list, you must first enclose a series of components within square brackets and then separate them with commas.

Syntax:

```
>>> List_name = [ ‘item1’, ‘item2’,’item3’, ........,’itemN’]
```

The above syntax ‘N’ items are assigned to the List. The items either to be homogeneous or heterogeneous. After creating of list, it can be modified later as per the requirement.

Example:

```
>>> Subject_List = [‘physics’ , ‘chemistry’ , ‘math’]

>>> Person_List = [‘Sai’, 18 , ‘CSE’ , ‘ANU’,78.89]
```

The above two list one homogeneous and other one is heterogeneous. The first list Student_List consist of 3 items of type string. Next, string is Person_List is a heterogeneous consist of 5 items of mixed combination of string, int and float.

7.2.2 Features of Python List

The list data type in Python possesses several key qualities, including the following:

- It is important to note that lists are sorted, which means that the order in which the items in the list are presented is maintained.
- Lists are changeable, which means that additions, deletions, and modifications can be made to the elements that make up the list.
- Lists can store elements of a variety of data kinds, making them heterogeneous. It is possible, for instance, to have a list that includes strings, floats, and integers.
- As a result of the fact that lists can be nested, it is possible to have one list contained within another list.
- Lists can expand or contract dynamically through the addition or removal of elements. Because of this, lists are extremely adaptable and diverse.
- Using indexing, elements included within a list can be retrieved. To obtain or edit the value of an element, you can make use of the index of that element.
- Lists are iterable, which means that you can use a loop to carry out the process of iterating over all of the items contained within the list.
- Lists come with several methods that are built in, which makes it simple to modify and interact with them.

7.2.3 Application of Python List

Here's how the list data type is used in Python.

- Lists are a common way for Python programs to store and change data. This means that a computer that reads data from a file can store that data in a list so that it can be used later.
- Lists can be used to make data structures like stacks and queues work, which are widely used in computer science.
- Lists can be used to store and show data in GUI programs. A list widget can be used in a GUI program to show, for example, a list of the things in a shopping cart.
- Web scraping tools like BeautifulSoup can be used to get data from web pages and store and change it in lists.
- Lists are a common way for machine learning apps to store and change data. One way to train a machine-learning model is to give it a list of features and a list of names.

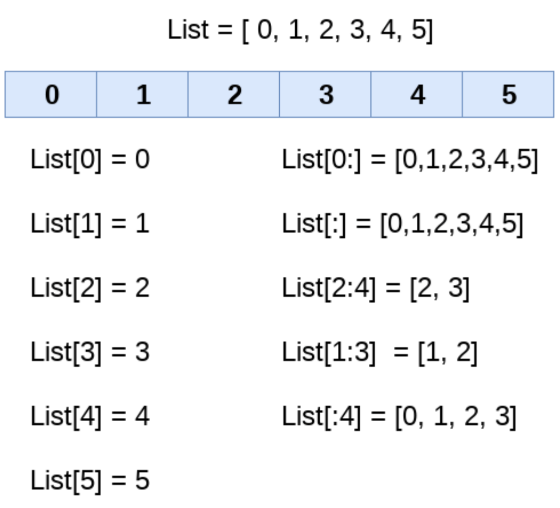
7.3 ACCESSING THE LIST

There are three different ways to get to items or objects in the list that was made in the last part. The following give the details:

- Indexing
- Negative Indexing
- Slicing

7.3.1 Indexing

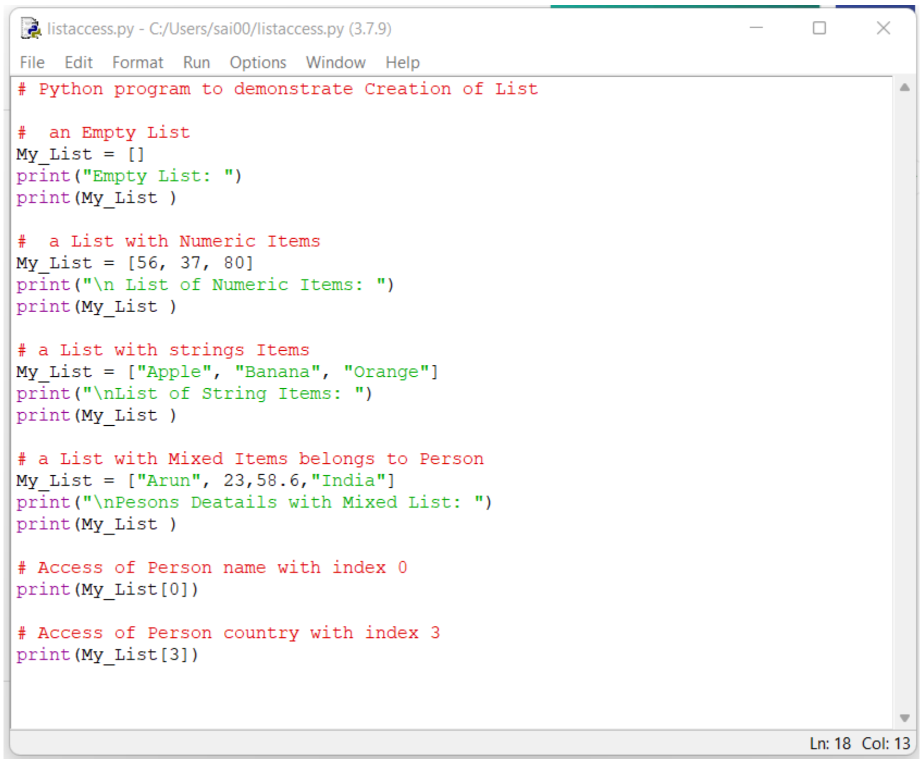
In Python, the Indexing method can be used to obtain individual items from a list. The index ranges from 0 to length-1. The first item is indexed at zero, the second at one, and so on. The idea of this method is shown in Figure 7.1



```
List = [ 0, 1, 2, 3, 4, 5]

List[0] = 0          List[0:] = [0,1,2,3,4,5]
List[1] = 1          List[:] = [0,1,2,3,4,5]
List[2] = 2          List[2:4] = [2, 3]
List[3] = 3          List[1:3]  = [1, 2]
List[4] = 4          List[:4] = [0, 1, 2, 3]
List[5] = 5
```

Fig 7.1. Accessing List of items with Indexing Method

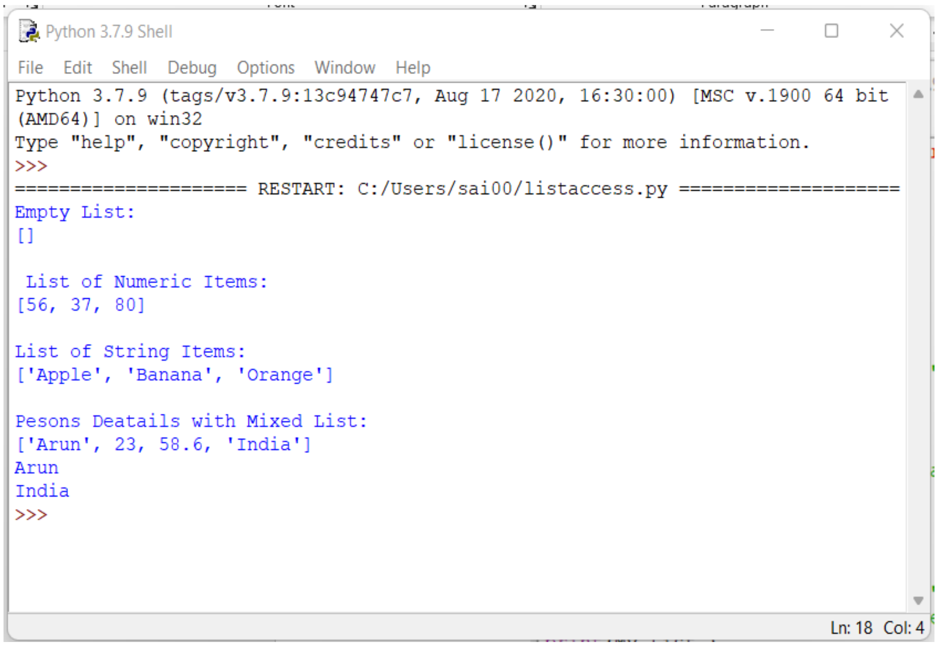
Example:


listaccess.py - C:/Users/sai00/listaccess.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 18 Col: 13

```
# Python program to demonstrate Creation of List

#  an Empty List
My_List = []
print("Empty List: ")
print(My_List )

#  a List with Numeric Items
My_List = [56, 37, 80]
print("\n List of Numeric Items: ")
print(My_List )

# a List with strings Items
My_List = ["Apple", "Banana", "Orange"]
print("\nList of String Items: ")
print(My_List )

# a List with Mixed Items belongs to Person
My_List = ["Arun", 23,58.6,"India"]
print("\nPesons Deatails with Mixed List: ")
print(My_List )

# Access of Person name with index 0
print(My_List[0])

# Access of Person country with index 3
print(My_List[3])
```

The above Python program build the first empty list, known as My_List, with zero items. Later, I redefined My_List with three members of the same type, namely integers. Similar to string elements and heterogeneous items, the same list is redefined. In addition, the list is retrieved using an indexing approach, as shown plainly in the code above.Output displays the results, which include each list as well as the extracted elements at indexes 0 and 3.

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/listaccess.py ====================
Empty List:
[]

 List of Numeric Items:
[56, 37, 80]

List of String Items:
['Apple', 'Banana', 'Orange']

Pesons Deatails with Mixed List:
['Arun', 23, 58.6, 'India']
Arun
India
>>>
```

7.3.2. Negative Indexing

In contrast to other programming languages, Python also allows you to access things with negative indexes. Negative indices are numbered from right to left. The index -1 denotes the final element on the List's right side, followed by the index -2 for the following member on the left, and so on until the last element on the left is reached. The idea of this method is shown in Figure 7.2.

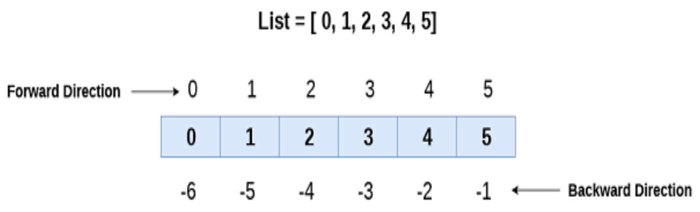
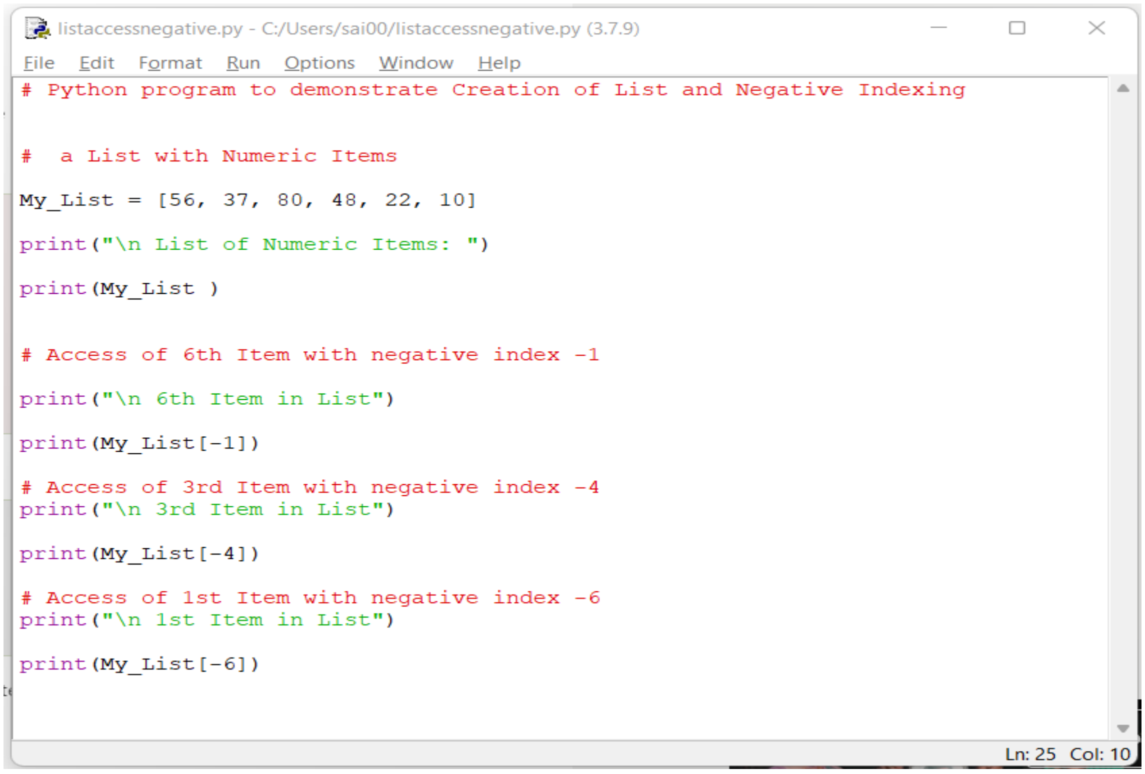


Fig 7.2. Accessing List of items with Indexing Method

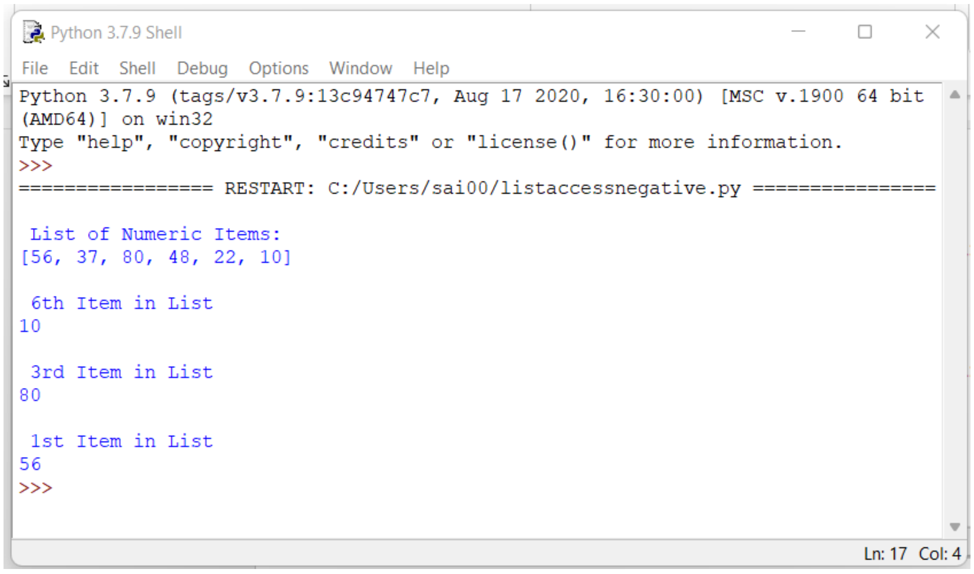
In the below example, generated pyton program created list My_List with six items. Later displayed all items . After that each element is indiviualy extracted with negetive indexing method and illustrated in the example. The output, displayed specific elements at index -1 , -4 and -6 respetively.

Example:


listaccessnegative.py - C:/Users/sai00/listaccessnegative.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 25 Col: 10

```
# Python program to demonstrate Creation of List and Negative Indexing

#  a List with Numeric Items

My_List = [56, 37, 80, 48, 22, 10]

print("\n List of Numeric Items: ")

print(My_List )

# Access of 6th Item with negative index -1

print("\n 6th Item in List")

print(My_List[-1])

# Access of 3rd Item with negative index -4
print("\n 3rd Item in List")

print(My_List[-4])

# Access of 1st Item with negative index -6
print("\n 1st Item in List")

print(My_List[-6])
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:/Users/sai00/listaccessnegative.py ================

 List of Numeric Items:
[56, 37, 80, 48, 22, 10]

 6th Item in List
10

 3rd Item in List
80

 1st Item in List
56
>>>
```

7.3.3 Slicing

In the last chapter, we saw how the Slicing function accesses a range of characters in the String. The Slicing Operator can also be used to extract a set of items or a sublist within an item denoted by a colon. List slicing generates a new list from an existing one.

Syntax:

list[start: stop: step]

where,

- “start" is an index position that indicates the point in a list from which the slicing will begin.
- stop is the index position before which the slicing process will come to an end in a list.
- In other words, the start index is modified after every n steps, and list slicing is carried out on that index.
- step is the number of steps that have been taken.

The figure 7.3, shows the concept of slicing method in python , where the complete list is devided into three slices and is shown in below:

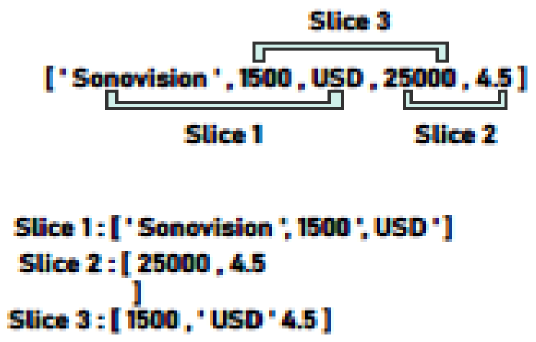
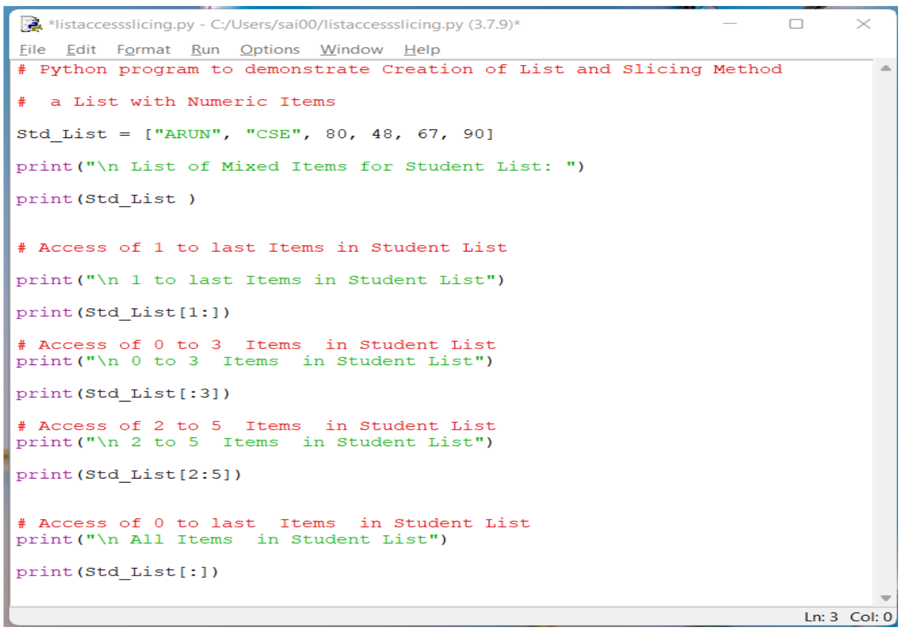

Slice 3
[' Sonovision ' , 1500 , USD , 25000 , 4.5]
Slice 1
Slice 2
Slice 1 : [' Sonovision ', 1500 ', USD ']
Slice 2 : [25000 , 4.5]
Slice 3 : [1500 , ' USD ' 4.5]

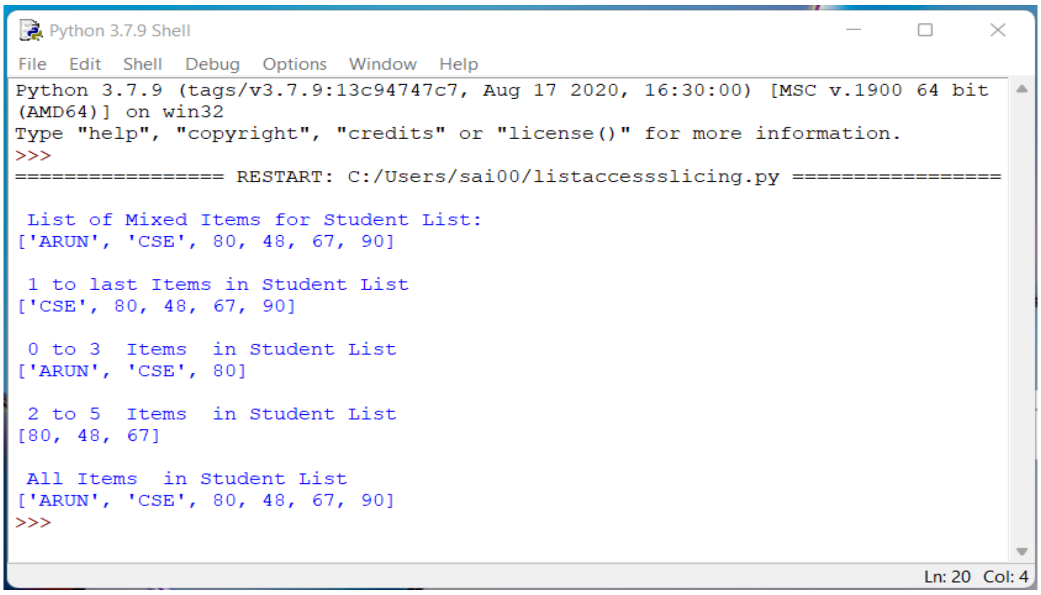
Fig 7.3. The concept of Slicing in Python

Example:


listaccessslicing.py - C:/Users/sai00/listaccessslicing.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 3 Col: 0

```
# Python program to demonstrate Creation of List and Slicing Method

#  a List with Numeric Items

Std_List = ["ARUN", "CSE", 80, 48, 67, 90]

print("\n List of Mixed Items for Student List: ")

print(Std_List )


# Access of 1 to last Items in Student List

print("\n 1 to last Items in Student List")

print(Std_List[1:])

# Access of 0 to 3  Items  in Student List
print("\n 0 to 3  Items  in Student List")

print(Std_List[:3])

# Access of 2 to 5  Items  in Student List
print("\n 2 to 5  Items  in Student List")

print(Std_List[2:5])


# Access of 0 to last  Items  in Student List
print("\n All Items  in Student List")

print(Std_List[:])
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:/Users/sai00/listaccessslicing.py =================

 List of Mixed Items for Student List:
['ARUN', 'CSE', 80, 48, 67, 90]

 1 to last Items in Student List
['CSE', 80, 48, 67, 90]

 0 to 3  Items  in Student List
['ARUN', 'CSE', 80]

 2 to 5  Items  in Student List
[80, 48, 67]

 All Items  in Student List
['ARUN', 'CSE', 80, 48, 67, 90]
>>>
```

Within the preceding illustration, a pyton program was used to construct a list called Std_List. This list contains six elements, all of which are of a heterogeneous nature, including both string and integer data types.
 Later on, things that were shown using the slicing method. There is a slice that fits all of the items presented [1:], and the following range of items in the list is displayed [0:3]. A slice of [2:5] is used to display the elements of the list in a complementary manner.

7.4 PYTHON LIST OPERATIONS

Python's simple list operations are an excellent method to begin learning more complex coding principles. It enables data manipulation and the creation of fundamental structures, both of which are necessary for resolving programming difficulties.
Basic list operations in Python include repeating a list, checking that an element is already in the list, and doing simple arithmetic on the numbers in the list.The complete list of operations in python shown in Table 7.1.

Table 7.1. Python List Operations

Operation	Python Expression	Description
Concatenation	L1 + L2	"Concatenation operator" is the name given to this operator, which is used to unite two or more lists of the same type or distinct types.
Repetition	L1 * n	The repetition operator is the name given to this. There will be several copies of the same list created by it.
Membership	in	The membership operator is the name given to this. Whether or whether a certain item is included in the list that was supplied is returned by it.

	not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular item is not present in the specified It also functions as a membership operator and performs the exact opposite of the in operation. It gives a return value of true if the item in question is not included in the list that was supplied.
Comparison	L1 == L2	Returns True if List, L1 is the same as List, L2. Otherwise, False.
	s1 != s2	Returns True if List, L1 is not the same as List, L2. Otherwise, False.
Iteration	for x in [1, 2, 3]: print x,	Iteration operator is the name given to this operator, and it prints elements of List in an iterative manner.

7.4.1 Concatenation Operator

When we are programming, we frequently find ourselves in situations where we need to join or concatenate two or more lists. When it comes to this particular matter, the plus operator (+) is utilized to join two or more lists that are of the same type or of different types.

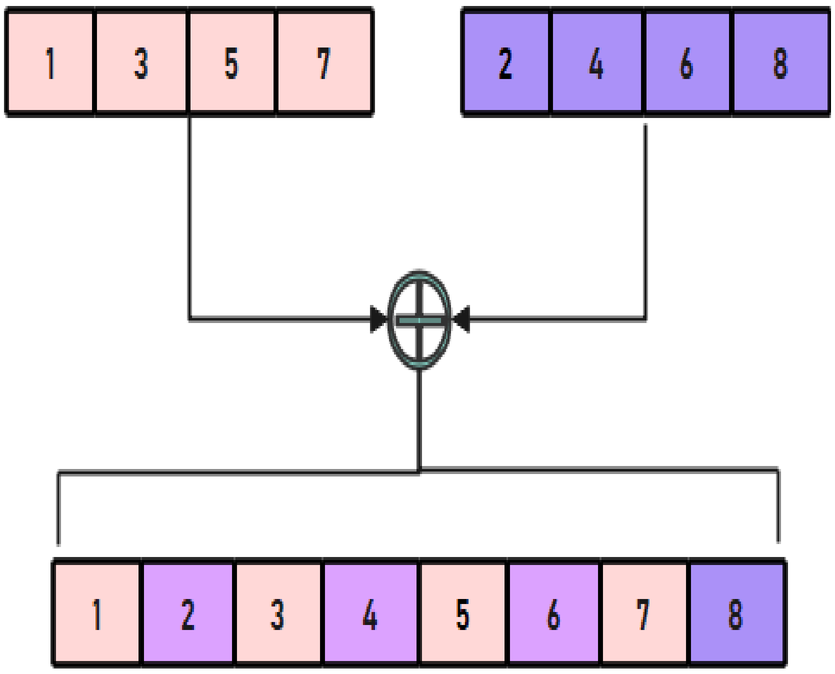
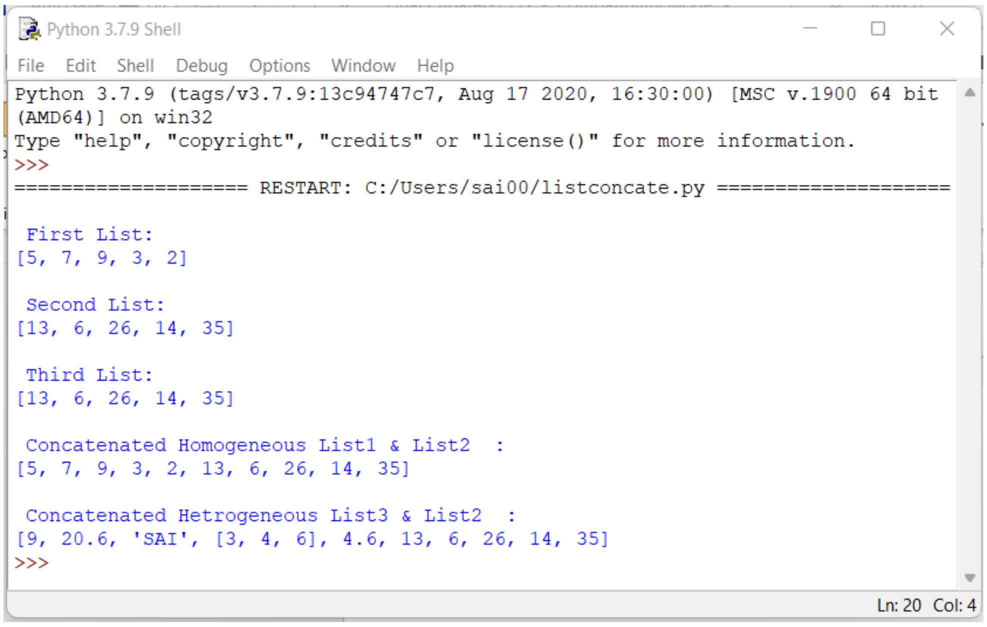


Fig 7.4. Concatenate of Two Lists with ‘+’ Operator

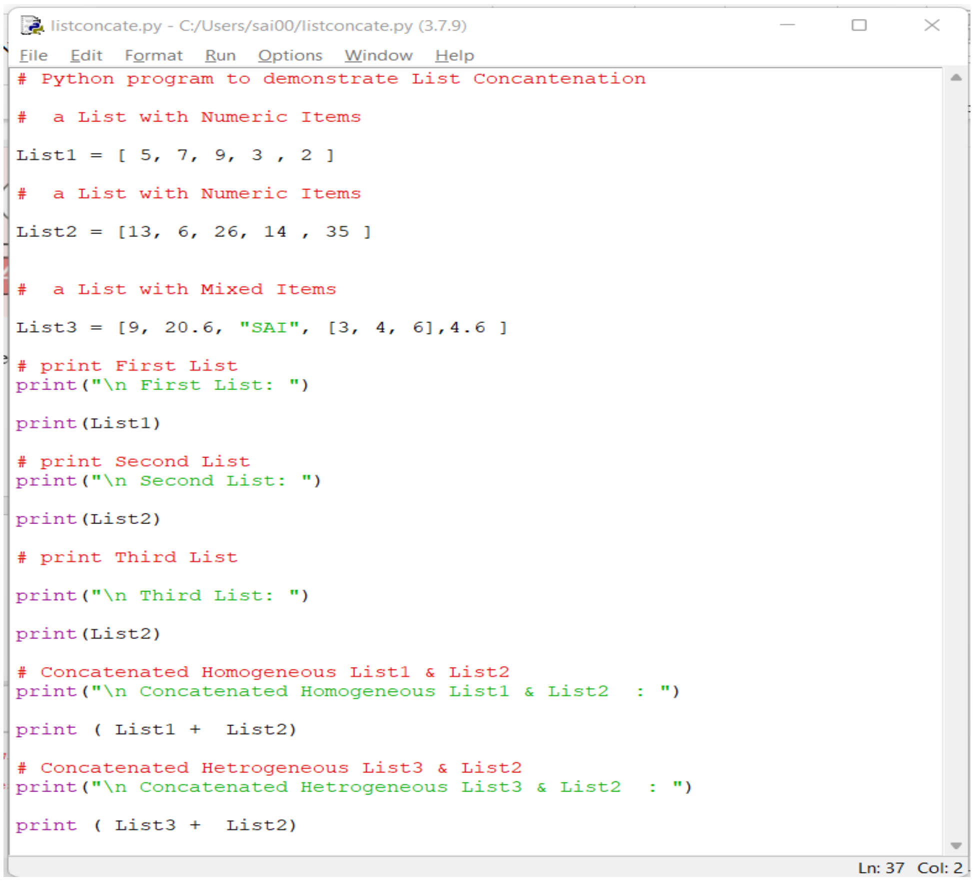
The two strings are not altered in any way by the concatenation of the two lists, as seen in the previous example. Instead, the operation generates a new List by combining the two Lists that was previously known as "first" and "second."

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/listconcate.py ====================

 First List:
[5, 7, 9, 3, 2]

 Second List:
[13, 6, 26, 14, 35]

 Third List:
[13, 6, 26, 14, 35]

 Concatenated Homogeneous List1 & List2  :
[5, 7, 9, 3, 2, 13, 6, 26, 14, 35]

 Concatenated Hetrogeneous List3 & List2  :
[9, 20.6, 'SAI', [3, 4, 6], 4.6, 13, 6, 26, 14, 35]
>>>
```

Example:


listconcate.py - C:/Users/sai00/listconcate.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 37 Col: 2

```
# Python program to demonstrate List Concantenation

#  a List with Numeric Items

List1 = [ 5, 7, 9, 3 , 2 ]

#  a List with Numeric Items

List2 = [13, 6, 26, 14 , 35 ]


#  a List with Mixed Items

List3 = [9, 20.6, "SAI", [3, 4, 6],4.6 ]

# print First List
print("\n First List: ")

print(List1)

# print Second List
print("\n Second List: ")

print(List2)

# print Third List

print("\n Third List: ")

print(List2)

# Concatenated Homogeneous List1 & List2
print("\n Concatenated Homogeneous List1 & List2  : ")

print ( List1 +  List2)

# Concatenated Hetrogeneous List3 & List2
print("\n Concatenated Hetrogeneous List3 & List2  : ")

print ( List3 +  List2)
```

The resulting Pyton program that was used in the example before produced the first list, which contained six elements of type integer. Created the second and third lists in the same manner, each containing six entries of the integer type. Subsequently, the action of adding or concatenating, denoted by the symbol +, was utilized in order to merge List1 and List2. A similar process, namely the addition operation, was conducted between List3 and List2. The outcome of the sample program is displayed in its entirety in the output.

7.4.2 Repetition Operator

This operator returns a repeating List a specified number of times. The new List includes the same List the number of times specified. The multiplication operator (*) is employed in this case. Suppose we have a List L and an integer N. Doing L * N means repeating L for N times.

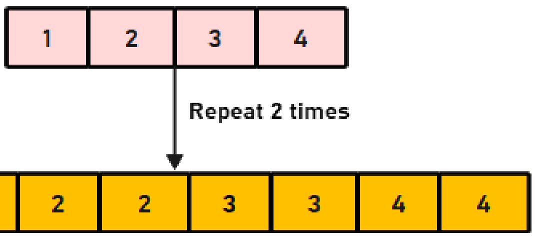
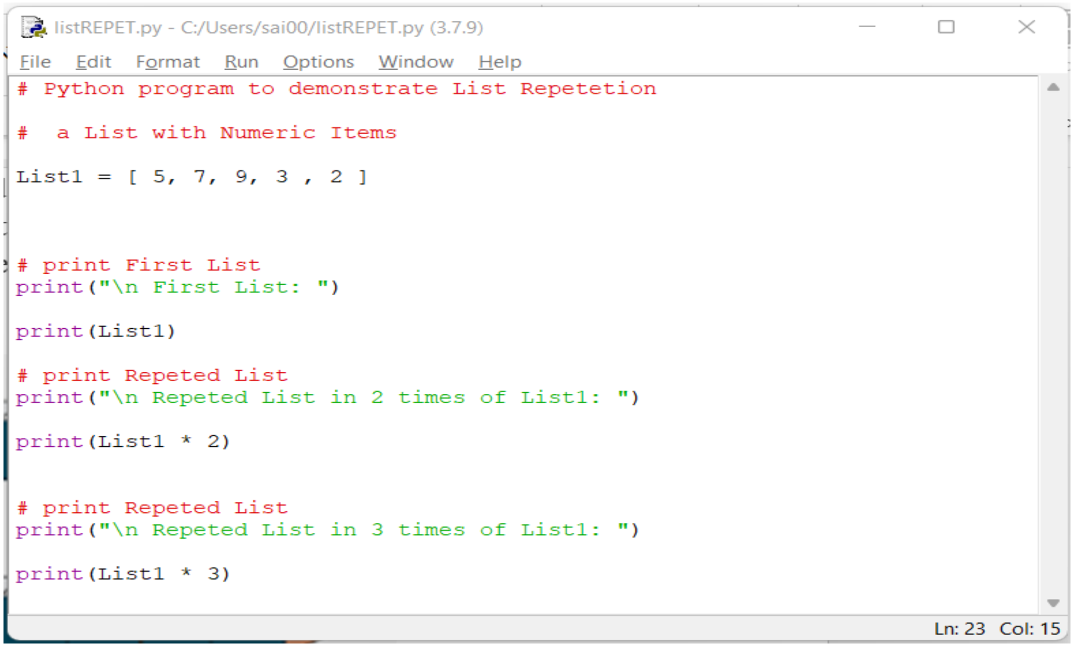


Fig 7.5. Repetition of Lists with '*' Operator

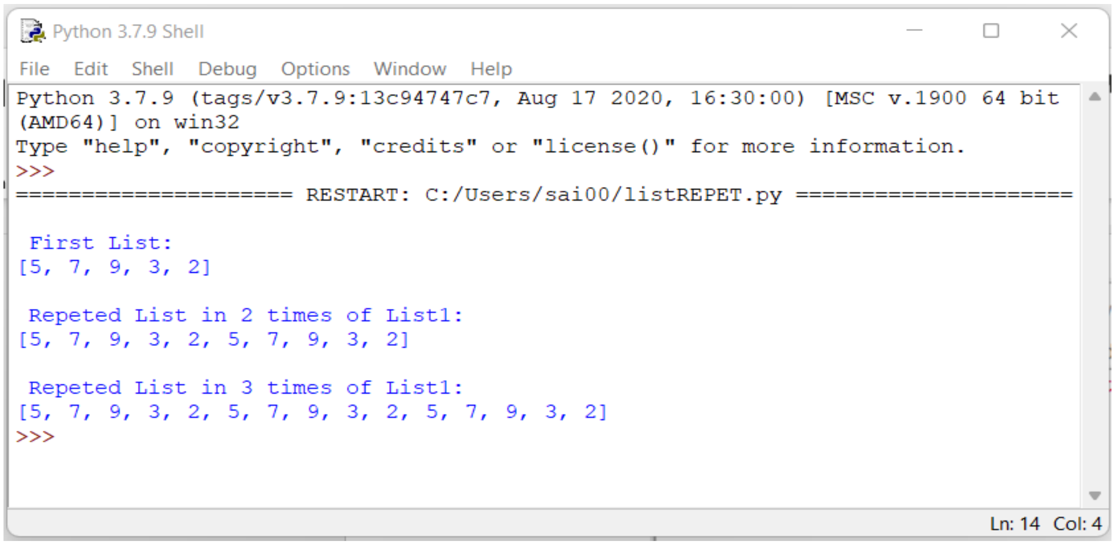
In the preceding example, the produced python program built the first list of six integers. Later, the repetition operation * was applied to List1 and repeated two or three times, with the result displayed on the output screen.

Example:


listREPET.py - C:/Users/sai00/listREPET.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 23 Col: 15

```
# Python program to demonstrate List Repetetion

#  a List with Numeric Items

List1 = [ 5, 7, 9, 3 , 2 ]


# print First List
print("\n First List: ")

print(List1)

# print Repeted List
print("\n Repeted List in 2 times of List1: ")

print(List1 * 2)


# print Repeted List
print("\n Repeted List in 3 times of List1: ")

print(List1 * 3)
```

Output:

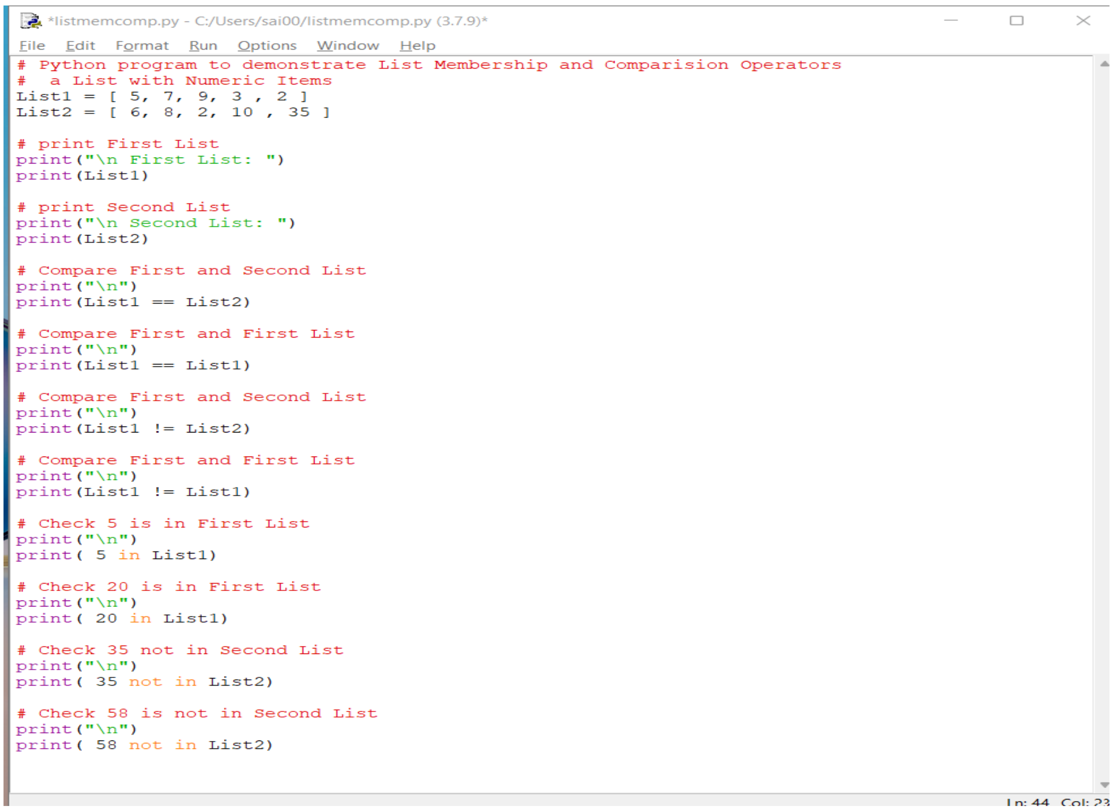


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/listREPET.py ====================

 First List:
[5, 7, 9, 3, 2]

 Repeted List in 2 times of List1:
[5, 7, 9, 3, 2, 5, 7, 9, 3, 2]

 Repeted List in 3 times of List1:
[5, 7, 9, 3, 2, 5, 7, 9, 3, 2, 5, 7, 9, 3, 2]
>>>
```

7.4.3 Membership Operator

These operators are often used to check if or if not an element or character exists in a particular string. The **in** returns **True** if a character **x** exists in a given string and **False** otherwise. The **not in** returns **True** if a character **x** does not exist in a given string and **False** otherwise.

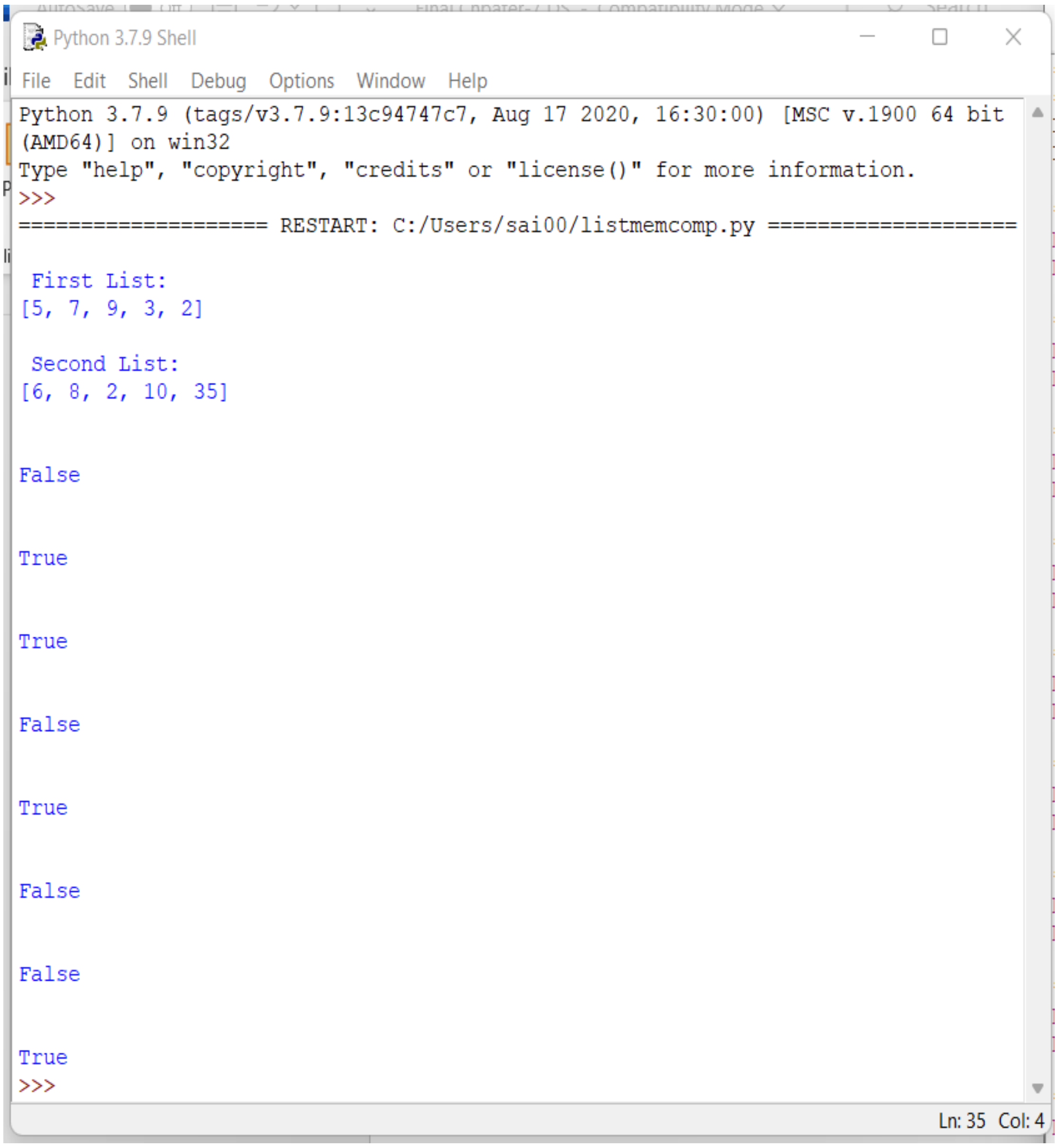
Example:


listmemcomp.py - C:/Users/sai00/listmemcomp.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 44 Col: 23

```
# Python program to demonstrate List Membership and Comparision Operators
#  a List with Numeric Items
List1 = [ 5, 7, 9, 3 , 2 ]
List2 = [ 6, 8, 2, 10 , 35 ]

# print First List
print("\n First List: ")
print(List1)

# print Second List
print("\n Second List: ")
print(List2)

# Compare First and Second List
print("\n")
print(List1 == List2)

# Compare First and First List
print("\n")
print(List1 == List1)

# Compare First and Second List
print("\n")
print(List1 != List2)

# Compare First and First List
print("\n")
print(List1 != List1)

# Check 5 is in First List
print("\n")
print( 5 in List1)

# Check 20 is in First List
print("\n")
print( 20 in List1)

# Check 35 not in Second List
print("\n")
print( 35 not in List2)

# Check 58 is not in Second List
print("\n")
print( 58 not in List2)
```

In the above example, the created pyton program demonstrates both comparison and membership functions. For this aim, I created the first and second lists of six integers. Later, both actions used the result shown on the output screen.

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
==================== RESTART: C:/Users/sai00/listmemcomp.py ====================

 First List:
[5, 7, 9, 3, 2]

 Second List:
[6, 8, 2, 10, 35]


False


True


True


False


True


False


False


True
>>> 
```

7.4.4 Comparison Operator

These operators in python just like their name specifies, are used to test the equivalence of two operands, in this case, 2 strings. Just because they return a boolean they are mostly used in conditional statements to evaluate the sameness of two strings. The == operator returns **True** when the two said strings are the same and **False** when the two said strings are not the same.

7.5 PYTHON LIST FUNCTIONS

Functions that are used with lists are known as list functions, and they are global functions. They return a value after receiving a list as an argument from the user. The following is a selection of examples of list functions:

You can choose from the following list functions in Python:

- len() : is a function that returns the total number of items in the list.
- sorted (): A new sorted list of the elements that were in the original list is returned by the function.
- The min (): function returns the element in the list that is the smallest.
- max (): function returns the item in the list that is the largest.
- sum (): This function returns the total value of all the items in the list.

7.5.1 len()

The number of items contained in a list can be displayed using the len() method. A list is used as the input, and the number of entries is returned as the value of the value returned. As shown in the following illustration, we will examine two lists that contain integers and create output, with the lengths of List_1 and List_2 being 2 and 3, respectively.

Example:

```
List_1 = [6,2]
List_2 = [42.56, 32.45, 87.12]
len(List_1)
len(List_2)
```

Output:

```
2
3
```

7.55.2. min ()

In a given list, the min () function displays the element that is the least significant. When given a list as input, it returns the minimum number of entries as the return value. In the next illustration, we will examine two lists that contain numbers and produce output based on the fact that the minimum number of entries in List_1 and List_2 are 2 and 32.45 respectively.

Example:

```
List_1 = [6,2]
List_2 = [42.56, 32.45, 87.12]
min(List_1)
min(List_2)
```

Output:

```
2
32.45
```

7.5.3. max ()

In a given list, the max () function displays the highest and most significant member. When given a list as input, it returns the maximum number of entries as the return value. In the next illustration, we will examine two lists that include numbers and create output that is, respectively, 6 and 87. The maximum number of items that can be found in List_1 and List_2 is twelve.

Example:

```
List_1 = [6,2]
List_2 = [42.56, 32.45, 87.12]
max(List_1)
max(List_2)
```

Output:

```
6
87.12
```

7.5.4. sort ()

The sort () function creates a sorted order of the elements in a list and displays them. It accepts an unordered list as input and returns a sorted list as the value after processing the input. In the following illustration, we will examine two lists that include numbers and produce output like [2,6] and [32.45, 42.56, 87.12], which are the sorted versions of List_1 and List_2, respectively.

Example:

```
List_1 = [6,2]
List_2 = [42.56, 32.45, 87.12]
sort.List_1
sort.List_2
```

Output:

```
[2,6]
[32.45, 42.56, 87.12]
```

7.6 List Methods

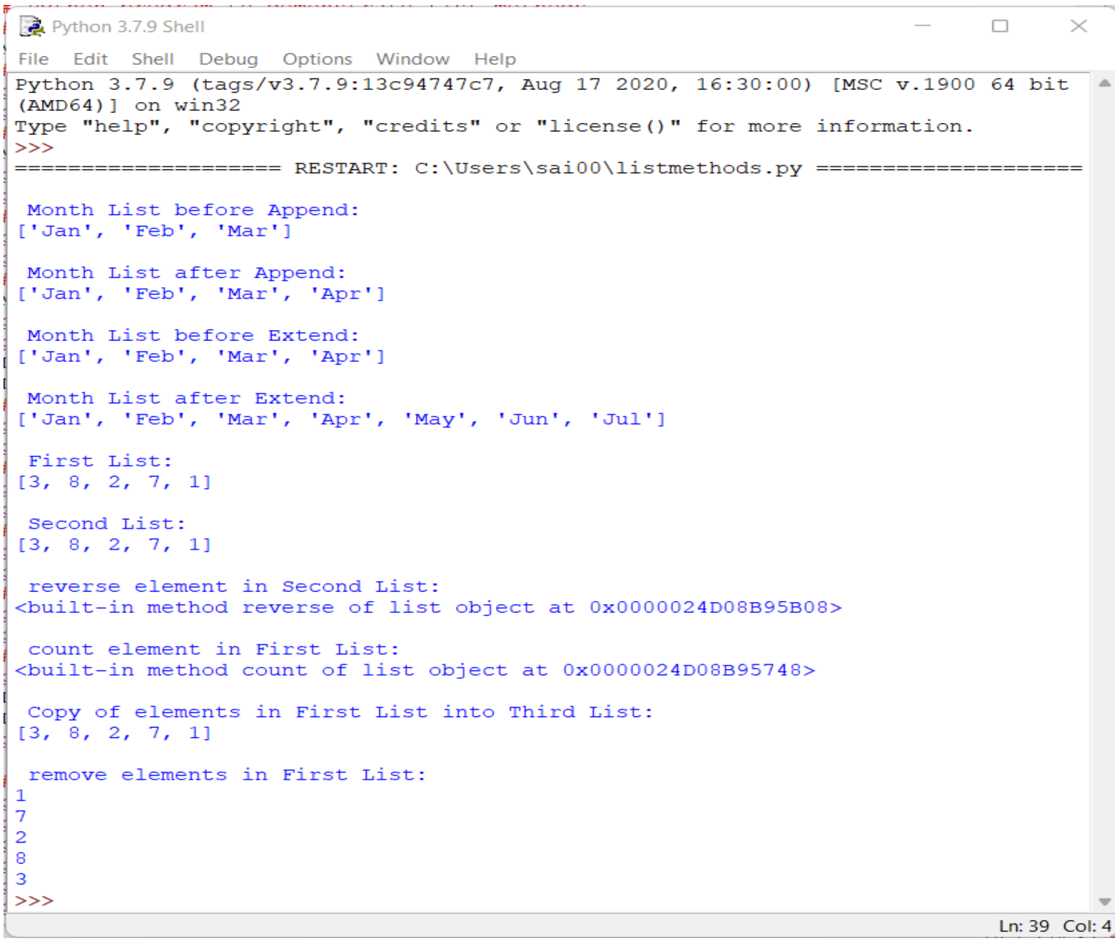
The built-in methods in lists called Python List Methods are used to manipulate Python lists and arrays. We've covered every technique you can use with Python lists below, including insert (), copy (), append (), and more, as show in Table 7.2. The result illustrates how each method is implemented in Python.

Table 7.2. Python List Methods

Method	**Description**
append ()	Used for adding elements to the end of the List.
copy ()	It returns a shallow copy of a list
clear ()	This method is used for removing all items from the list.
count ()	These methods count the elements.
extend ()	Adds each element of an iterable to the end of the List
index ()	Returns the lowest index where the element appears.
insert ()	Inserts a given element at a given index in a list.
pop ()	Removes and returns the last value from the List or the given index value.
remove ()	Removes a given object from the List.
reverse ()	Reverses objects of the List in place.

The generated Python program illustrates the operation of list methods in the example below. To do this, lists with month collections were established. Afterwards, many procedures were used, including pop (), copy (), reverse (), append (), and extend (). The output displays the entire outcome.

Example:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:\Users\sai00\listmethods.py ====================

 Month List before Append:
['Jan', 'Feb', 'Mar']

 Month List after Append:
['Jan', 'Feb', 'Mar', 'Apr']

 Month List before Extend:
['Jan', 'Feb', 'Mar', 'Apr']

 Month List after Extend:
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul']

 First List:
[3, 8, 2, 7, 1]

 Second List:
[3, 8, 2, 7, 1]

 reverse element in Second List:
<built-in method reverse of list object at 0x0000024D08B95B08>

 count element in First List:
<built-in method count of list object at 0x0000024D08B95748>

 Copy of elements in First List into Third List:
[3, 8, 2, 7, 1]

 remove elements in First List:
1
7
2
8
3
>>>
```

7.7 SUMMARY

Python lists are a fundamental and extremely versatile data structure. They allow you to store ordered groupings of elements that may be of different data kinds. Lists include a wide range of built-in methods and operations for adding, removing, altering, sorting, searching, and manipulating members. Lists are extremely useful for organizing and processing data in Python since they are dynamic and mutable. Whether you are a beginner or an expert programmer, learning lists is critical for releasing Python's full potential in disciplines such as data research, web development, automation scripting, and more.

7.8 TECHNICAL TERMS

List, Indexing, Negative Indexing, Max, Min, Count, Index, and Slicing

7..9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. How is a List created and called? Explain.
2. What are the various List methods? Explain.
3. Explain about List functions with example.

Short Notes:

1. Write about indexing access method.
2. How List is different from the Tuple and Dictionary.
3. Explain about concatenation operation of List.

7.10 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. "Think Python: How to Think Like a Computer Scientist" by Allen Downey
4. "Python Cookbook" by David Beazley and Brian K. Jones
5. "Programming Python" by Mark Lutz

Mrs. A. SARVANI

LESSON- 08

PYTHON TUPLE

AIMS AND OBJECTIVES

The main aim of this chapter is understanding the concept of tuples in Python Programming. The discussion related to understand what tuple and its characteristics is. After completion of this chapter, student will be able to know what tuple is, how it is different from other data types. Also able to know access tuples by various methods, operations, functions, and methods in tuples.

STRUCTURE

8.1 INTRODUCTION

Python is a popular high-level, general-purpose programming language that excels at creating graphical user interfaces and web applications. It is also a popular choice for application development due to its dynamic type and binding features. In this chapter we'll learn about tuples, an important data structure in Python programming.

Python tuples are a data structure that is quite like a list. The primary distinction between the two is that tuples are immutable, which means they cannot be modified once generated. This makes them excellent for storing non-modifiable data, such as database records. A tuple can contain any number of objects of various types, including strings, integers, floats, lists, and so on. Let's look at how to generate and use a tuple to make our programming work easier.

8.2 PYTHON TUPLE

A sequence of any items that are separated by commas and wrapped in parenthesis is referred to as a tuple. We use tuples to represent fixed collections of elements since they are immutable objects, which means they cannot be modified. Tuples are used to carry out this function. Tuple items are placed in a specific order, cannot be altered, and permit duplicate values. When we say that tuples are ordered, we are referring to the fact that the items in the tuple have a predetermined order on which they will remain indefinitely. Tuples and Python lists share some similarities in terms of indexing, nested objects, and repetition; nevertheless, the most significant distinction between the two is that a Python tuple is immutable, whereas a Python list is mutable. Tuples are used in Python programming languages. Since tuples are indexed, the first item has an index of [0], the second item uses an index of [1], and so on.

8.2.1 Creating Python Tuples

It is possible to create a tuple by associated with all of the items (elements) in parentheses () rather than square brackets [], and by separating each element with commas. It is possible for a tuple to include any number of objects of different types, including integers, floats, lists, strings, and so on. In addition, you have the option of specifying nested tuples, which can include one or more items that are either dictionaries, lists, or tuples.

The given example shows how to create simple Tuple in python:

Example:

```
emp_tup = ()    #   Empty Tuple

int_tup = (2, 8, 1, 6, 15, 3)    # A Tuple with integers

mixed_tup = (12, "sai", 81.3)    # A Tuple with mixed data items

nested_tup = ("Python", [8,5,17,6], (2, 6, 1, 20))    # Nested Tuple
```

We produced four different sorts of tuples in the example that was just presented: empty, int-type, mixed type, and nested type. It is after the initialization of data items that the size of the empty tuple is calculated. Nevertheless, the elements that are part of the int type and the mixed type are the numbers 6 and 3. An example of a nested tuple is a special sort of tuple in which each element also contains additional elements. A string, a list, and a tuple were the three elements that were present in the nested tuple that was defined before.

8.2.2 Advantages of Tuple over List

Since they are so comparable, tuples and lists are applied in scenarios that are comparable. On the other hand, there are a few benefits that come along with utilizing a tuple rather than a list.

- In contrast to lists, the Tuples cannot be modified in any way. The addition, removal, or replacement of a tuple is not possible.
- Tuples are often utilized for heterogeneous data kinds, which are distinct from one another, whereas lists are typically utilized for homogeneous data types, which are comparable to one another.
- As a result of the immutability of tuples, iterating through them is a more efficient process than iterating through a list. As a consequence of this, there is a slight improvement in performance.
- Dictionary keys can be derived from tuples that include elements that cannot be changed. When it comes to lists, this is not possible.
- If you have data that does not change, implementing it as a tuple will ensure that it continues to be protected from being written to.
- If you wish to make changes to the information contained in a tuple, we will first need to transform it into a list.

8.3 ACCESSING TUPLE

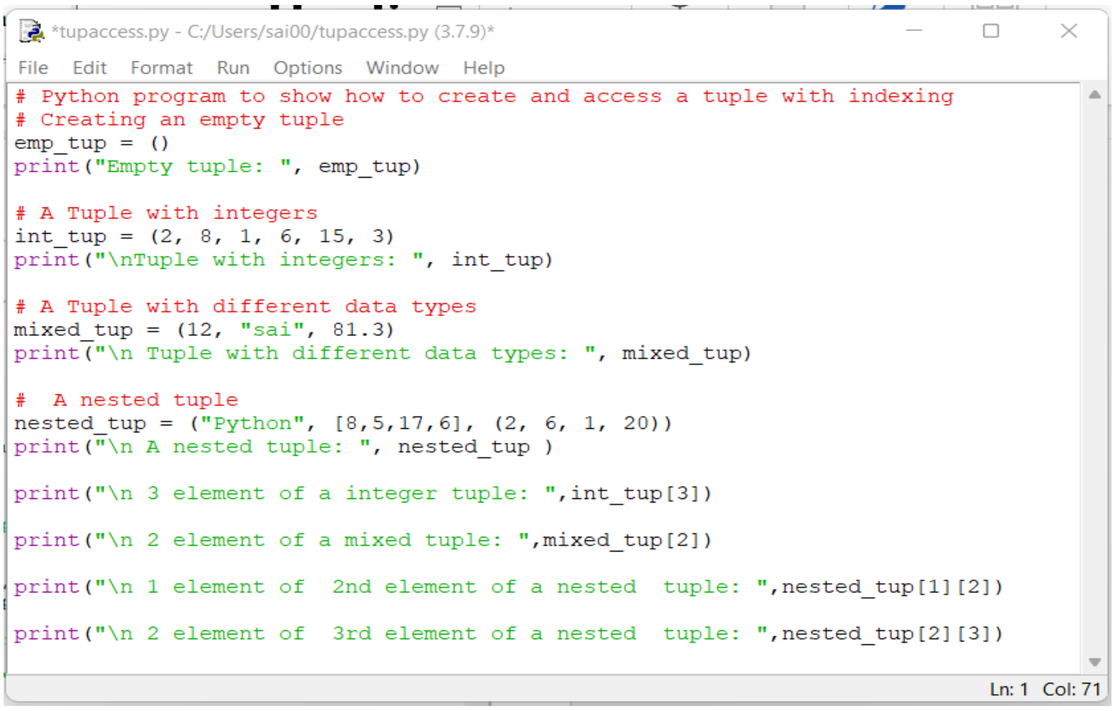
A tuple's objects can be accessed in 3 different types of ways which includes:

- Indexing
- Negative Indexing
- Slicing

8.3.1 Indexing

Accessing an item within a tuple that has an index that begins at 0 can be accomplished using the index operator []. A tuple that contains five items will have indices that range from 0 to 4, inclusive. An index that is higher than four will be considered out of range.

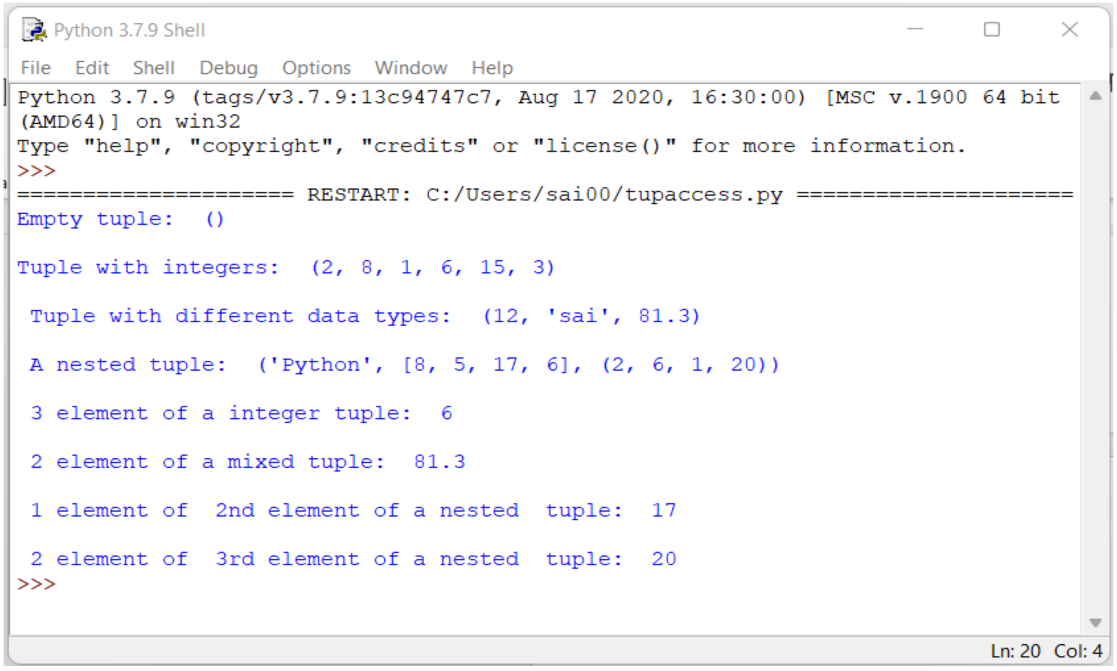
Example:


tupaccess.py - C:/Users/sai00/tupaccess.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 1 Col: 71

```
# Python program to show how to create and access a tuple with indexing
# Creating an empty tuple
emp_tup = ()
print("Empty tuple: ", emp_tup)

# A Tuple with integers
int_tup = (2, 8, 1, 6, 15, 3)
print("\nTuple with integers: ", int_tup)

# A Tuple with different data types
mixed_tup = (12, "sai", 81.3)
print("\n Tuple with different data types: ", mixed_tup)

#  A nested tuple
nested_tup = ("Python", [8,5,17,6], (2, 6, 1, 20))
print("\n A nested tuple: ", nested_tup )

print("\n 3 element of a integer tuple: ",int_tup[3])

print("\n 2 element of a mixed tuple: ",mixed_tup[2])

print("\n 1 element of  2nd element of a nested  tuple: ",nested_tup[1][2])

print("\n 2 element of  3rd element of a nested  tuple: ",nested_tup[2][3])
```

The four types of tuples that have previously been constructed in the example above—empty, int-type, mixed type, and nested type—are accessed using an index. This operator is quite helpful in accessing particular elements from the tuple. Different elements are accessible from different types of tuples in the code above. Three elements from the integer tuple and two from the mixed tuple, for instance. Similar access is made using the indexing method in nested tuples.

Output:

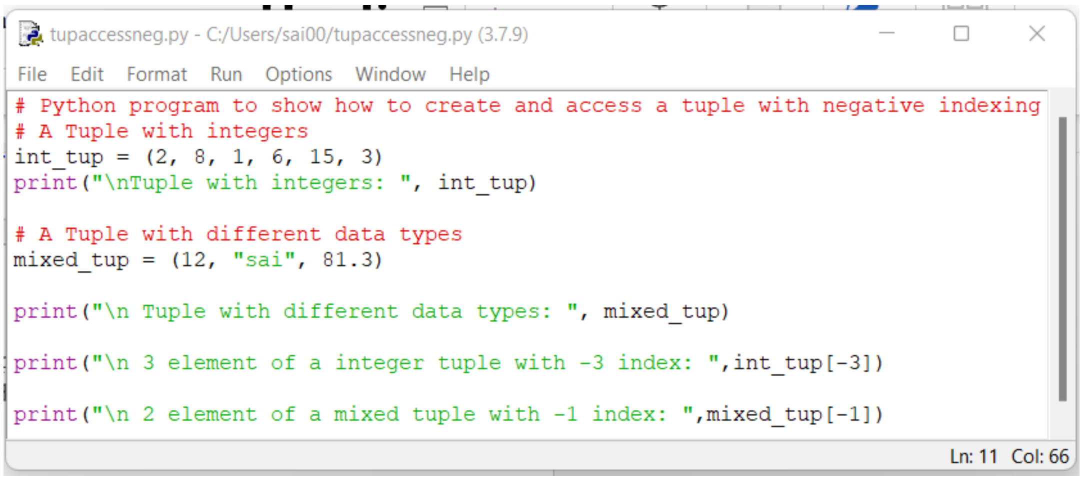


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/Users/sai00/tupaccess.py =====================
Empty tuple:  ()

Tuple with integers:  (2, 8, 1, 6, 15, 3)

 Tuple with different data types:  (12, 'sai', 81.3)

 A nested tuple:  ('Python', [8, 5, 17, 6], (2, 6, 1, 20))

 3 element of a integer tuple:  6

 2 element of a mixed tuple:  81.3

 1 element of  2nd element of a nested  tuple:  17

 2 element of  3rd element of a nested  tuple:  20
>>>
```

8.3.2 Negative Indexing

Tuple, a type of sequence object in Python, also allows negative indexing. -1 addresses the final item in the selection, -2 addresses the second-to-last item, and so on.

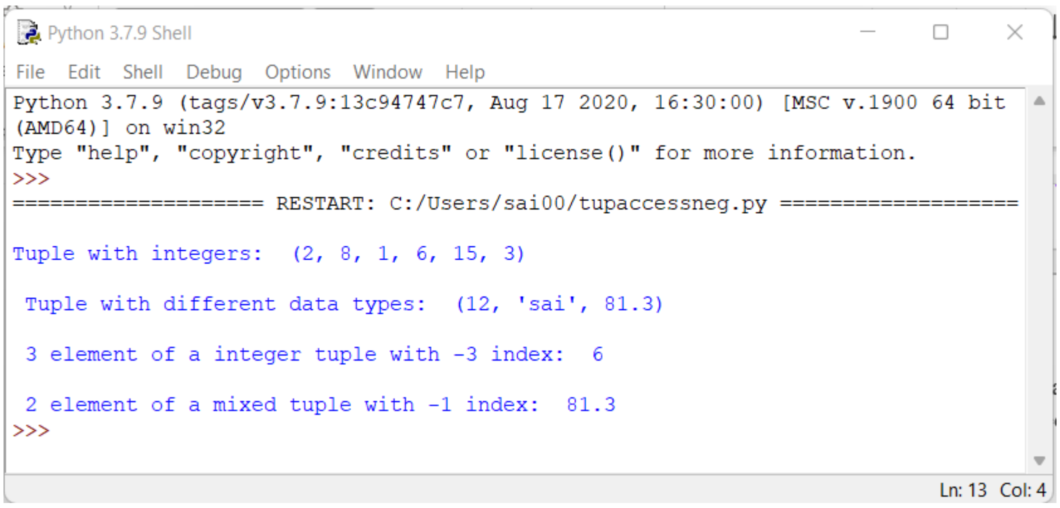
Example:


tupaccessneg.py - C:/Users/sai00/tupaccessneg.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 11 Col: 66

```
# Python program to show how to create and access a tuple with negative indexing
# A Tuple with integers
int_tup = (2, 8, 1, 6, 15, 3)
print("\nTuple with integers: ", int_tup)

# A Tuple with different data types
mixed_tup = (12, "sai", 81.3)

print("\n Tuple with different data types: ", mixed_tup)

print("\n 3 element of a integer tuple with -3 index: ",int_tup[-3])

print("\n 2 element of a mixed tuple with -1 index: ",mixed_tup[-1])
```

Two sorts of tuples, empty and int-type, are already constructed in the example above and are accessed using a negative index. For instance, the -3 and -1 indexes are used to retrieve the elements 3 and 2 of the integer and mixed tuples, respectively. In a similar vein, nested tuples can also use this type of access.

Output:

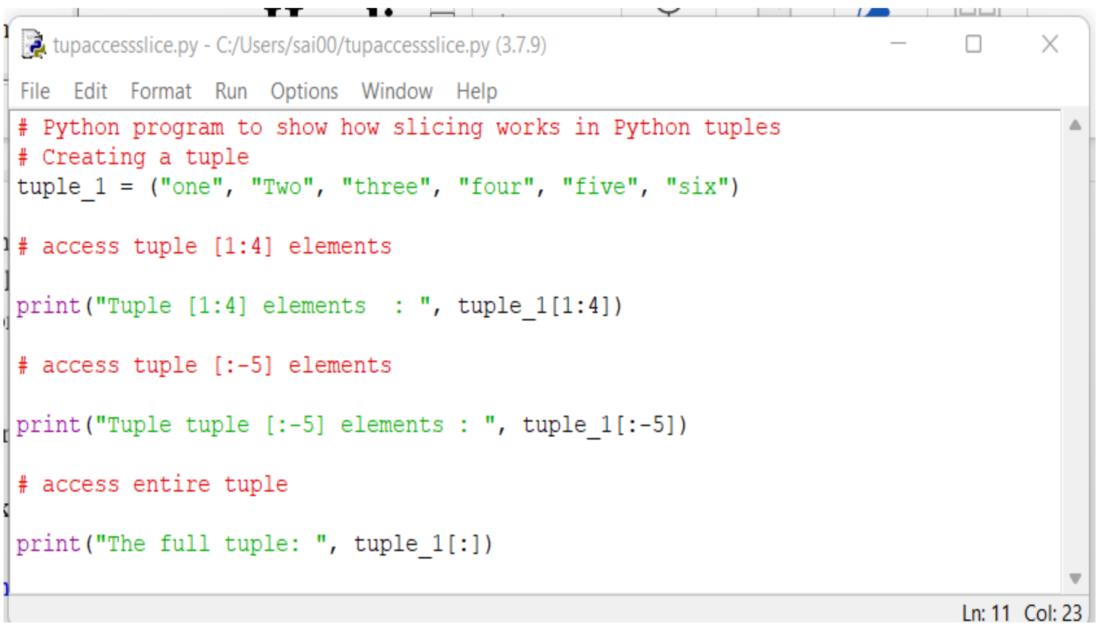


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/tupaccessneg.py ===================

Tuple with integers:  (2, 8, 1, 6, 15, 3)

 Tuple with different data types:  (12, 'sai', 81.3)

 3 element of a integer tuple with -3 index:  6

 2 element of a mixed tuple with -1 index:  81.3
>>>
```

8.3.3. Slicing

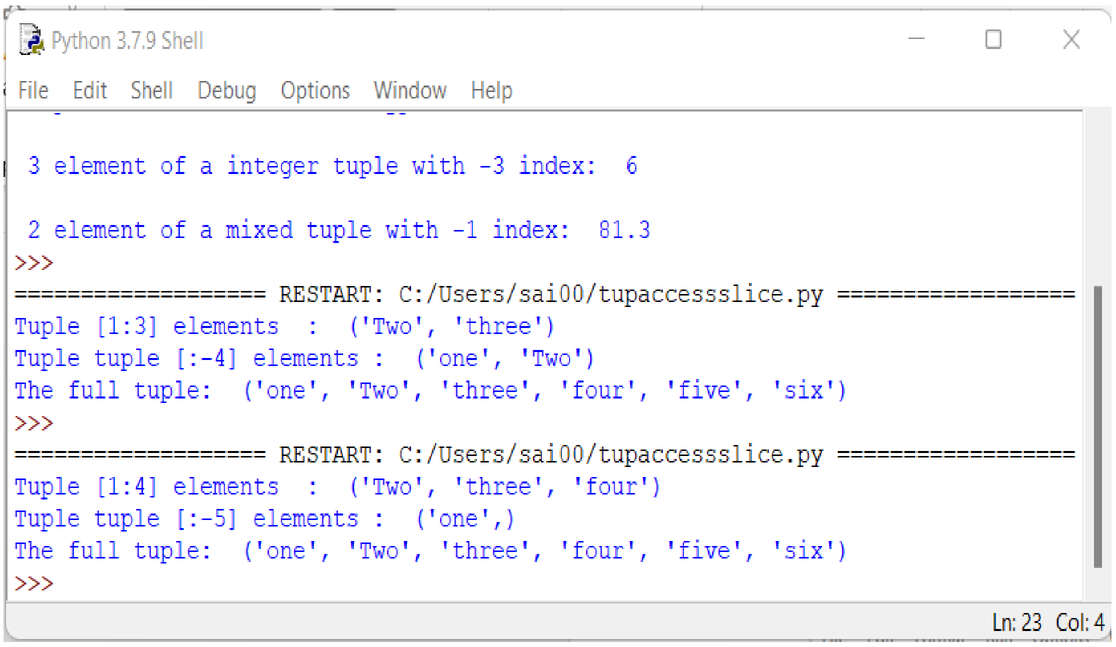
In Python, tuple slicing is a widely used technique that programmers use to solve real-world problems. Examine a Python tuple. To access a range of a tuple's elements, slice it. One method is to use the colon as a simple slicing operator (:).We can use the slicing operator colon (:) to access different tuple components.

Example:


tupaccessslice.py - C:/Users/sai00/tupaccessslice.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 11 Col: 23

```
# Python program to show how slicing works in Python tuples
# Creating a tuple
tuple_1 = ("one", "Two", "three", "four", "five", "six")

# access tuple [1:4] elements

print("Tuple [1:4] elements  : ", tuple_1[1:4])

# access tuple [:-5] elements

print("Tuple tuple [:-5] elements : ", tuple_1[:-5])

# access entire tuple

print("The full tuple: ", tuple_1[:])
```

Output:



```
 3 element of a integer tuple with -3 index:  6

 2 element of a mixed tuple with -1 index:  81.3
>>>
================== RESTART: C:/Users/sai00/tupaccessslice.py ==================
Tuple [1:3] elements  :  ('Two', 'three')
Tuple tuple [:-4] elements :  ('one', 'Two')
The full tuple:  ('one', 'Two', 'three', 'four', 'five', 'six')
>>>
================== RESTART: C:/Users/sai00/tupaccessslice.py ==================
Tuple [1:4] elements  :  ('Two', 'three', 'four')
Tuple tuple [:-5] elements :  ('one',)
The full tuple:  ('one', 'Two', 'three', 'four', 'five', 'six')
>>>
```

8.4 PYTHON TUPLE OPERATIONS

Tuple is a sequence in Python. As a result, we can use the + operator to concatenate two tuples and the "*" operator to concatenate many copies of a tuple. Tuple objects are used by the membership operators "in" and "not in."

Table 8.1 Python Tuple Operations

Python Expression	**Results**	**Description**
len((1, 2, 3))	**3**	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

8.4.1 Concatenation of Tuples

The process of connecting two or more tuples is called concatenation. The operator "+" is used for concatenation. Tuple concatenation is always performed starting at the end of the original tuple. On tuples, other arithmetic operations are not applicable. Concatenation can only be used to join datatypes that are the same, joining a list and a tuple result in an error. The idea of tuple concatenation is shown in Figure 8.1.

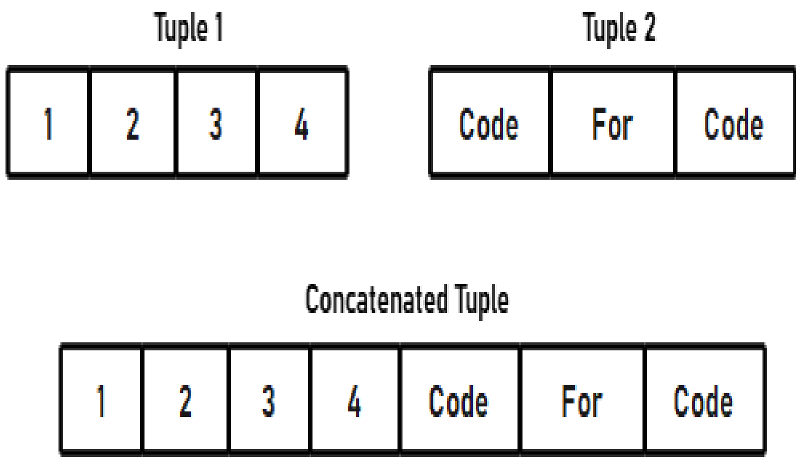
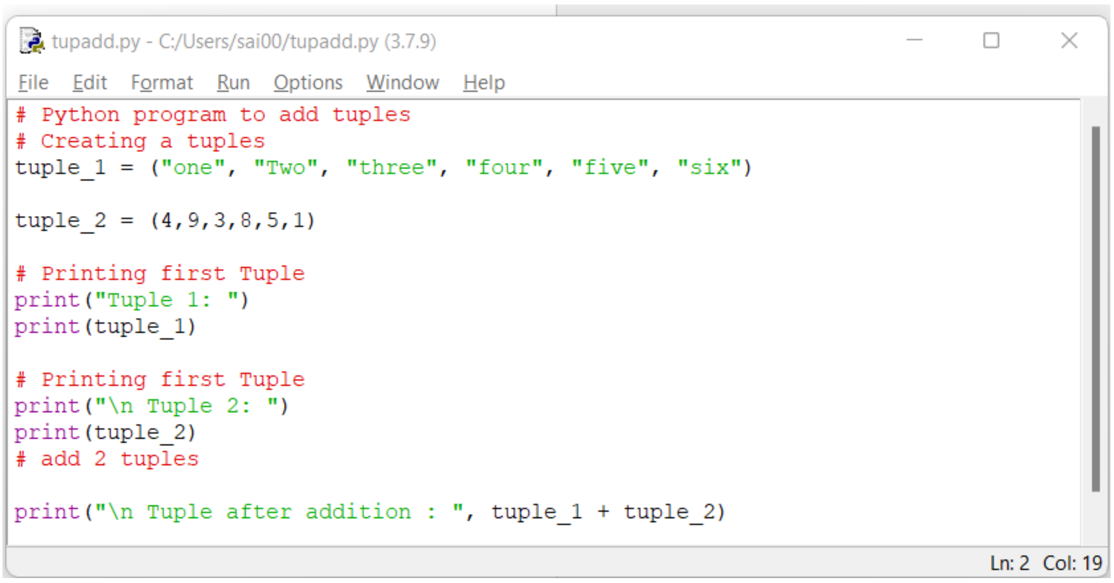


Fig 8.1 Concatenation of Tuples in Python

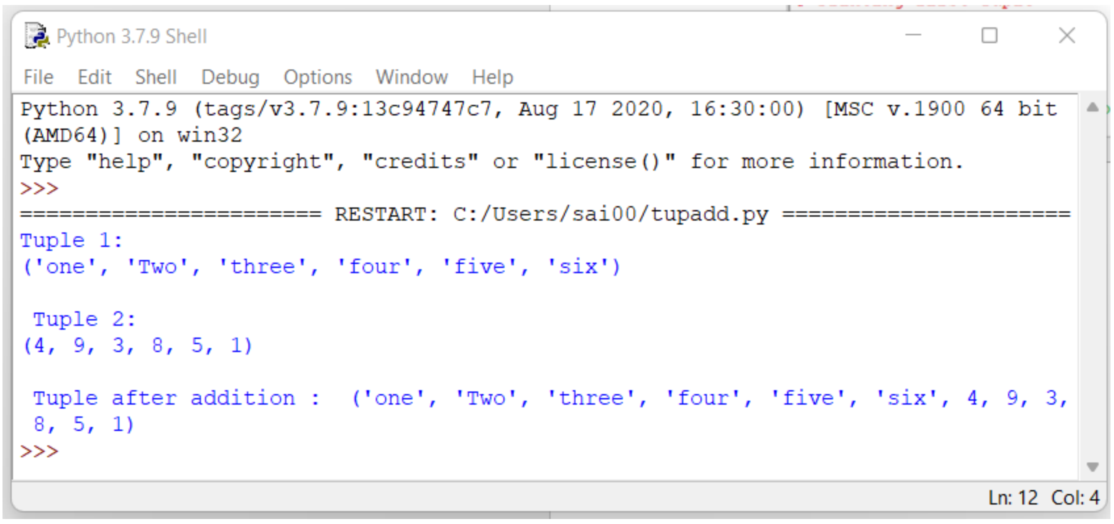
Example:


tupadd.py - C:/Users/sai00/tupadd.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 2 Col: 19

```
# Python program to add tuples
# Creating a tuples
tuple_1 = ("one", "Two", "three", "four", "five", "six")

tuple_2 = (4,9,3,8,5,1)

# Printing first Tuple
print("Tuple 1: ")
print(tuple_1)

# Printing first Tuple
print("\n Tuple 2: ")
print(tuple_2)
# add 2 tuples

print("\n Tuple after addition : ", tuple_1 + tuple_2)
```

Two tuples of the types character and integer were constructed in the example above, designated as tuple_1 and tuple_2. The outcome of later addition operations applied to two tuples is reported. The result, which combines the contents of tuples 1 and 2 into a single tuple, is displayed on screen.

Output:

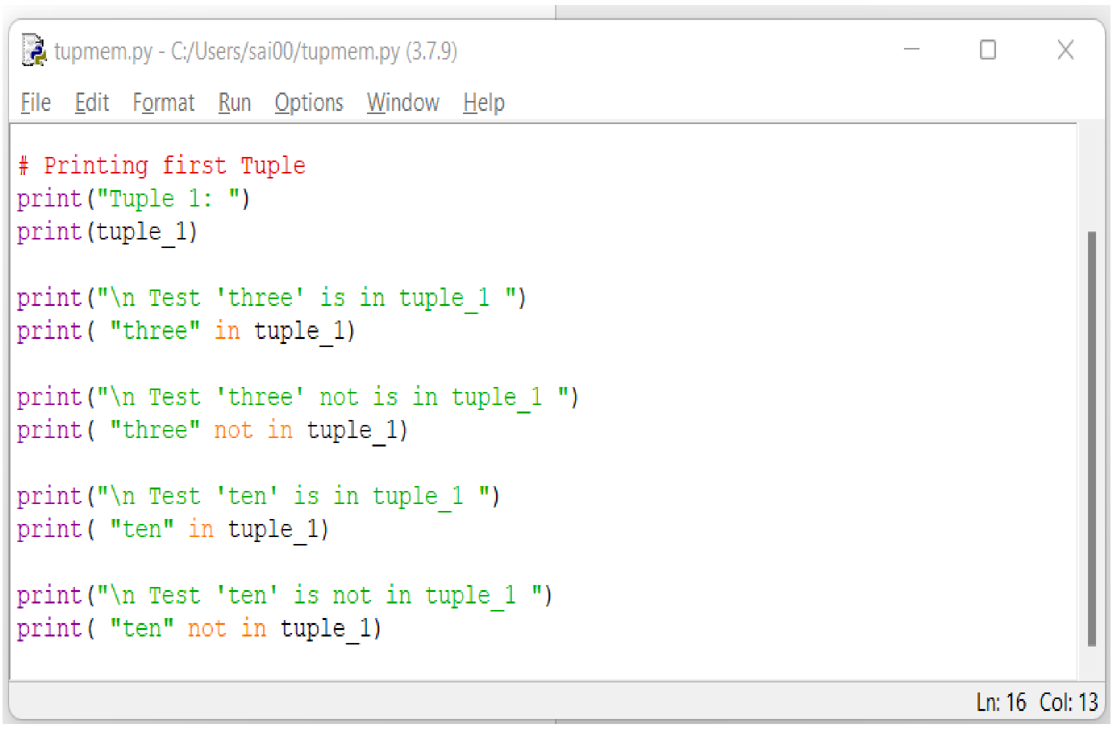


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/tupadd.py =====================
Tuple 1:
('one', 'Two', 'three', 'four', 'five', 'six')

 Tuple 2:
(4, 9, 3, 8, 5, 1)

 Tuple after addition :  ('one', 'Two', 'three', 'four', 'five', 'six', 4, 9, 3,
 8, 5, 1)
>>>
```

8.4.2 Tuple Membership

The existence of an item in a tuple can be ascertained by using the in and not in keywords.

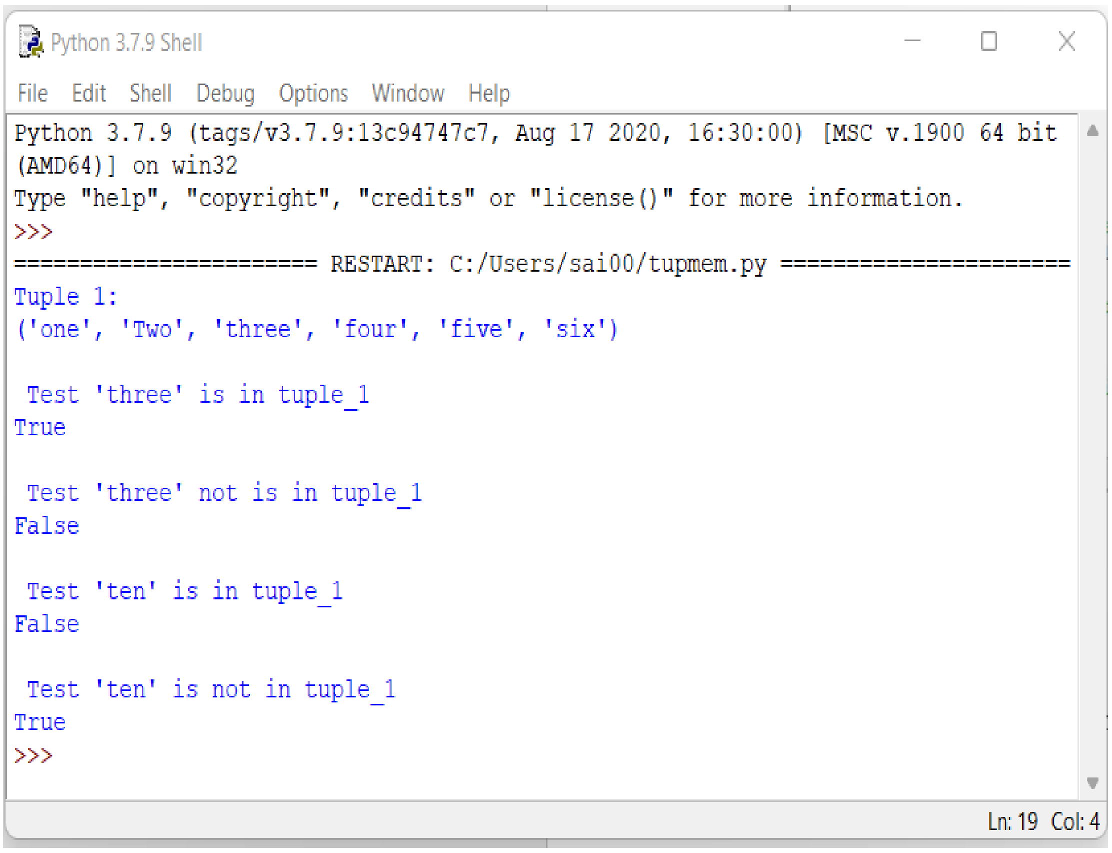
Example:



tupmem.py - C:/Users/sai00/tupmem.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 16 Col: 13

```
# Printing first Tuple
print("Tuple 1: ")
print(tuple_1)

print("\n Test 'three' is in tuple_1 ")
print( "three" in tuple_1)

print("\n Test 'three' not is in tuple_1 ")
print( "three" not in tuple_1)

print("\n Test 'ten' is in tuple_1 ")
print( "ten" in tuple_1)

print("\n Test 'ten' is not in tuple_1 ")
print( "ten" not in tuple_1)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/tupmem.py =====================
Tuple 1:
('one', 'Two', 'three', 'four', 'five', 'six')

 Test 'three' is in tuple_1
True

 Test 'three' not is in tuple_1
False

 Test 'ten' is in tuple_1
False

 Test 'ten' is not in tuple_1
True
>>>
```

Applying membership procedures on the two produced tuples, tuple_1 and tuple_2, as demonstrated in the preceding example. The results of these membership operations, such as is and is not, are TRUE or FALSE. Verified whether the term "there" is available in the case above. In a same manner, look up further words.

8.5 PYTHON TUPLE FUNCTIONS

Python offers a variety of functions for carrying out tasks. Functions such as cmp(), max(), min(), and so forth are used to carry out particular tasks. Each function's explanation can be found in Table 8.1.

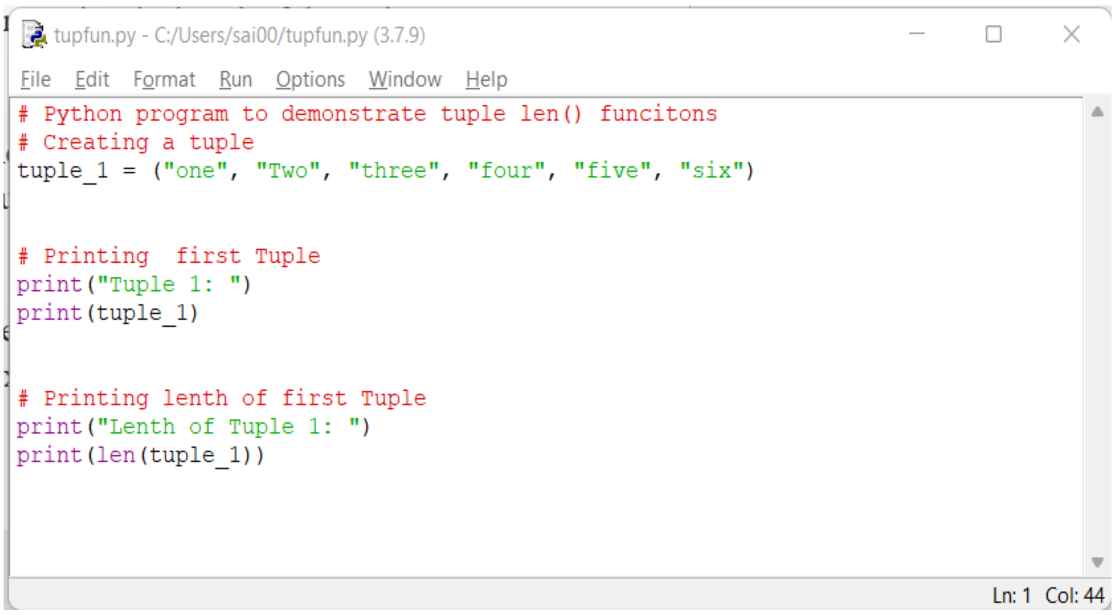
Table 8.1 Python Tuple Functions

Function	Description
cmp(tuple1, tuple2)	Compares elements of both the tuples
len(tuple)	Returns the total length of the tuple
max(tuple)	Returns the largest element from the tuple
min(tuple)	Returns the smallest element from the tuple
tuple(seq)	Converts a list into tuple

8.5.1 len()

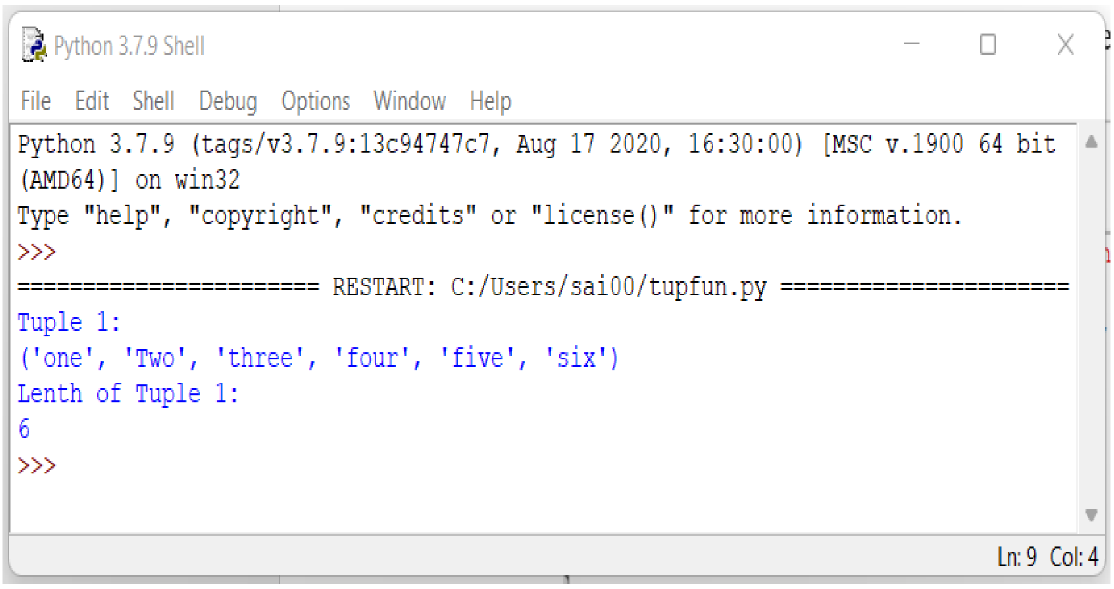
The number of elements in a tuple can be obtained using the len() method. It accepts a tuple as an input and outputs an integer number that is the tuple's length.

Example:


tupfun.py - C:/Users/sai00/tupfun.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 1 Col: 44

```
# Python program to demonstrate tuple len() funcitons
# Creating a tuple
tuple_1 = ("one", "Two", "three", "four", "five", "six")


# Printing  first Tuple
print("Tuple 1: ")
print(tuple_1)


# Printing lenth of first Tuple
print("Lenth of Tuple 1: ")
print(len(tuple_1))
```

Output:



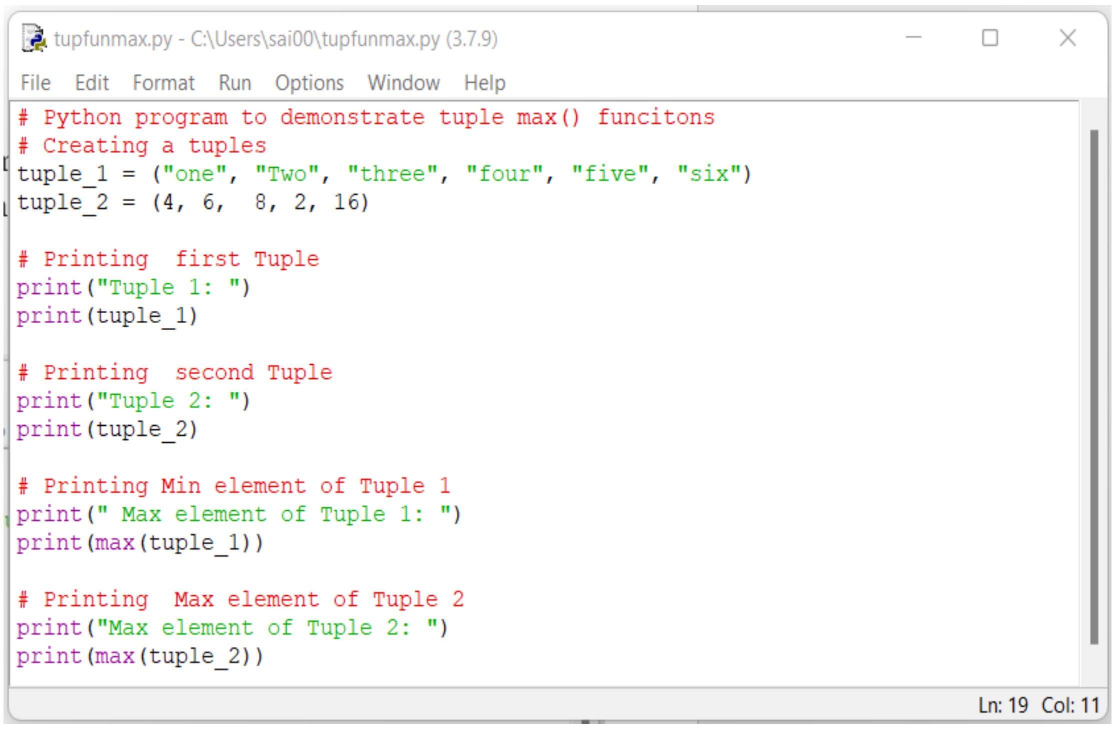
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/Users/sai00/tupfun.py =====================
Tuple 1:
('one', 'Two', 'three', 'four', 'five', 'six')
Lenth of Tuple 1:
6
>>>
```

We have defined a tuple called my_tuple with five items in the example above. The length of the tuple, which is 5, was then obtained using the len() method.

8.5.2 max ()

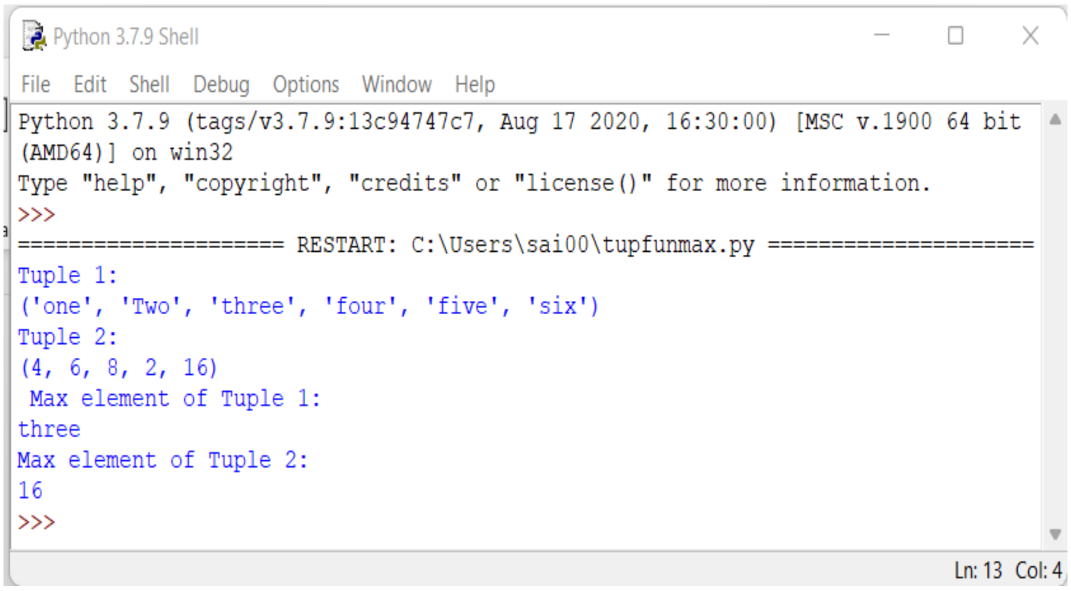
To get the maximum value in a tuple, use the max () function. It accepts a tuple as an input and outputs the tuple's maximum value.

Example:


tupfunmax.py - C:\Users\sai00\tupfunmax.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 19 Col: 11

```
# Python program to demonstrate tuple max() funcitons
# Creating a tuples
tuple_1 = ("one", "Two", "three", "four", "five", "six")
tuple_2 = (4, 6,  8, 2, 16)

# Printing  first Tuple
print("Tuple 1: ")
print(tuple_1)

# Printing  second Tuple
print("Tuple 2: ")
print(tuple_2)

# Printing Min element of Tuple 1
print(" Max element of Tuple 1: ")
print(max(tuple_1))

# Printing  Max element of Tuple 2
print("Max element of Tuple 2: ")
print(max(tuple_2))
```

Output:



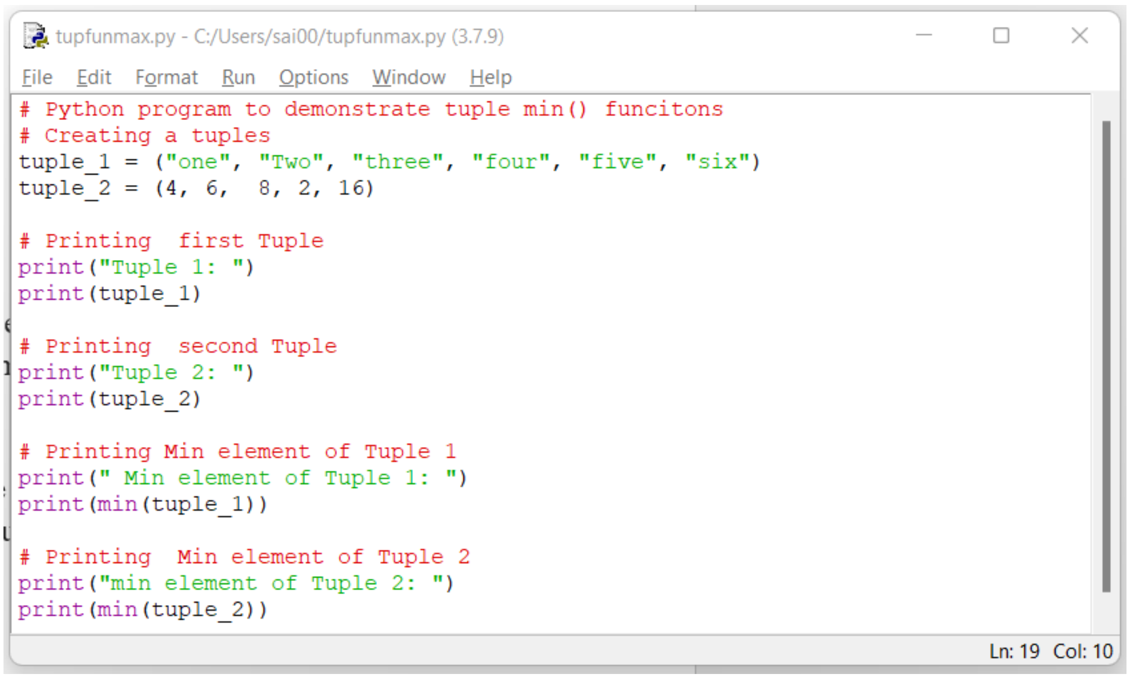
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:\Users\sai00\tupfunmax.py ====================
Tuple 1:
('one', 'Two', 'three', 'four', 'five', 'six')
Tuple 2:
(4, 6, 8, 2, 16)
 Max element of Tuple 1:
three
Max element of Tuple 2:
16
>>>
```

We have defined a tuple called my_tuple with five items in the example above. The maximum value in the tuple, which is 9, was then obtained using the max() method.

8.5.3 min ()

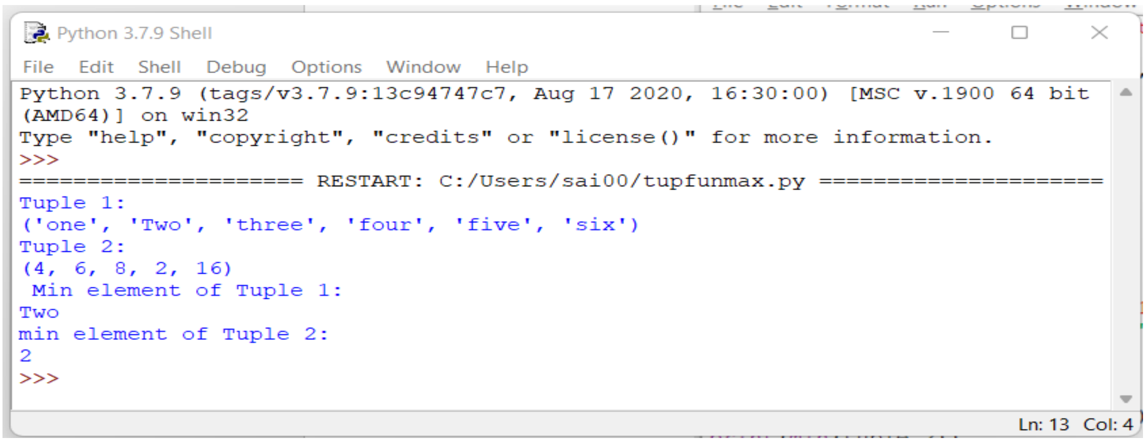
To get the lowest value in a tuple, use the min () function. It accepts a tuple as an input and outputs the tuple's minimal value.

Example:


tupfunmax.py - C:/Users/sai00/tupfunmax.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 19 Col: 10

```python
# Python program to demonstrate tuple min() funcitons
# Creating a tuples
tuple_1 = ("one", "Two", "three", "four", "five", "six")
tuple_2 = (4, 6,  8, 2, 16)

# Printing  first Tuple
print("Tuple 1: ")
print(tuple_1)

# Printing  second Tuple
print("Tuple 2: ")
print(tuple_2)

# Printing Min element of Tuple 1
print(" Min element of Tuple 1: ")
print(min(tuple_1))

# Printing  Min element of Tuple 2
print("min element of Tuple 2: ")
print(min(tuple_2))
```

Output:



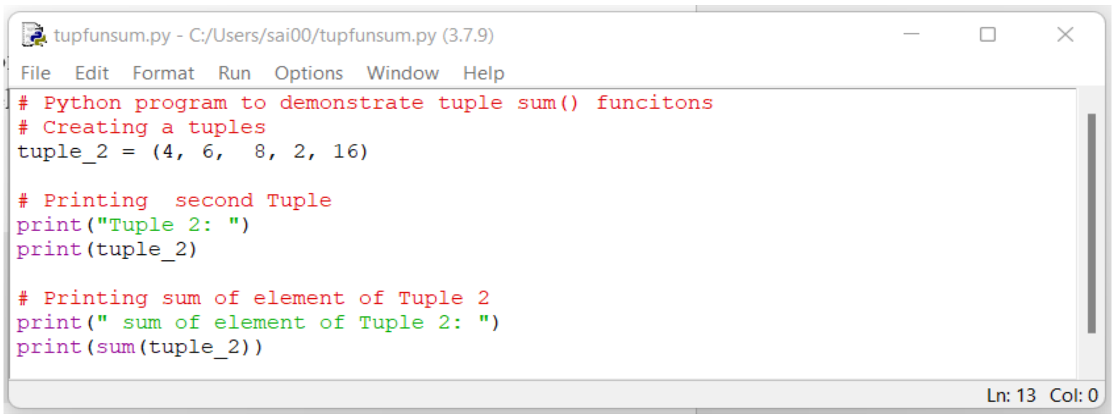
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/tupfunmax.py ====================
Tuple 1:
('one', 'Two', 'three', 'four', 'five', 'six')
Tuple 2:
(4, 6, 8, 2, 16)
 Min element of Tuple 1:
Two
min element of Tuple 2:
2
>>>
```

We have defined two tuples tuple_1 and tuple_2 with six and five items in the example above. Next, we obtained the tuple's minimal values Two, and 2 by using the min() function.

8.5.4 sum ()

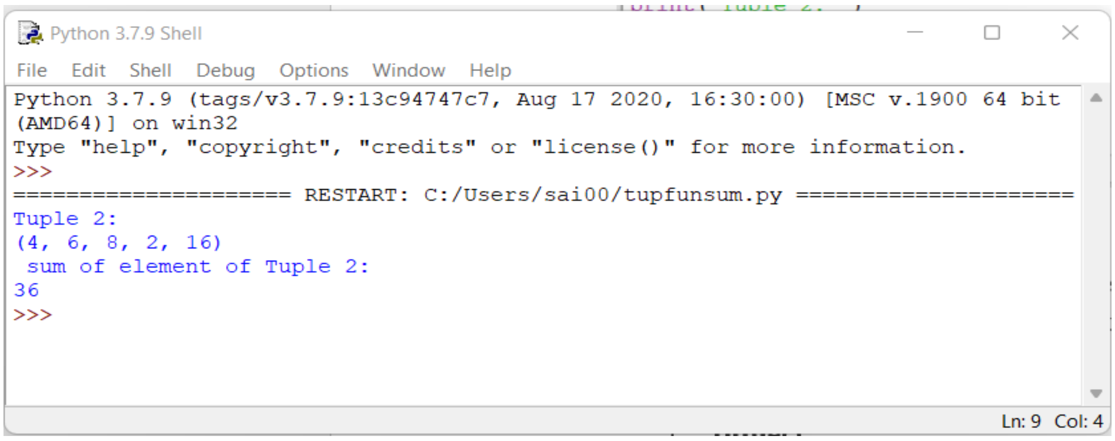
The sum of each element in a tuple can be obtained using the sum () function. It accepts a tuple as an input and outputs the total of each tuple's elements.

Example:


tupfunsum.py - C:/Users/sai00/tupfunsum.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 13 Col: 0

```
# Python program to demonstrate tuple sum() funcitons
# Creating a tuples
tuple_2 = (4, 6,  8, 2, 16)

# Printing  second Tuple
print("Tuple 2: ")
print(tuple_2)

# Printing sum of element of Tuple 2
print(" sum of element of Tuple 2: ")
print(sum(tuple_2))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/tupfunsum.py ====================
Tuple 2:
(4, 6, 8, 2, 16)
 sum of element of Tuple 2:
36
>>>
```

We have defined a tuple called tuple_2 with five items in the example above. The total of all the elements in the tuple, which is 36, was then obtained using the sum () method.

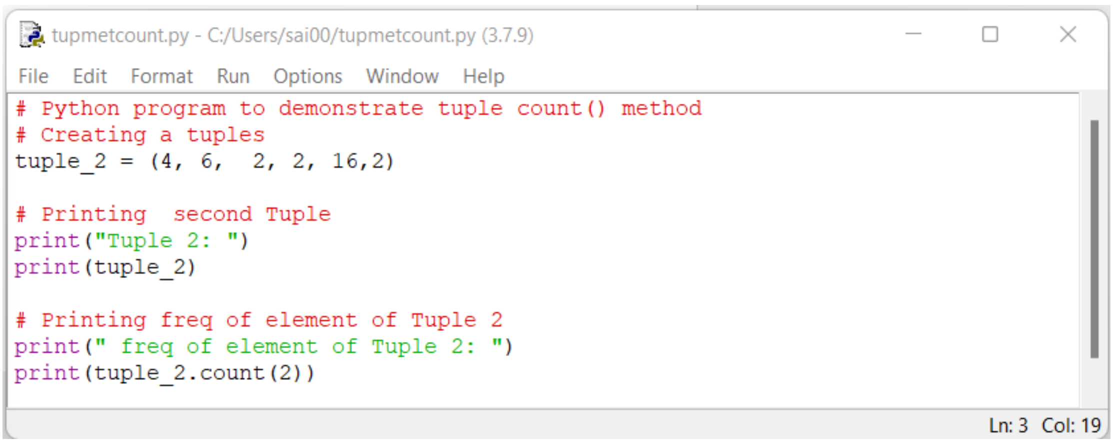
8.6 TUPLE METHODS

Python's tuple routines offer an extensive range of functionalities for working with tuples. Programmers can find the length, maximum or minimum value, total of all items, and create tuples from iterables using these functions. Easy finding and counting of particular elements within tuples is also made possible by the index() and count() operations.

8.6.1 Count () Method

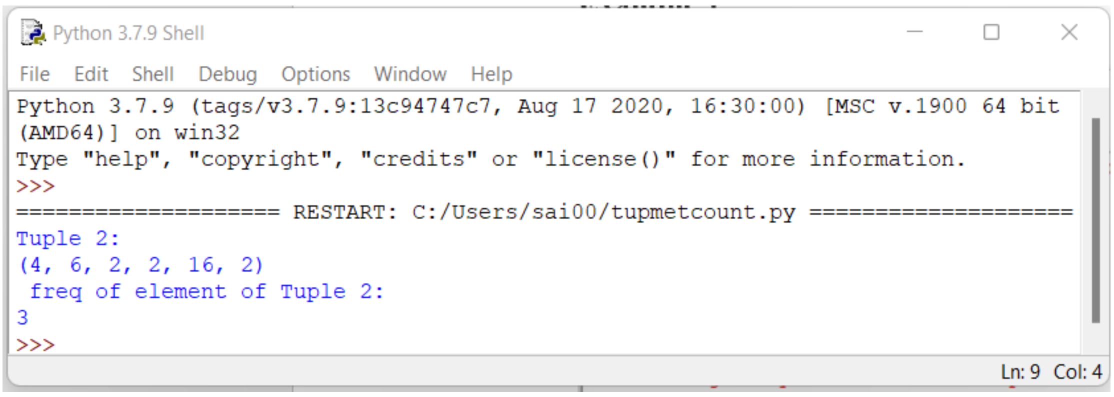
A built-in Python function called count () can be used to determine how many times a certain element appears in a tuple. The value to be counted is the only input that the method accepts.

Example:


tupmetcount.py - C:/Users/sai00/tupmetcount.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 3 Col: 19

```
# Python program to demonstrate tuple count() method
# Creating a tuples
tuple_2 = (4, 6,  2, 2, 16,2)

# Printing  second Tuple
print("Tuple 2: ")
print(tuple_2)

# Printing freq of element of Tuple 2
print(" freq of element of Tuple 2: ")
print(tuple_2.count(2))
```

Output:



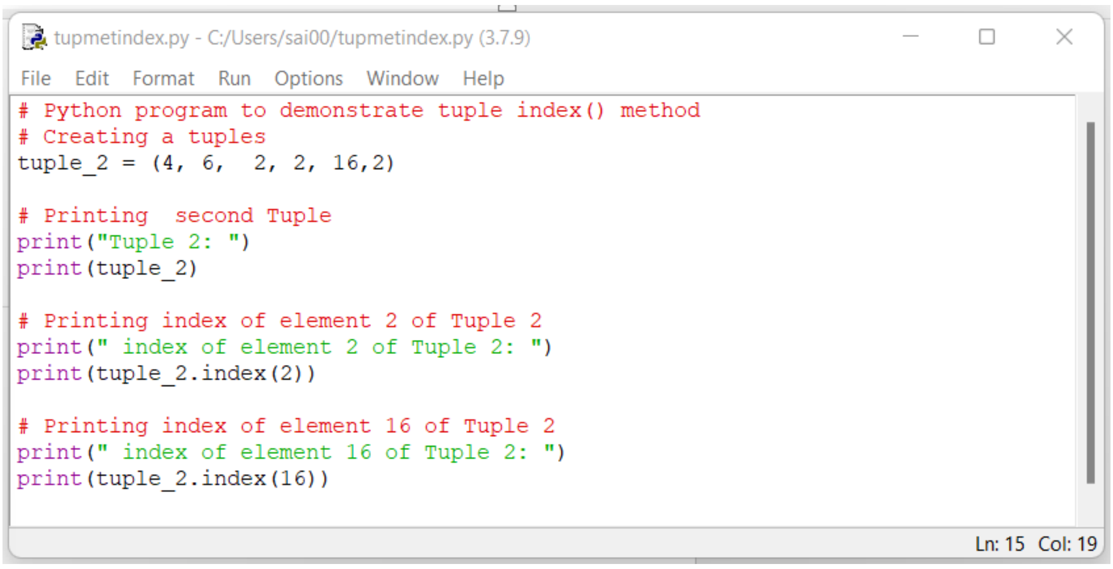
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/tupmetcount.py ====================
Tuple 2:
(4, 6, 2, 2, 16, 2)
 freq of element of Tuple 2:
3
>>>
```

In the above example, we first create a tuple tuple_2 with some elements. Then we use the count () method to count the number of occurrences of the value 2 in the tuple. The method returns the count of 2 which is 3. Finally, we print the count.

8.6.2. Index () Method

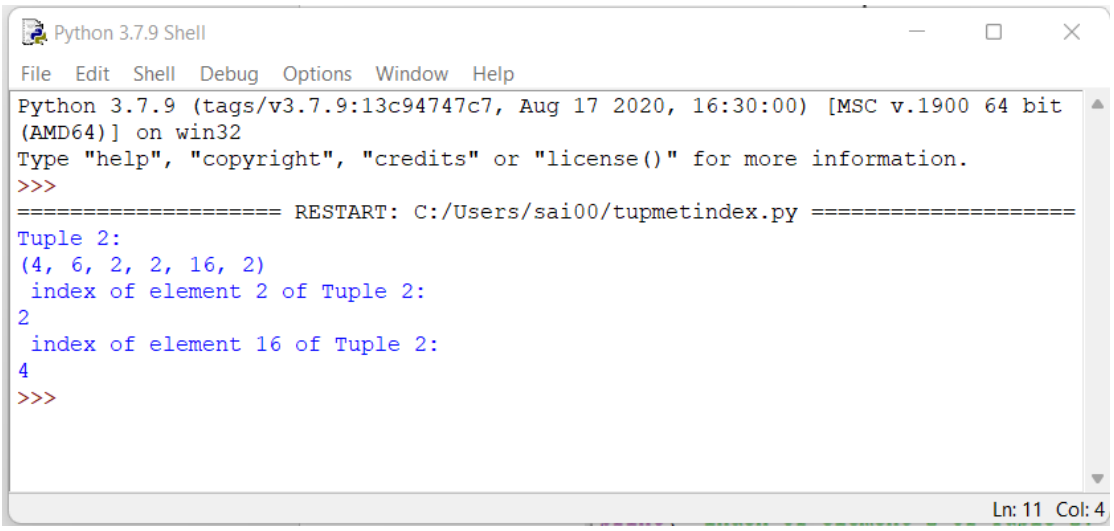
A built-in Python function called index () can be used to determine the index of a given element's first instance in a tuple. The value to be searched in the tuple is the only input required by the method.

Example:


tupmetindex.py - C:/Users/sai00/tupmetindex.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 15 Col: 19

```
# Python program to demonstrate tuple index() method
# Creating a tuples
tuple_2 = (4, 6,  2, 2, 16,2)

# Printing  second Tuple
print("Tuple 2: ")
print(tuple_2)

# Printing index of element 2 of Tuple 2
print(" index of element 2 of Tuple 2: ")
print(tuple_2.index(2))

# Printing index of element 16 of Tuple 2
print(" index of element 16 of Tuple 2: ")
print(tuple_2.index(16))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/tupmetindex.py ====================
Tuple 2:
(4, 6, 2, 2, 16, 2)
 index of element 2 of Tuple 2:
2
 index of element 16 of Tuple 2:
4
>>>
```

In the preceding example, we first create a tuple, tuple_2, with certain elements. Then we use the index () method to discover the index of the tuple's first occurrence of the value 2. The method returns the index of the first occurrence of 2 (which is 1). Finally, we will print the index.Tuples are widely used in Python for a variety of purposes, including returning multiple values from a function, representing fixed groupings of data, and serving as keys in dictionaries.

The methods discussed above make it simple to interact with tuples in Python, allowing you to extract and change their contents. The count () function in Python is useful for determining the number of repetitions of a certain element in a tuple. The index () function in Python is useful for determining the index of the first occurrence of a certain element in a tuple.

8.7 SUMMARY

Tuples enable integer-based indexing and duplicate elements, which improves data organization and retrieval. They can be defined with or without parentheses; however, without parentheses, a following comma is required to represent a tuple. Tuples are best used for their original purpose; misapplication can result in inefficiencies, such as substituting lists, sets, or dictionaries. To ensure efficient data processing and manipulation, choose the suitable data structure after carefully considering the use cases.

8.8 TECHNICAL TERMS

Tuple, Indexing, Negative Indexing, Max, Min , Count,.Index, and Slicing

8.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. How is a tuple created and called? Explain.
2. What are the various tuple methods? Explain.
3. Explain about tuple functions with example.

Short Notes:

1. Write about indexing access method.
2. How Tuple is different form the List and Dictionary.
3. Explain about membership operator in tuple.

8.10 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. "Python Pocket Reference" by Mark Lutz
4. "Python Essential Reference" by David Beazley
5. "Python Programming: An Introduction to Computer Science" by John Zelle
6. "Introduction to Computation and Programming Using Python" by John Guttag

Mrs. A. SARVANI

LESSON- 09

PYTHON DICTIONARY

AIMS AND OBJECTIVES

The main aim of this chapter is understanding the concept of dictionary in Python Programming. The discussion related to understand what dictionary and its characteristics. After completion of this chapter, student will be able to know what dictionary, how it is different from other data types. Also able to know operations, functions, and methods in dictionary.

STRUCTURE

9.1. INTRODUCTION

Python, a programming language, is equipped with a wide variety of tools and functions. The dictionary is one example of such a feature. In the Python programming language, a dictionary is a collection of key-value pairs. Uniqueness is required for the dictionary keys. A value of any kind could be assigned to the dictionary. Python's dictionary is a data structure that makes it possible for us to develop code that is both simple and very effective. The fact that the keys of this data structure can be hashed is the reason why it is referred to as a hash table in many different languages. In a moment, we will comprehend the significance of this.

Using Python dictionaries, we can easily obtain a value that has been associated with a specific key and then immediately access that value. It is recommended that we make use of them if we are looking for a certain Python object, also known as a lookup method.

9.2 PYTHON DICTIONARY

A dictionary in Python is a set of objects that let's us store information in key-value pairs. With Python dictionaries, we may rapidly obtain a value by associating it with a distinct key. Using them whenever we need to locate (search for) a certain Python object is a good concept. For this purpose, lists can also be used, but they operate far more slowly than dictionaries.

9.2.1. The Characteristic of Dictionary

- In the first place, the dictionary will have information in the form of key-value pairs.
- A colon ":" sign is used to visually define the key and the values.
- The representation of an item can consist of a single key-value pair.
- It is not permitted to have duplicate keys.
- It is possible to acknowledge duplicate values.
- It is quite OK to use heterogeneous objects for both keys and values.
- The order of the insertion is not maintained.
- a dictionary object that is capable of being altered.
- Dictionary entries behave in a dynamic manner.
- The notions of indexing and slicing are not applicable in this situation.

9.2.2. Creating Python Dictionary

In python, a dictionary is created using the key:value pairs using the curly brackets {} and is separated by commas. The syntax for creating dictionary is shown below:

Syntax:

```
my_dict = {
                "key1":"value1",
                "key2":"value2"
            }
```

In the above syntax my_dict is a dictionary created with two pair of items differentiated with different keys and values.

Example:

```
# creating a dictionary
        country_capitals = {
                        "Germany": "Berlin",
                        "Canada": "Ottawa",
                        "England": "London"

                       }
```

In the above example country_capitals is a dictionary created with three pair of items which includes {"Germany": "Berlin"}, {"Canada": "Ottawa"} and { “England": "London"}.

9.3. ACCESSING DICTIONARY ELEMENTS

To access an element from the dictionary there are three ways and are described below:

- Access by Key
- Access by get () function
- Access of nested dictionary

9.3.1 Access Dictionary by Key

We can access the value of a dictionary item by placing the key inside square brackets. It accesses and prints the values associated with the keys. The keys and values showcasing can be of different data types (string and integer).

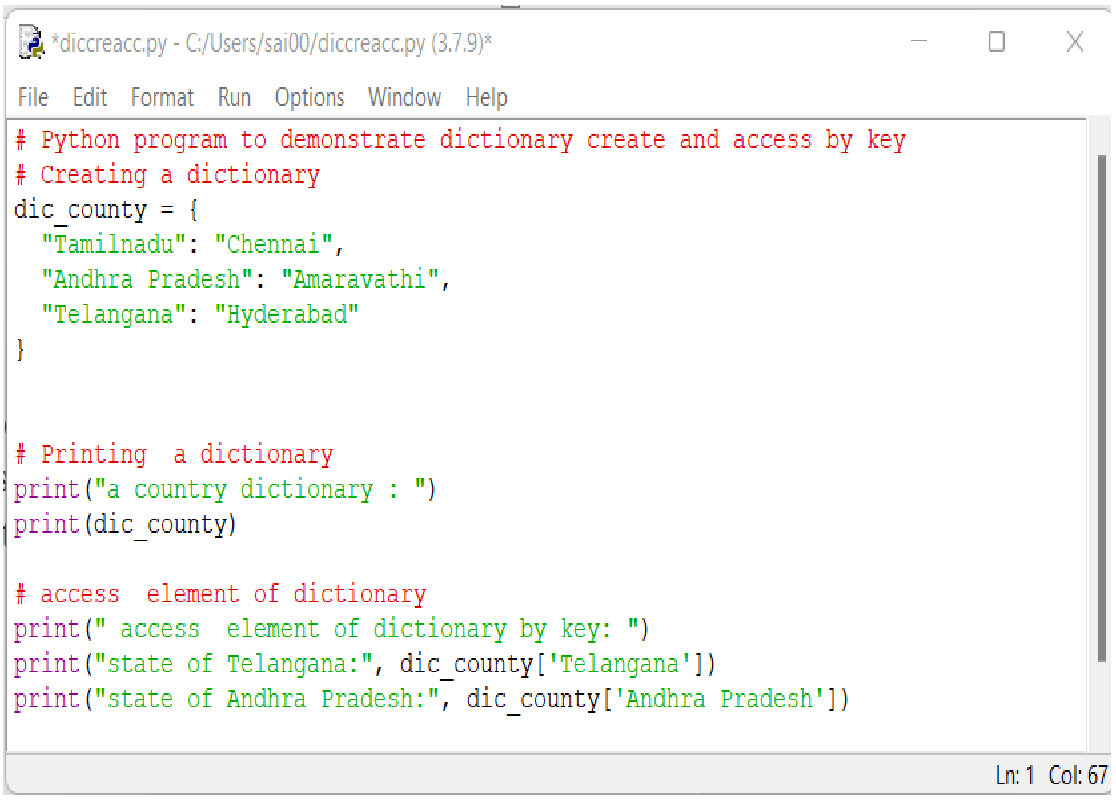
Syntax:

Value= dictionary_name[‘Key’]

Example:

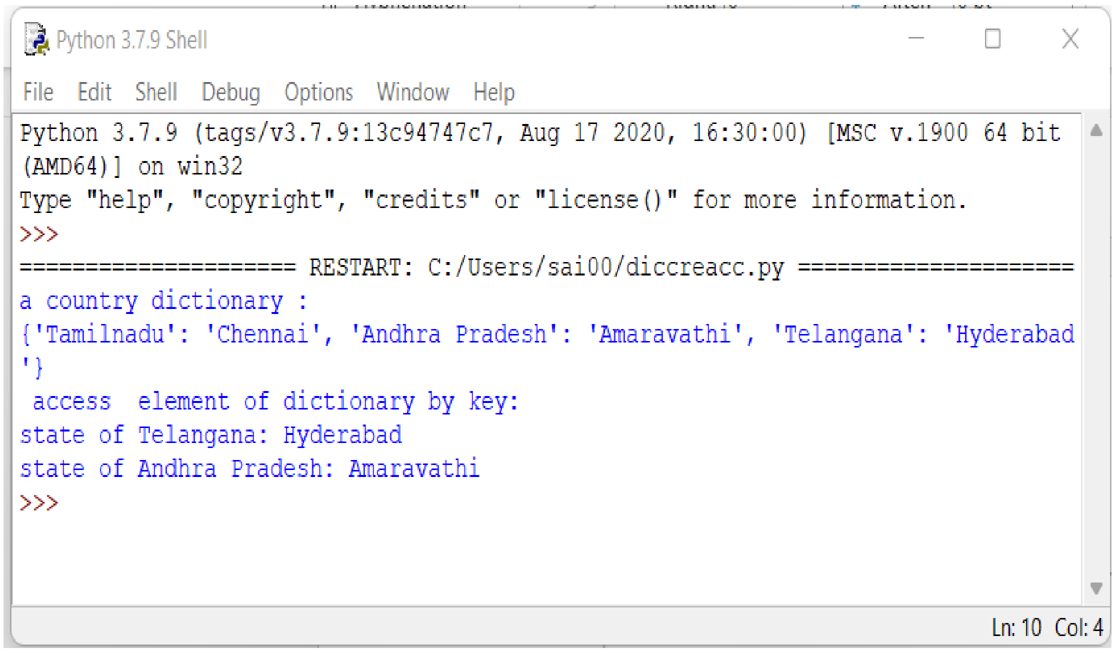
State= dic_county [‘Andhra Pradesh’]

Example:


diccreacc.py - C:/Users/sai00/diccreacc.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 1 Col: 67

```
# Python program to demonstrate dictionary create and access by key
# Creating a dictionary
dic_county = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad"
}


# Printing  a dictionary
print("a country dictionary : ")
print(dic_county)

# access  element of dictionary
print(" access  element of dictionary by key: ")
print("state of Telangana:", dic_county['Telangana'])
print("state of Andhra Pradesh:", dic_county['Andhra Pradesh'])
```

Output:

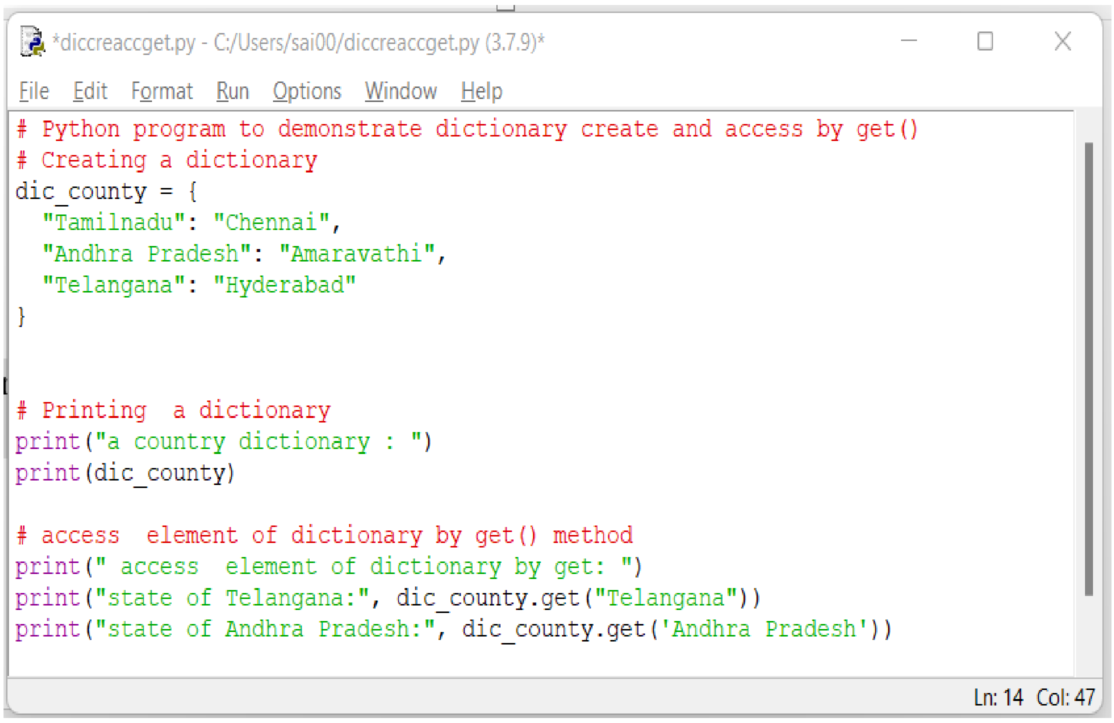


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/diccreacc.py ====================
a country dictionary :
{'Tamilnadu': 'Chennai', 'Andhra Pradesh': 'Amaravathi', 'Telangana': 'Hyderabad
'}
 access  element of dictionary by key:
state of Telangana: Hyderabad
state of Andhra Pradesh: Amaravathi
>>>
```

9.3.2 Access dictionary by get() method

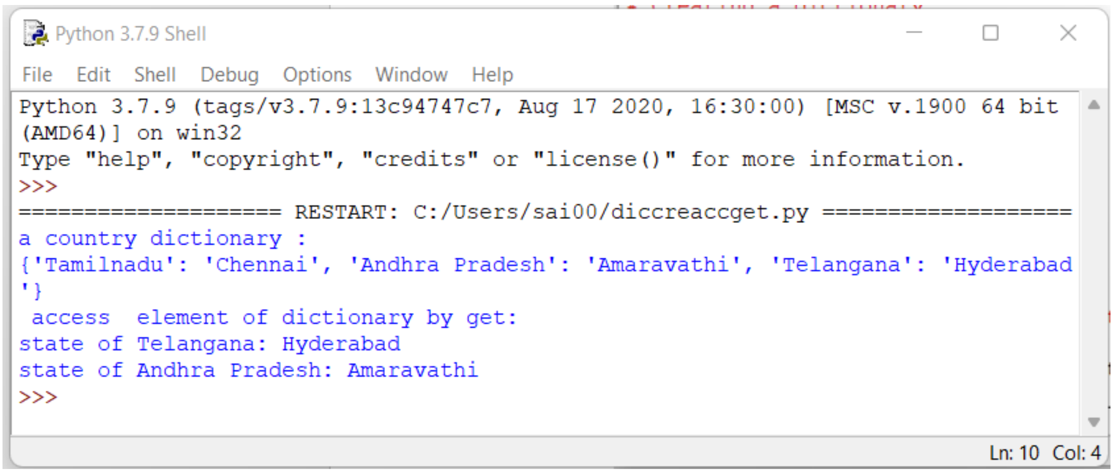
The code demonstrates accessing a dictionary element using the get() method. It retrieves and prints the value associated with the key 3 in the dictionary ‘Dict’. This method provides a safe way to access dictionary values, avoiding KeyError if the key doesn’t exist.

Example:


diccreaccget.py - C:/Users/sai00/diccreaccget.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 14 Col: 47

```
# Python program to demonstrate dictionary create and access by get()
# Creating a dictionary
dic_county = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad"
}


# Printing  a dictionary
print("a country dictionary : ")
print(dic_county)

# access  element of dictionary by get() method
print(" access  element of dictionary by get: ")
print("state of Telangana:", dic_county.get("Telangana"))
print("state of Andhra Pradesh:", dic_county.get('Andhra Pradesh'))
```

Output:

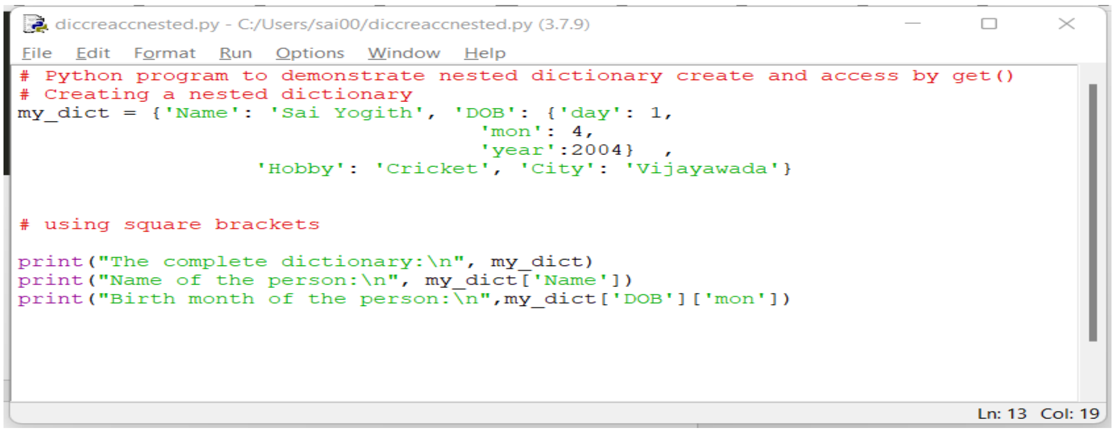


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/diccreaccget.py ==================
a country dictionary :
{'Tamilnadu': 'Chennai', 'Andhra Pradesh': 'Amaravathi', 'Telangana': 'Hyderabad
'}
 access  element of dictionary by get:
state of Telangana: Hyderabad
state of Andhra Pradesh: Amaravathi
>>>
```

9.3.3 Access of Nested Dictionary

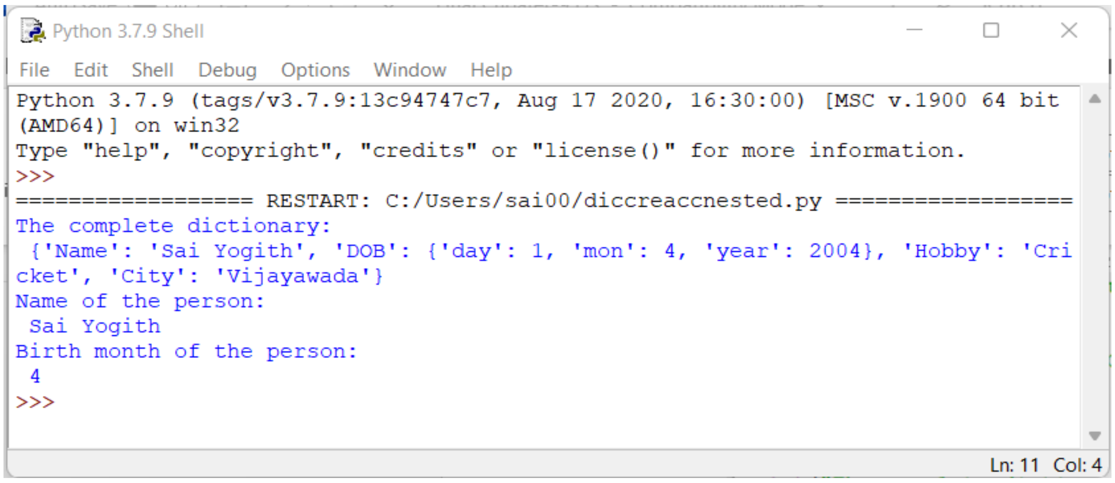
To access the value of any key in the nested dictionary, use indexing [] syntax. It first accesses main dictionary associated with the key and then, it accesses a specific value by navigating through the nested dictionaries.

Example:


diccreaccnested.py - C:/Users/sai00/diccreaccnested.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 13 Col: 19

```
# Python program to demonstrate nested dictionary create and access by get()
# Creating a nested dictionary
my_dict = {'Name': 'Sai Yogith', 'DOB': {'day': 1,
                                         'mon': 4,
                                         'year':2004}  ,
                     'Hobby': 'Cricket', 'City': 'Vijayawada'}


# using square brackets

print("The complete dictionary:\n", my_dict)
print("Name of the person:\n", my_dict['Name'])
print("Birth month of the person:\n",my_dict['DOB']['mon'])
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/diccreaccnested.py ==================
The complete dictionary:
 {'Name': 'Sai Yogith', 'DOB': {'day': 1, 'mon': 4, 'year': 2004}, 'Hobby': 'Cri
cket', 'City': 'Vijayawada'}
Name of the person:
 Sai Yogith
Birth month of the person:
 4
>>>
```

9.4 DICTIONARY METHODS

Dictionary methods are used to perform specific functionality over dictionary that may be updating, adding, extracting a, removing and etc operations on keys and items. Some of the functions includes in given Table 9.1.

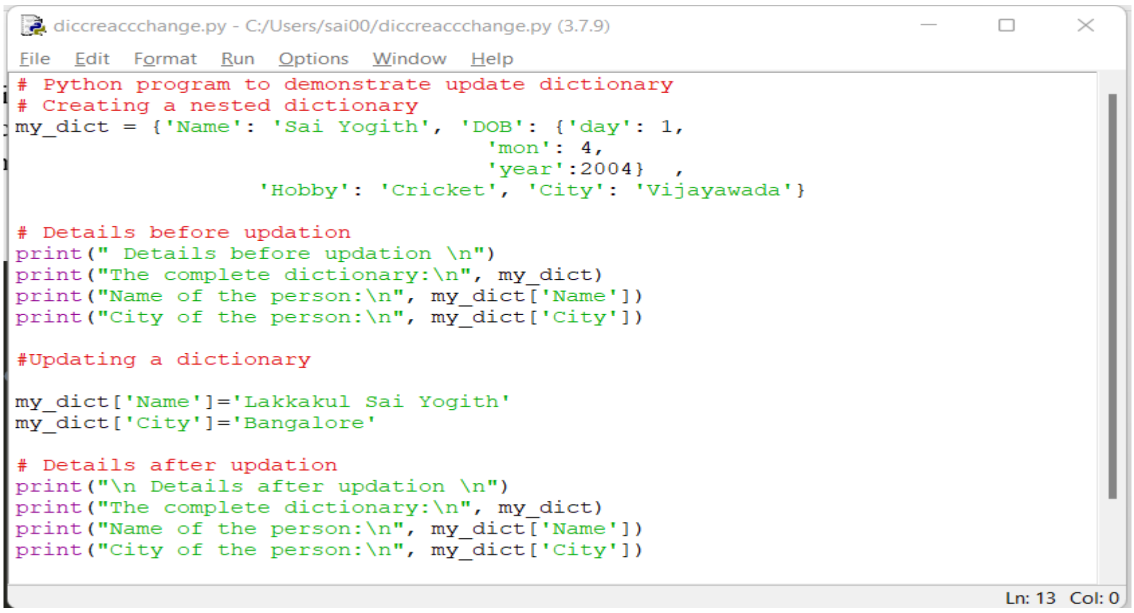
Table 9.1. Dictionary Methods

Function	Description
pop()	Removes the item with the specified key.
update()	Adds or changes dictionary items.
clear()	Remove all the items from the dictionary.
keys()	Returns all the dictionary's keys.
values()	Returns all the dictionary's values.
get()	Returns the value of the specified key.
popitem()	Returns the last inserted key and value as a tuple.
copy()	Returns a copy of the dictionary.

9.4.1. Update Elements Methods

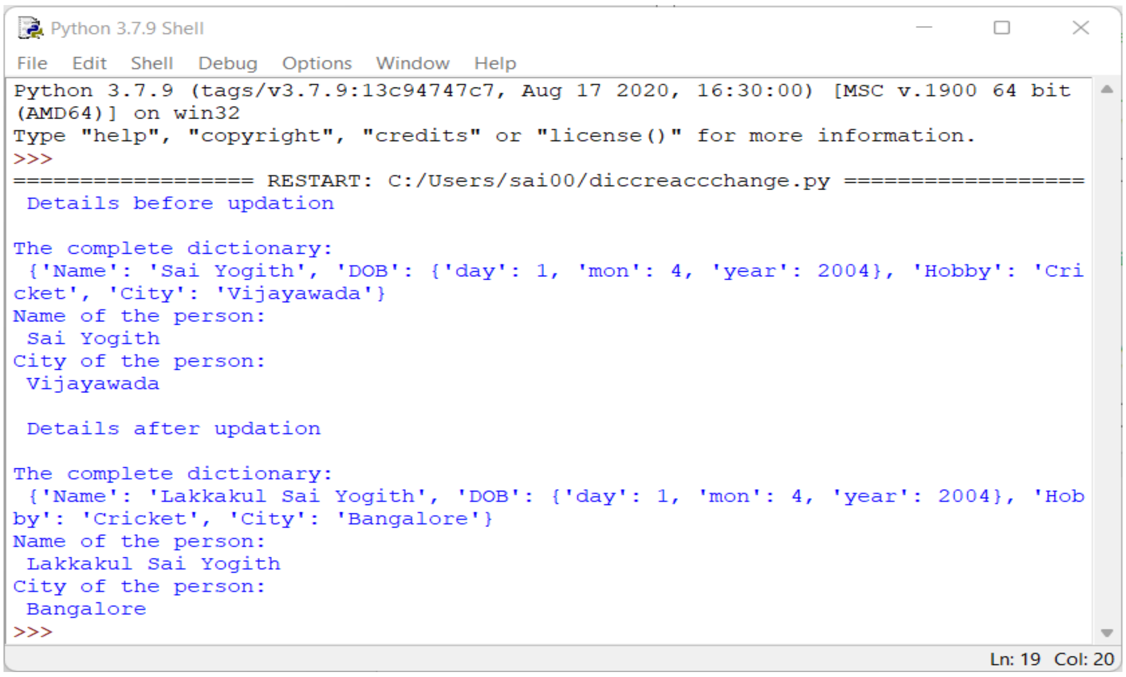
Dictionaries are subject to change. Using an assignment operator, we can add new things or change the value of existing items.

Example:


diccreaccchange.py - C:/Users/sai00/diccreaccchange.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 13 Col: 0

```
# Python program to demonstrate update dictionary
# Creating a nested dictionary
my_dict = {'Name': 'Sai Yogith', 'DOB': {'day': 1,
                                  'mon': 4,
                                  'year':2004}  ,
                   'Hobby': 'Cricket', 'City': 'Vijayawada'}

# Details before updation
print(" Details before updation \n")
print("The complete dictionary:\n", my_dict)
print("Name of the person:\n", my_dict['Name'])
print("City of the person:\n", my_dict['City'])

#Updating a dictionary

my_dict['Name']='Lakkakul Sai Yogith'
my_dict['City']='Bangalore'

# Details after updation
print("\n Details after updation \n")
print("The complete dictionary:\n", my_dict)
print("Name of the person:\n", my_dict['Name'])
print("City of the person:\n", my_dict['City'])
```

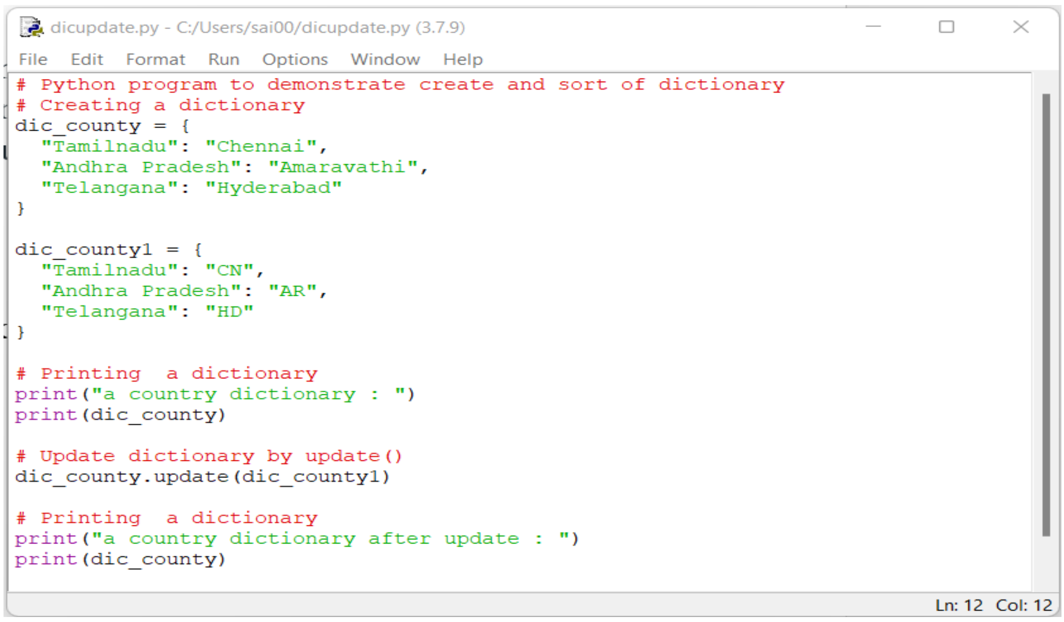
Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/diccreaccchange.py ==================
 Details before updation

The complete dictionary:
 {'Name': 'Sai Yogith', 'DOB': {'day': 1, 'mon': 4, 'year': 2004}, 'Hobby': 'Cri
cket', 'City': 'Vijayawada'}
Name of the person:
 Sai Yogith
City of the person:
 Vijayawada

 Details after updation

The complete dictionary:
 {'Name': 'Lakkakul Sai Yogith', 'DOB': {'day': 1, 'mon': 4, 'year': 2004}, 'Hob
by': 'Cricket', 'City': 'Bangalore'}
Name of the person:
 Lakkakul Sai Yogith
City of the person:
 Bangalore
>>>
```

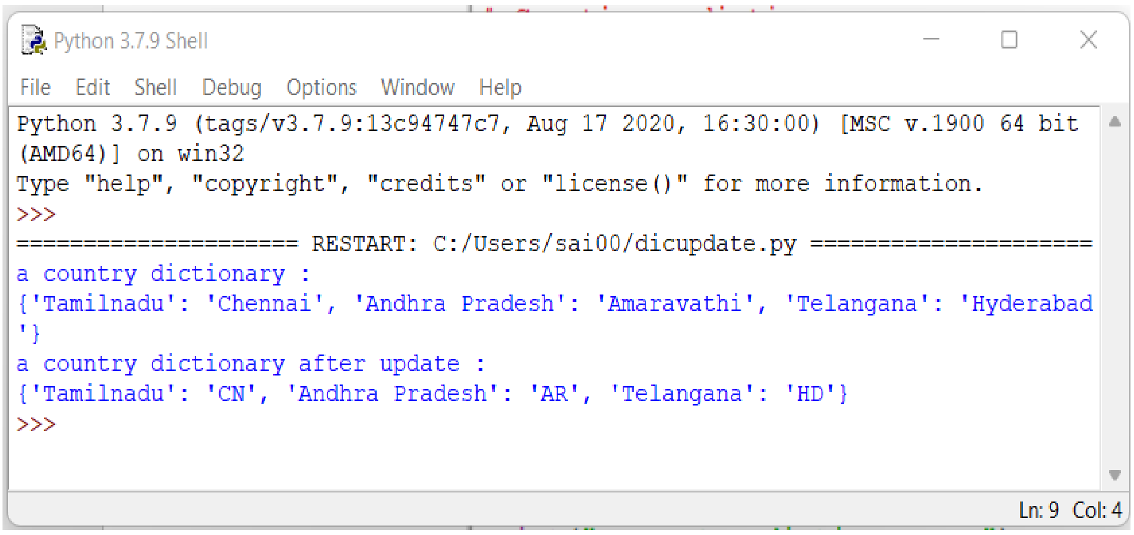
By declaring value together with the key, for example, Dict[Key] = ‘Value’, one value at a time can be added to a Dictionary. Another approach is to use Python’s update () function. Python’s update () method is a built-in dictionary function that updates the key-value pairs of a dictionary using elements from another dictionary or an iterable of key-value pairs. With this method, you can include new data or merge it with existing dictionary entries.

Example:


dicupdate.py - C:/Users/sai00/dicupdate.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 12 Col: 12

```python
# Python program to demonstrate create and sort of dictionary
# Creating a dictionary
dic_county = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad"
}

dic_county1 = {
  "Tamilnadu": "CN",
  "Andhra Pradesh": "AR",
  "Telangana": "HD"
}

# Printing  a dictionary
print("a country dictionary : ")
print(dic_county)

# Update dictionary by update()
dic_county.update(dic_county1)

# Printing  a dictionary
print("a country dictionary after update : ")
print(dic_county)
```

Output:



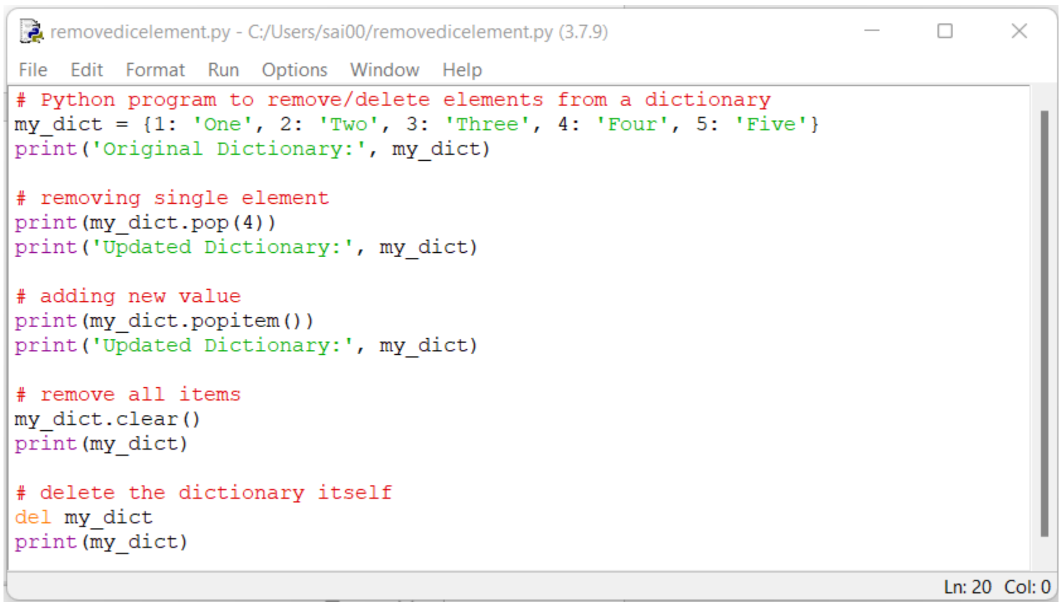
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/dicupdate.py ====================
a country dictionary :
{'Tamilnadu': 'Chennai', 'Andhra Pradesh': 'Amaravathi', 'Telangana': 'Hyderabad
'}
a country dictionary after update :
{'Tamilnadu': 'CN', 'Andhra Pradesh': 'AR', 'Telangana': 'HD'}
>>>
```

9.4.2. Removing Elements Methods

A key can be removed from a dictionary in three ways: from an individual entry, from all entries, or from the entire dictionary.

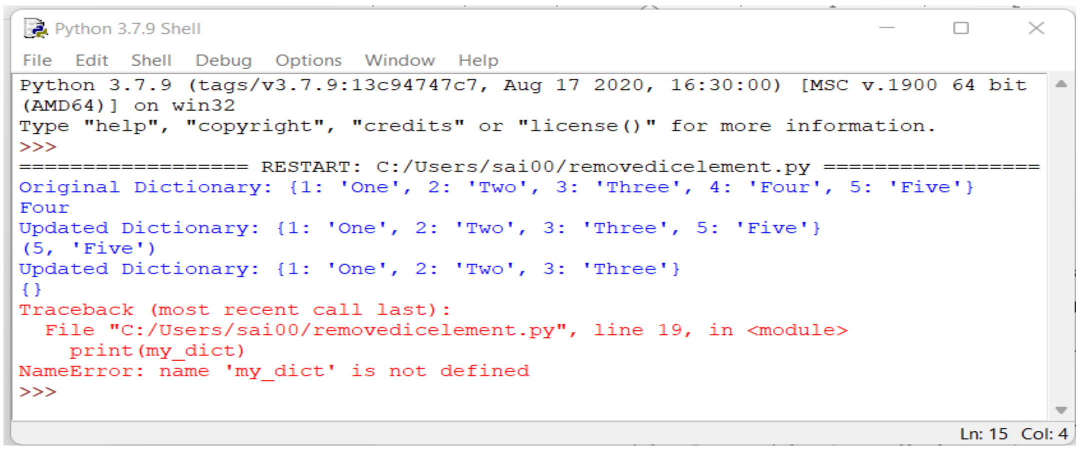
1. The pop () function can be used to remove a single element. The value of the key that has been specified to be eliminated is returned by the pop () function.
2. To randomly remove any elements (key-value pairs) of the dictionary, we can use the popitem() It returns the arbitrary key-value pair that has been removed from the dictionary.
3. Using the clear () method, all elements can be eliminated at once. The del keyword is used to completely delete the entire dictionary.

Example:


removedicelement.py - C:/Users/sai00/removedicelement.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 20 Col: 0

```python
# Python program to remove/delete elements from a dictionary
my_dict = {1: 'One', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five'}
print('Original Dictionary:', my_dict)

# removing single element
print(my_dict.pop(4))
print('Updated Dictionary:', my_dict)

# adding new value
print(my_dict.popitem())
print('Updated Dictionary:', my_dict)

# remove all items
my_dict.clear()
print(my_dict)

# delete the dictionary itself
del my_dict
print(my_dict)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/removedicelement.py =================
Original Dictionary: {1: 'One', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five'}
Four
Updated Dictionary: {1: 'One', 2: 'Two', 3: 'Three', 5: 'Five'}
(5, 'Five')
Updated Dictionary: {1: 'One', 2: 'Two', 3: 'Three'}
{}
Traceback (most recent call last):
  File "C:/Users/sai00/removedicelement.py", line 19, in <module>
    print(my_dict)
NameError: name 'my_dict' is not defined
>>>
```

9.4.3 keys () and values() Methods

In Python, the keys () function returns a view object that contains dictionary keys, which enables quick access and iteration across the dictionary.The values() method in Python returns a view object that contains all of the dictionary values. This view object can be accessed and iterated through in an effective manner within Python.

Syntax:

d = {'key': 'value'}

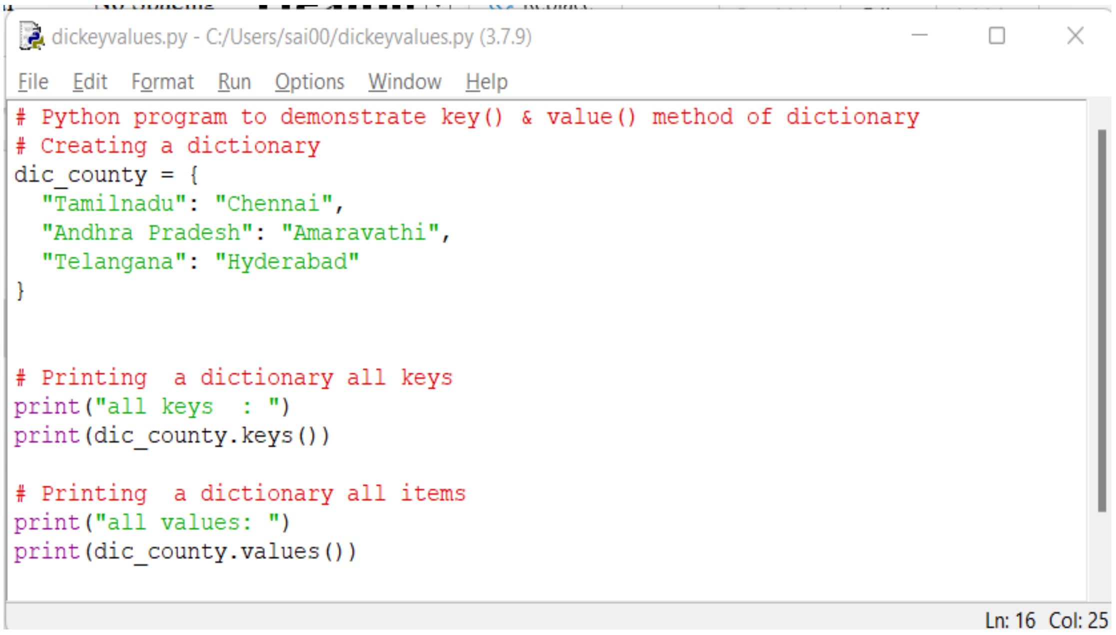
d.keys()

Syntax:

d = {'key': 'value'}

d. values ()

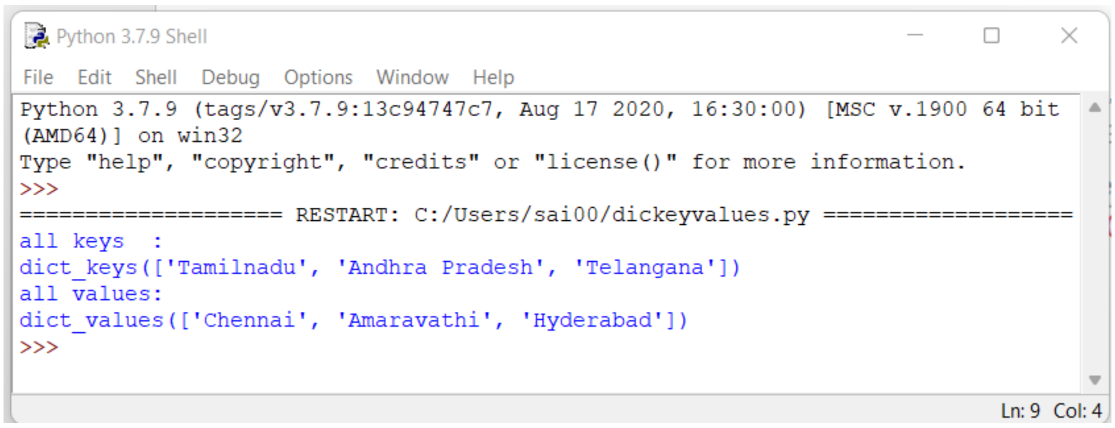
Example:


dickeyvalues.py - C:/Users/sai00/dickeyvalues.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 16 Col: 25

```
# Python program to demonstrate key() & value() method of dictionary
# Creating a dictionary
dic_county = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad"
}


# Printing  a dictionary all keys
print("all keys  : ")
print(dic_county.keys())

# Printing  a dictionary all items
print("all values: ")
print(dic_county.values())
```

In the above example , created dictionary called dic_country with three elements with the usage of keys() and values() fucntions displayed the information related every key and values associated with elements stored in dic_county dictionary. The reslut shown in output.

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/dickeyvalues.py ===================
all keys  :
dict_keys(['Tamilnadu', 'Andhra Pradesh', 'Telangana'])
all values:
dict_values(['Chennai', 'Amaravathi', 'Hyderabad'])
>>>
```

9.5 PYTHON DICTIONARY FUNCTIONS

The Python dictionary offers a wide range of methods that may be utilized to conduct operations on key-value pairs in an easy and convenient manner. The following is a list of functions using the Python dictionary shown in Table 9.1.

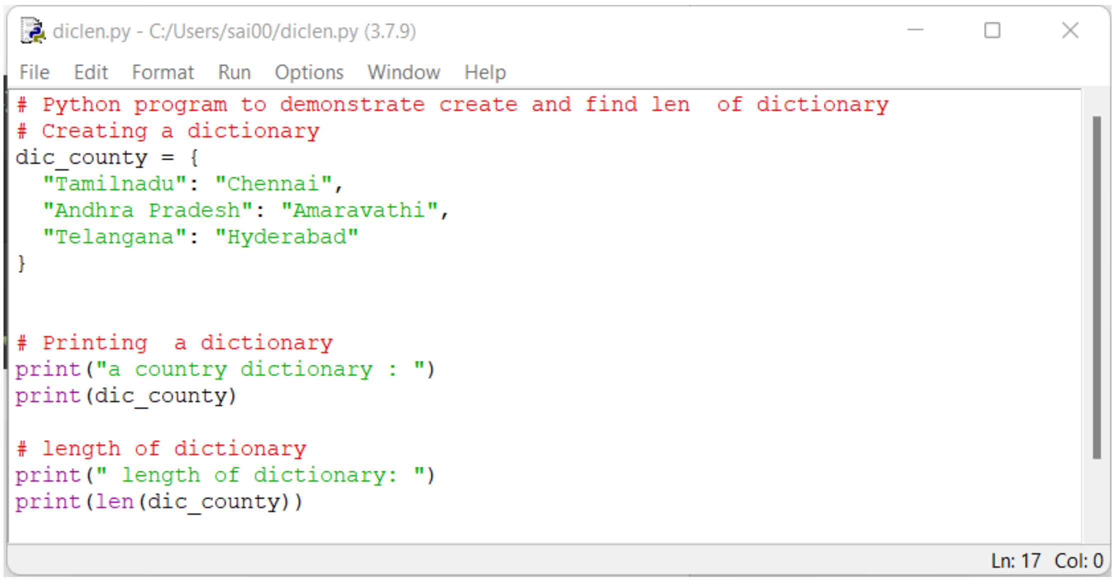
Table 9.1. Python Dictionary Functions

Function	Python Expression	Description
len()	len(my_dictionary)	Returns the length of the dictionary (key count).
sorted ()	sorted (dictionary_name)	Returns the dictionary with keys sorted in ascending order.
all ()	all (dictionary_name)	Returns True if all the keys in the dictionary are True (not 0 and False).
any ()	any(dictionary_name)	Returns True is any of the keys in the dictionary is True.
str ()	str (dictionary_name)	Returns a string representation of the dictionary passed as the argument.

9.5.1 len() function

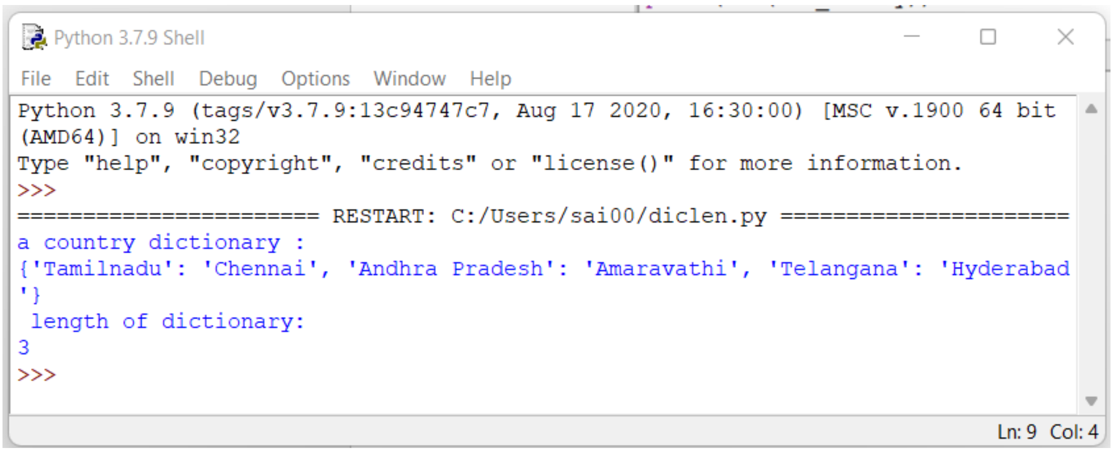
Using the len() method, which returns the item count, one can determine the length of a dictionary by its use. Printing the length of my dictionary is as follows:

Example:


diclen.py - C:/Users/sai00/diclen.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 17 Col: 0

```
# Python program to demonstrate create and find len  of dictionary
# Creating a dictionary
dic_county = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad"
}


# Printing  a dictionary
print("a country dictionary : ")
print(dic_county)

# length of dictionary
print(" length of dictionary: ")
print(len(dic_county))
```

Output:



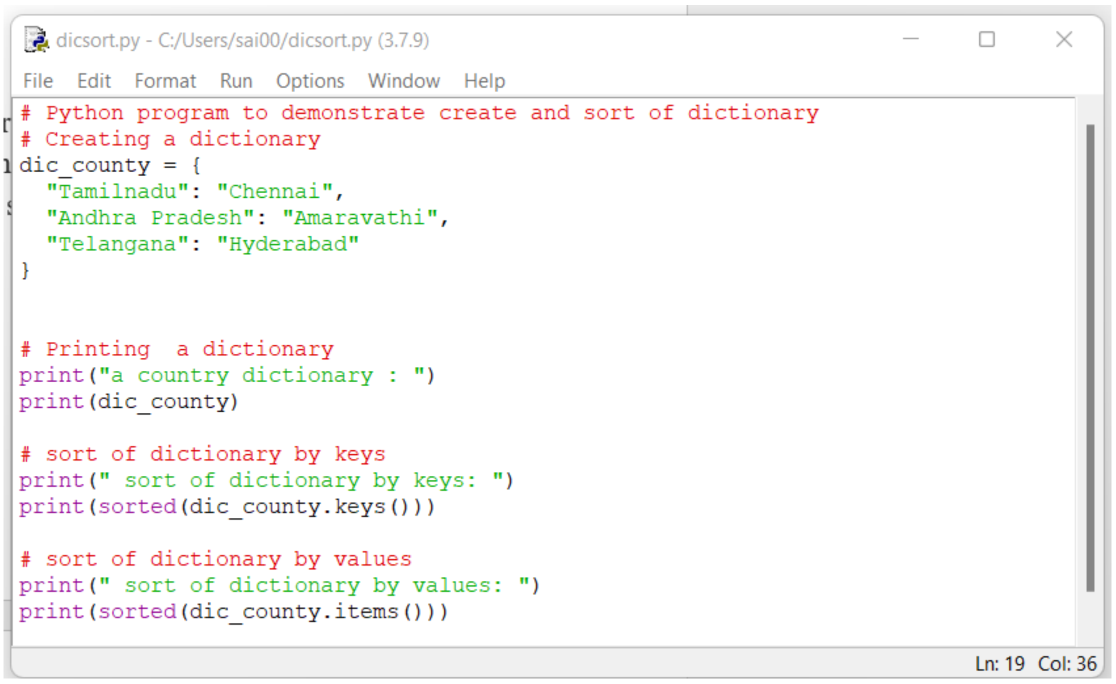
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/diclen.py =====================
a country dictionary :
{'Tamilnadu': 'Chennai', 'Andhra Pradesh': 'Amaravathi', 'Telangana': 'Hyderabad
'}
 length of dictionary:
3
>>>
```

In the above example , dic_country elements count is determined by calling the len() function and displayed lengh of the dictionary 3 it means dictinay holds the three elemtns and reslut shown in output.

9.5.2 sorted () function.

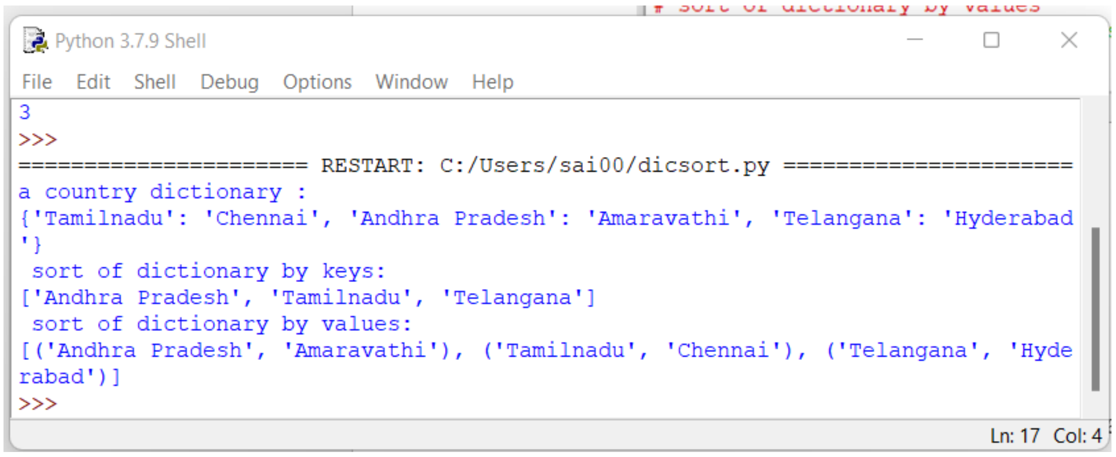
Sorting the dictionary can be accomplished with Python's built-in keys functions, which include the keys () and values () functions. Any iterable can be used as an argument, and it will return the sorted list of keys you provided. The dictionary can be arranged in ascending order by using the keys to sort the entries. First, let's get familiar with the below example.

Example:


dicsort.py - C:/Users/sai00/dicsort.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 19 Col: 36

```
# Python program to demonstrate create and sort of dictionary
# Creating a dictionary
dic_county = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad"
}


# Printing  a dictionary
print("a country dictionary : ")
print(dic_county)

# sort of dictionary by keys
print(" sort of dictionary by keys: ")
print(sorted(dic_county.keys()))

# sort of dictionary by values
print(" sort of dictionary by values: ")
print(sorted(dic_county.items()))
```

Output:



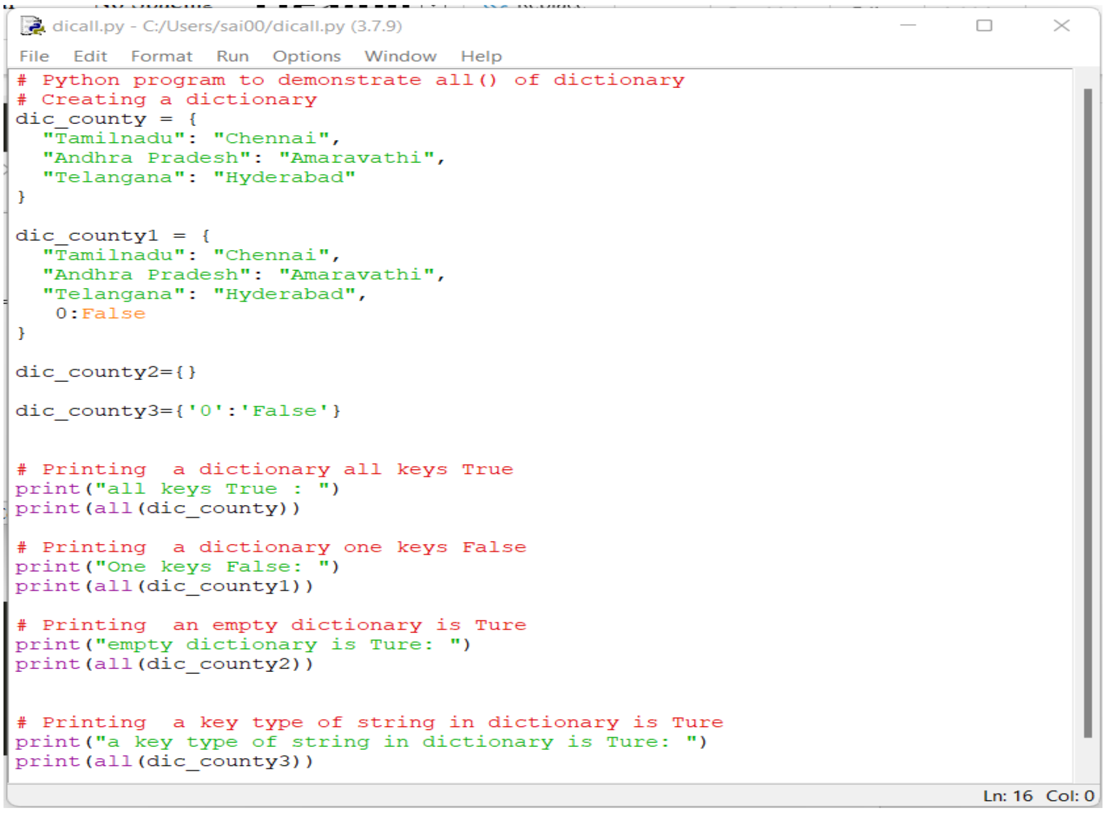
```
3
>>>
====================== RESTART: C:/Users/sai00/dicsort.py ======================
a country dictionary :
{'Tamilnadu': 'Chennai', 'Andhra Pradesh': 'Amaravathi', 'Telangana': 'Hyderabad
'}
 sort of dictionary by keys:
['Andhra Pradesh', 'Tamilnadu', 'Telangana']
 sort of dictionary by values:
[('Andhra Pradesh', 'Amaravathi'), ('Tamilnadu', 'Chennai'), ('Telangana', 'Hyde
rabad')]
>>>
```

We have declared a dictionary of names in the code that was just presented. We made use of the built-in function in conjunction with the sorted() method, which provided us with a list of the keys that had been sorted. We then proceeded to utilize the items() function in order to obtain the dictionary in the order that it was sorted.

9.5.3 all() function

A dictionary's keys are the only elements that are examined when the all() method is applied to it; the values are not examined. In the event that not all of the keys in the dictionary are true, the all() method will return FALSE; but, if all of the keys are true, it will return false. In the event that the dictionary does not consist of any entries, the all() function also returns a value of TRUE.

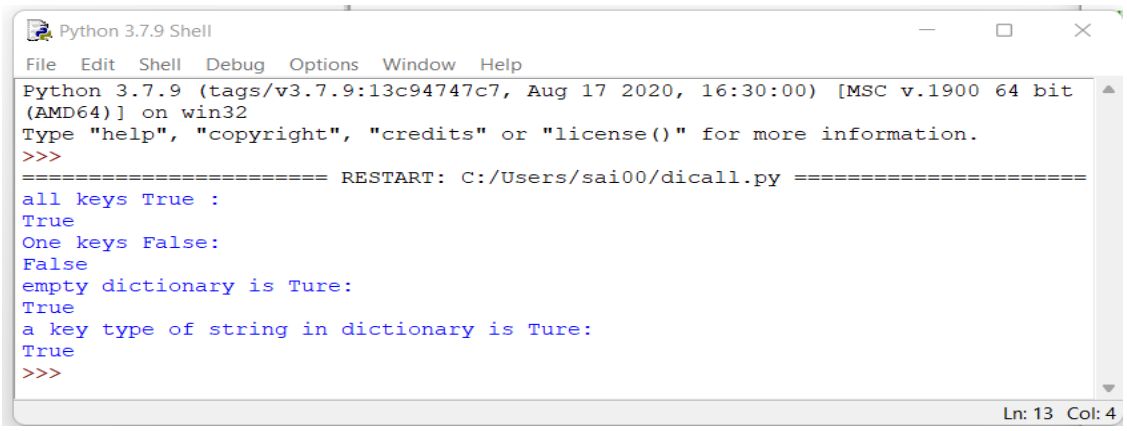
Example:



dicall.py - C:/Users/sai00/dicall.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 16 Col: 0

```
# Python program to demonstrate all() of dictionary
# Creating a dictionary
dic_county = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad"
}

dic_county1 = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad",
   0:False
}

dic_county2={}

dic_county3={'0':'False'}


# Printing  a dictionary all keys True
print("all keys True : ")
print(all(dic_county))

# Printing  a dictionary one keys False
print("One keys False: ")
print(all(dic_county1))

# Printing  an empty dictionary is Ture
print("empty dictionary is Ture: ")
print(all(dic_county2))


# Printing  a key type of string in dictionary is Ture
print("a key type of string in dictionary is Ture: ")
print(all(dic_county3))
```

Output:

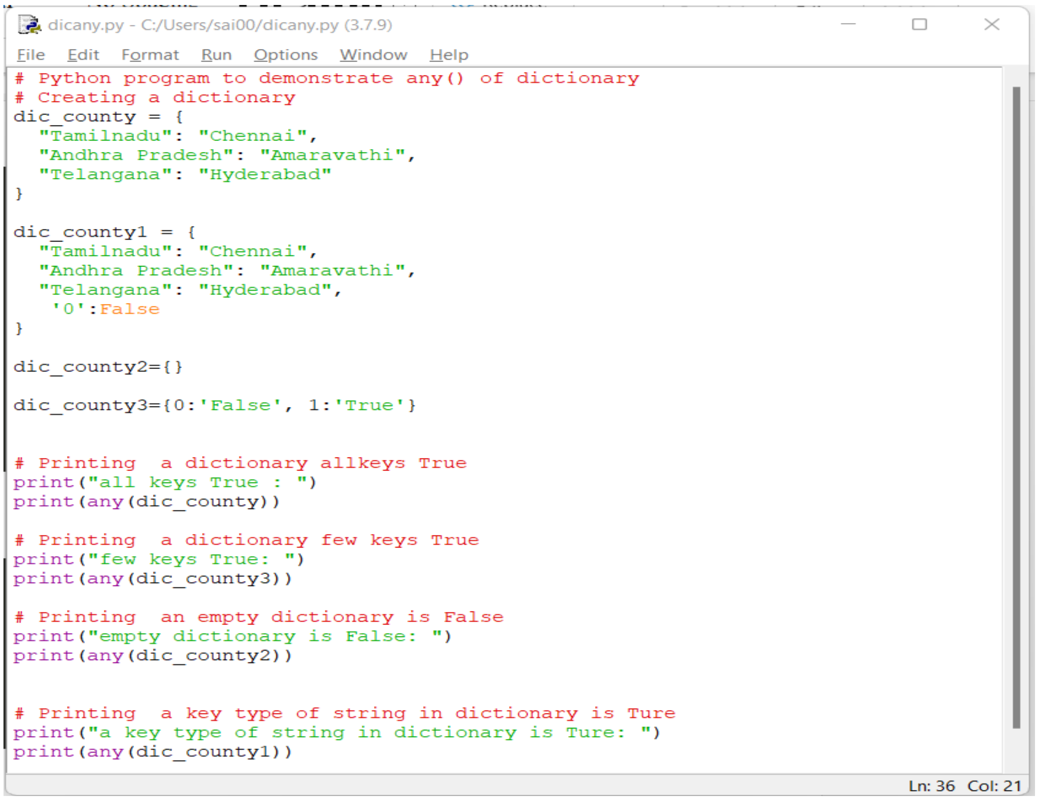


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/dicall.py ======================
all keys True :
True
One keys False:
False
empty dictionary is Ture:
True
a key type of string in dictionary is Ture:
True
>>>
```

9.5.4 any () function

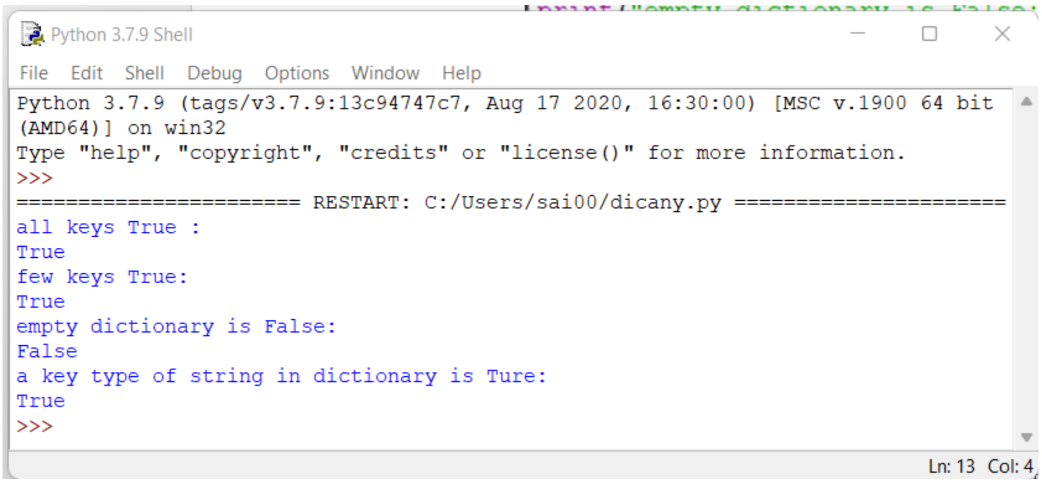
The any () method only verifies the keys of a dictionary when it is applied to a dictionary; it does not verify the values. If any of the keys associated with the dictionary are true, the any () method will return TRUE; otherwise, it will return FALSE. In the event that the dictionary does not consist of any entries, the any () function also returns FALSE.

Example:


dicany.py - C:/Users/sai00/dicany.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 36 Col: 21

```
# Python program to demonstrate any() of dictionary
# Creating a dictionary
dic_county = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad"
}

dic_county1 = {
  "Tamilnadu": "Chennai",
  "Andhra Pradesh": "Amaravathi",
  "Telangana": "Hyderabad",
   '0':False
}

dic_county2={}

dic_county3={0:'False', 1:'True'}


# Printing  a dictionary allkeys True
print("all keys True : ")
print(any(dic_county))

# Printing  a dictionary few keys True
print("few keys True: ")
print(any(dic_county3))

# Printing  an empty dictionary is False
print("empty dictionary is False: ")
print(any(dic_county2))


# Printing  a key type of string in dictionary is Ture
print("a key type of string in dictionary is Ture: ")
print(any(dic_county1))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======================= RESTART: C:/Users/sai00/dicany.py ======================
all keys True :
True
few keys True:
True
empty dictionary is False:
False
a key type of string in dictionary is Ture:
True
>>>
```

9.6 SUMMARY

Python is an excellent programming language that comes with a wide variety of feature sets. The fact that it provides a structured code makes it much simpler to comprehend. Since Python is currently one of the most widely used programming languages in the modern day, it is essential to have a comprehensive understanding of this programming language. This chapter will provide you with practical experience on how to work with dictionary along methods and functions.

9.7 TECHNICAL TERMS

Dictionary, Update, any, key, value, Get Method, Pop, Clear, and pop Items.

9.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. How is a dictionary created and called? Explain.
2. What are the various dictionary methods? Explain.
3. Explain about dictionary functions with example.

Short Notes:

1. Write about get () access method.
2. How dictionary is different form the List.

9.9 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. "Python Pocket Reference" by Mark Lutz
4. "Python Essential Reference" by David Beazley
5. "Python Programming: An Introduction to Computer Science" by John Zelle
6. "Introduction to Computation and Programming Using Python" by John Guttag

Mr. G. V. SURESH

LESSON- 10

PYTHON FUNCTION

AIMS AND OBJECTIVES

The main aim of this chapter is understanding the concept of Functions in Python Programming. The discussion related to creating new functions and working with built-n functions. After completion of this chapter, student will be able to know what function is, how different types of functions created. Also able to know use of recursive and lamba functions. The nested functions and scope of the variable with good number of examples is the objective of this chapter.

STRUCTURE

10.1. INTRODUCTION

Various teams collaborate on a single project and subsequently create distinct modules. Developers working on various modules frequently break up their code into smaller, more understandable pieces called functions to facilitate debugging. This makes it easier for developers to discover the fault in terms of function. Functions in Python can be built-in or user-defined; this chapter covers the full features.

Even if you have only been programming for a few days, you have probably already come across functions like print (), len (), and type (). All of them are integrated features. This chapter is going to demonstrate to you how to define and reuse your own functions in addition to how to use the built-in functions.

10.2. PYTHON FUNCTIONS

Python functions help to simplify and organize programming by enabling the creation of smaller code segments. As a result, code is easier for people to understand when they view it. Functions improve reusability and reduce repetition in code, which is their primary benefit.

10.2.1 Creating A Function.

When defining a function in Python, we must adhere to the following guidelines and syntax is shown in figure 10.1:

The function definition is initiated with the def keyword.

- The function name that follows the def keyword is followed by parentheses with the user-passed arguments and a colon at the end.
- The function's body begins with an indented block on a new line following the addition of the colon.
- The caller receives a result object from the return statement. Return none is the same as a return statement without an argument.

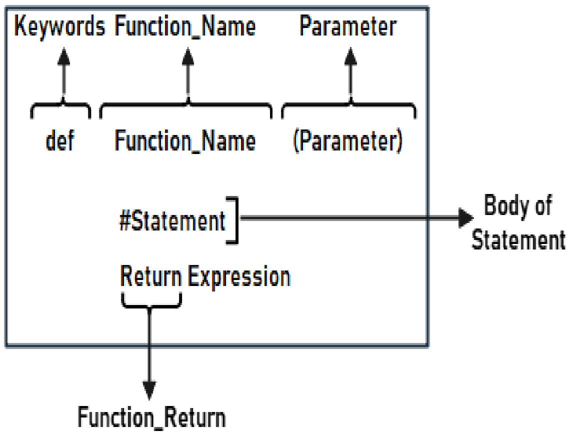


Fig10.1. Syntax for declaring function.

10.2.2 Calling a Function

After defining a function in Python, we may call it by using its name followed by parenthesis containing the function's parameters i.e, greet().When the function greet () is called, the program's control moves to the function definition. All the code within the function is executed. Following the function call, program control moves on to the following statement. The necessary steps are described in the figure 10.2 below.

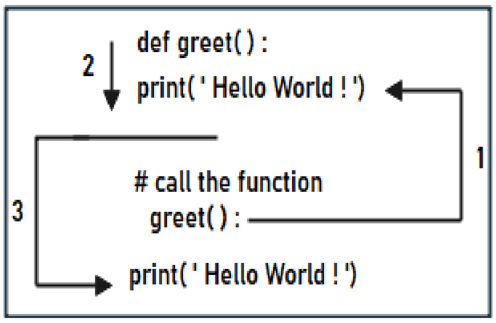


Fig 10.2. The python function working

10.2.3. Advantages of using Functions

- Python functions increase code modularity by breaking it down into smaller portions that can be solved independently, making implementation easier.
- Python functions reduce redundancy and save time rewriting code. All we must do is call the function once it has been defined.
- Defining a function in Python allows for unlimited calls, increasing code reuse.
- Using functions to separate huge programs improves readability and debugging.

10.3. TYPES OF FUNCTIONS

There are two types of functions in python which are shown in Figure 10.3:

- User-Defined Functions
- Built-in Functions

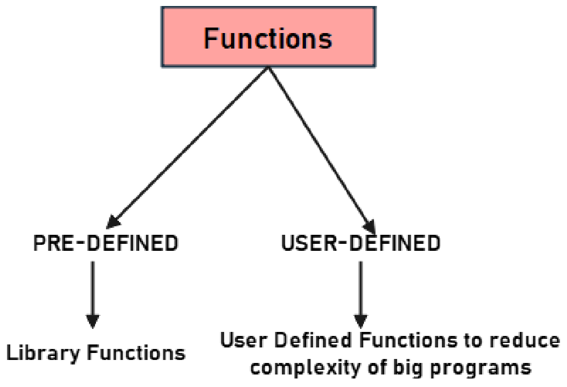
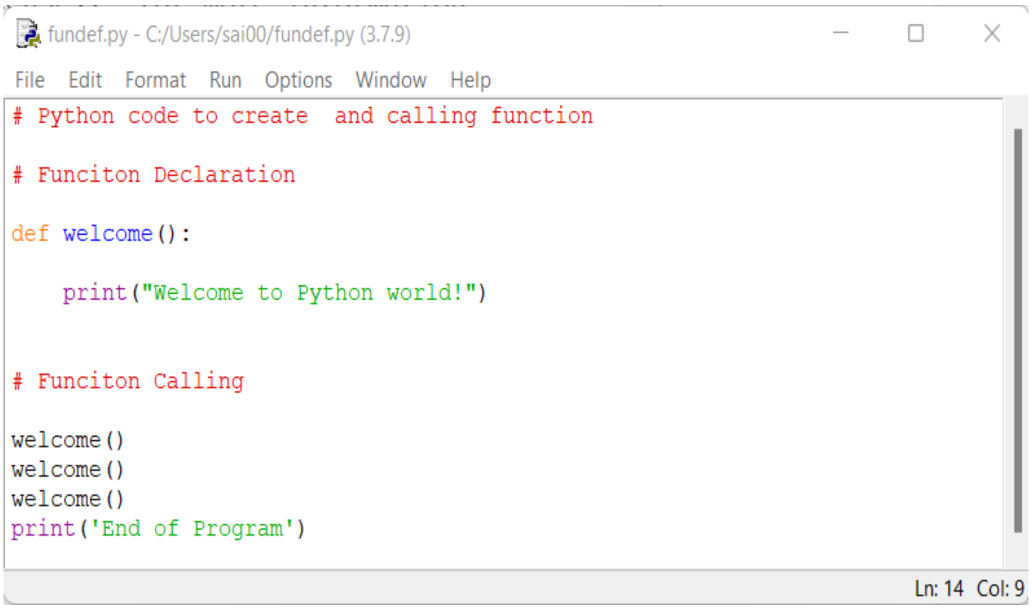


Fig10.3. Types of User-defined functions

10.3.1. User-Defined Functions

These functions are defined by the user to carry out any given task. Defining a function allows you to reuse code, making it more modular and easier to read. In Python, you can define a function with the def keyword, followed by the function name and any required parameters in parentheses. The python code that defines the function and is shown in below example.

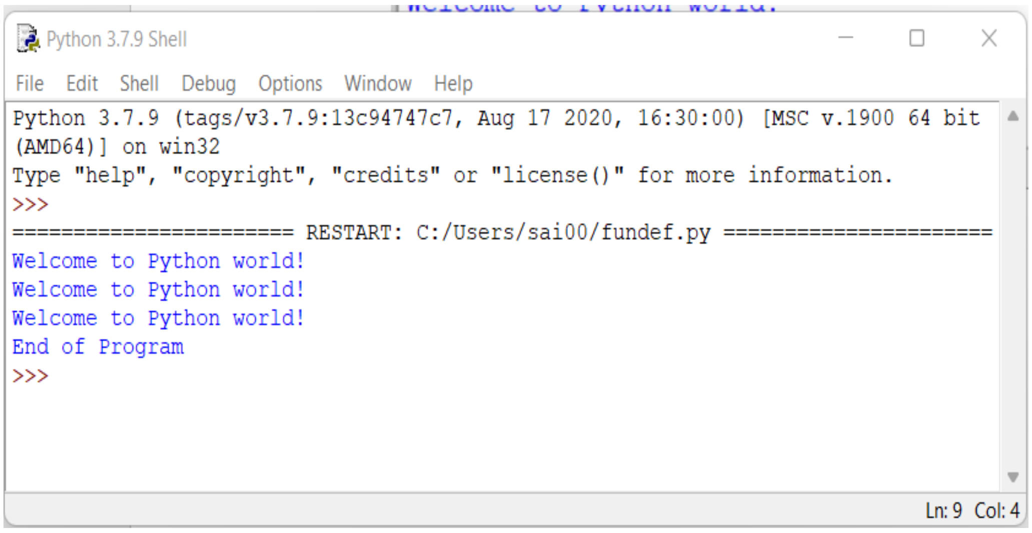
Example:


fundef.py - C:/Users/sai00/fundef.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 14 Col: 9

```
# Python code to create  and calling function

# Funciton Declaration

def welcome():

    print("Welcome to Python world!")


# Funciton Calling

welcome()
welcome()
welcome()
print('End of Program')
```

In the above example, the **welcome ()** function is completely user-defined and outputs the message **"Welcome to Python world!".** The function is only defined once but has different types. This demonstrates the value of function reusability. Similarly, we can build any function to fulfill a certain goal.

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/fundef.py ======================
Welcome to Python world!
Welcome to Python world!
Welcome to Python world!
End of Program
>>>
```

10.3.2.Built-in Functions

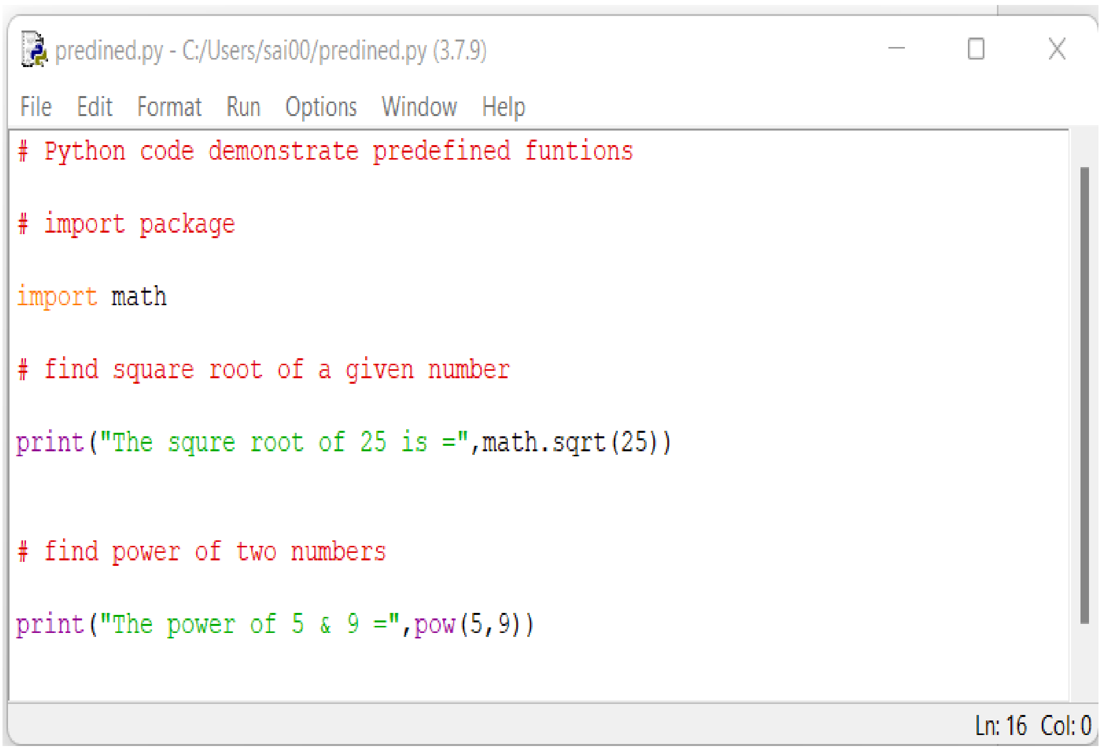
Python's built-in functions are already defined. A user must remember the name and parameters of a certain function. There is no need to redefine these functions because they have already been defined.Some of the widely used built-in functions are given below and shown in Table 10.1:

Table 10.1 Built-in Functions in Python

Function	**Description**
pow()	Returns the power of two numbers
abs()	Returns the absolute value of a number
max()	Returns the largest item in a python iterable
min()	Returns the largest item in a python iterable
sum()	Sum() in Python returns the sum of all the items in an iterator
type()	The type() in Python returns the type of a python object
Sqrt()	Executes the python built-in to find sqrt of the given number

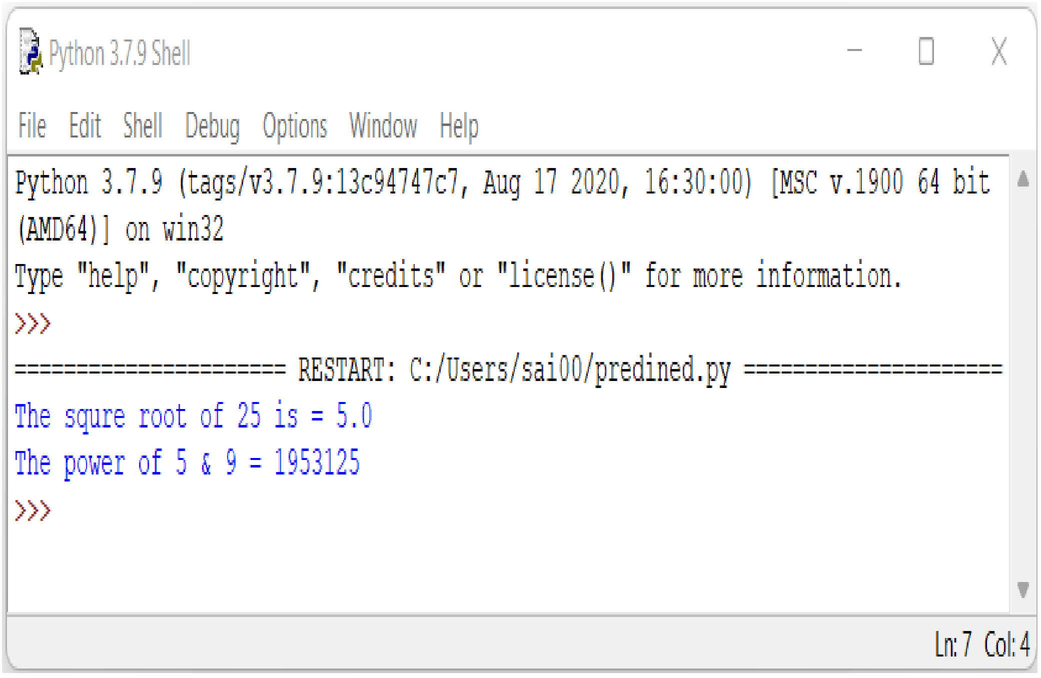
The following two example python codes shown in below demonstrate the usage of built-in functions to fulfil the specific task. In the first example python code imported math module and later performed the two functions pow () and sqrt() operations. The result of each function is produced on the output.

Example


predined.py - C:/Users/sai00/predined.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 16 Col: 0

```
# Python code demonstrate predefined funtions

# import package

import math

# find square root of a given number

print("The squre root of 25 is =",math.sqrt(25))


# find power of two numbers

print("The power of 5 & 9 =",pow(5,9))
```

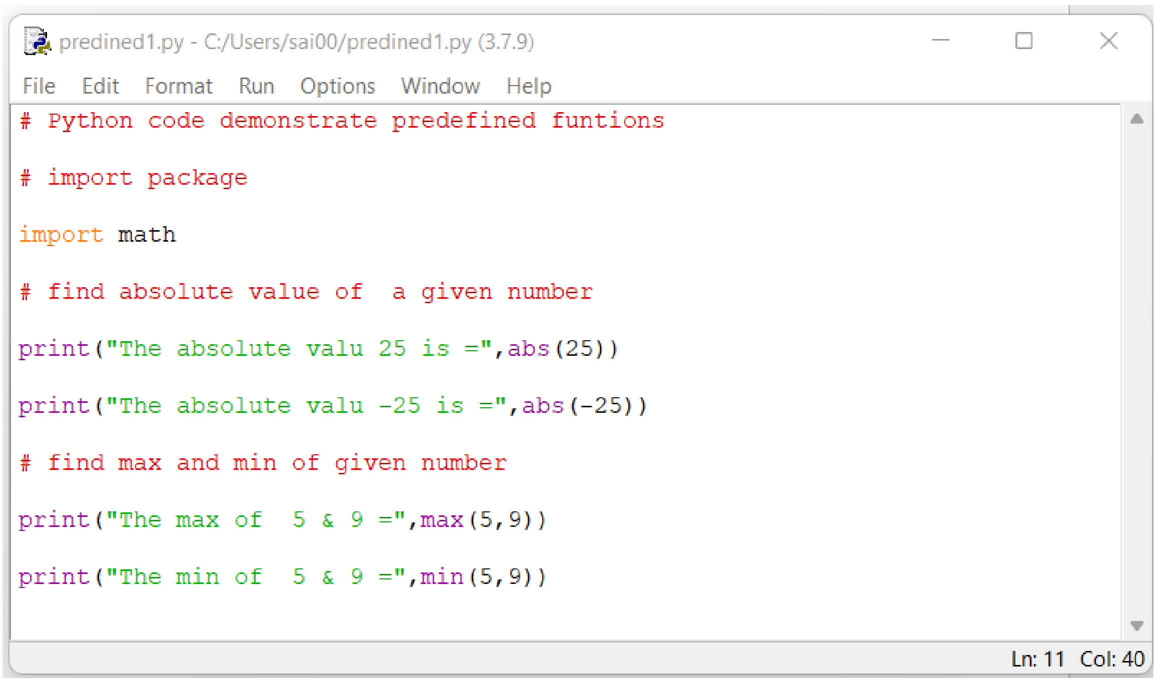
Output



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/predined.py ====================
The squre root of 25 is = 5.0
The power of 5 & 9 = 1953125
>>>
```

Similarly, the second example also imported math module and perform the abs (), max () and min () operations respectively. The absolute function took the -25 is a negative number and produced the output as 25. The maximum of 5 and 9 is determined by max () and minimum is returned by min () function.

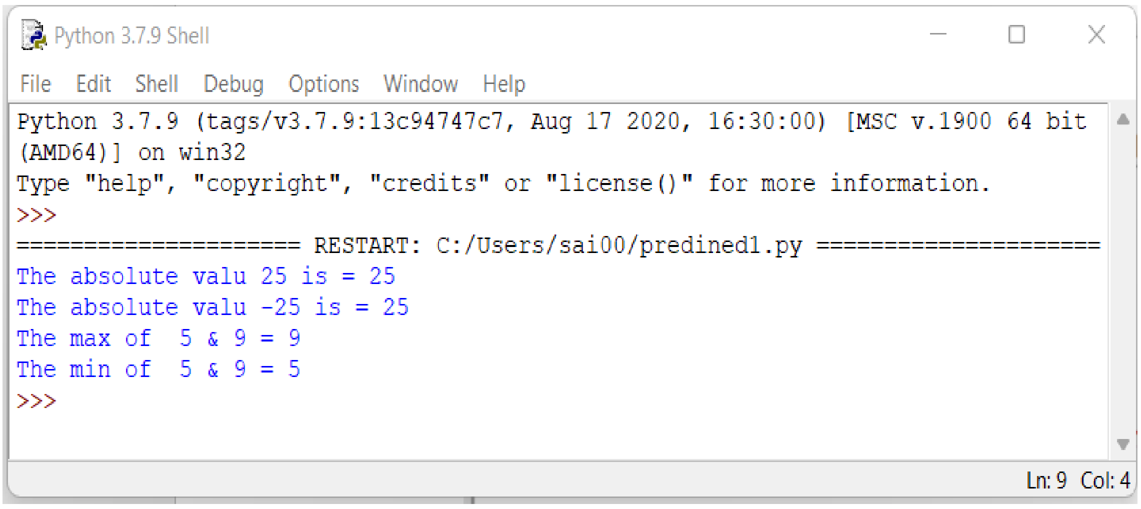
Example



predined1.py - C:/Users/sai00/predined1.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 11 Col: 40

```
# Python code demonstrate predefined funtions

# import package

import math

# find absolute value of  a given number

print("The absolute valu 25 is =",abs(25))

print("The absolute valu -25 is =",abs(-25))

# find max and min of given number

print("The max of  5 & 9 =",max(5,9))

print("The min of  5 & 9 =",min(5,9))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/Users/sai00/predined1.py =====================
The absolute valu 25 is = 25
The absolute valu -25 is = 25
The max of  5 & 9 = 9
The min of  5 & 9 = 5
>>>
```

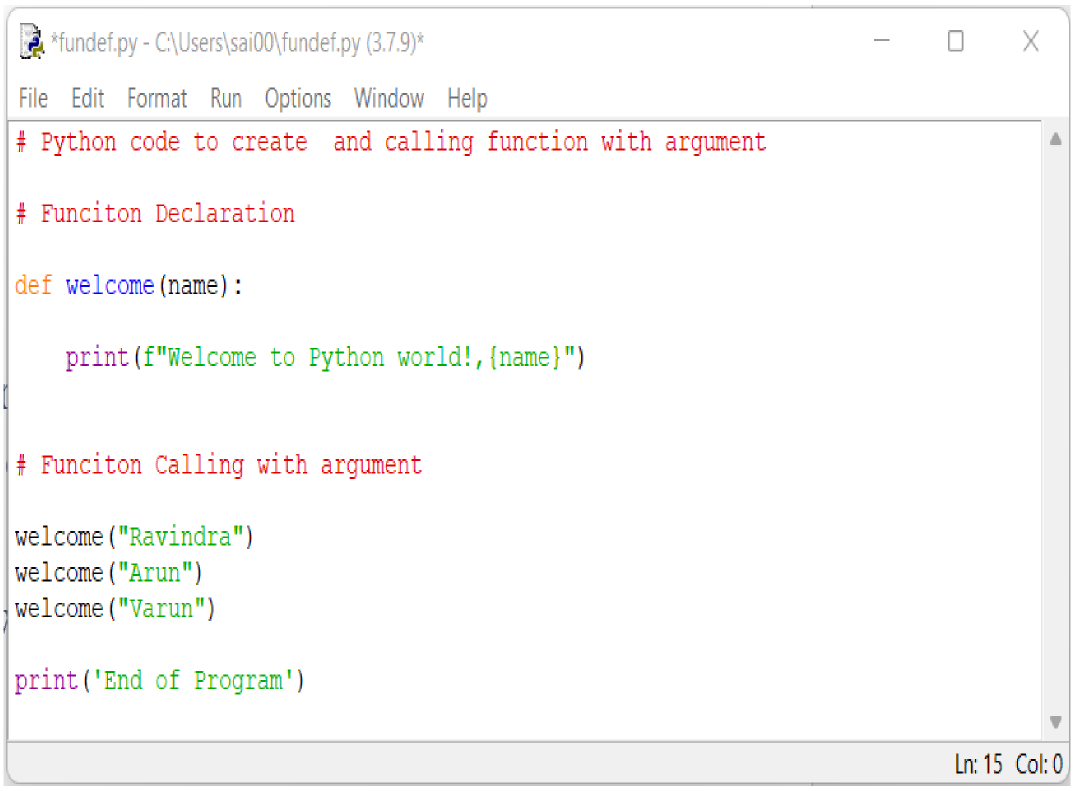
10.4. FUNCTION ARGUMENTS

The function is defined with arguments, and it given choice to enter different inputs to the function. The argument is passed as a parameter in the function definition. The parameters of different number and different types i.e., int, string, list etc.

The syntax of define function with arguments:

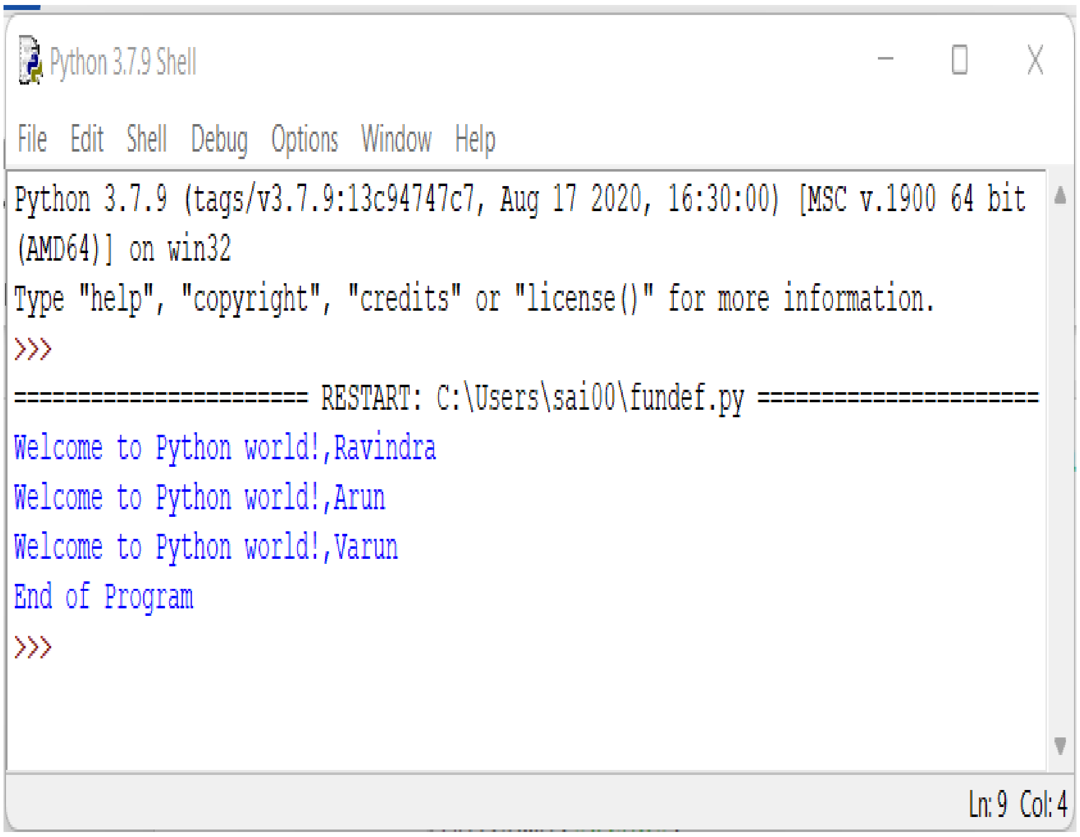
```
def fun_name (argument-1, argument-2, ……, argument-N):

                    Function body
```

Example:


fundef.py - C:\Users\sai00\fundef.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 15 Col: 0

```
# Python code to create  and calling function with argument

# Funciton Declaration

def welcome(name):

    print(f"Welcome to Python world!,{name}")


# Funciton Calling with argument

welcome("Ravindra")
welcome("Arun")
welcome("Varun")

print('End of Program')
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:\Users\sai00\fundef.py =====================
Welcome to Python world!,Ravindra
Welcome to Python world!,Arun
Welcome to Python world!,Varun
End of Program
>>>
```

In this example, the **welcome ()** function takes one parameter, name, to personalize the welcome message. When the function is called, it will print out a message to the console that greets the specified name.

To call a user-defined function, type its name followed by any required parameters in parentheses. Here's an example:

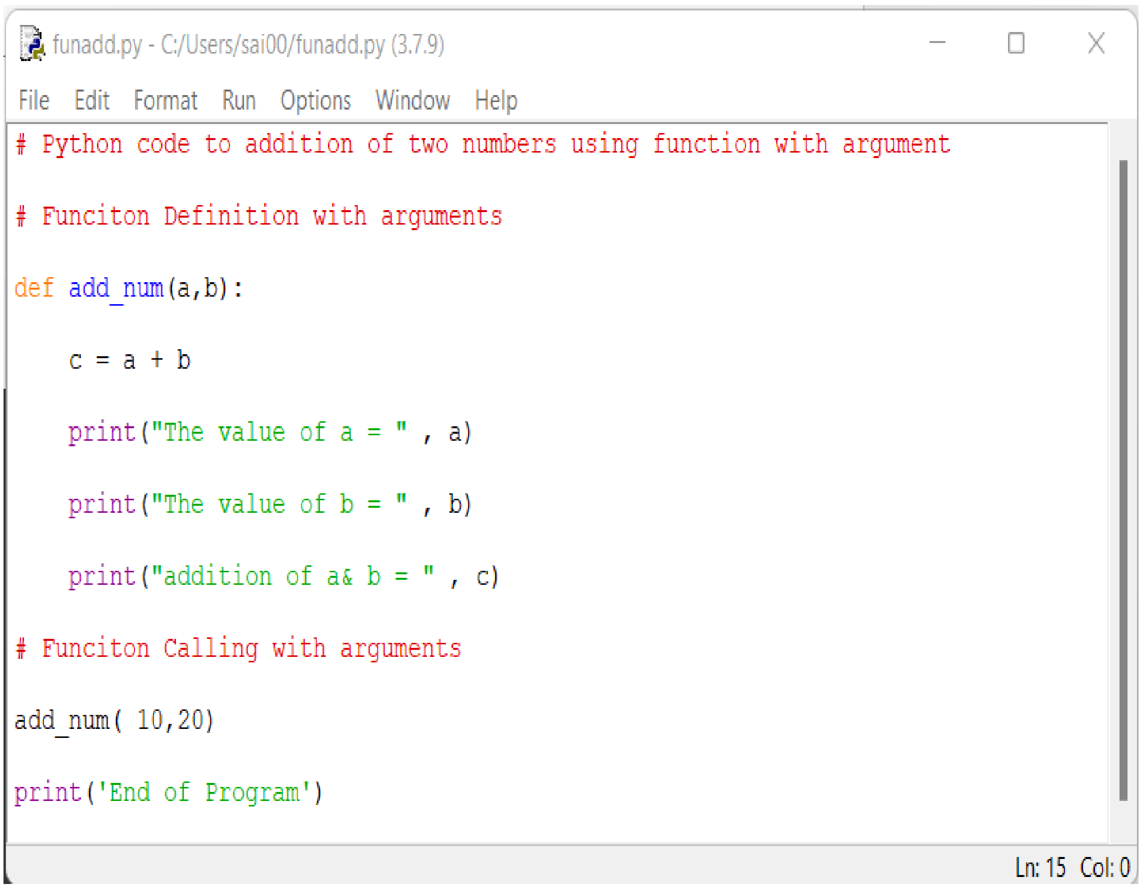
```
welcome ("Ravindra")

welcome("Varun")
```

This would output the following message to the screen:

```
Welcome to Python world! Ravindra

Welcome to Python world! Varun

End of Program
```

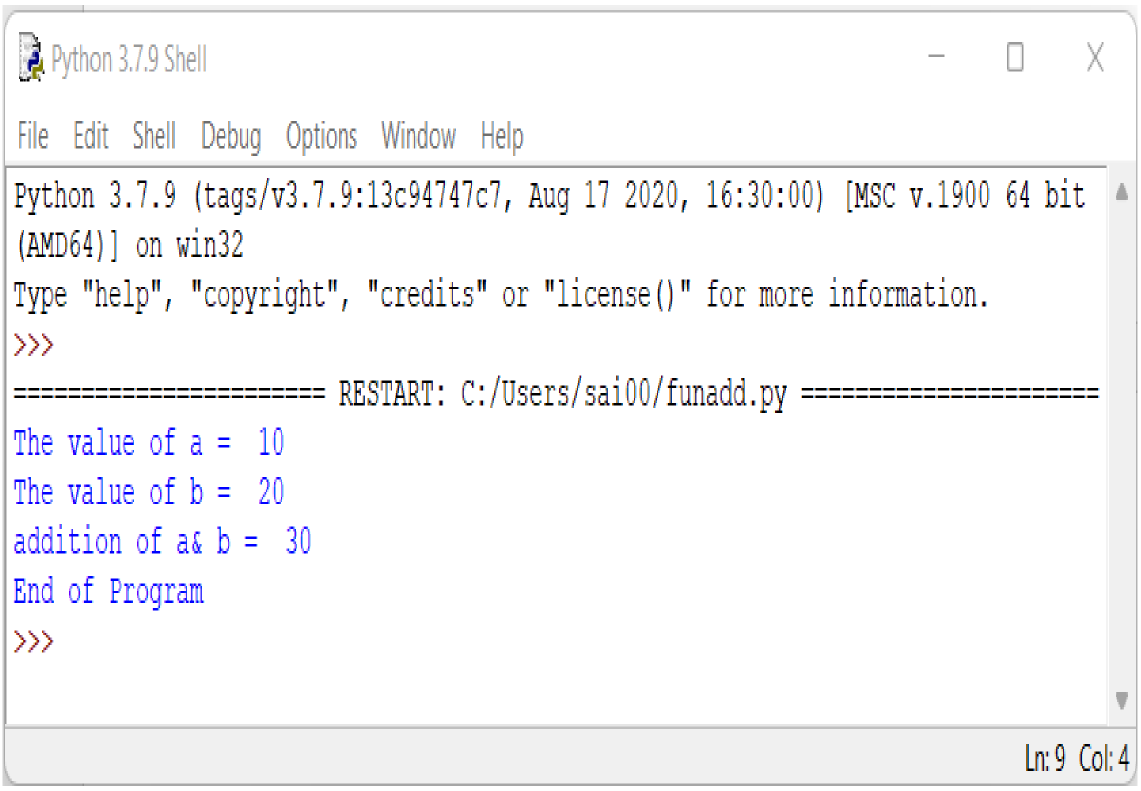
User-defined functions can be as simple or complicated as required. They can contain any number of statements, control structures, and other functions, and they can return one or more values as needed. When writing a function, use a descriptive name that really defines what it does, as well as clear, clear code that is simple to read and understand.

Example:


funadd.py - C:/Users/sai00/funadd.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 15 Col: 0

```
# Python code to addition of two numbers using function with argument

# Funciton Definition with arguments

def add_num(a,b):

    c = a + b

    print("The value of a = " , a)

    print("The value of b = " , b)

    print("addition of a& b = " , c)

# Funciton Calling with arguments

add_num( 10,20)

print('End of Program')
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/funadd.py =====================
The value of a =  10
The value of b =  20
addition of a& b =  30
End of Program
>>>
```

In the above example, we have created a function named **add_num ()** with arguments: a and b. The 10 and 20 are given inputs to the **add_num()** function. Once the function is called immediately the control will be gone to definitition and complete body is executed and produced result and is shown in Output screen shown in above.

10.5 RETURN STATEMENT

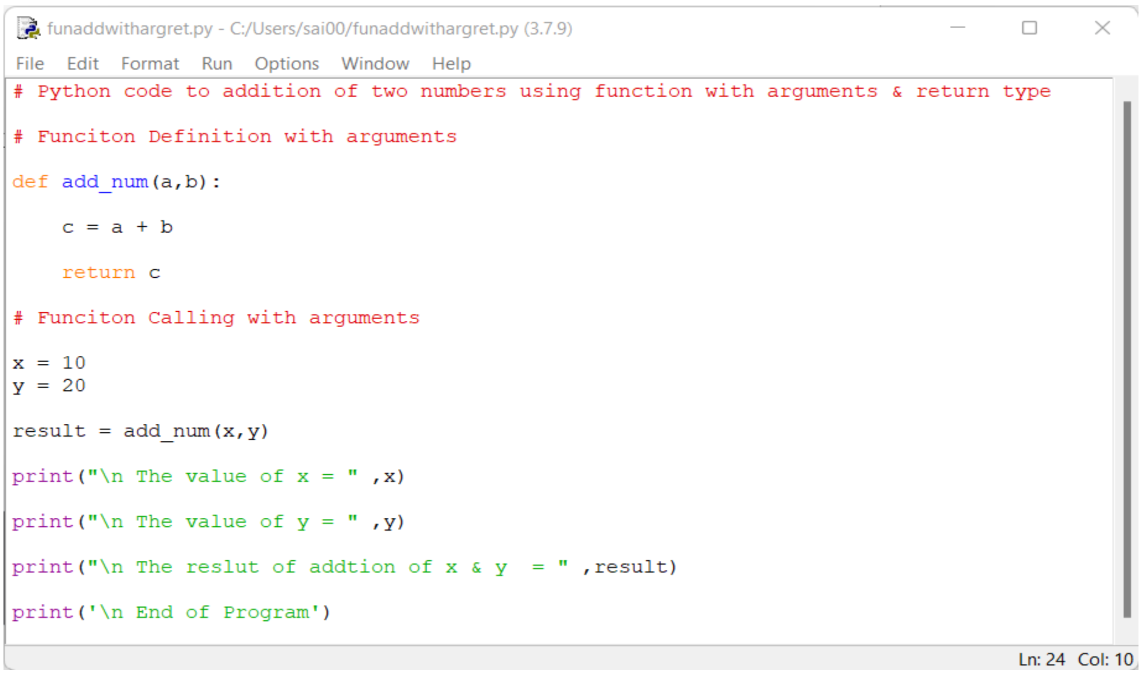
The function is defined with parameters and a return type, and it provides the option to enter a variety of inputs to the function in addition to return values. After the argument has been evaluated with the various arguments, the value is eventually returned. The argument is passed through a series of parameters.

The syntax of define function with arguments:

```
def fun_name (argument-1, argument-2, ……, argument-N):
        Function body
 return eval(argument-1, argument-2, ……, argument-N))
```

Like function defined in the previous section **add_num ()** with arguments: a and b is defined in the below example. The 10 and 20 are given inputs to the **add_num()** function. However, in the body of function included the return statement which returns the value of c and is result of addition among the a and b variables. The complete result is shown in Output screen shown in above.

Example

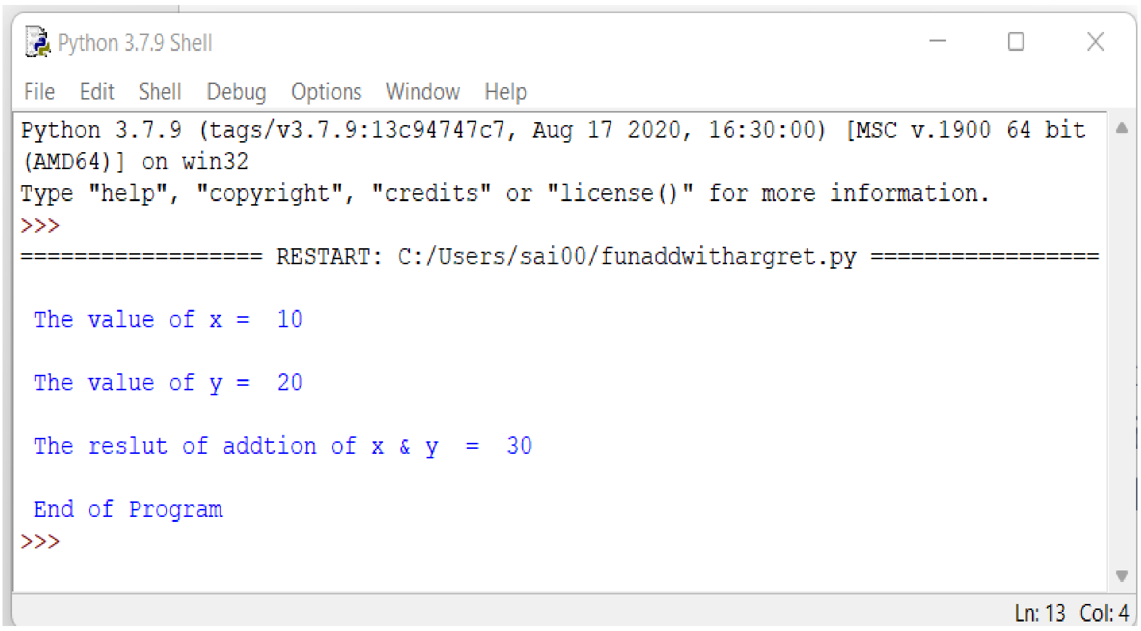


funaddwithargret.py - C:/Users/sai00/funaddwithargret.py (3.7.9)

```
# Python code to addition of two numbers using function with arguments & return type

# Funciton Definition with arguments

def add_num(a,b):

    c = a + b

    return c

# Funciton Calling with arguments

x = 10
y = 20

result = add_num(x,y)

print("\n The value of x = " ,x)

print("\n The value of y = " ,y)

print("\n The reslut of addtion of x & y  = " ,result)

print('\n End of Program')
```

Ln: 24 Col: 10

Output



Python 3.7.9 Shell

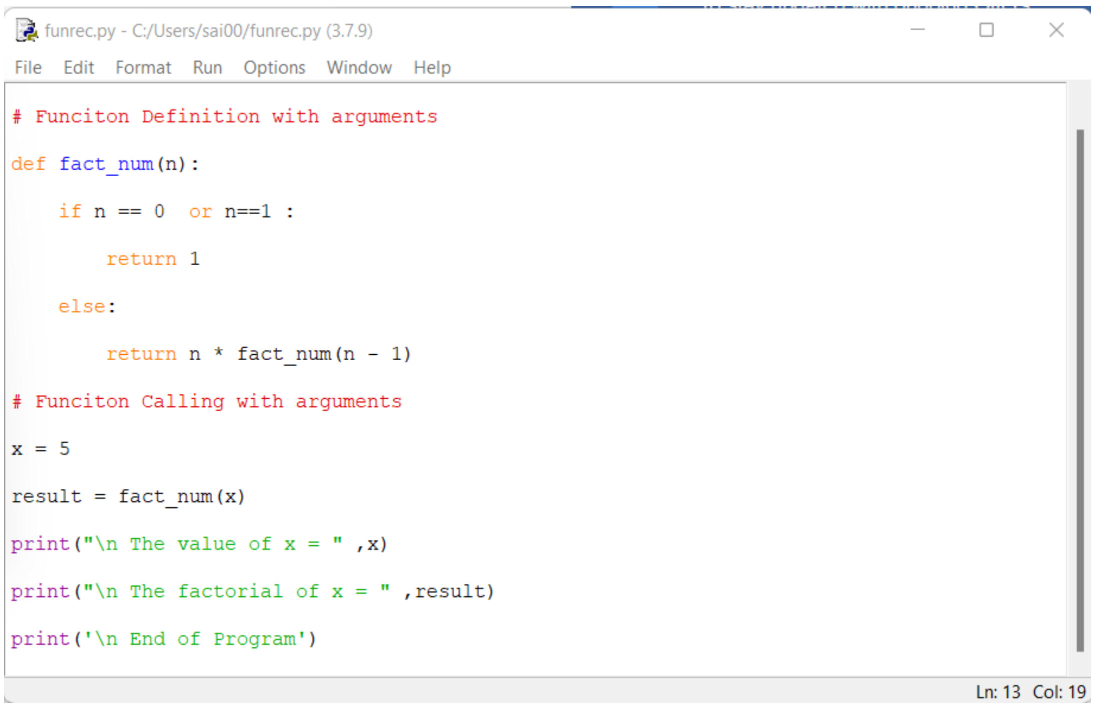
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/funaddwithargret.py =================

 The value of x =  10

 The value of y =  20

 The reslut of addtion of x & y  =  30

 End of Program
>>>
```

Ln: 13 Col: 4

10.6 PYTHON RECURSIVE FUNCTION

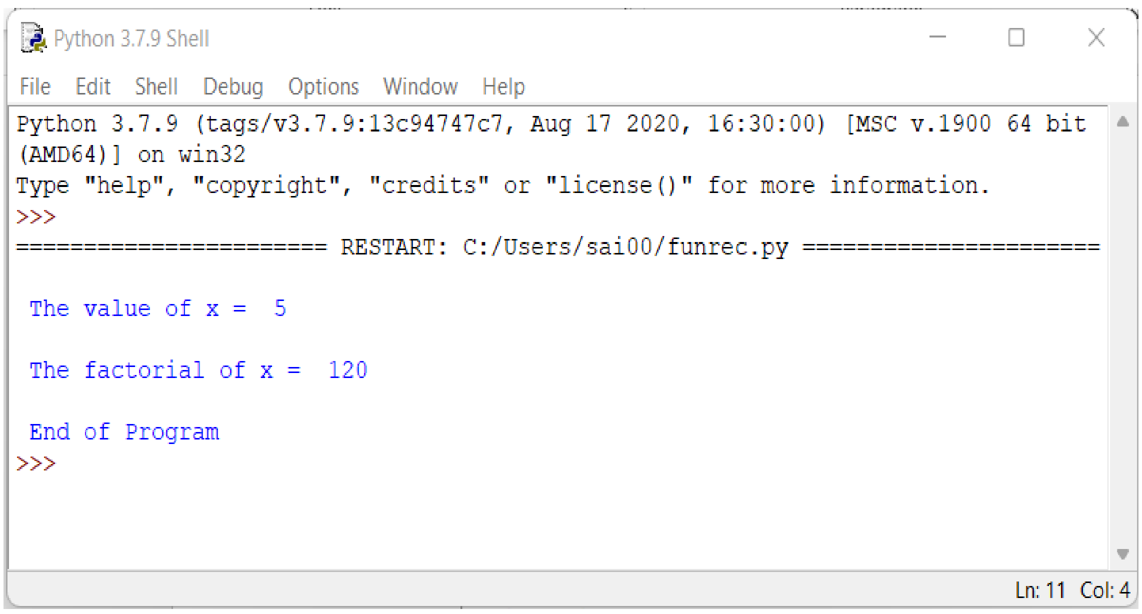
Python's recursive functions are those that call themselves to carry out a task in an iterative manner until a particular condition is satisfied. If an issue can be split down into smaller sub-problems that can be addressed using the same approach, then recursive functions can be used to solve the problem. The following is an illustration of a simple recursive function in Python that was designed to compute the factorial of a given number:

Example:



```
# Funciton Definition with arguments
def fact_num(n):
    if n == 0  or n==1 :
        return 1
    else:
        return n * fact_num(n - 1)
# Funciton Calling with arguments
x = 5
result = fact_num(x)
print("\n The value of x = " ,x)
print("\n The factorial of x = " ,result)
print('\n End of Program')
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/funrec.py ======================

 The value of x =  5

 The factorial of x =  120

 End of Program
>>>
```

In this example, the fact_num() recursive function is explained as follows:

- Create a recursive function called factorial, which accepts an integer n as input.
- Create a base case: If n equals 0 or 1, return 1 because the factorial of 0 or 1 is 1.
- In the recursive scenario, call the factorial function with n-1 as the argument, then multiply the result by n.
- Return the result of the recursive call.
- To calculate the factorial of a given number, call the factorial function and pass the required value as an argument.

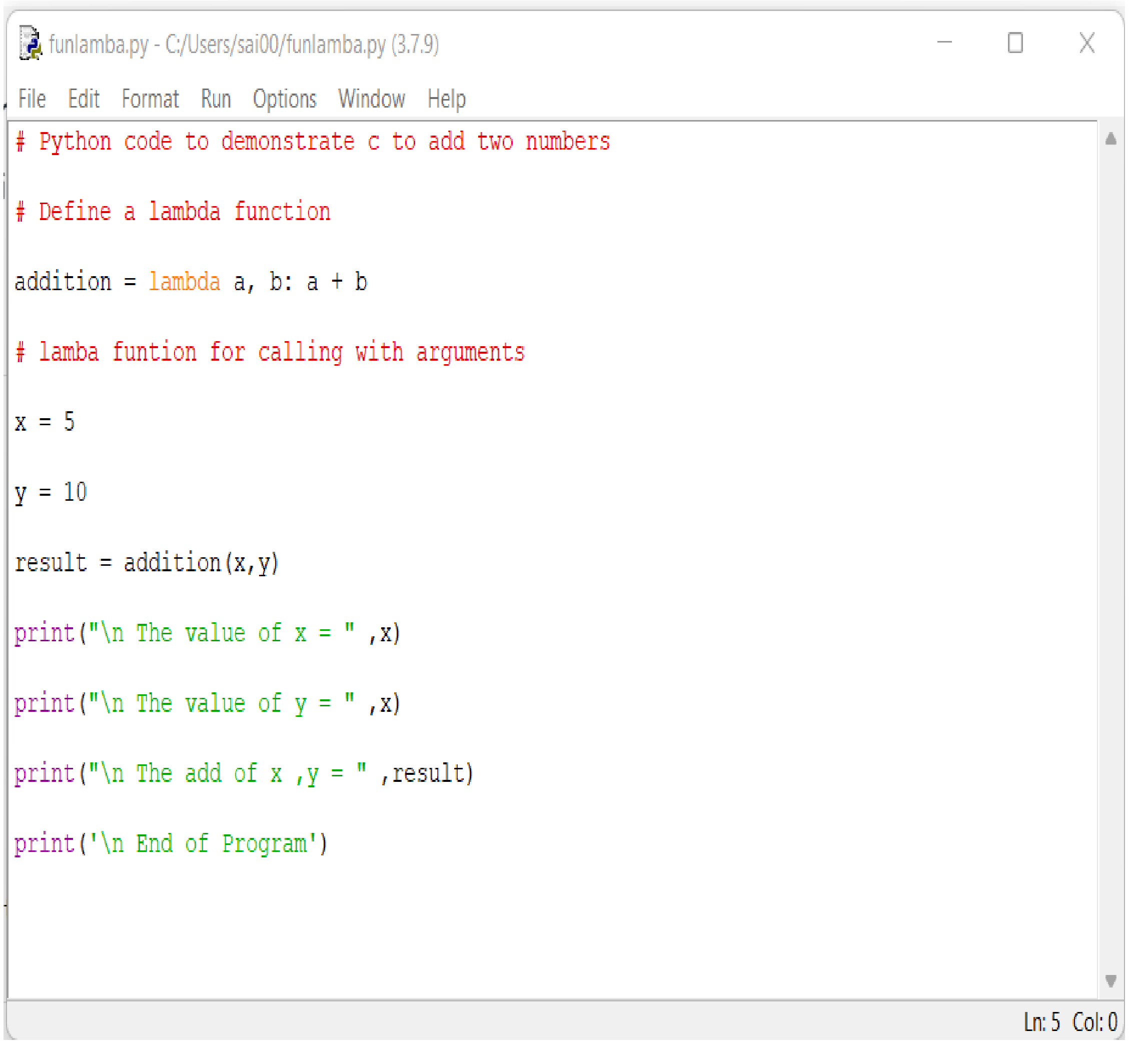
Recursion is a technique in which a function solves a problem by breaking it down into smaller subproblems of the same type. By continuously solving these smaller subproblems and aggregating their findings, the original problem is solved. Recursive functions are slower and use more memory than iterative solutions, therefore they may not be the best option for complex problems. Furthermore, recursive functions can be more difficult to debug and understand than iterative methods, thus they should only be used when they clearly outperform other approaches.

10.7 PYTHON LAMBDA FUNCTION

In Python, a lambda function is a short enough anonymous function that can accept any number of parameters but has only one expression. Lambda functions are also referred to as "anonymous functions" because they do not require a named function to be defined.

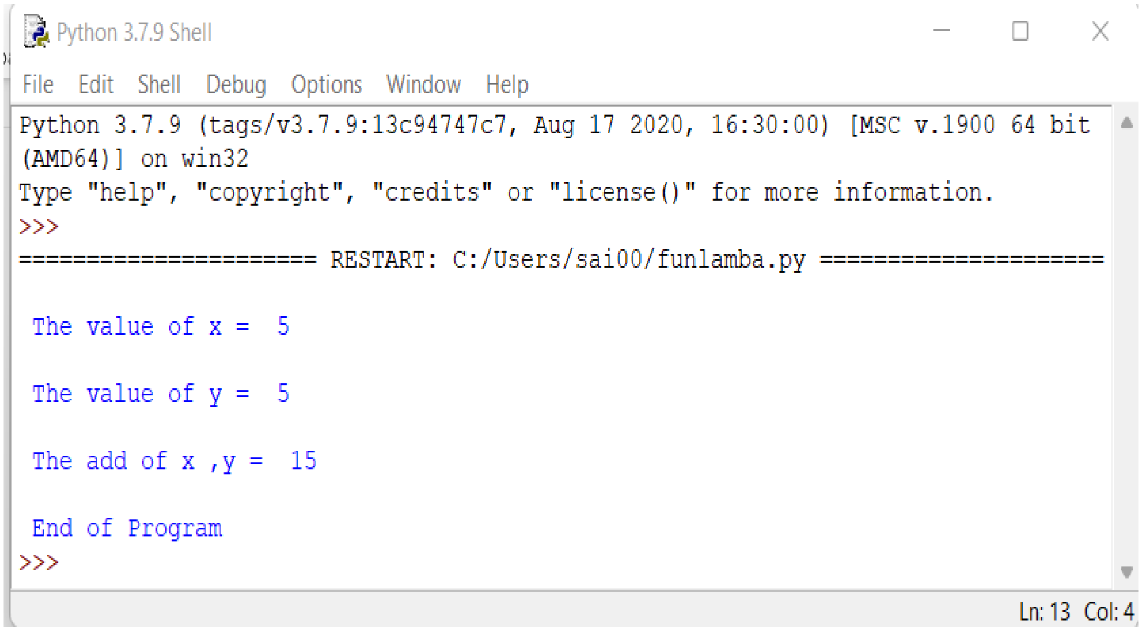
Here's an example of a simple lambda function for adding two numbers.In this example, the lambda function takes two inputs (x and y) and returns their sum.To use a lambda function, assign it to a variable and call it like a regular function.

Example:


funlamba.py - C:/Users/sai00/funlamba.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 5 Col: 0

```
# Python code to demonstrate c to add two numbers

# Define a lambda function

addition = lambda a, b: a + b

# lamba funtion for calling with arguments

x = 5

y = 10

result = addition(x,y)

print("\n The value of x = " ,x)

print("\n The value of y = " ,x)

print("\n The add of x ,y = " ,result)

print('\n End of Program')
```

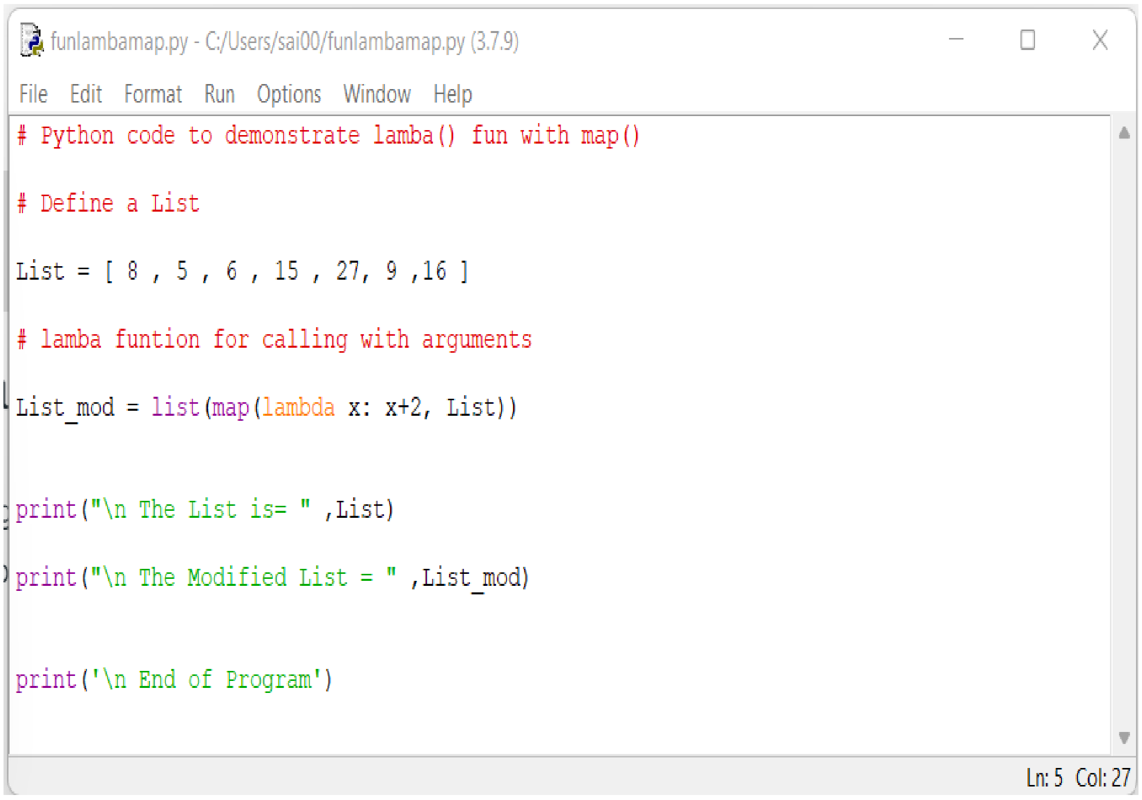
Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/Users/sai00/funlamba.py ====================

 The value of x =  5

 The value of y =  5

 The add of x ,y =  15

 End of Program
>>>
```

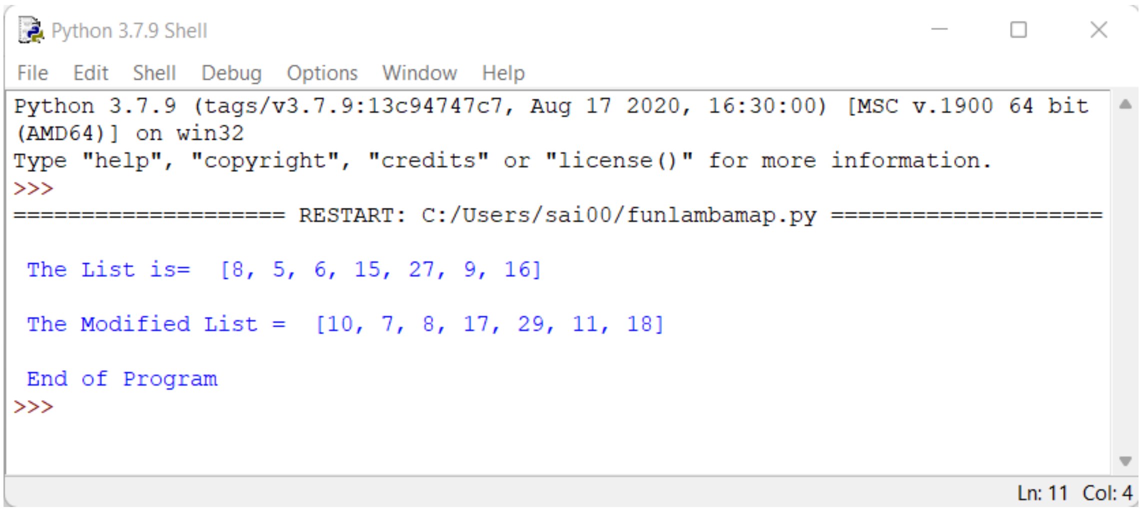
Python's map() function accepts a function and a list as its arguments. The function calls itself "map." The function is invoked with a lambda function and a list, and the function then returns a new list that contains all of the lambda-modified items that were returned by that function for each item.

Example:


funlambamap.py - C:/Users/sai00/funlambamap.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 5 Col: 27

```python
# Python code to demonstrate lamba() fun with map()

# Define a List

List = [ 8 , 5 , 6 , 15 , 27, 9 ,16 ]

# lamba funtion for calling with arguments

List_mod = list(map(lambda x: x+2, List))


print("\n The List is= " ,List)

print("\n The Modified List = " ,List_mod)


print('\n End of Program')
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/funlambamap.py ====================

 The List is=  [8, 5, 6, 15, 27, 9, 16]

 The Modified List =  [10, 7, 8, 17, 29, 11, 18]

 End of Program
>>>
```

In order to apply a function to each individual item in a list, the following is an example of how to use a lambda function together with the map() function. For the purpose of this illustration, the lambda function is utilized to square each individual element in the numbers list, and the map () function is utilized to apply the lambda function to each individual element of the list. After that, the list of squared values that produced the result is displayed on the screen.

Table 10.2. Lamba Function Vs Non-Lamba Function

With lambda function	Without lambda function
Work for single-line statements that return a value.	Work for multi-line statements inside function
Ideal for executing quick tasks or data manipulations.	Ideal for executing multi-line code
Using the lambda function might reduce code readability.	This allows comments and necessary function descriptions for good readability.

In this, we seen the Lambda function that is available in Python. An n-th number of arguments can be passed to a lambda function at the same time. Having said that, it only returns a single argument at a time. In the above section, we will go over certain lambda functions together with the Python program code, and we will also provide some examples of these functions. In addition, we talked about the Lambda function in Python, including the list, and the map function.

10.8 SUMMARY

Functions are an essential part of any programming language. Python functions are defined with the def keyword, and they can accept any number of arguments. Python also allows anonymous functions. They can return a single value or a list of values, one by one. Python functions are reusable code blocks that execute specified tasks, making it easier to divide software into smaller, more modular portions. They help to structure and manage the code, as well as eliminate duplication.

10.9 TECHNICAL TERMS

Function, Recursive Function, Arguments, Lamba Function, Builit-in Function, Absolute Value, Minimum and Maximum.

10.10 SELF ASSESSMENT QUESTIONS

Essay Questions:

1. How is user defined function is created and called? Explain.
2. What are the various ways to create user-defined function? Explain.
3. Explain about Recursive Function with example.

Short Notes:

1. Write about Lamba Function
2. List functions in Math Module.

10.11 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. Smith, J. - Python Programming: A Comprehensive Guide. Publisher.
4. Brown, A. - Mastering Python Modules. Publisher.
5. Python Software Foundation- Python Documentation. Retrieved from https://docs.python.org/3/

Mr. G. V. SURESH

LESSON- 11

PYTHON LOCAL AND GLOBAL VARIABLES

AIMS AND OBJECTIVES

The main aim of this chapter is understanding the different types of variables used in the python programming. The discussion related to local and global variables is focused on this chapter. After completion of this chapter, student will be able to know what the scope of variable in python code is. Also able to differentiate the local and global variables efficiently. The nested functions and scope of the variable with good number of examples is the objective of this chapter.

STRUCTURE

11.1. INTRODUCTION

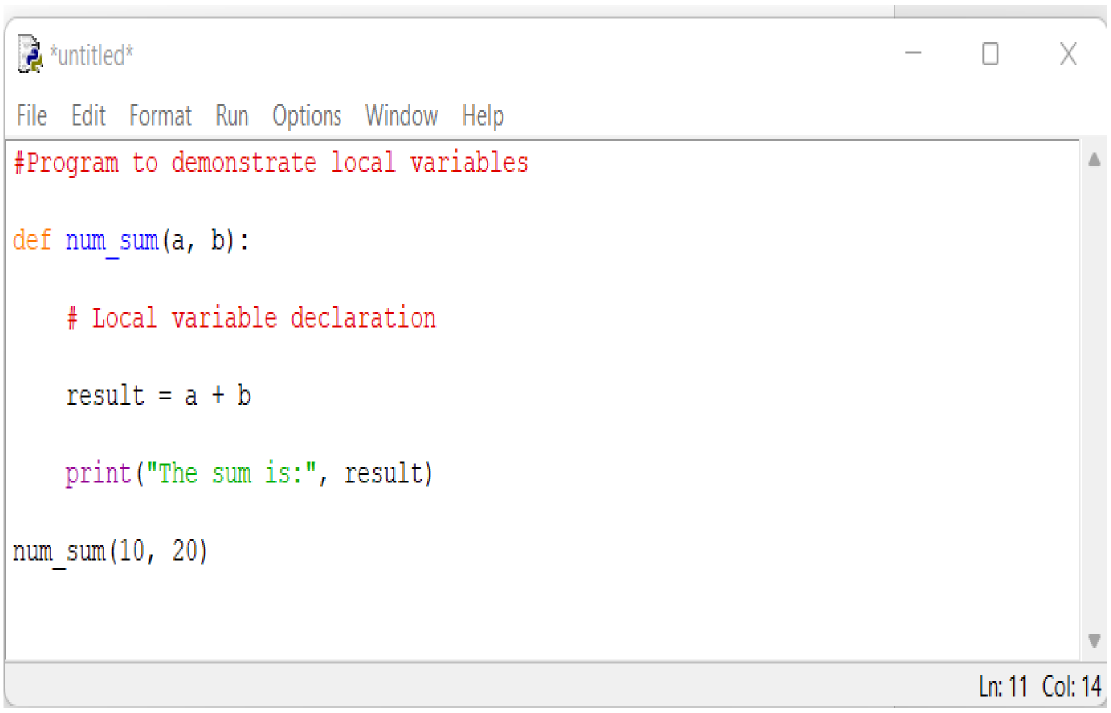
Variables are the containers that are used in Python for the storage of any data values. Python, in contrast to other languages, does not have a "statically typed" syntax. Neither the declaration of variables nor the declaration of their types is required before we use them. As soon as we give a value to a variable for the first time, we have created that variable. Our program does not make all of its variables accessible from every single area of it, and not all of the variables are present for the same period of time. The way a variable is defined determines both the locations at which it can be accessed and the length of time it has been in existence. The part of a program that allows a variable to be accessed is referred to as the scope, and the amount of time that the variable is present is referred to as the lifetime of the variable.

When it comes to programming, both local and global variables are quite important. In order to write code that is both modular and efficient, it is vital to have a solid understanding of how local and global variables operate within functions. This lesson will cover the idea of local and global variables in functions, including their declaration, scope, and best practices. We will also explore the best practices for using these variables.

11.2. LOCAL VARIABLE

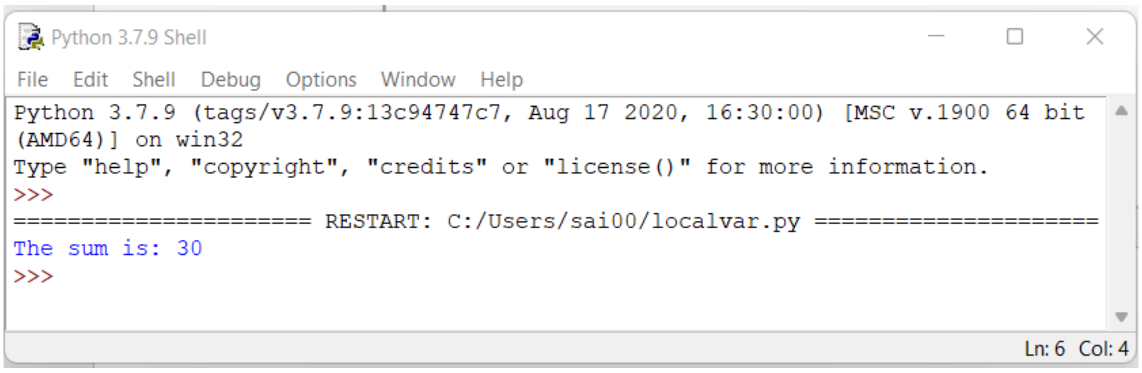
A local variable is a variable that is either declared within a function or used as a parameter in that function. These variables are named local because they can only be accessed and used within the function or block of code in which they are declared. Local variables are inaccessible from outside the function. When a local variable is declared within a function, it is defined and assigned a value when the function is called or executed. The variable remains in memory while the function is running and is destroyed when the function is finished.

Example:



```
#Program to demonstrate local variables

def num_sum(a, b):

    # Local variable declaration

    result = a + b

    print("The sum is:", result)

num_sum(10, 20)
```

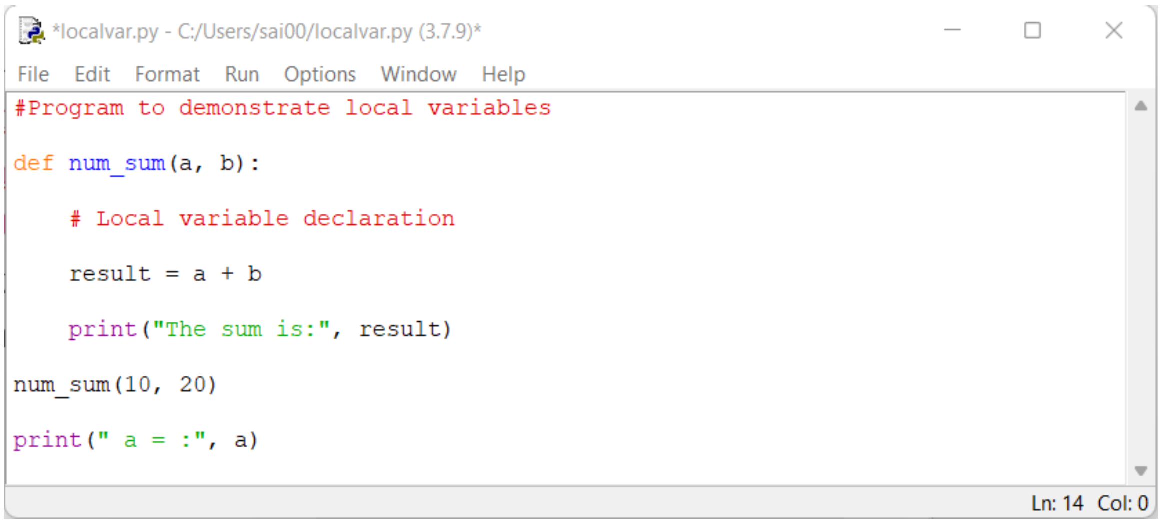
Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/localvar.py =====================
The sum is: 30
>>>
```

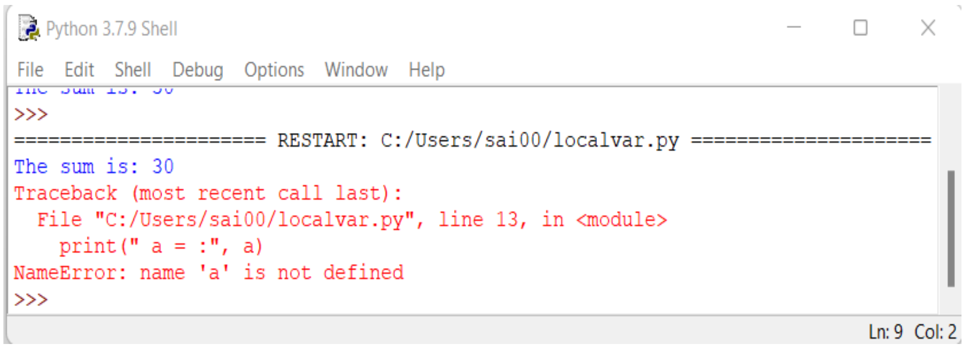
In this example, the function num_sum accepts two parameters, and b. Within the function, a local variable named result is declared and given the value a + b. This local variable's value is only accessible within the num_sum function.When the function is invoked with the arguments 10 and 20 (num_sum (10, 20)), the local variable result is computed and printed, with the sum shown as 30.

Example:


localvar.py - C:/Users/sai00/localvar.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 14 Col: 0

```python
#Program to demonstrate local variables

def num_sum(a, b):

    # Local variable declaration

    result = a + b

    print("The sum is:", result)

num_sum(10, 20)

print(" a = :", a)
```

Output:



```
>>>
===================== RESTART: C:/Users/sai00/localvar.py =====================
The sum is: 30
Traceback (most recent call last):
  File "C:/Users/sai00/localvar.py", line 13, in <module>
    print(" a = :", a)
NameError: name 'a' is not defined
>>>
```

Outside of the function, the local variable result is inaccessible. If you try to access it with print(result), you will receive a NameError since the variable is not specified in the global scope.

11.2.1. Advantages of Local Variables

- Local variables are only accessible within the function or code block they are defined in. This makes the code more legible and understandable because developers can see exactly where and when a variable is used.
- Local variables are limited to their defined scope, reducing name conflicts.
- Local variables are only available within their defined scope, preventing hostile actors from accessing sensitive data. This increased security can assist prevent data breaches and other security risks.
- Simplified debugging: Limiting the scope of variables can help identify the root cause of a fault. If a variable is only accessible within a certain function, the search for the problem can be limited to that function.

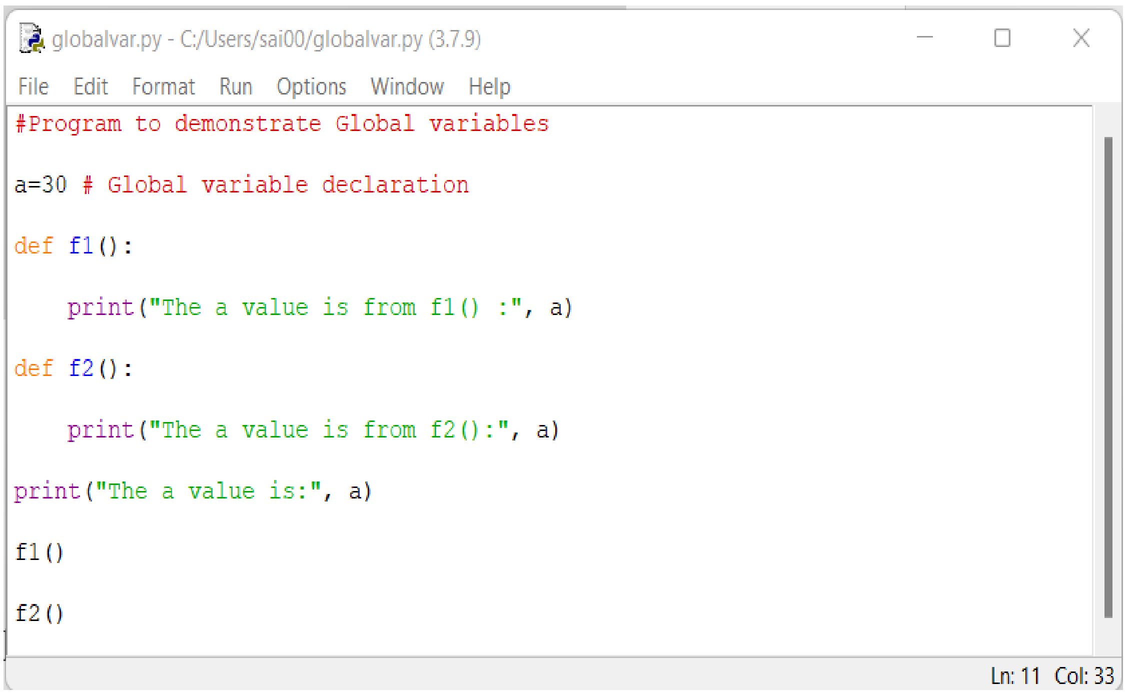
11.2.1 Disadvantages of Local Variables:

- The most significant drawback of local variables is that they can only be accessed inside the scope in which they are defined.
- Sharing data between different functions or code blocks can become more challenging because of this behaviour.
- Because local variables are only stored in memory while the function or code block in which they are defined is running, this results in an increase in the amount of memory that is being used.
- It is necessary to define local variables within the function or code block in which they are being used, which can add additional effort and complexity to the development process.

11.3. GLOBAL VARIABLE

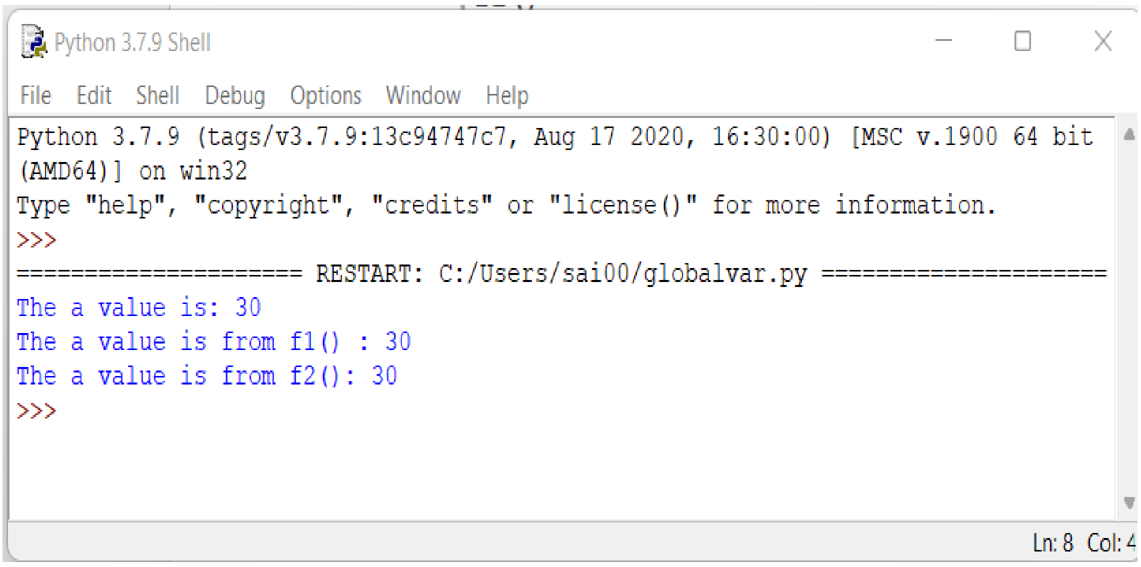
Variables that are defined outside of any function or block of code are referred to as global variables. These variables are accessible from any location within the program to which they are assigned. They are visible and accessible throughout the entirety of the program because they have a global scope, which means they are accessible. In most cases, the declaration of thet occurs at the beginning of the program.

Example:


globalvar.py - C:/Users/sai00/globalvar.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 11 Col: 33

```
#Program to demonstrate Global variables

a=30 # Global variable declaration

def f1():

    print("The a value is from f1() :", a)

def f2():

    print("The a value is from f2():", a)

print("The a value is:", a)

f1()

f2()
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/globalvar.py ====================
The a value is: 30
The a value is from f1() : 30
The a value is from f2(): 30
>>>
```

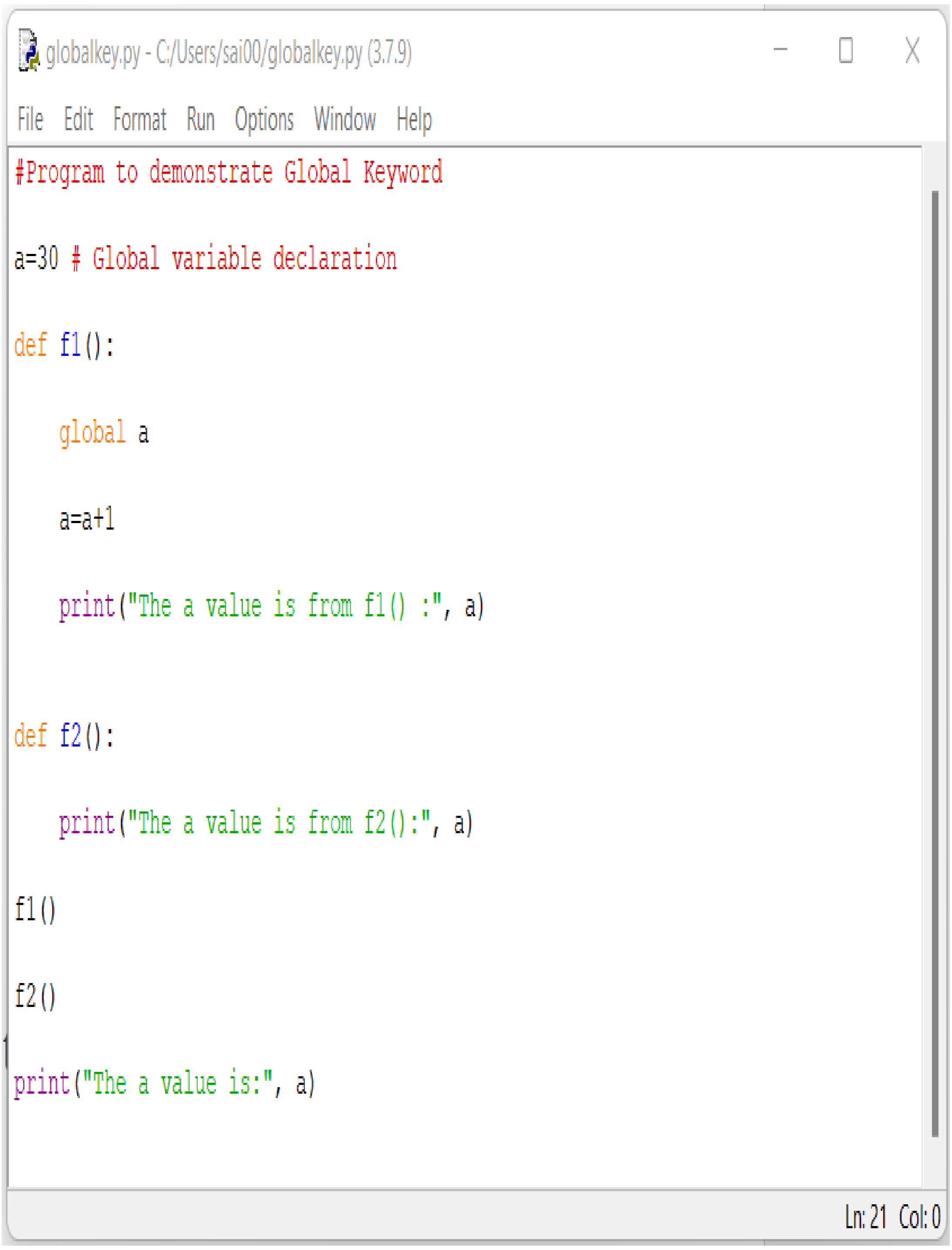
The variable an is declared as a global variable in this example, which means that it is not contained within any function or block of code. Because of this, it is accessible from any location within the application and has a worldwide scope.

The global variable and is within the scope of both the f1() and f2() functions, which means that they are able to access and output the value of the variable. This demonstrates that global variables can be accessed and utilized within separate functions, which provides a mechanism for data to be shared among multiple components of the program.

11.3.1. Use of Global Keyword

The global keyword is a crucial component of Python, as it allows for the modification of global variables inside a local scope, such as within a function. It communicates to the Python interpreter in a clear and concise manner that a variable is global, which enables you to modify the value of the variable from within a function. To preserving state or data throughout the execution of your application, this is absolutely necessary.

Example:

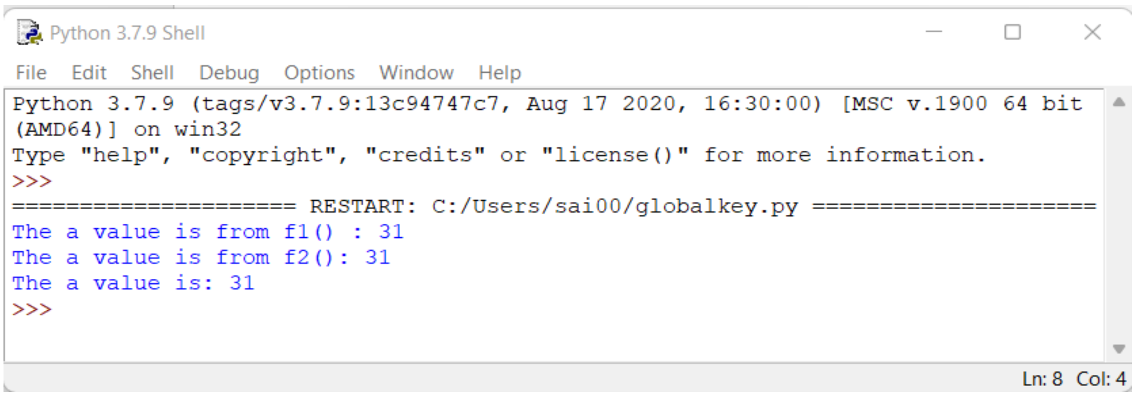

globalkey.py - C:/Users/sai00/globalkey.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 21 Col: 0

```
#Program to demonstrate Global Keyword

a=30 # Global variable declaration

def f1():

    global a

    a=a+1

    print("The a value is from f1() :", a)


def f2():

    print("The a value is from f2():", a)

f1()

f2()

print("The a value is:", a)
```

You might use the global keyword in the following manner, for instance, if you have a global variable called "a" and you want to increment it while you are working within a function:

```
global a
a=a+1
```

Output:



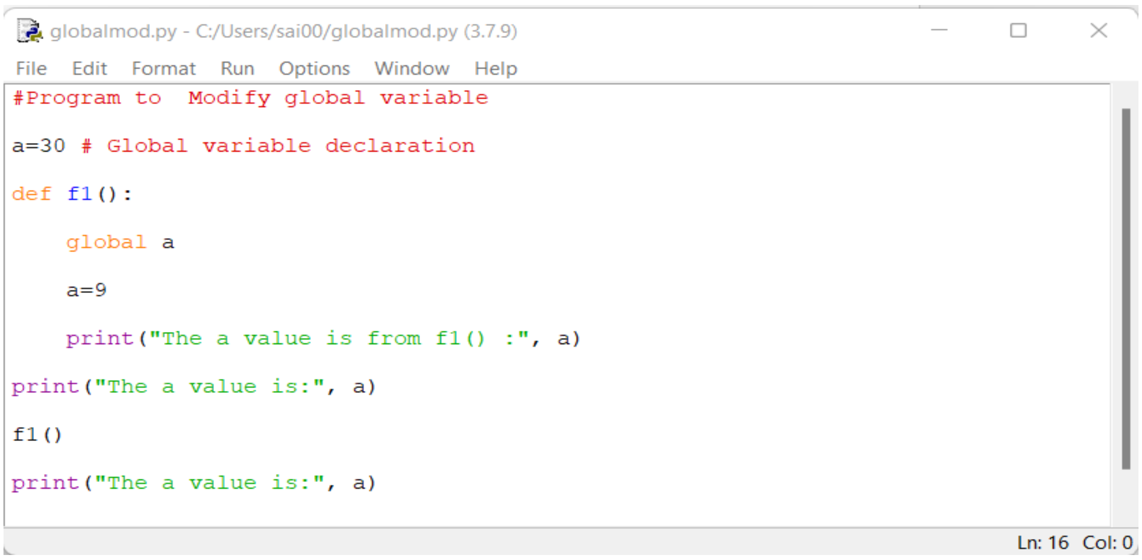
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/Users/sai00/globalkey.py =====================
The a value is from f1() : 31
The a value is from f2(): 31
The a value is: 31
>>>
```

The above code demonstrates how the global keyword bridges the scope between local and global variables by essentially incrementing the global 'a' variable from within the f1() function using the global keyword.

11.3.2. Update of Global Variables

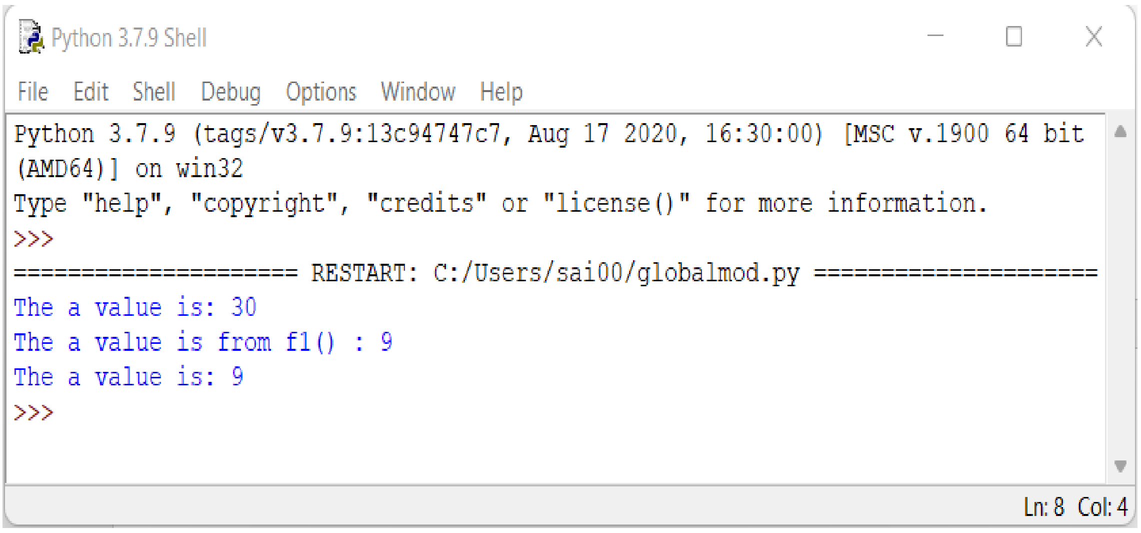
Modifying global variables within functions can be achieved using the global keyword, as shown earlier. However, there are alternative techniques, such as passing global variables as arguments to functions and returning modified values.

Example:


globalmod.py - C:/Users/sai00/globalmod.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 16 Col: 0

```
#Program to  Modify global variable

a=30 # Global variable declaration

def f1():

    global a

    a=9

    print("The a value is from f1() :", a)

print("The a value is:", a)

f1()

print("The a value is:", a)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/globalmod.py ====================
The a value is: 30
The a value is from f1() : 9
The a value is: 9
>>>
```

You are required to make use of the global keyword whenever you want to access or edit a global variable that is contained within a function.First, the value 100 is assigned to the global variable an in this particular illustration. It is possible to access and edit the global variable a by utilizing the global keyword, which is utilized within the function f() to signal that we want to do so. After that, a new value of nine is assigned to it, and it is printed, which results in a = nine. When the print("The a value is:", a) command is executed outside of the function, it accesses the changed global variable a, which is still 9. This results in the final output displaying a value of 9.

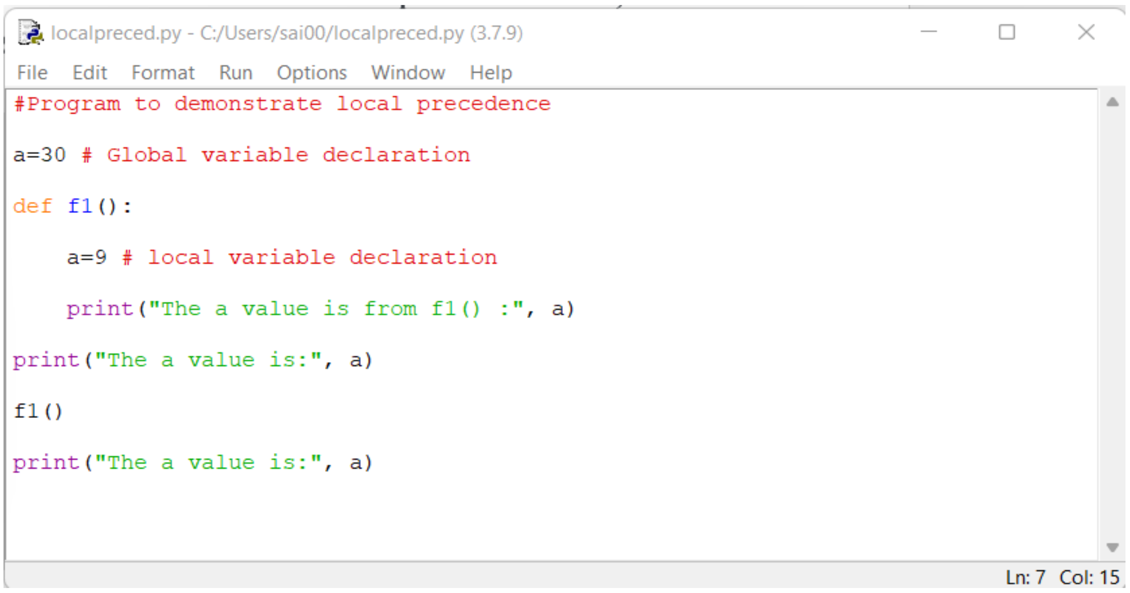
When utilizing the global keyword, it is essential to keep the following considerations in mind in order to decrease the likelihood of making mistakes:

- Prior to assigning a value to the variable that is contained within the function, the global statement ought to be declared.
- If a value is assigned to a variable without first declaring it to be global, the result will be the creation of a new local variable rather than the modification of the global variable.
- They are a source of complexity and potential dangers, such as unintended side effects and difficulties in debugging.
- It is vital to apply this technique with caution and carefully record any alterations made to global variables.

11.3.4 Precedence: Local Variables and Global Variables

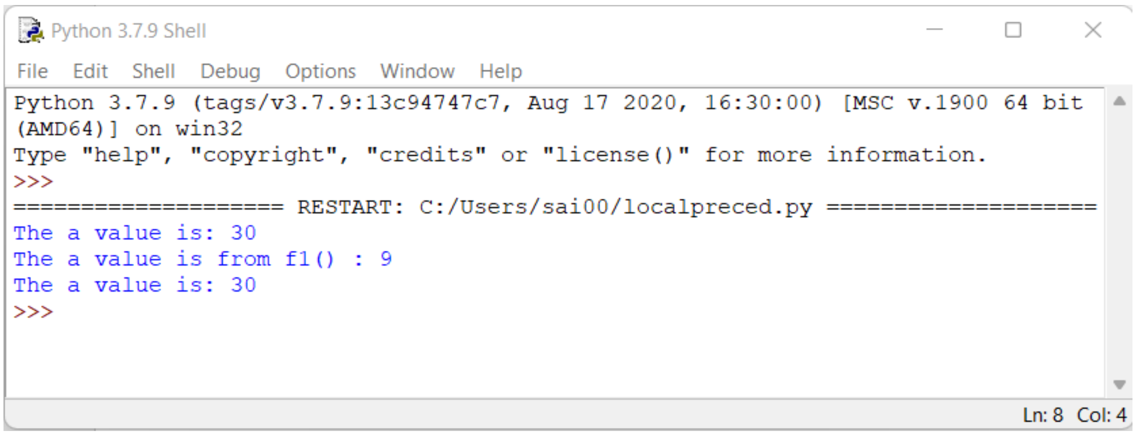
If you define a variable that is local to a function and it has the same name as a global variable, then the local variable will take priority within the scope of the function. The global variable, on the other hand, is unaffected by the function within which it is contained.

Example:


localpreced.py - C:/Users/sai00/localpreced.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 7 Col: 15

```
#Program to demonstrate local precedence

a=30 # Global variable declaration

def f1():

    a=9 # local variable declaration

    print("The a value is from f1() :", a)

print("The a value is:", a)

f1()

print("The a value is:", a)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/localpreced.py ====================
The a value is: 30
The a value is from f1() : 9
The a value is: 30
>>>
```

As an illustration, a global variable denoted by the variable ‘a’ is defined with a starting value of 30 . A value 9 is assigned to a local variable called ‘a’ , which is declared within the function f(). Each time the function f() is used, the value of the local variable a=9 is printed out. When the print("The a value is:", a) statement is executed outside of the function, it references the global variable a (30). This is since the scope of the local variable is restricted to the function itself. This is the reason why the output displays There is ‘a’ value of 30.

11.3.5 Advantages and Disadvantages of Global Variables

Advantages of Global Variables:

- In addition to being simple to use, global variables are also simple to retrieve because they may be accessed from any location within the code.

- Global variables can be utilized to facilitate the sharing of data between several classes or functions.
- You can use global variables to store data that needs to be persistent during the lifetime of the program.

Disadvantages of Global Variables:

- As was discussed previously, it is possible for global variables and local variables to share the same name, which might increase the likelihood of confusion.
- When global variables are adjusted, there is a possibility that the code will be affected in a manner that is not intended.
- Global variables provide a security concern since they can be accessed and modified by anybody who has access to the code without the need for authorization.

11.4 BEST PRACTICES FOR USING LOCAL AND GLOBAL VARIABLES IN PYTHON

When it comes to using local and global variables in Python, the best practices are as follows:

1. Prefer local variables and restrict the use of global variables. It is advisable to minimize the utilization of global variables, as they have the potential to make the code more difficult to comprehend and maintain.
2. When declaring variables, it is important to ensure that they are declared in the shortest scope possible, especially in situations when they are required. By doing so, the complexity is reduced, and any undesired side effects or conflicts with other variables are avoided simultaneously.
3. Select names that are descriptive for variables. Having names that have meaning enhances the readability and comprehension of the code.
4. Using function parameters and return values, you can transfer data from one function to another. As a result, this effectively encourages modularity and prevents an excessive dependence on global variables.
5. Make use of local variables to store intermediate results or temporary values within functions. By doing so, the code remains uncluttered and prevents the global namespace from being cluttered.

11.5 DIFFERENCE BETWEEN LOCAL VARIABLE AND GLOBAL VARIABLE

- **Scope:** On the other hand, global variables have a global scope, which means that they can be accessed and edited from any location inside the code. On the other hand, local variables are characterized by their limited scope, which means that they can only be accessed and modified within the confines of the domain of the function or class in which they are defined.

- **Visibility:** Local variables are only accessible within the scope in which they are defined. Within the boundaries of their designated scope, they cannot be accessed or modified in any way. Global variables, on the other hand, are accessible and modifiable from any area of the program's code because they are visible throughout the entirety of the program.

- **Initialization:** It is necessary to explicitly declare and initialize local variables within the block of code or function in which they are utilized. On the other hand, global variables can be initialized at any point in the program, including the point of definition or any other portion of the program.

- **Memory Allocation:** Local variables are normally given memory when the block of code or function to which they belong is run. The memory is then deallocated after the scope is exited. Global variables, on the other hand, are memory that is allocated at the beginning of the program and remain in memory until the program is terminated.

- **Change of Implicit:** The modification of the value of a local variable within a function or code block does not influence the value of the variable outside of that scope. Altering the value of a global variable, on the other hand, can influence the value of the variable throughout the entire program.

- **Precedence:** When a local variable and a global variable have the same name, the local variable is given priority inside its scope. This is because the local variable is more specific. It is because of this that the local variable will be utilized rather than the global variable within the code block or function in which it is defined.

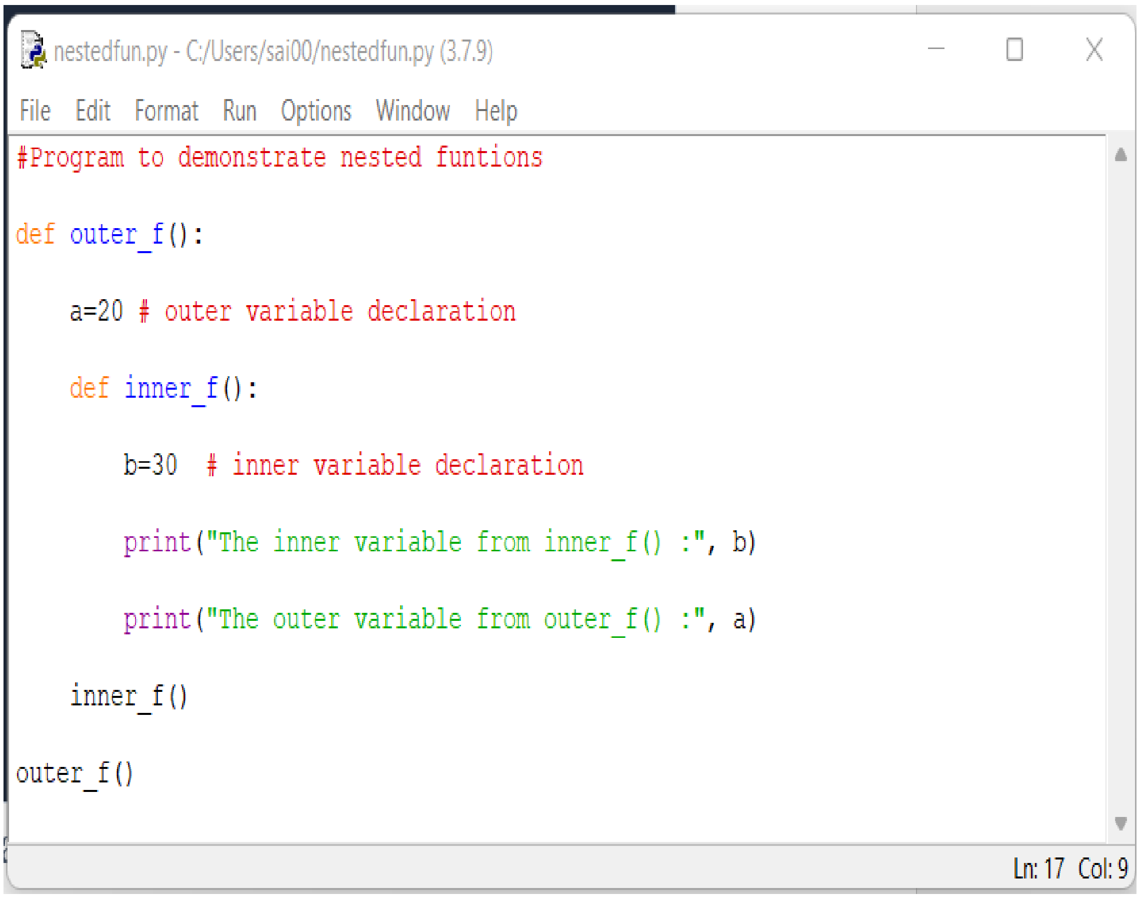
11.6. NESTED FUNCTIONS AND VARIABLE SCOPE

The concept of nested functions, which are functions within functions, might be thought of as an inception of programming. They produce a hierarchical structure of variable scopes, with each nested function having its own local scope. It is possible to access variables in nested functions from the scope that is the most inner to the scope that is the most outer. If a variable is not discovered within the local scope, Python will look for it within the scopes that surround it.

11.6.1. Accessing variables from outer functions

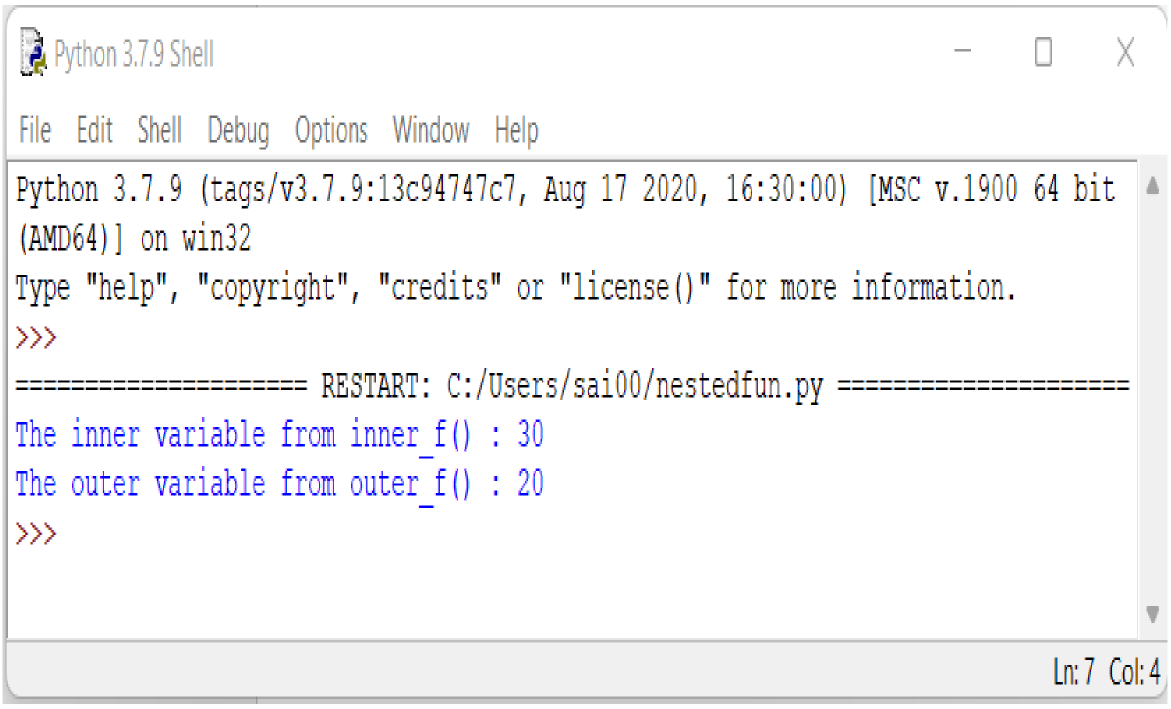
The code in the example demonstrates how nested functions can access variables on both the local and outer scopes of their respective functions.

Example:


nestedfun.py - C:/Users/sai00/nestedfun.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 17 Col: 9

```
#Program to demonstrate nested funtions

def outer_f():

    a=20 # outer variable declaration

    def inner_f():

        b=30  # inner variable declaration

        print("The inner variable from inner_f() :", b)

        print("The outer variable from outer_f() :", a)

    inner_f()

outer_f()
```

Output:



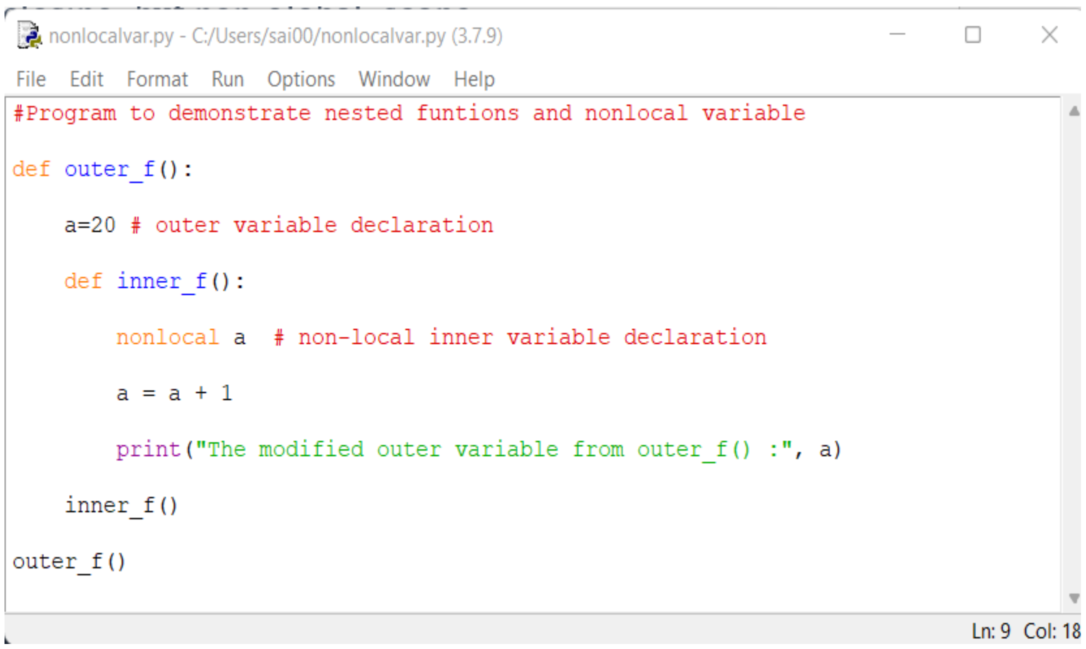
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/nestedfun.py ====================
The inner variable from inner_f() : 30
The outer variable from outer_f() : 20
>>>
```

The above code where defined **outer_f(),** which accepts the variable a as outer one and assign value 20. Later defined local **inner_f()** function and declared inner variable b with value 30. After that both inner and outer functions are invoked and produced values 30 and 20 respectively with precedence.

11.5.2. Nonlocal Variables

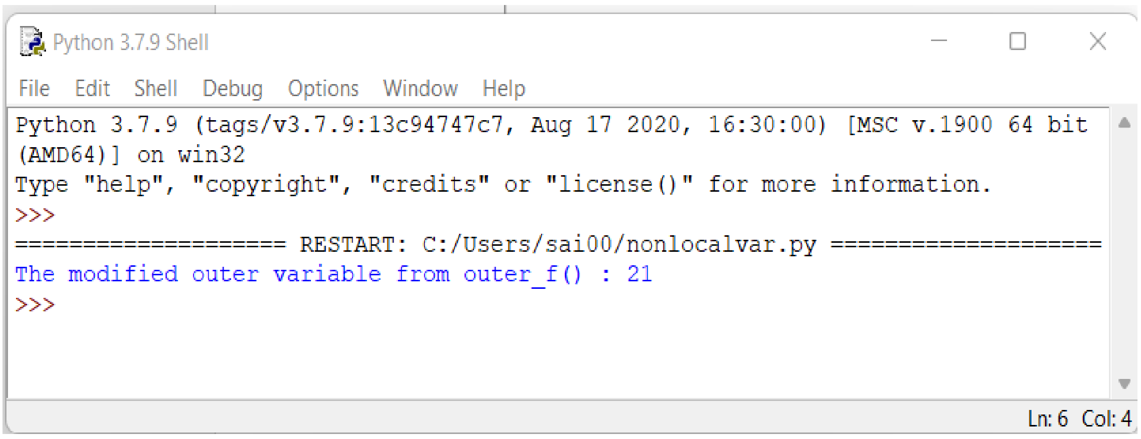
Variables that are not local serve as a connection between the local and global scopes. When utilized in nested functions, they are employed to modify variables that are contained within an enclosing scope that is not global.

Example:


nonlocalvar.py - C:/Users/sai00/nonlocalvar.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 9 Col: 18

```
#Program to demonstrate nested funtions and nonlocal variable

def outer_f():

    a=20 # outer variable declaration

    def inner_f():

        nonlocal a  # non-local inner variable declaration

        a = a + 1

        print("The modified outer variable from outer_f() :", a)

    inner_f()

outer_f()
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/nonlocalvar.py ====================
The modified outer variable from outer_f() : 21
>>>
```

For the sake of this illustration, the nonlocal keyword enables us to edit the 'a' in the outer scope, which is not the global scope. When it is necessary to make changes to variables that are in outer scopes within nested functions, nonlocal variables come in handy. It is possible to manipulate data within a restricted scope without having to resort to global variables because to their capabilities.

11.7 SUMMARY

Python is a programming language that has both local and global variables, each of which serves a distinct purpose and has a different scope. Different from global variables, which may be accessed from any part of the program, local variables can only be accessible from inside the section of code or function in which they are defined. Global variables can be accessed from anywhere in the program.

Local variables provide modularity, maintainability, and clarity to the structure of the information by limiting their visibility and ensuring that their value is not changed by other components of the program. This allows the information to be organized for easier comprehension. The utilization of global variables should be approached with caution to avoid any potential issues, even though they offer a wide range of accessibility. By restricting their visibility and ensuring that their value is not influenced by other components of the program, local variables offer modularity, maintainability, and clarity to the structure of the information.

11.8 TECHNICAL TERMS

Variable Scope, Local Variable, Global Variable, Non-Local Variable, Scope, Precedence, Visibility.

11.9 SELF ASSESSMENT QUESTIONS

Essay Questions:

1. Compare Local and Global variables.
2. Illustrate the concept of Non-Local Variables
3. Explain best practices to use local and global variables.

Short Notes:

1. Define Variable Scope.
2. List advantages of Local Variable
3. Mention the usage of global variables.

11.10 SUGGESTED READINGS

1. Steven Cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. Smith, J. - Python Programming: A Comprehensive Guide. Publisher.
4. Brown, A. - Mastering Python Modules. Publisher.

Dr. U. SURYA KAMESWARI

LESSON- 12

PYTHON MODULE

AIMS AND OBJECTIVES

The aim is to significance understand of python modules. A discussion of how to create new module and import the existing modules is focused on this chapter. After completion of this chapter, student will be able to know how to create and import modules. Random and Math modules are also understood thoroughly with examples.

STRUCTURE

12. 1 INTRODUCTION

As the size and complexity of our programs continue to grow, the necessity of organizing our code will become increasingly important. It is recommended that a large and complicated program be broken down into files and functions that each carry out a particular function. When we add more and more functions into a program, we ought to think about arranging the functions by placing them in modules. This is something that we should consider doing.

A module is nothing more than a file that includes code written in Python. In the process of dividing a program into modules, each module ought to include functions that carry out activities that are connected to one another. For instance, if the information contained in a book is not indexed or organized into separate chapters, the book will become tedious and difficult to understand. For this reason, the book is broken up into chapters, which makes it much simpler to comprehend.

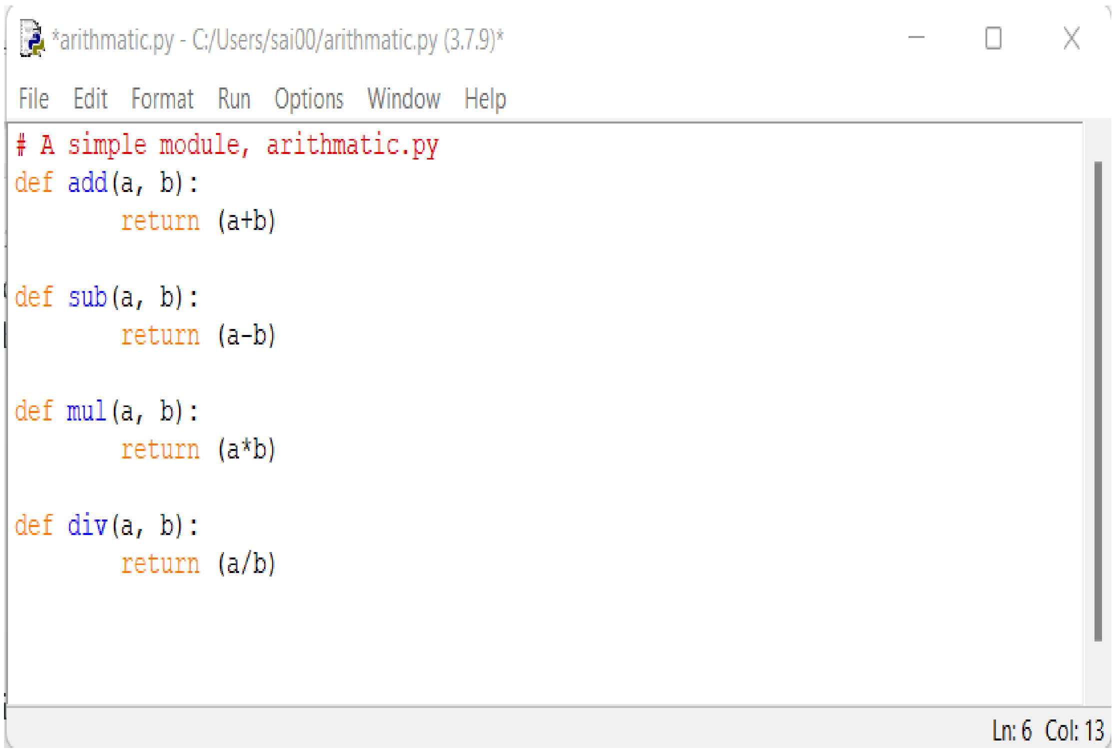
12.2. PYTHON MODULE

Python modules, on the other hand, are files that contain code that is very similar to one another. As a result, a module makes the Python code that defines classes, variables, and functions easier to understand and use.

12.2.1. Creating Python Module

To create a Python module, write the desired code and save that in a file with **.py** extension. Let's create a simple arithmatic.py in which we define three functions, add(),sub(),mul and div().

Example:


arithmatic.py - C:/Users/sai00/arithmatic.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 6 Col: 13

```
# A simple module, arithmatic.py
def add(a, b):
        return (a+b)

def sub(a, b):
        return (a-b)

def mul(a, b):
        return (a*b)

def div(a, b):
        return (a/b)
```

12.2.2. Advantages of Python Modules

The following are some of the benefits that Python modules offer:

- One of the reasons why putting code into modules is beneficial is because it allows one to import the functionality of the module.
- It is possible to utilize a module in combination with other Python programs. Consequently, it offers the capability of code reusability.
- The use of a module enables us to structure our Python code in a sensible manner.
- Creating a module that contains code that is related to one another makes the code simpler to comprehend and use.
- It is possible to classify and place in a single module attribute that are conceptually like one another.

12.3 IMPORT PYTHON MODULE

We can import the functions, and classes defined in a module to another module using the **import statement** in some other Python source file. When the interpreter encounters an import statement, it imports the module if the module is present in the search path.
The syntax to import python module is:

import module_name

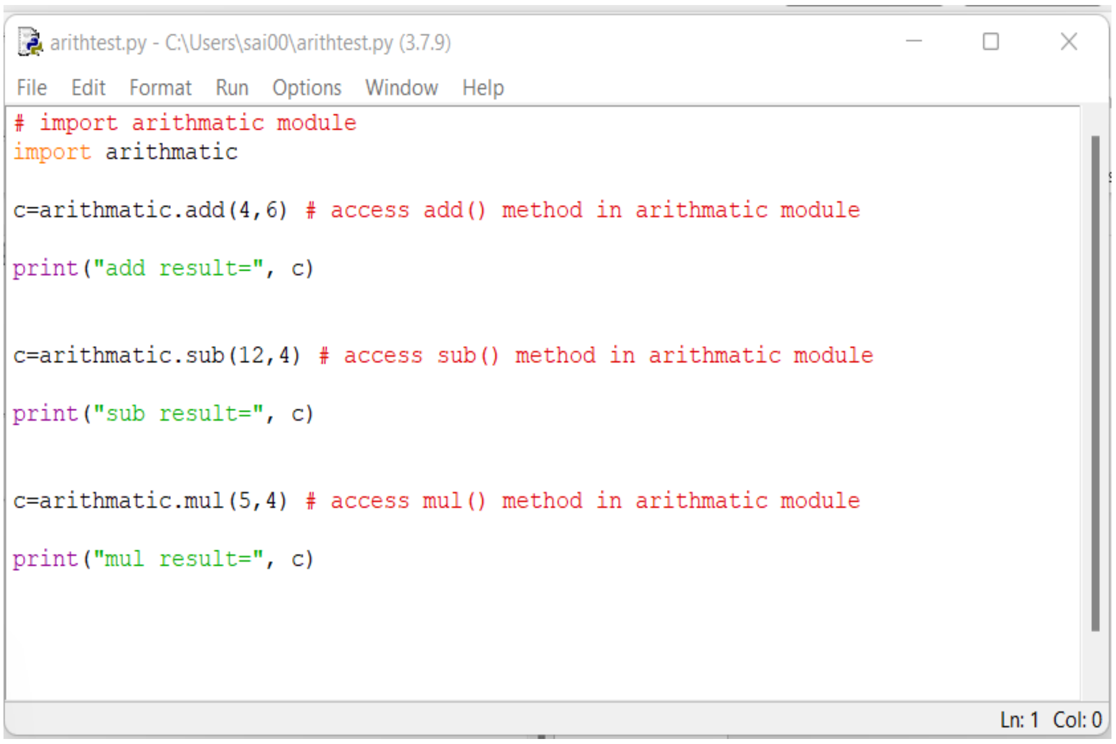
For example, to import the module arithmatic.py, we need to put the following command at the top of the script.

import arithmatic

This does not import the functions or classes directly instead imports the module only. To access the functions inside the module the dot(.) operator is used.

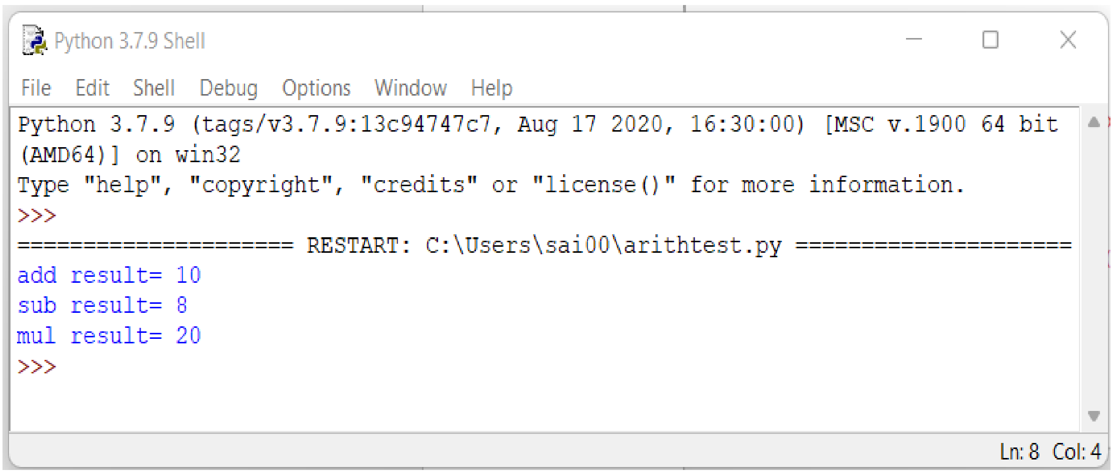
import arithmetic. add
import arithmetic. sub

Example:


arithtest.py - C:\Users\sai00\arithtest.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```
# import arithmatic module
import arithmatic

c=arithmatic.add(4,6) # access add() method in arithmatic module

print("add result=", c)


c=arithmatic.sub(12,4) # access sub() method in arithmatic module

print("sub result=", c)


c=arithmatic.mul(5,4) # access mul() method in arithmatic module

print("mul result=", c)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:\Users\sai00\arithtest.py ====================
add result= 10
sub result= 8
mul result= 20
>>>
```

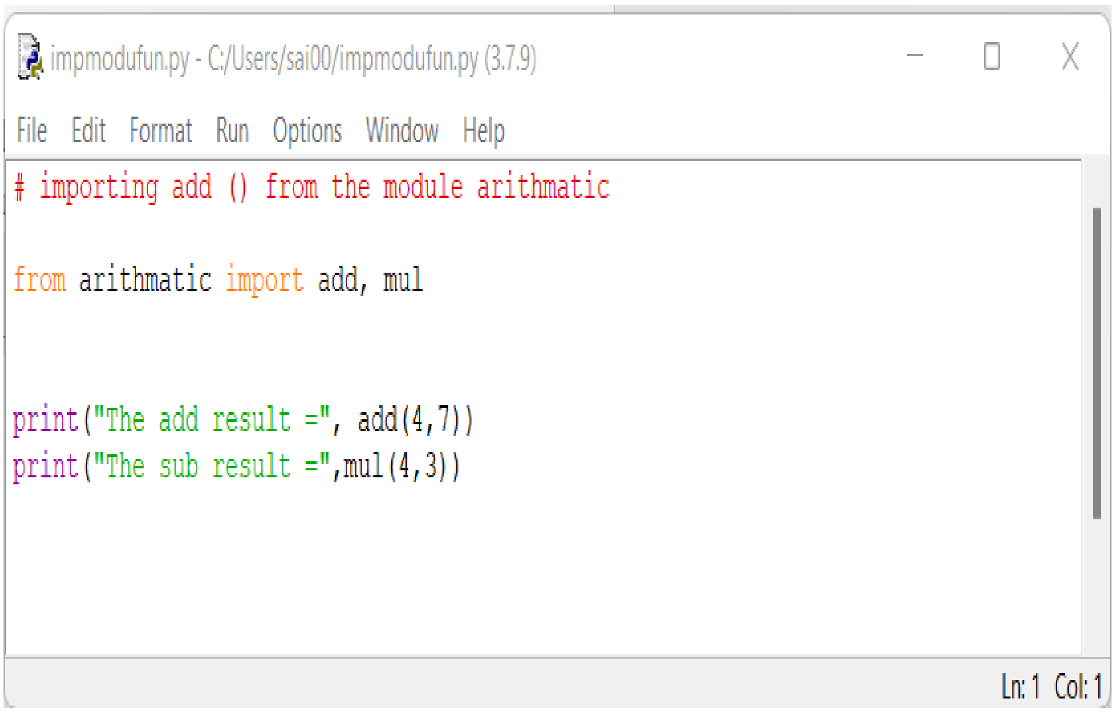
In the above code accessed addition functions with arithmetic.add (4.6) and produced result add result = 10. Similarly, we can access other two methods and produced results related to subtraction and multiplication operations.

12.3.1 Import Specific Attributes from Module

With Python's from statement, it is possible to import particular properties from a module without importing the module in its entirety. To do anything like import the add () method from the arithmatic.py module, for instance, we need to place the following command at the very beginning of the script.

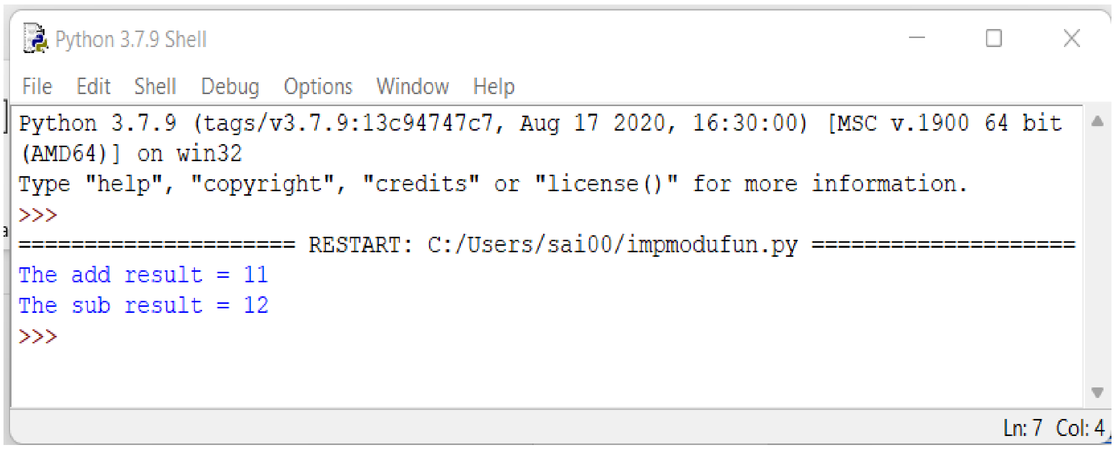
from module import function-name (or) class-name

Example:


impmodufun.py - C:/Users/sai00/impmodufun.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 1 Col: 1

```
# importing add () from the module arithmatic

from arithmatic import add, mul


print("The add result =", add(4,7))
print("The sub result =",mul(4,3))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/impmodufun.py ===================
The add result = 11
The sub result = 12
>>>
```

In the above code accessed only two functions add() and mul() are imported separately with the statement from arithmetic import add, mul respectively. Later accessed and produced result add result = 11 and sub result=12 with various a and b values. The result is shown in output.

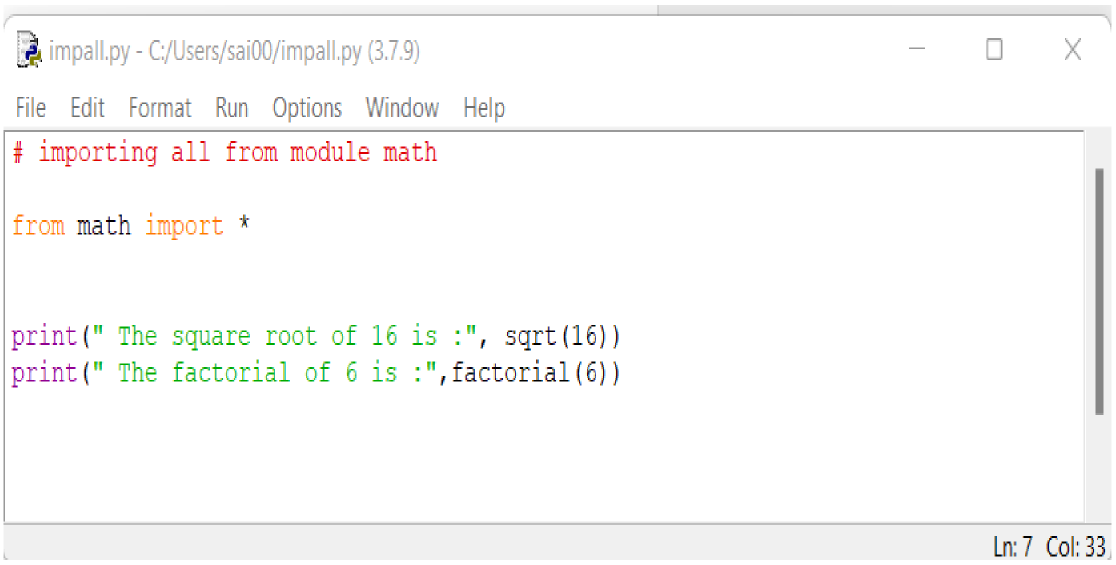
12.3.2 Import All Attributes in Module

When the import statement is used when combined with the * symbol, all of the names from a module are imported into the namespace that is now being used. Using the symbol * comes with both positive and negative implications. However, if you are unsure about what you will require from the module, it is not suggested that you use the * symbol; otherwise, you should use it.

Syntax:

from module_name import *

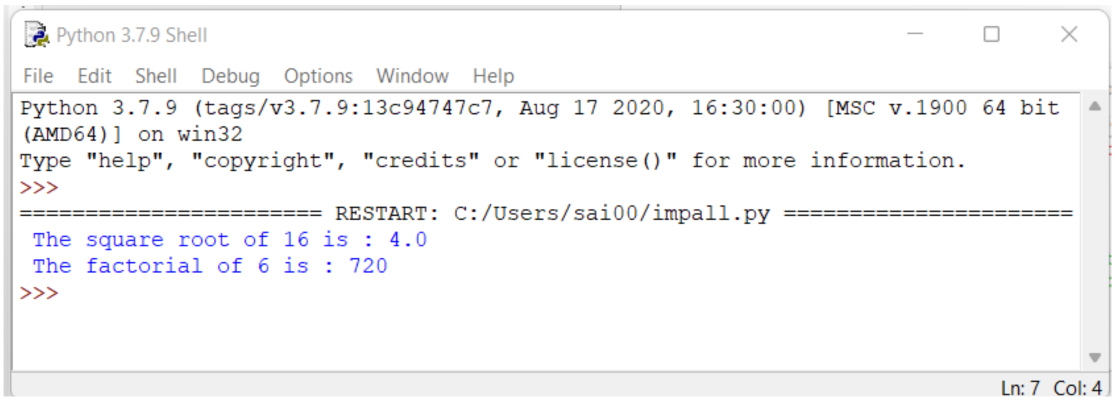
Example:



impall.py - C:/Users/sai00/impall.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 7 Col: 33

```
# importing all from module math

from math import *


print(" The square root of 16 is :", sqrt(16))
print(" The factorial of 6 is :",factorial(6))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/impall.py =====================
 The square root of 16 is : 4.0
 The factorial of 6 is : 720
>>>
```

In the above code imported all functions of math modules which is a built-in module at a time with the statement from arithmetic import * respectively. Later accessed and produced square root result = 4 of a given value 16. Similarly, factorial result of 6 is 720 respectively and is shown in output.

12.4 RANDOM MODULE

Among the built-in modules in Python, the Python Random module is responsible for the generation of random integers that fall within the range of 0.0 to 1.0. When it comes to the implementation of a randomization technique, the Python Random Module is helpful. The occurrence of these numbers is completely arbitrary and does not adhere to any rules or instructions. By utilizing this module, we can generate random integers, display a random item for a list or string, and perform a variety of other functions.

Additionally, we have the capability to select a sequence of numbers from a list of elements, offering us the ability to choose from a sequence of numbers. It is possible to use it to generate integers from a certain range in order to select the relevant number from among them. Nevertheless, it generates the numbers in a pseudorandom method, which, if we go deeper into its working mechanism under the hood, cannot be considered entirely random. Despite this, it continues to be useful for usage.

The following are some examples of typical applications:

- To participate in a game of chance in which the computer is required to roll a handful of dice, select a number, or flip a coin,
- To randomly let a new hostile spaceship to appear and shoot at you, to randomly shuffle a deck of playing cards, and to randomly allow a new enemy spaceship to appear.
- To conduct a computerized model for the purpose of calculating the environmental impact of constructing a dam, we will simulate the possibility of rainfall.
- To encrypting your use of the internet for banking purposes.
- Simulate Data for Time series analysis

- Random Number Generations for various typical applications

There are numerous categories of functions that can be accessed through the random module. There is a description of those in the table that is provided.

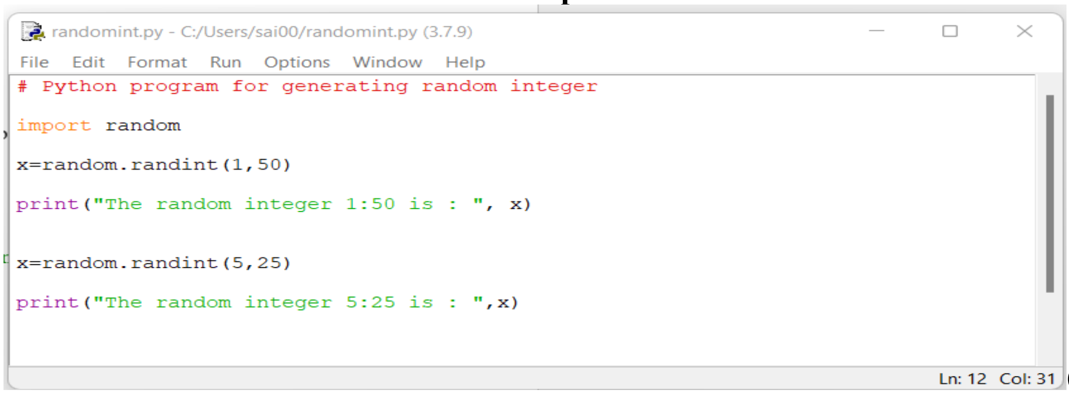
Table 12.1 Python Random Module Functions

Function	Meaning
randint(a, b)	Generate a random integer number within a range a to b.
randrange(start,stop, step)	Returns a random integer number within a range by specifying the step increment.
choice(seq)	Select a random item from a seq such as a list, string.
sample(population, k)	Returns a k sized random samples from a population such as a list or set
choices(population, weights, k)	Returns a k sized weighted random choices with probability (weights) from a population such as a list or set
seed(a=None, version=2)	Initialize the pseudorandom number generator with a seed value a.
shuffle(x[, random])	Shuffle or randomize the sequence x in-place.
uniform(start, end)	Returns a random floating-point number within a range
triangular(low, high, mode)	Generate a random floating-point number N such that low <= N <= high and with the specified mode between those bounds
betavariate(alpha, beta)	Returns a random floating-point number with the beta distribution in such a way that alpha > 0 and beta > 0.
expovariate(lambd)	It returns random floating-point numbers, exponentially distributed. If lambda is positive, it returns values range from 0 to positive infinity. Else from negative infinity to 0 if lambda is negative.
gammavariate(alpha, beta)	Returns a random floating-point number N with gamma distribution such that alpha > 0 and beta > 0

12.4.1. randint() Function

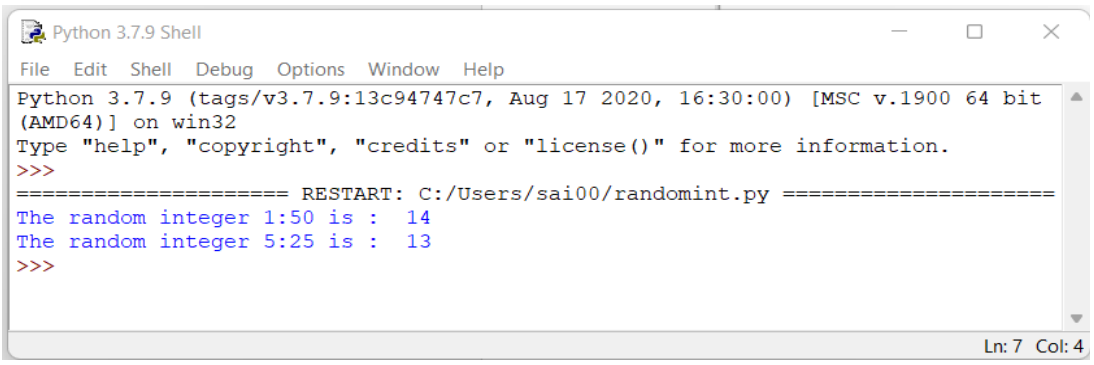
The random.randint() function generates a random integer from the range of numbers supplied.

Example:


randomint.py - C:/Users/sai00/randomint.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 12 Col: 31

```
# Python program for generating random integer

import random

x=random.randint(1,50)

print("The random integer 1:50 is : ", x)


x=random.randint(5,25)

print("The random integer 5:25 is : ",x)
```

Output:



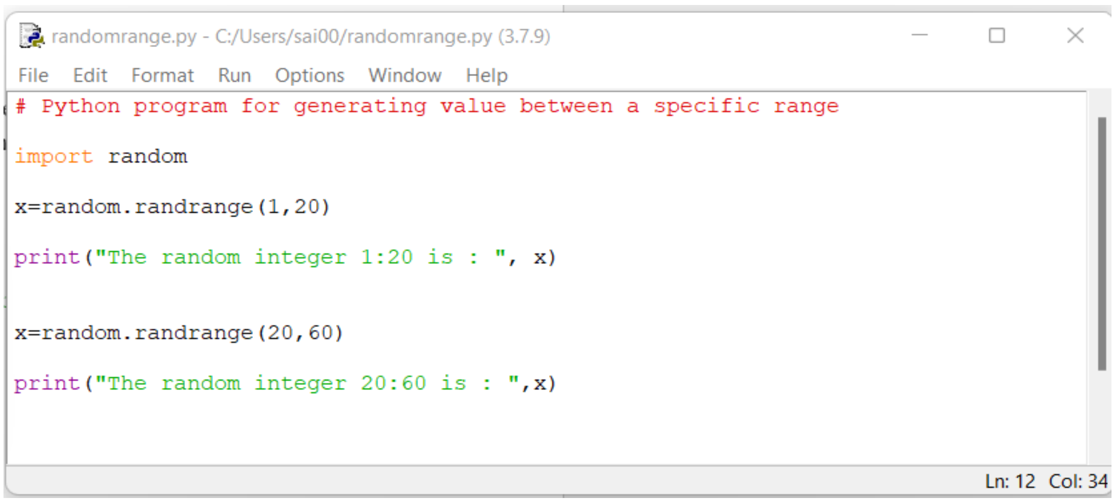
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/randomint.py ====================
The random integer 1:50 is :  14
The random integer 5:25 is :  13
>>>
```

In the above code imported random module, and generated random integer number with randomint() function. Two times called those functions and generated different random integer numbers every time 14 and 13 among different ranges respectively.

12.4.2. randrange() Function

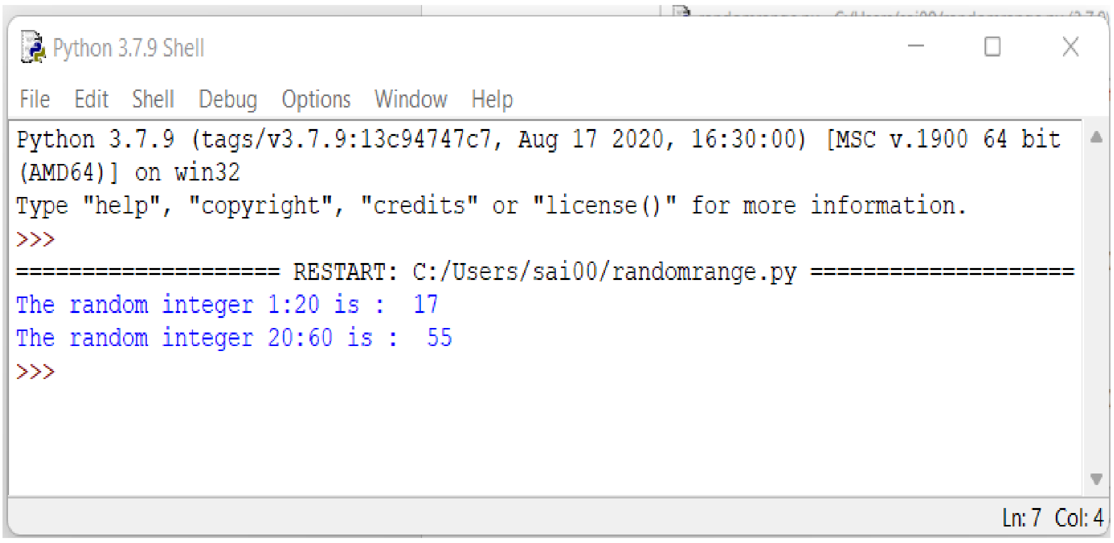
The function selects an item randomly from the given range defined by the start, the stop, and the step parameters. By default, the start is set to 0. Likewise, the step is set to 1 by default.

Example:


randomrange.py - C:/Users/sai00/randomrange.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 12 Col: 34

```
# Python program for generating value between a specific range

import random

x=random.randrange(1,20)

print("The random integer 1:20 is : ", x)


x=random.randrange(20,60)

print("The random integer 20:60 is : ",x)
```

Output:



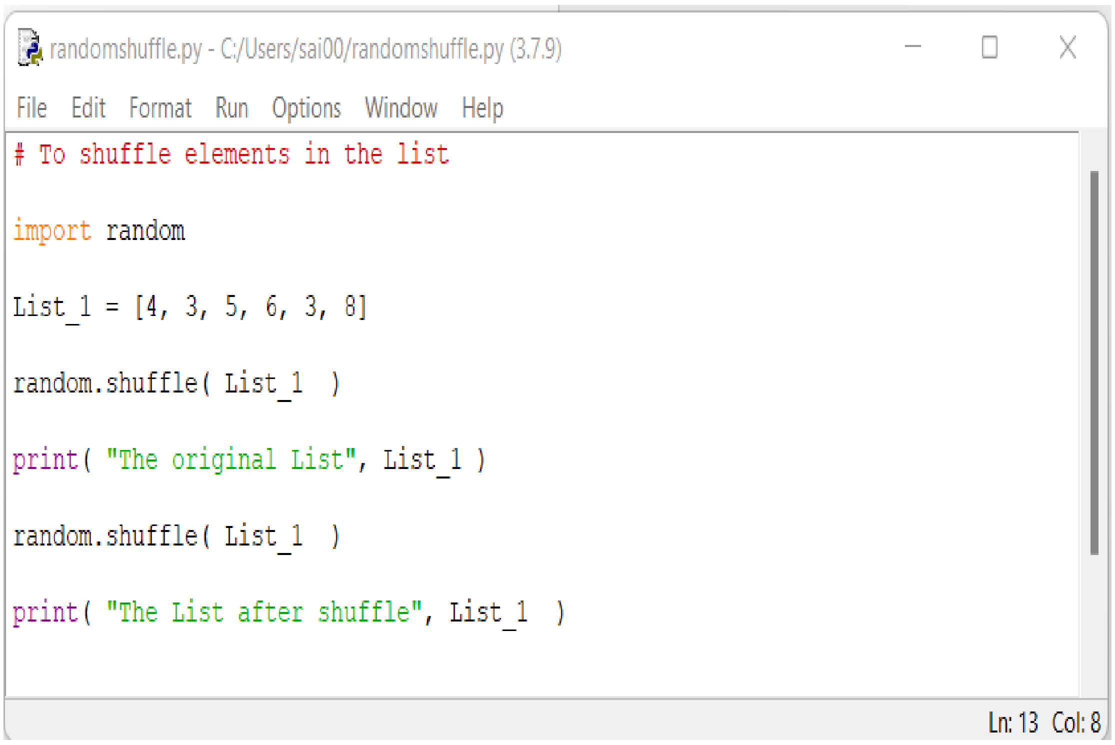
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/randomrange.py ===================
The random integer 1:20 is :  17
The random integer 20:60 is :  55
>>>
```

In the code that was just presented, the random module was imported, and the randomrange() method was used to generate a random integer value. By calling those functions twice, we were able to generate distinct random integer numbers each time, with 17 and 55 being derived from a different range of values, respectively.

12.4.3. shuffle () Function

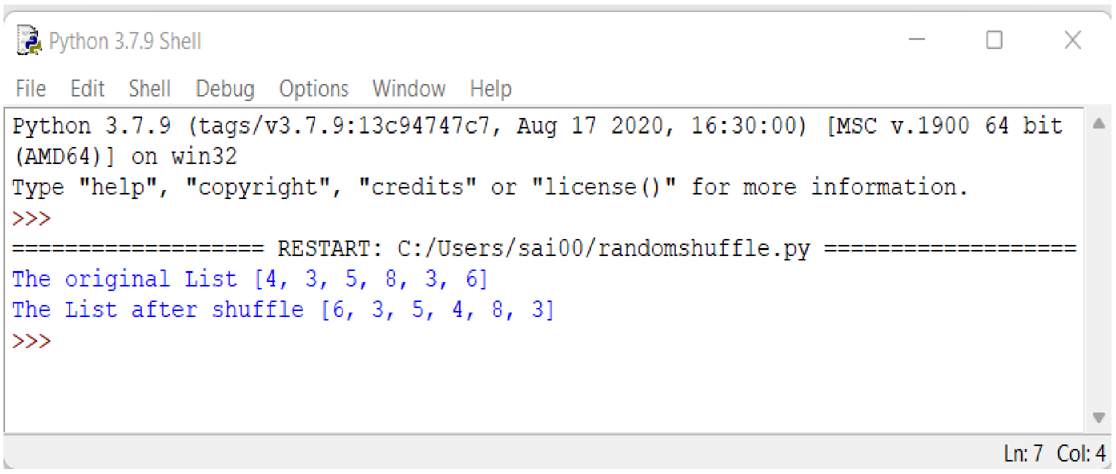
The random.shuffle() function shuffles the given list randomly.

Example:


randomshuffle.py - C:/Users/sai00/randomshuffle.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 13 Col: 8

```python
# To shuffle elements in the list

import random

List_1 = [4, 3, 5, 6, 3, 8]

random.shuffle( List_1  )

print( "The original List", List_1 )

random.shuffle( List_1  )

print( "The List after shuffle", List_1  )
```

Output:



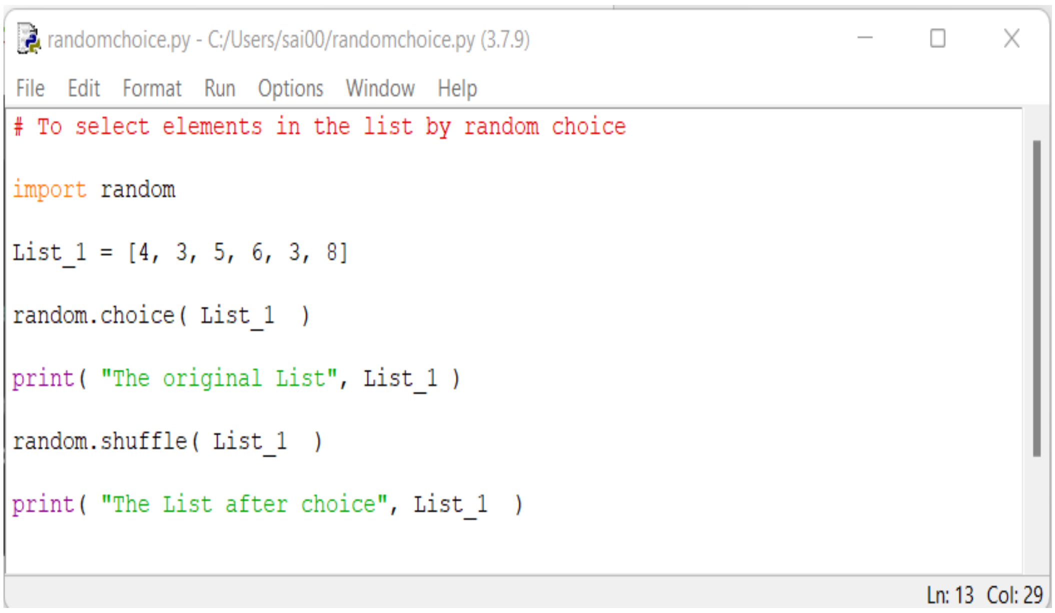
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/randomshuffle.py ==================
The original List [4, 3, 5, 8, 3, 6]
The List after shuffle [6, 3, 5, 4, 8, 3]
>>>
```

In the code that was just presented, the random module was imported, and the shuffle () method was used to shuffle a random list of numbers. By calling that function list of numbers swapped and we can apply this functions to other types of list with different data types also. The list after and before shuffle is shown in output respectively.

12.4.4. choice () Function

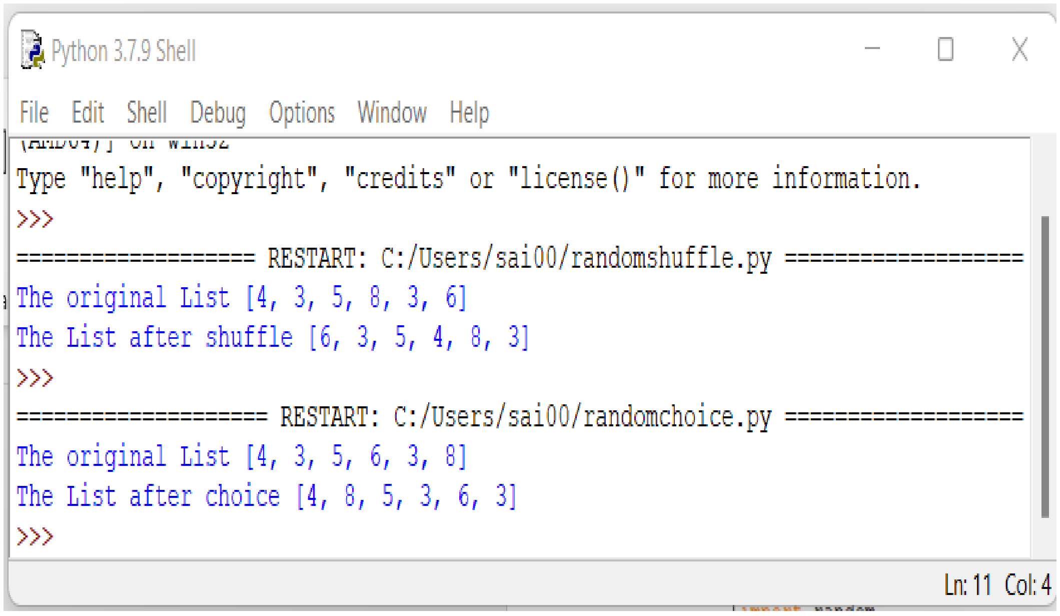
Utilizing the 'choice ()' function is a helpful tool to have at your disposal when you need to select a random element from a sequence. In the given example, we make use of the 'choice()' function to pick a color at random from the list of colors.In the real world, recommendation systems make use of the choice() function to select items at random from a list. These systems are used to make suggestions to consumers regarding products, movies, or music depending on their tastes. Additionally, these systems ensure that material on websites and apps will remain current.

Example:


randomchoice.py - C:/Users/sai00/randomchoice.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 13 Col: 29

```python
# To select elements in the list by random choice

import random

List_1 = [4, 3, 5, 6, 3, 8]

random.choice( List_1  )

print( "The original List", List_1 )

random.shuffle( List_1  )

print( "The List after choice", List_1  )
```

Output:

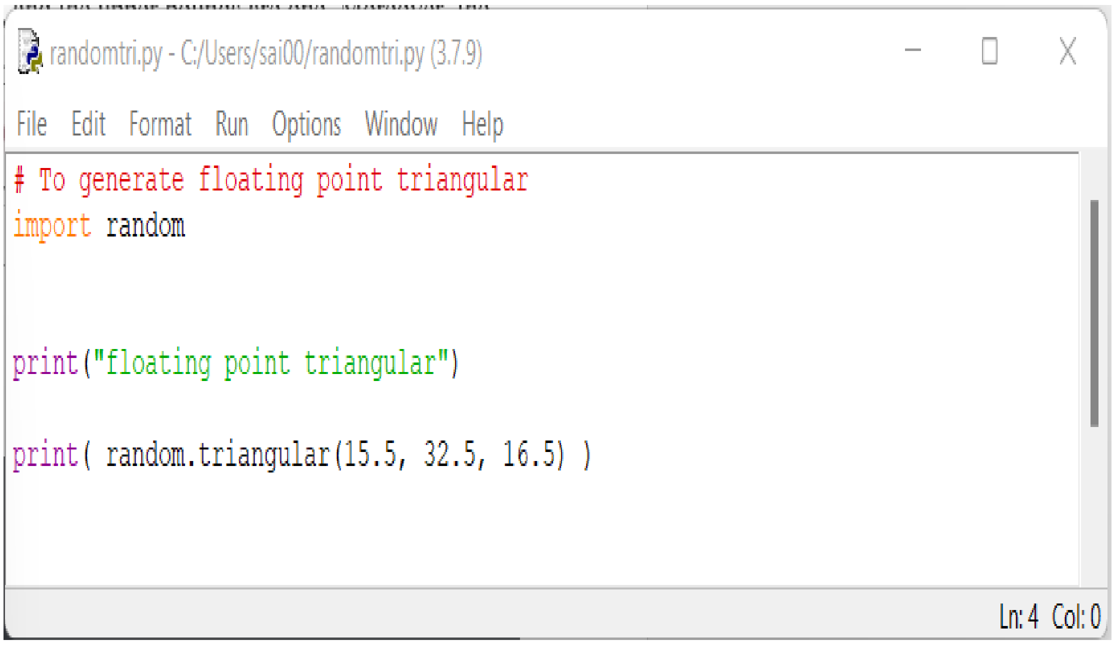


```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/randomshuffle.py ==================
The original List [4, 3, 5, 8, 3, 6]
The List after shuffle [6, 3, 5, 4, 8, 3]
>>>
=================== RESTART: C:/Users/sai00/randomchoice.py ==================
The original List [4, 3, 5, 6, 3, 8]
The List after choice [4, 8, 5, 3, 6, 3]
>>>
```

12.4.5. random.triangular( ) Function

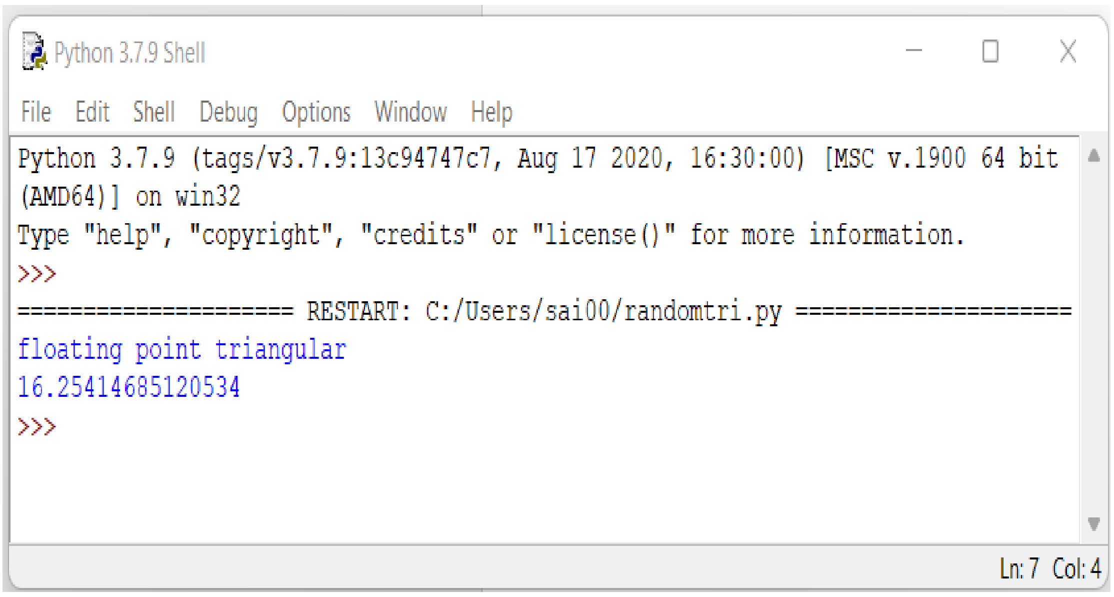
A random floating-point integer N is returned by the random.triangular() function. This number is chosen in such a way that lower is equal to N and upper is equal to N, and the mode that is supplied falls between these two bounds.When a lower bound is not specified, the default value is ZERO, and the upper bounds are set to one. Additionally, the peak parameter by default is set to the point that is in the middle of the boundaries, which results in a distribution that is symmetric.If you want to use these numbers in a simulation, you can generate random numbers for triangular distribution by using the random.triangular() function. in other words, to get value from a probability distribution that is triangular.

Example:


randomtri.py - C:/Users/sai00/randomtri.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 4 Col: 0

```
# To generate floating point triangular
import random


print("floating point triangular")

print( random.triangular(15.5, 32.5, 16.5) )
```

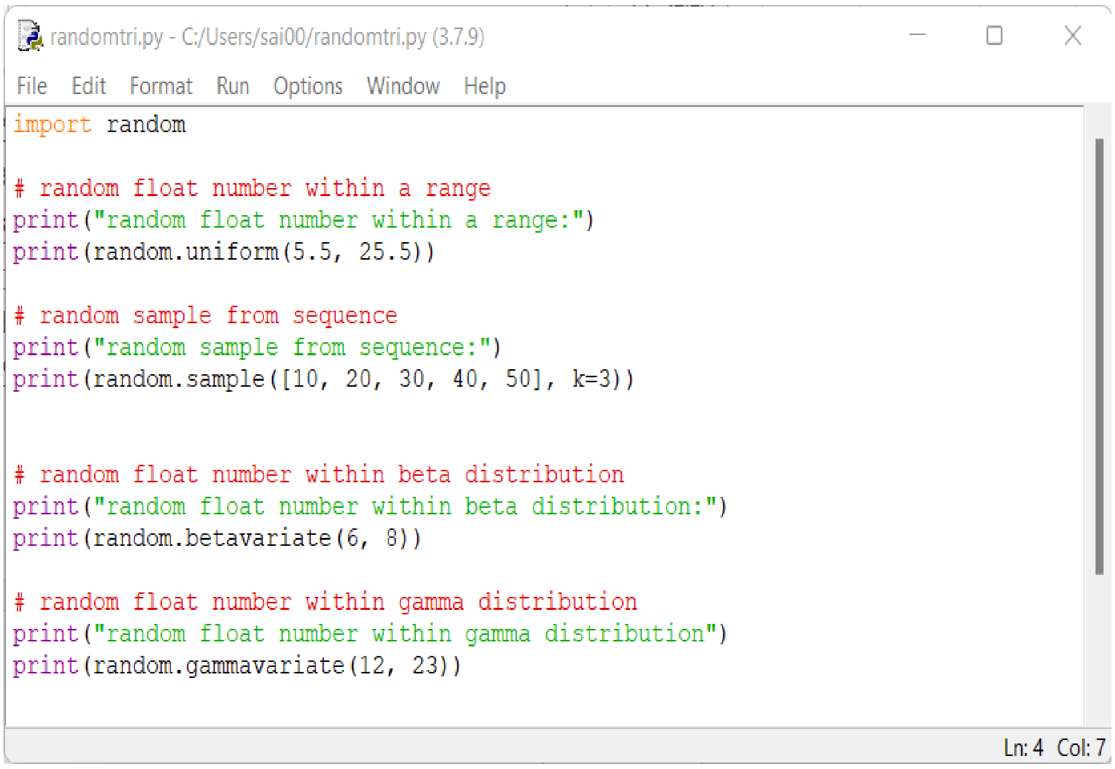
Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/randomtri.py ====================
floating point triangular
16.25414685120534
>>>
```

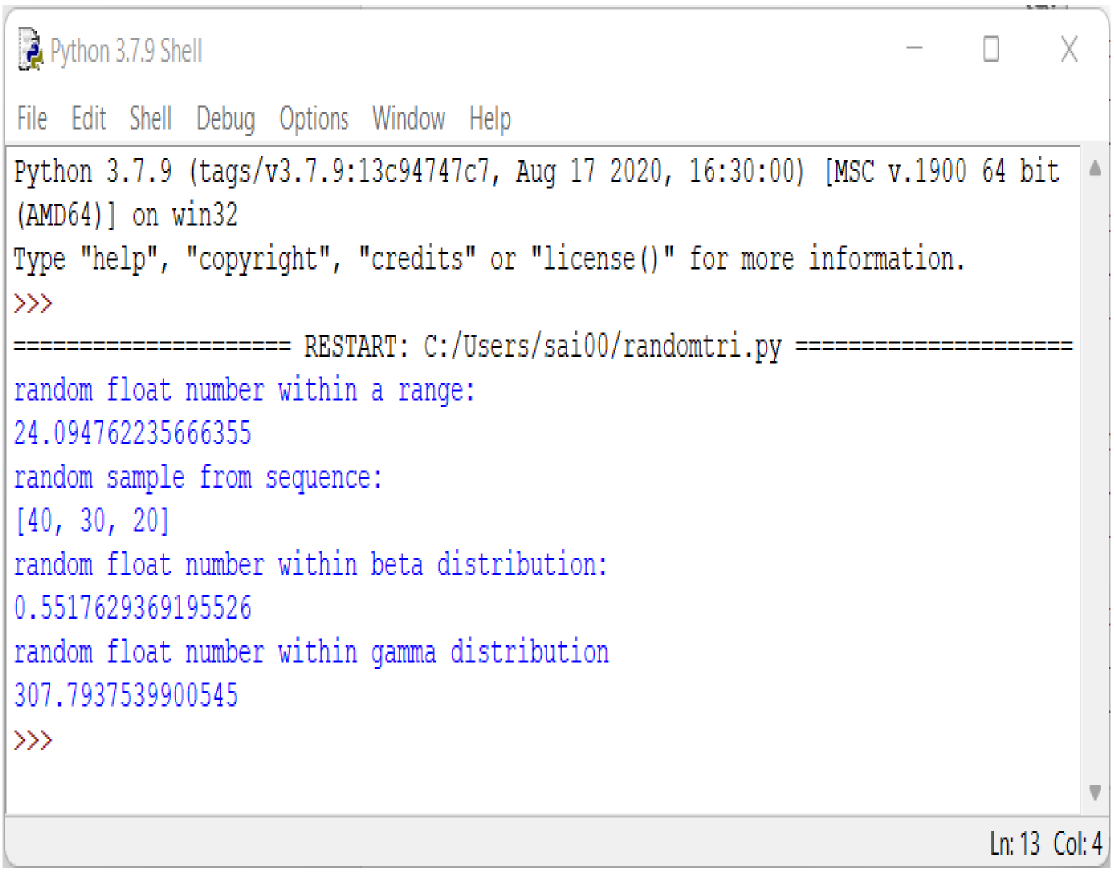
The random.uniform() function returns a random floating-point number between the range what we supplied. In the next example, generated 24.09 as float value reterived from the range(5.5,25.5). Other method is sample () which is pick random numbers from the given list of values based on the k value 3. The other two methods are betavariate() and gammavariate(0 are used to return float numbers with gamma and beta distribution respectively.

Example:


randomtri.py - C:/Users/sai00/randomtri.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 4 Col: 7

```python
import random

# random float number within a range
print("random float number within a range:")
print(random.uniform(5.5, 25.5))

# random sample from sequence
print("random sample from sequence:")
print(random.sample([10, 20, 30, 40, 50], k=3))


# random float number within beta distribution
print("random float number within beta distribution:")
print(random.betavariate(6, 8))

# random float number within gamma distribution
print("random float number within gamma distribution")
print(random.gammavariate(12, 23))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/randomtri.py ====================
random float number within a range:
24.094762235666355
random sample from sequence:
[40, 30, 20]
random float number within beta distribution:
0.5517629369195526
random float number within gamma distribution
307.7937539900545
>>>
```

A useful tool for incorporating randomness into your programs and projects is the random module that is available in the Python programming language. The purpose of this module is to give a comprehensive collection of functions that will assist you in accomplishing your objectives, whether you are developing games, running simulations, or need to make judgments based on probabilities. The capabilities of the "random" module are extensive and diverse, ranging from the generation of random numbers with a variety of distributions to the shuffling of sequences and the selection of random pieces.

12.5 PYTHON MATH MODULE

Functions and constants in mathematics are the components that make up the Math Module. For mathematical activities, it is a built-in module that was developed. The math module includes the mathematical functions that are used to perform fundamental operations like addition (+), subtraction (-), multiplication (*), and division (/), as well as more complex operations like trigonometric, logarithmic, and exponential functions.

In the math module, the following is a list of all the mathematical functions that you can use whenever you find yourself in need of them in your program.However, in this section all functions are categorized into three types:

- Numeric Functions
- Logarithmic and Power Functions
- Trigonometric and Angular Functions

12.5.1 Numeric Functions

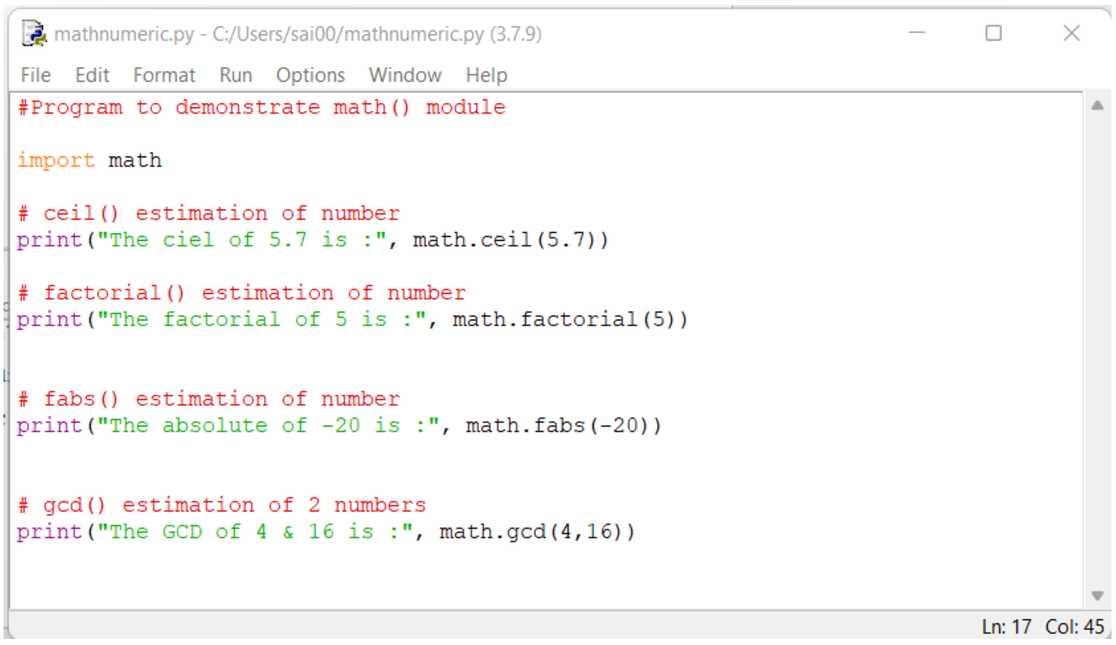
In this section, we will discuss the functions that are utilized in number theory as well as representation theory. For example, we will discuss how to determine the factorial of a number. Functions such as ceil(), floor(), factorial(), and fabs() are examples of some of the numerical functions. The Ceil value represents the integral value that is the smallest and greater than the number, whereas the floor value represents the integral value that is the greatest and smaller than the number overall. The ceil() and floor() methods, respectively, can be utilized to perform this calculation with relative ease. A single line of code is all that is required to get the factorial of a number when we make use of the factorial() method. If the integer is not integral, an error message will be provided. Fabs() is a function that returns the absolute value of the number. 3. Finding the GCD.

Table 12.2. Python Math Module Numeric Functions

Function	Description
ceil(x)	Returns the smallest integral value greater than the number
floor(x)	Returns the greatest integral value smaller than the number
factorial(x)	Returns the factorial of the number
fabs(x)	Returns the absolute value of the number
ceil(x)	Returns the smallest integral value greater than the number
floor(x)	Returns the highest integral value greater than the number
isfinite(x)	Check whether the value is neither infinity not Nan
isinf(x)	Check whether the value is infinity or not
isnan(x)	Returns true if the number is “nan” else returns false
ldexp(x, i)	Returns x * (2**i)
modf(x)	Returns the fractional and integer parts of x
trunc(x)	Returns the truncated integer value of x
gcd(x, y)	Compute the greatest common divisor of 2 numbers

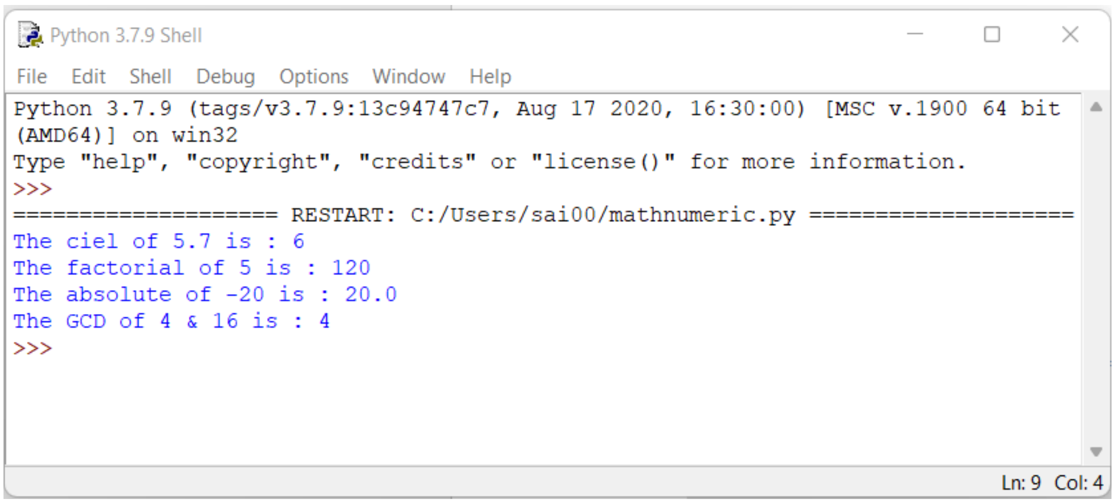
The math module is imported into this code, the value 5.7 is assigned to the variable a, and then the ceiling and floor of an are calculated and printed out according to the results. First, the math module is imported into the program, then the value 5 is assigned to the variable a, and last, the factorial of an is computed and finally printed. In this piece of code, the math module is imported, the values 4 and 16 are assigned to the variables a and b, respectively, and then the greatest common divisor (GCD) of a and b is calculated and printed out. Importing the math module, assigning the value -20to the variable a, and then calculating and printing the absolute value of an are all operations that are performed by this code.

Example:


mathnumeric.py - C:/Users/sai00/mathnumeric.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 17 Col: 45

```
#Program to demonstrate math() module

import math

# ceil() estimation of number
print("The ciel of 5.7 is :", math.ceil(5.7))

# factorial() estimation of number
print("The factorial of 5 is :", math.factorial(5))


# fabs() estimation of number
print("The absolute of -20 is :", math.fabs(-20))


# gcd() estimation of 2 numbers
print("The GCD of 4 & 16 is :", math.gcd(4,16))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/mathnumeric.py ===================
The ciel of 5.7 is : 6
The factorial of 5 is : 120
The absolute of -20 is : 20.0
The GCD of 4 & 16 is : 4
>>>
```

12.5.2 Logarithmic and Power Functions

It is possible to write power functions using the notation pow(x,n), where n represents the power of x. On the other hand, logarithmic functions are believed to be the inverse of exponential functions. Among the functions is the pow() function, which is responsible for computing x**y. Following the conversion of its parameters into float, this function then computes the power of the argument.

- The logarithmic value of an as expressed with base b is returned by the log() function. It is the natural log that is used to compute the value if the base is not specified.

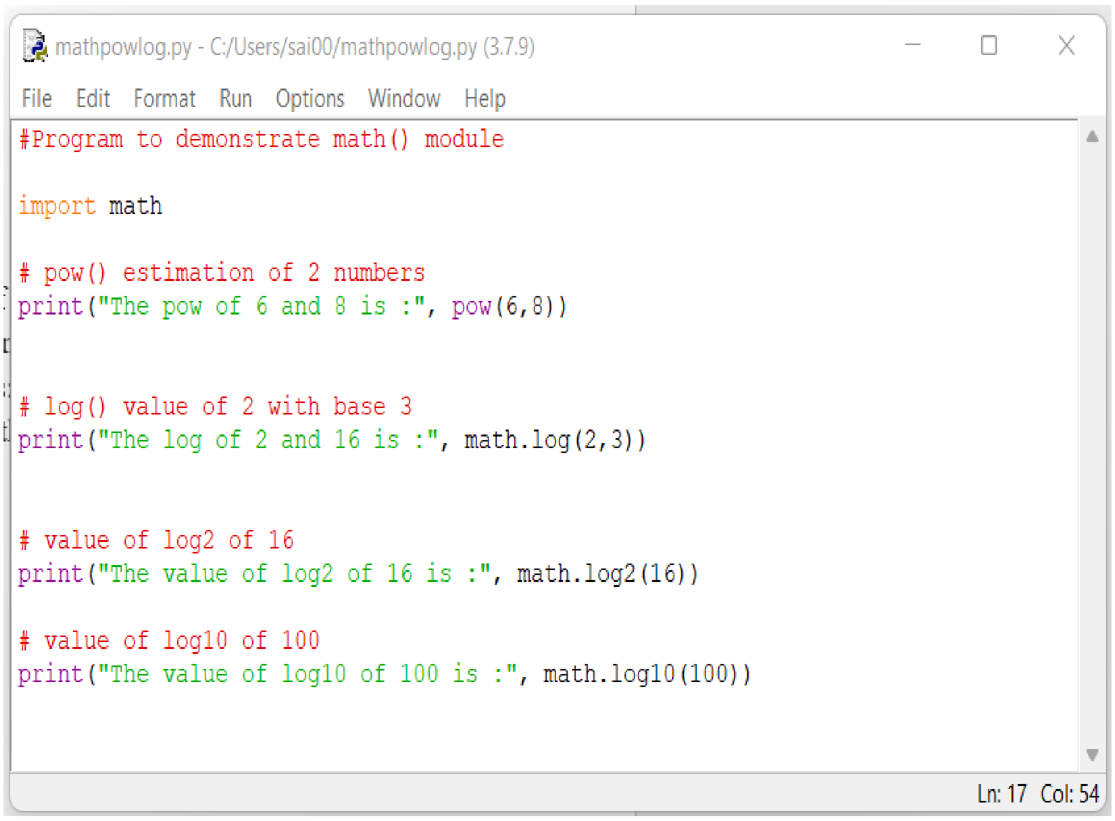
- With base 2, the log2(a) function calculates the value of the logarithm of a. The accuracy of this number is higher than that of the value of the function that was explained earlier.

- With base 10, the log10(a) function calculates the value of the logarithm of a. The accuracy of this number is higher than that of the value of the function that was explained earlier.

Table 12.3 Math Module Logarithmic and Power Functions

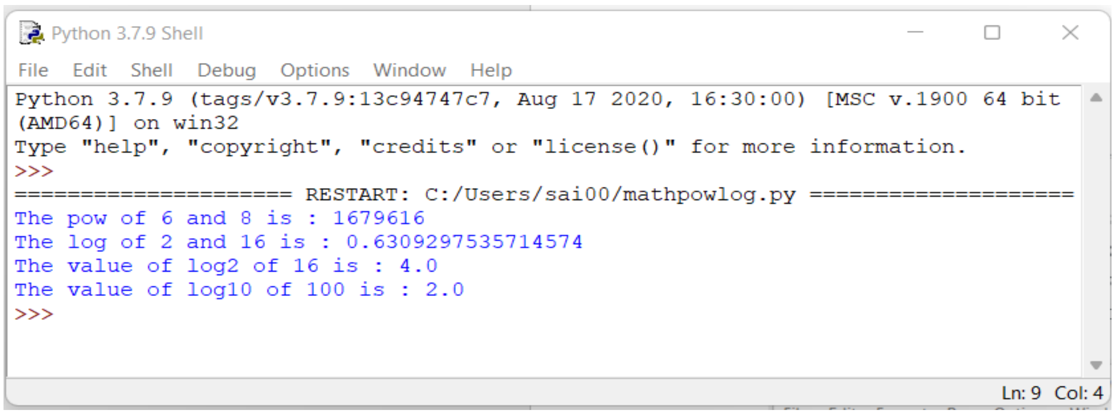
Function	Description
exp(x)	Returns the value of e raised to the power x(e**x)
expm1(x)	Returns the value of e raised to the power a (x-1)
log(x[, b])	Returns the logarithmic value of a with base b
log1p(x)	Returns the natural logarithmic value of 1+x
log2(x)	Computes value of log a with base 2
log10(x)	Computes value of log a with base 10
pow(x, y)	Compute value of x raised to the power y (x**y)

Example:


mathpowlog.py - C:/Users/sai00/mathpowlog.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 17 Col: 54

```
#Program to demonstrate math() module

import math

# pow() estimation of 2 numbers
print("The pow of 6 and 8 is :", pow(6,8))


# log() value of 2 with base 3
print("The log of 2 and 16 is :", math.log(2,3))


# value of log2 of 16
print("The value of log2 of 16 is :", math.log2(16))

# value of log10 of 100
print("The value of log10 of 100 is :", math.log10(100))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/mathpowlog.py ====================
The pow of 6 and 8 is : 1679616
The log of 2 and 16 is : 0.6309297535714574
The value of log2 of 16 is : 4.0
The value of log10 of 100 is : 2.0
>>>
```

First, the math module is imported into the program, and then the logarithms of three different values are computed and printed accordingly. Logarithms can be worked with using several functions that are provided by the math module. These functions include log(), log2(), and log10().The pow(6,8) is estimated with and then various log( ) functions with different base values are tested respectively. The complete results is shown in output.

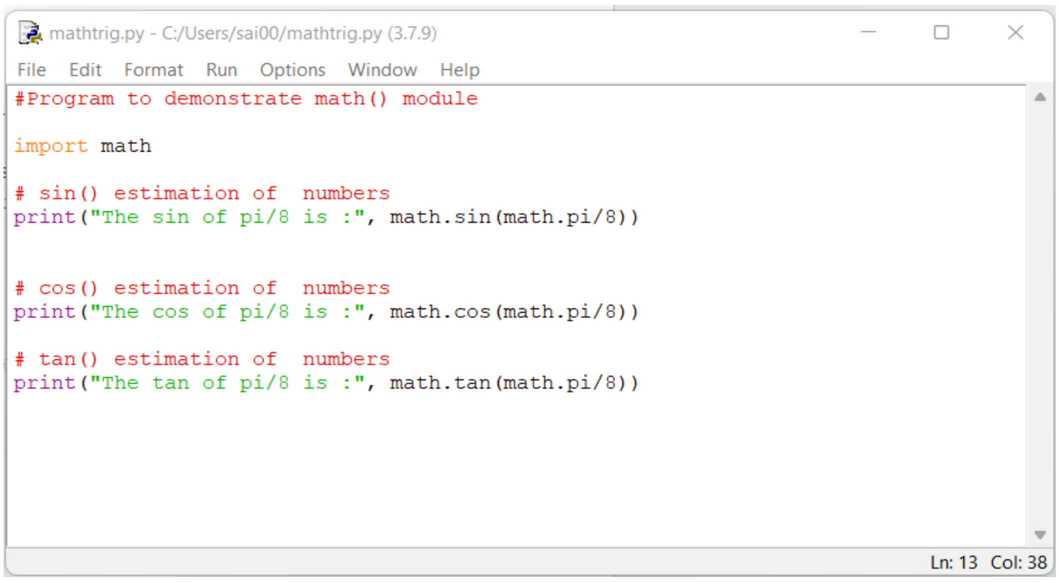
12.5.3 Trigonometric and Angular Functions

You are all required to be familiar with trigonometry and the fact that it can be challenging to determine the sine and cosine values of any angle. The math module encompasses built-in functions that allow for the discovery of such values and even the modification of values between degrees and radians. The sine, cosine, and tangent of the value that was supplied as an argument are returned by the sin(), cos(), and tan() methods, respectively. The radians should be used for the value that is passed through this function.

Table 12.4 Math Module Trigonometric and Angular Functions

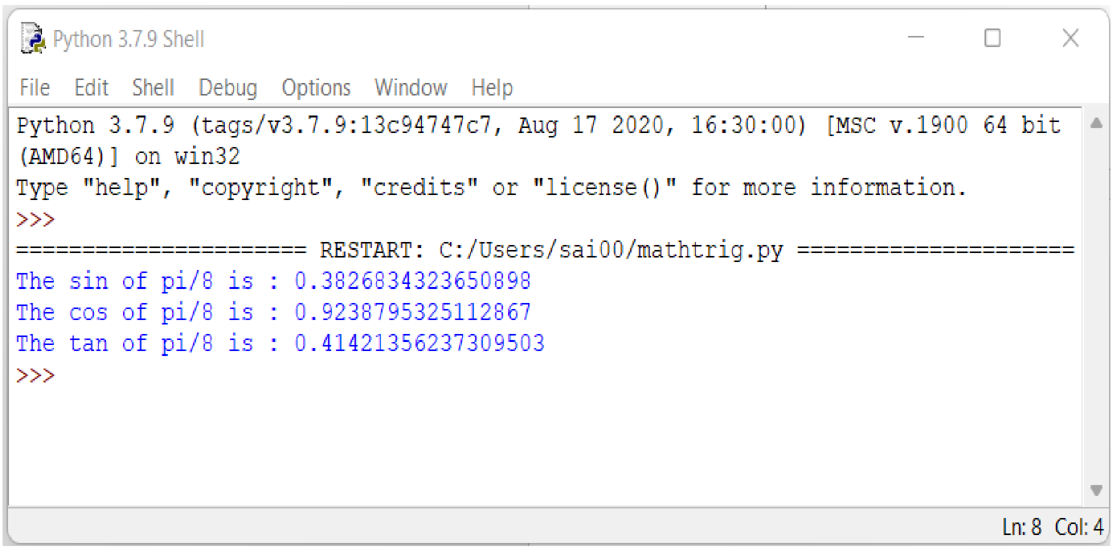
Function Name	Description
acos(x)	Returns the arc cosine of value passed as argument
asin(x)	Returns the arc sine of value passed as argument
atan(x)	Returns the arc tangent of value passed as argument
atan2(y, x)	Returns atan(y / x)
cos(x)	Returns the cosine of value passed as argument
hypot(x, y)	Returns the hypotenuse of the values passed in arguments
sin(x)	Returns the sine of value passed as argument
tan(x)	Returns the tangent of the value passed as argument
gamma(x)	Return the gamma function of the argument
lgamma(x)	Return the natural log of the absolute value of the gamma function

Example:


mathtrig.py - C:/Users/sai00/mathtrig.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 13 Col: 38

```
#Program to demonstrate math() module

import math

# sin() estimation of  numbers
print("The sin of pi/8 is :", math.sin(math.pi/8))


# cos() estimation of  numbers
print("The cos of pi/8 is :", math.cos(math.pi/8))

# tan() estimation of  numbers
print("The tan of pi/8 is :", math.tan(math.pi/8))
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Users/sai00/mathtrig.py =====================
The sin of pi/8 is : 0.3826834323650898
The cos of pi/8 is : 0.9238795325112867
The tan of pi/8 is : 0.41421356237309503
>>>
```

Initially, this code imports the math module, which is responsible for providing a wide range of mathematical functions. Afterwards, it establishes sin value of pi/8. Pi is the mathematical constant that represents the ratio of the circumference of a circle to its diameter. Similarly performed with cos() and tan() values of pi/8.

12.6 SUMMARY

To work with OOP in Python, classes and methods are extremely significant concepts they contribute to the creation of code that is not only comprehensible but also reusable. By building a class, you have the ability to put together a collection of information and capabilities into a single entity that can be utilized in the construction of a variety of things. You can have access to the methods and properties of an object after it has been formed by utilizing the dot notation. By having a solid understanding of Python's classes and objects, you will be able to create code that is more logical, efficient, and maintainable.

12.7 TECHNICAL TERMS

Module, Import Package, Random Module, Math Module, Gamma Distribution, Logarithmic Operations, Exponential Operations.

12.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. How is python module created? Explain.
2. What are the various ways to import python module? Explain.
3. Explain different functions in Random Module.

Short Notes:

1. Define Python Module?
2. List functions in Math Module.

12.9 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. Python Software Foundation- Python Documentation. Retrieved from https://docs.python.org/3/

Dr. U. SURYA KAMESWARI

LESSON - 13

PYTHON CLASSES AND OBJECTS

AIMS AND OBJECTIVES

The student will be able to understand the significance of classes and objects in Python, describe how classes and objects created, and explain how it is processed through the course of this chapter. A discussion of the class variables and object variables also focused here. Following the completion of this chapter, you will be able to provide an explanation of the differences between the public and private data members. An explanation of how built-in class attributes works also be aware by the students in python.

STRUCTURE

13.1. INTRODUCTION

The programming paradigm known as object-oriented programming (OOPs) is utilized in Python. This paradigm makes use of objects and classes in the programming process. Inheritance, polymorphisms, encapsulation, and other real-world elements are some of the concepts that it intends to incorporate into the programming. One of the most important ideas behind object-oriented programming is to link the data and the functions that act on that data together as a single unit. This ensures that no other part of the code may access the data. The following is a list of some of the OOP's concepts that are supported by Python and are shown in Figure 13.1:

- Objects
- Classes
- Polymorphism
- Encapsulation
- Inheritance

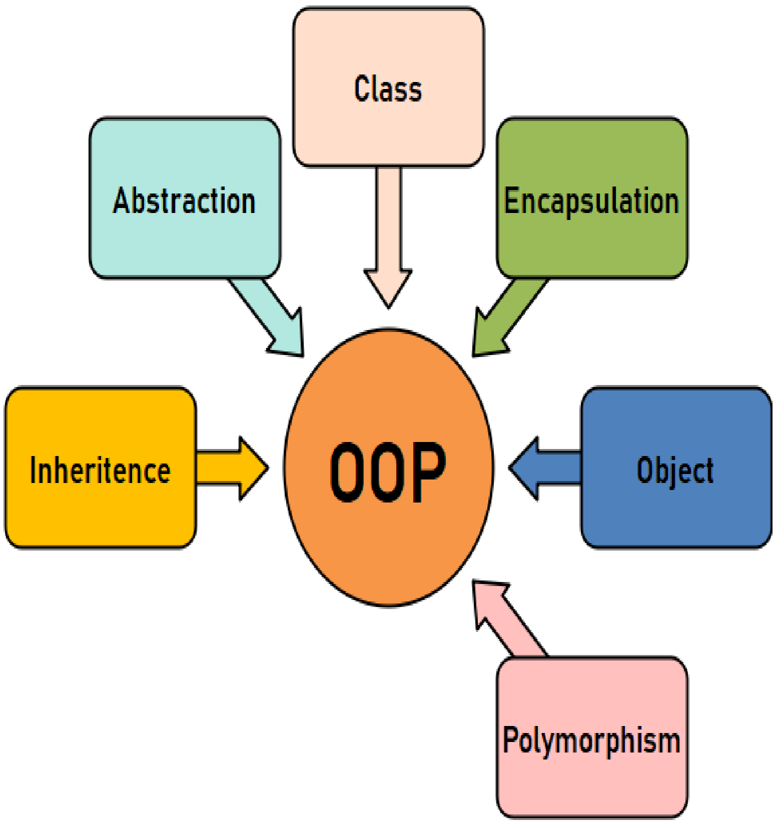


Fig 13.1. OOP Concepts in Python

13.2. PYTHON CLASS

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods. To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, and age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it’s the exact need for classes.

13.2. Creating a Python Class with Self argument

Class is a combination of set of attributes and methods. However, the methods are useful to process or perform specific operations over attributes. Classes are created by keyword called **class. The a**ttributes are the variables that belong to a class. Attributes are always public and can be accessed using the dot (.) operator.

Syntax:

```
class Class Name:
  # Statement-1
  …
  …
  ….
  # Statement-N
```

Example:

```
class Student:
        def __init__(self, name, age,marks):
                self.name = name
                self.age = age
                self.marks = marks
s1 = Student("Rama", 15, 80)
s2 = Student(“Krishna”, 12, 60)
print(s1.name)
print(s1.age)
print(s1.marks)
```

In the above example, a class named Student using the class keyword is created with the attributes name, age and marks. Class is initialized as self with the values of name=Rama, age=15 and marks=80 with object s1. Finally individual elements related to the student class are displayed. Similarly, other object s2 was also created and instantiated.

13.2.2. Advantages of Python Class

- Classes offer a convenient means of storing the data members and methods in a single location, which contributes to the program's overall organization and helps to keep it more organized.
- This object-oriented programming paradigm offers a few additional capabilities, one of which is inheritance, which may be accessed through the utilization of classes.
- Classes are also useful for overriding any standard operator that may be present.

- Reusing code is made possible by the utilization of classes, which results in the increase of the program's overall efficiency.
- Increasing the readability of the program can be accomplished by establishing a clear structure for the code by grouping functions that are related to one another and storing them in a single location (inside a class).

13.3. PYTHON OBJECTS

A state and a behavior are both associated with the object, which is an entity. It might be anything taken from the real world, such as a mouse, keyboard, chair, table, pen, or anything else. There are many different types of objects, including integers, texts, floating-point numbers, even arrays, and dictionaries. On a more specific level, an object can be defined as any single number or any single string.
The following three components are used to compose an object:

- **State:** The attributes of an object are what are used to represent the state of an object. At the same time, it reflects the characteristics of an object.
- **Behavior:** it is exemplified by the techniques that an item possesses. The way in which a thing reacts to other objects is another aspect that it represents.
- **Identity:** It provides an object with a name that is distinctly its own and makes it possible for one object to communicate with other objects.

Let us use the example of the class dog, which was explained earlier, to comprehend the identification, behavior, and state of the student. A possible interpretation of the identification is that it is the name of the student. The name, age, and marks of the student are all examples of attributes that can be the student's state. The behavior can be interpreted as indicating whether the student is reading or writing now.

13.3.1. Creating a Python Object

Using this, an object of the class Student, which was declared earlier, will be created with the name obj. Let's have a fundamental understanding of some terms that will be utilized while working with objects and classes before we delve into the specifics of objects and classes with example shown in Figure 13.2.

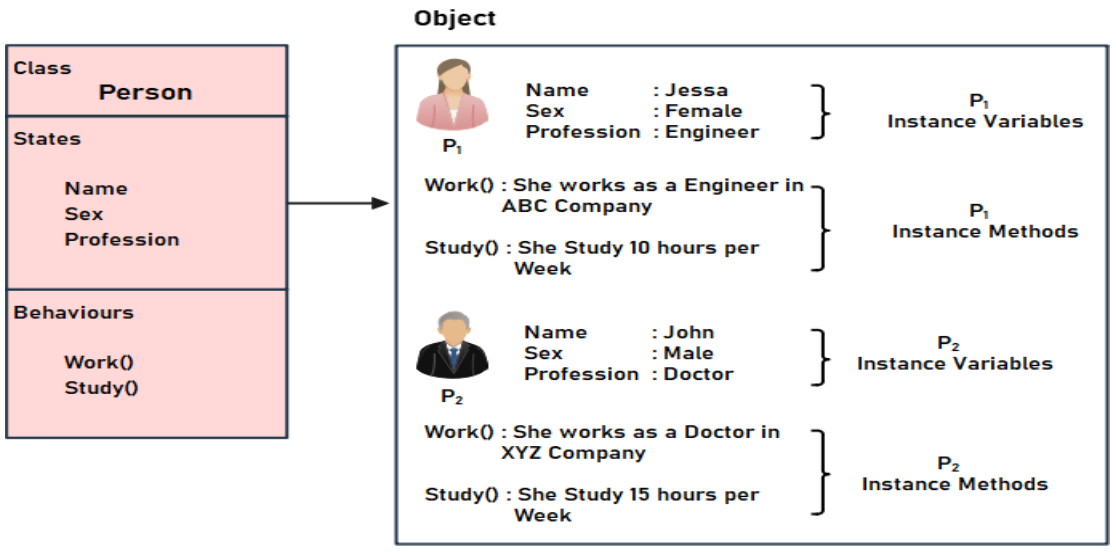
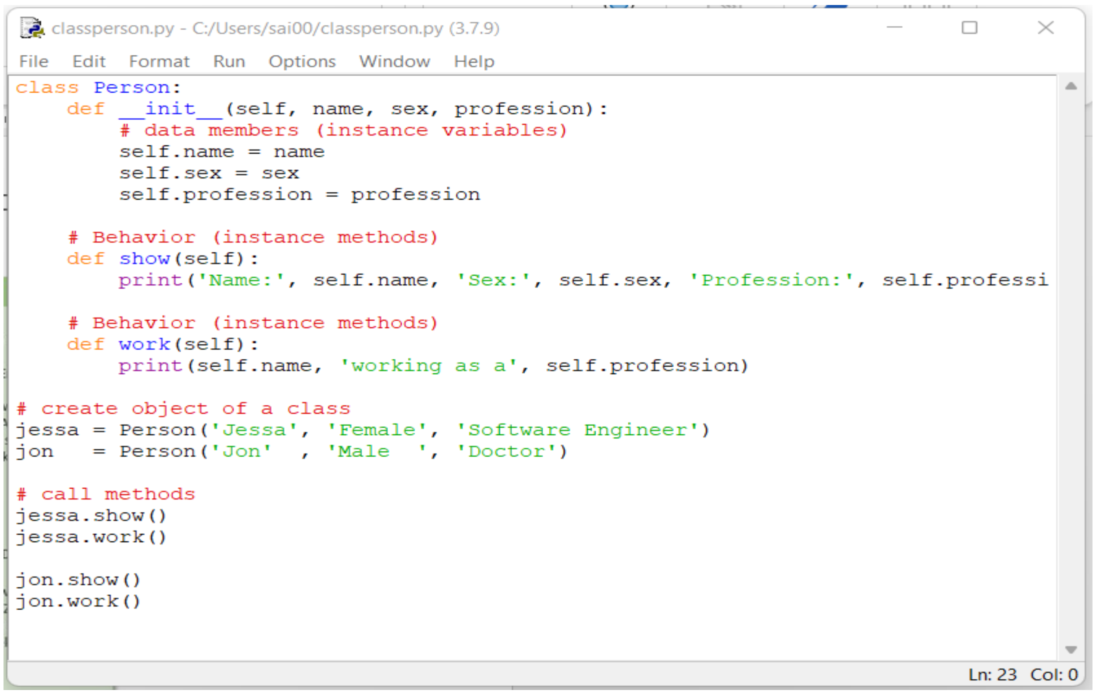


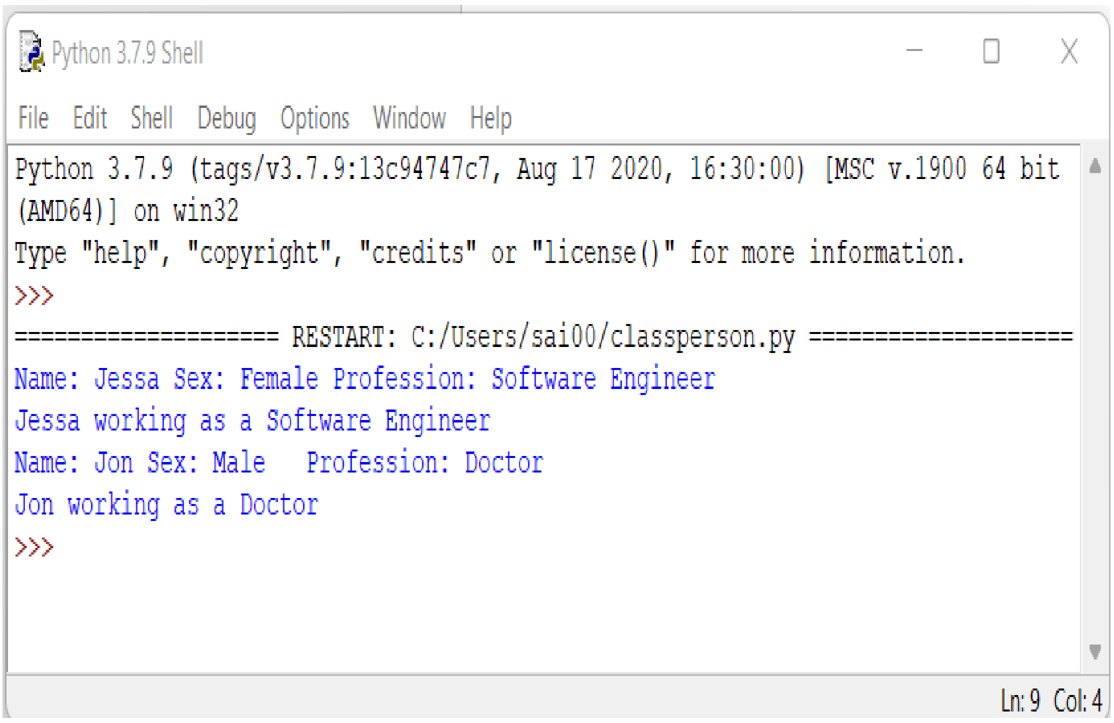
Fig 13.2. A Python Class and Object: Example

Example:


classperson.py - C:/Users/sai00/classperson.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 23 Col: 0

```
class Person:
    def __init__(self, name, sex, profession):
        # data members (instance variables)
        self.name = name
        self.sex = sex
        self.profession = profession

    # Behavior (instance methods)
    def show(self):
        print('Name:', self.name, 'Sex:', self.sex, 'Profession:', self.professi

    # Behavior (instance methods)
    def work(self):
        print(self.name, 'working as a', self.profession)

# create object of a class
jessa = Person('Jessa', 'Female', 'Software Engineer')
jon   = Person('Jon'  , 'Male  ', 'Doctor')

# call methods
jessa.show()
jessa.work()

jon.show()
jon.work()
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/classperson.py ===================
Name: Jessa Sex: Female Profession: Software Engineer
Jessa working as a Software Engineer
Name: Jon Sex: Male   Profession: Doctor
Jon working as a Doctor
>>>
```

In the above example the Person class is defined with the three states which includes **Name, Sex and Profession**. The behaviors of the Person class includes **work () and study ()**. After that two objects known as **'Jon'** and **'Jessa'** are created. The fact that Jessa is a woman and that she is employed as a software engineer is readily apparent. Jon, on the other hand, who is a male and a lawyer, is a different story. In this case, both objects are formed from the same class; nevertheless, their states and behaviors are distinct from one another.

13.3.2. Class Variable and Object Variable

Both instance variables and class variables are utilized during the process of designing a class and the example shown in Figure 13.3.

There are two distinct ways in which attributes can be defined in Class:

- Instance Variable: Bound to Object
- Class Variable: Bound to Class

Instance variables are attributes that are connected to an instance of a class. Instance variables are also known as instance variables. The constructor, often known as the __init__() method of a class, is where we define individual instance variables. No share to any other object. Every object has its own.

Class Variables: A class variable is a variable that is declared within a class, but it is not declared within any instance method or __init__() method.It a variable belongs to class so that it can be shared by every object associated with the class.

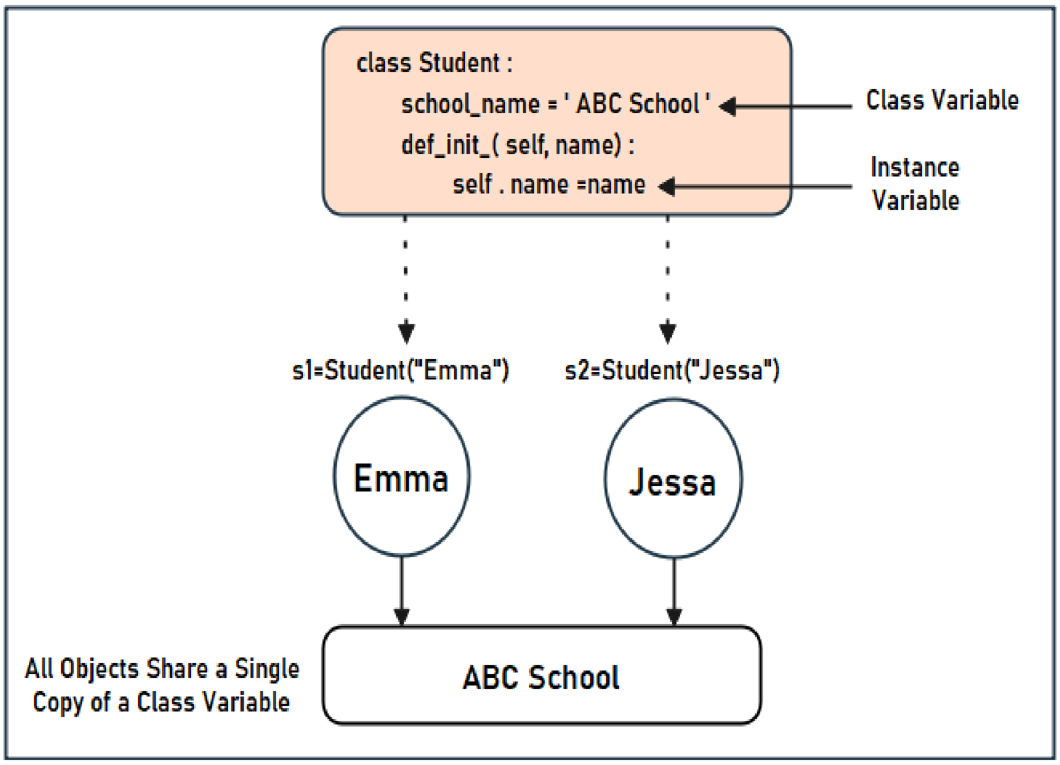


Fig 13.3. Pyhon Class Attributes Example

There is no sharing of instance attributes between objects. Every object, on the other hand, possesses its own copy of the instance attribute, which is exclusive to that item. The class variables are shared by each instance of a class. The value of a class variable, on the other hand, does not change from one instance variable to another, in contrast to instance variables.

13.4 CLASS METHODS

Within a class, we can define the three different kinds of methods that are described below in object-oriented programming and are shown in Figure 13.4.

- **Instance method:** This method is utilized to access or modify the state of the object. These methods are referred to as instance methods, and they are used when we employ instance variables within a method.
- **A class method** is a method that is used to access or modify the state of a class. When it comes to the implementation of methods, if we only use class variables, then we should declare such kinds of methods as class methods.
- **A static method**: is a general utility method that is used to carry out a task with no other component present. Because this static method does not have access to the class attributes, we do not make use of any instance or class variables within this method.

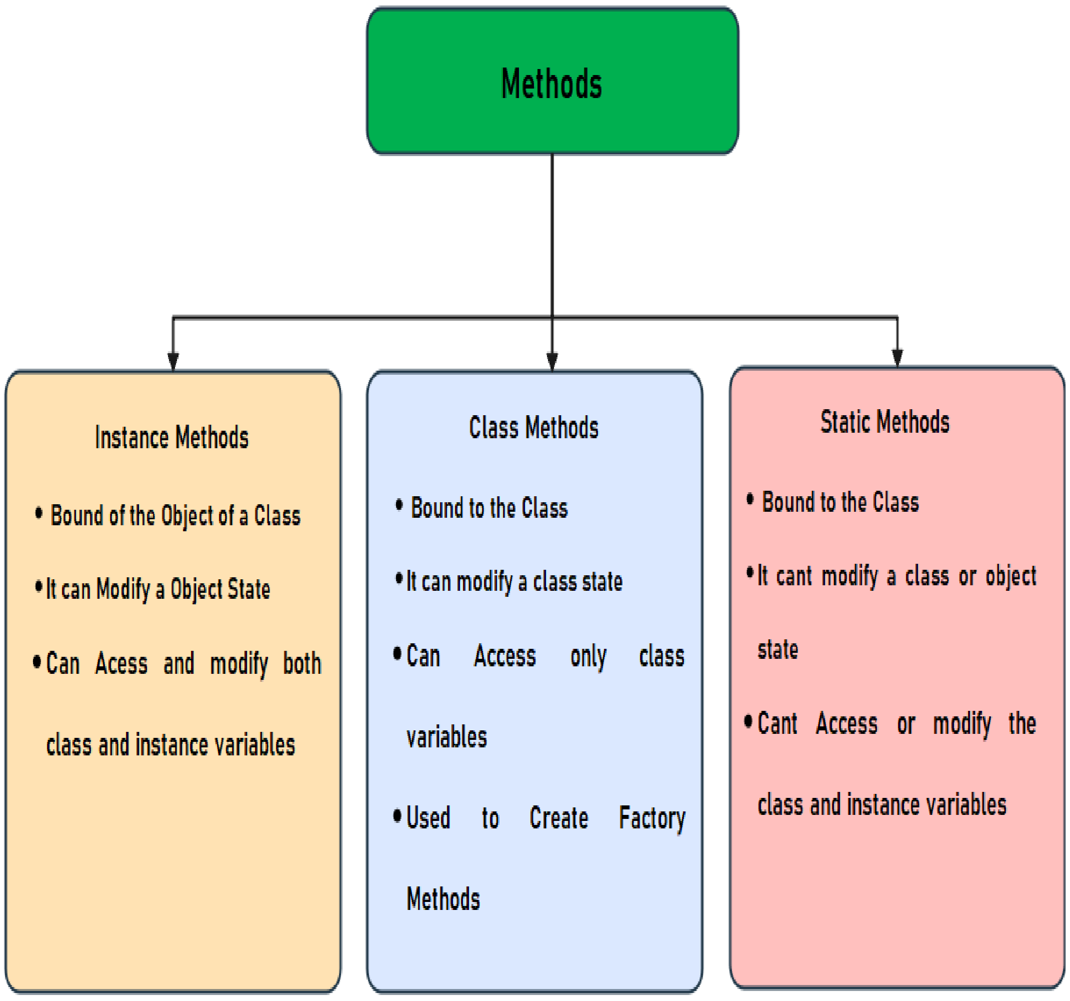
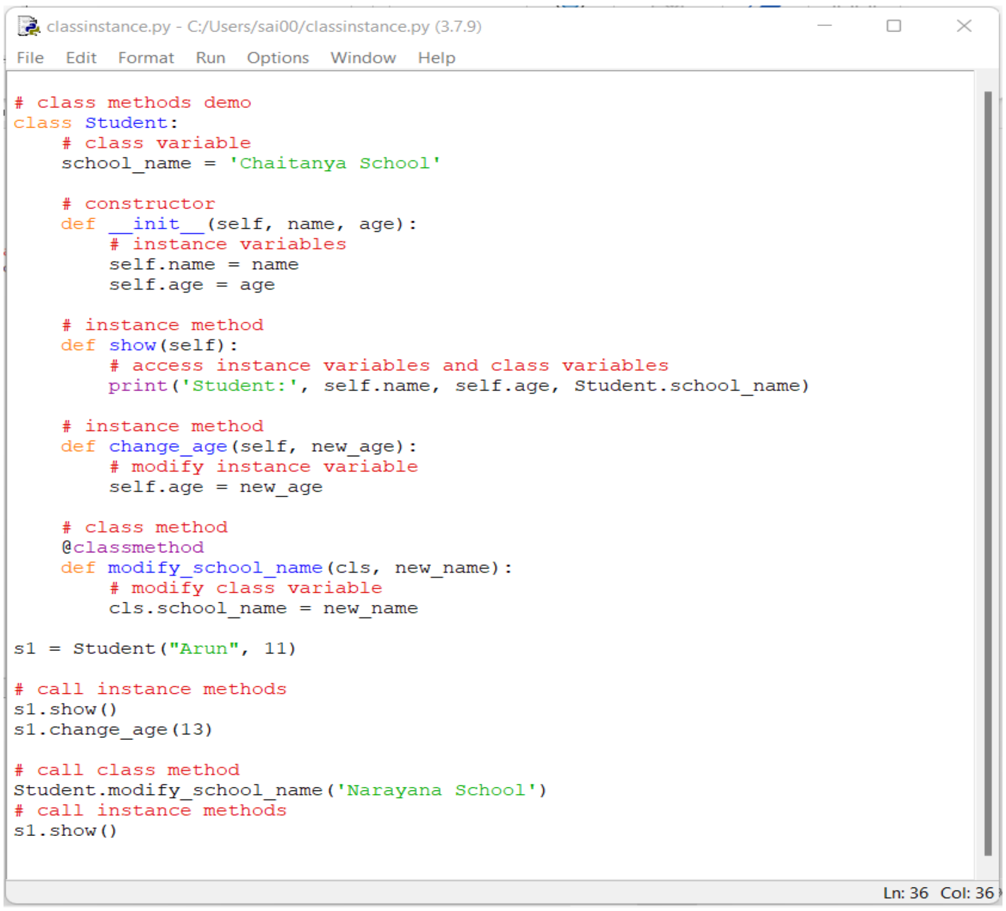

Methods
Instance Methods
• Bound of the Object of a Class
• It can Modify a Object State
• Can Acess and modify both class and instance variables
Class Methods
• Bound to the Class
• It can modify a class state
• Can Access only class variables
• Used to Create Factory Methods
Static Methods
• Bound to the Class
• It cant modify a class or object state
• Cant Access or modify the class and instance variables

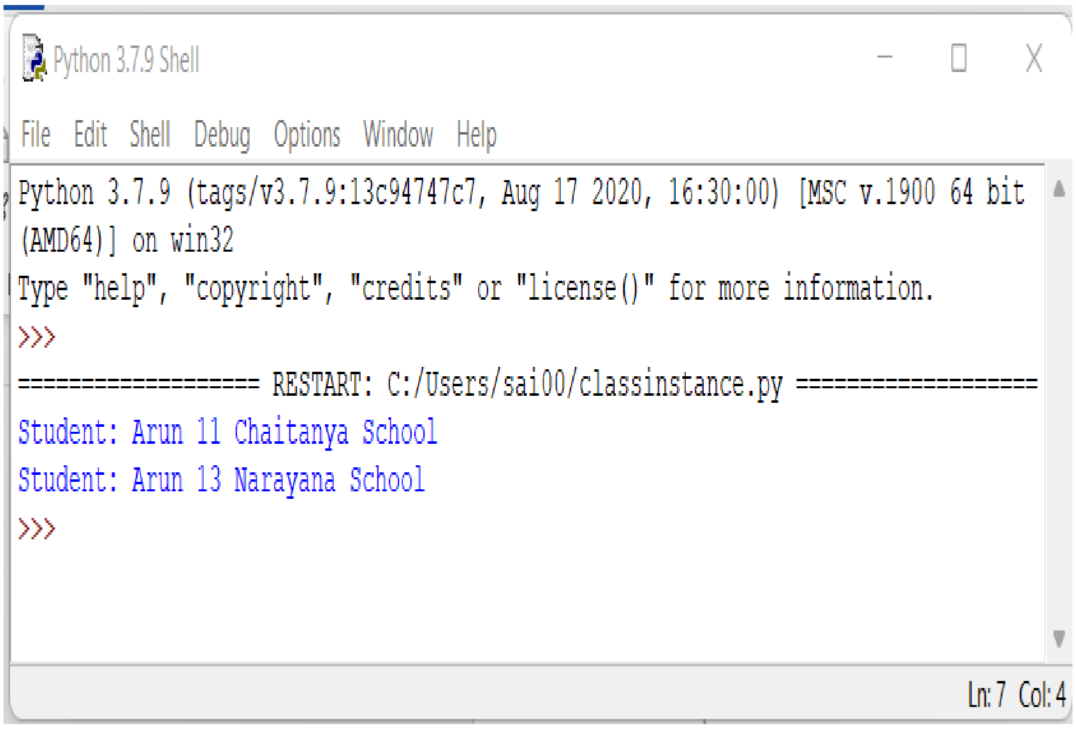
Fig 13.4. Python Class Methods

Example:


classinstance.py - C:/Users/sai00/classinstance.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 36 Col: 36

```
# class methods demo
class Student:
    # class variable
    school_name = 'Chaitanya School'

    # constructor
    def __init__(self, name, age):
        # instance variables
        self.name = name
        self.age = age

    # instance method
    def show(self):
        # access instance variables and class variables
        print('Student:', self.name, self.age, Student.school_name)

    # instance method
    def change_age(self, new_age):
        # modify instance variable
        self.age = new_age

    # class method
    @classmethod
    def modify_school_name(cls, new_name):
        # modify class variable
        cls.school_name = new_name

s1 = Student("Arun", 11)

# call instance methods
s1.show()
s1.change_age(13)

# call class method
Student.modify_school_name('Narayana School')
# call instance methods
s1.show()
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/classinstance.py ==================
Student: Arun 11 Chaitanya School
Student: Arun 13 Narayana School
>>>
```

At the instance level, also known as the object level, work. Take, for instance, the case when we have object that were formed from the student class known as **‘Arun’** . These object had **name, age and school_name**. Accessing and modifying the instance variables is possible through the utilization of instance methods. Hence with the help of the method known as **modify_shool_name()** changed name of the **school_ name** from **‘Chaitanya School ‘** to **‘Narayana School’** respectively.

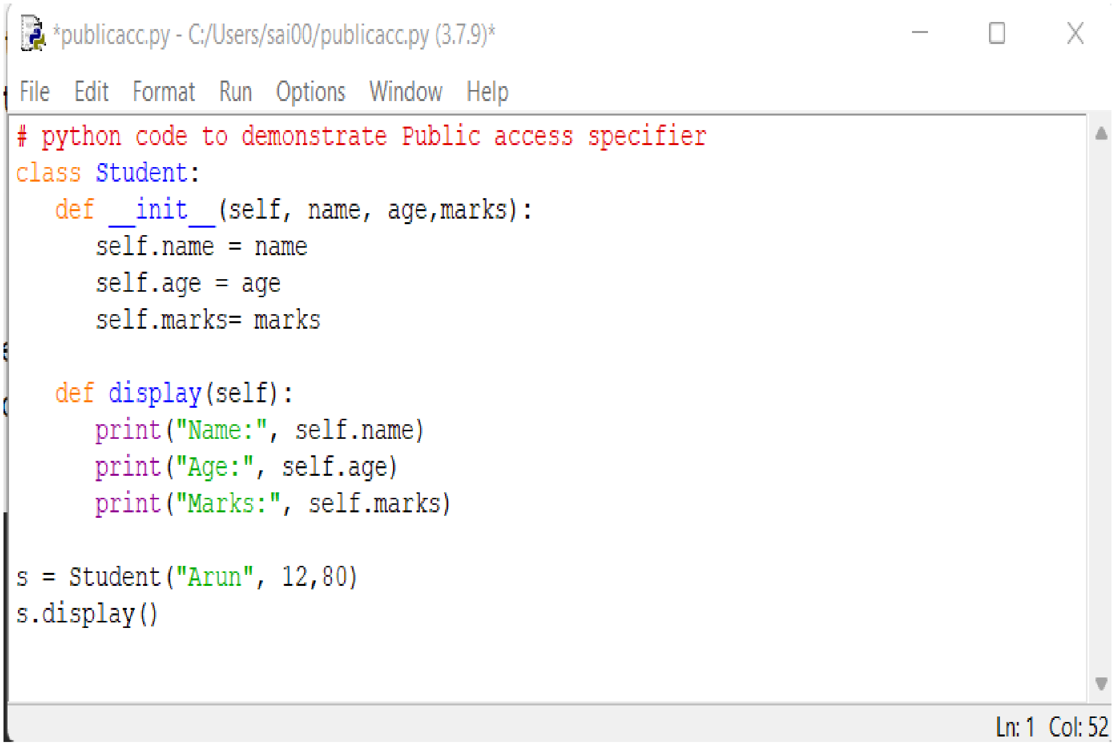
13.5 ACCESS SPECIFIERS

Python is a programming language that is known for its simplicity and ease of use, yet it is also known for its versatility and power. Python is distinguished from other programming languages by its support for object-oriented programming (OOP), which is one of the major aspects that sets it distinct. In the programming language Python, methods are functions that are linked to an object and describe the behavior of that object. Encapsulation is a principle of object-oriented programming that safeguards the data that is contained within a class by utilizing access modifiers. Public, private, and protected access modifiers are the three types of access modifiers that Python offers. These access modifiers place limits on the ability of any object that is not a member of the class to access the member variables and methods of the class.

13.5.1 Public Access Specifier

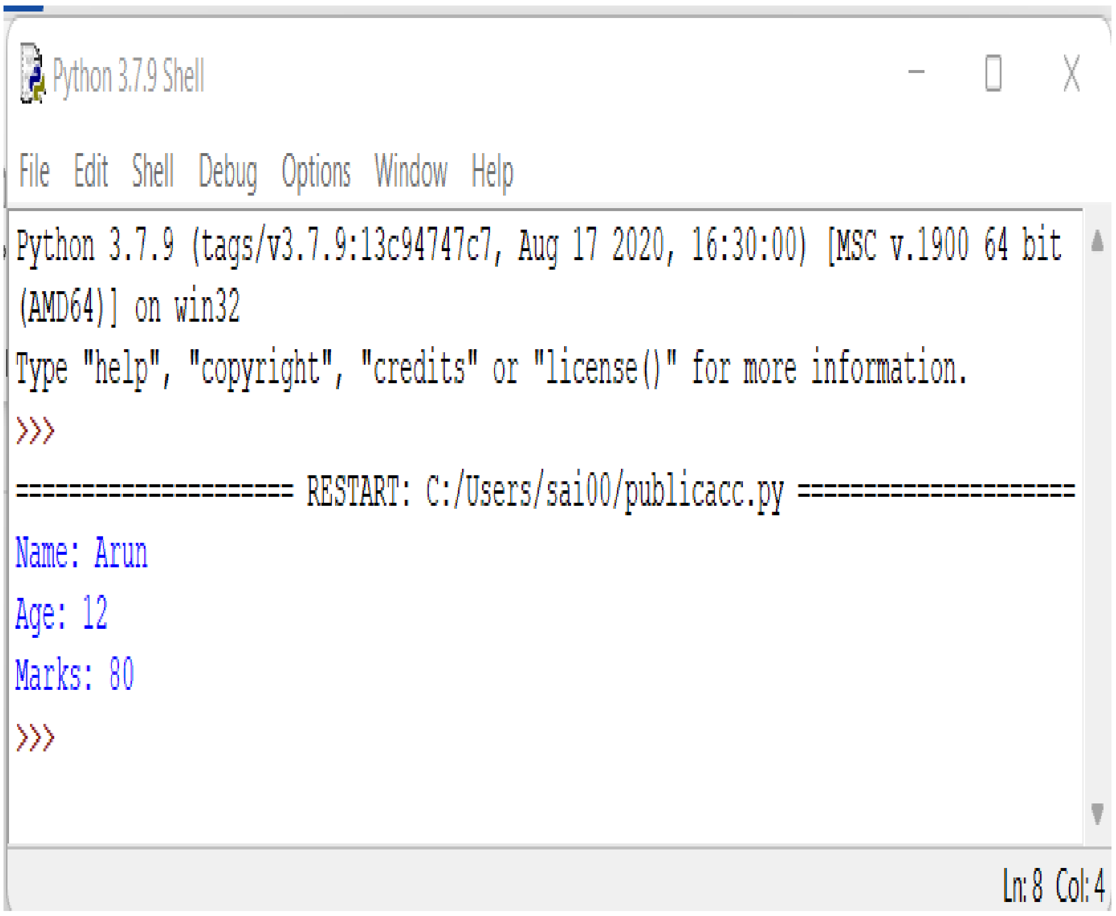
The member variables and methods are public by default, which means that they can be accessed from any location within or outside of the class where the class is being declared. If you want to make the class, its methods, and its attributes public, you do not need to use the public keyword.

Example:


publicacc.py - C:/Users/sai00/publicacc.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 1 Col: 52

```
# python code to demonstrate Public access specifier
class Student:
   def __init__(self, name, age,marks):
      self.name = name
      self.age = age
      self.marks= marks

   def display(self):
      print("Name:", self.name)
      print("Age:", self.age)
      print("Marks:", self.marks)

s = Student("Arun", 12,80)
s.display()
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/publicacc.py ====================
Name: Arun
Age: 12
Marks: 80
>>>
```

In the above example, developed a constructor for a Python class that we titled **'Student'** and constructed a constructor that accepts three arguments that are named real_name, age, and marks. After that, we created a variable outside of the class that we called **'s'**, instantiated the **'Student'** class with the appropriate arguments, and then printed the values of the variable with the method **display ().**

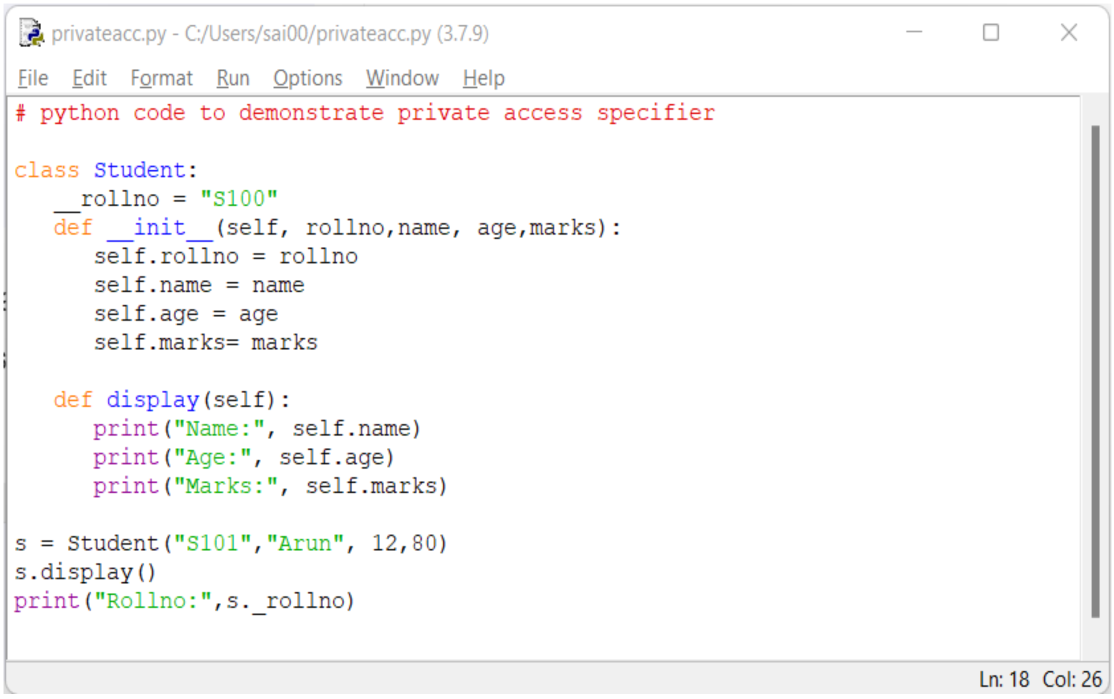
13.5.1 Private Access Specifier

The "Private" access modifier, as its name suggests, is responsible for limiting the variables and methods that are declared within a particular class to the environment of that class. To put it another way, the variables and methods that are declared within a class can only be accessed within the environment of that class since they are not accessible outside of that class.

The Python programming language does not have any mechanism that restricts access to the methods or variables. On the other hand, there is a path that we may take to restrict access to the variables and methods that are available in Python. To mimic the effect of the private access modifier, Python recommends using a double underscore when executing the code.

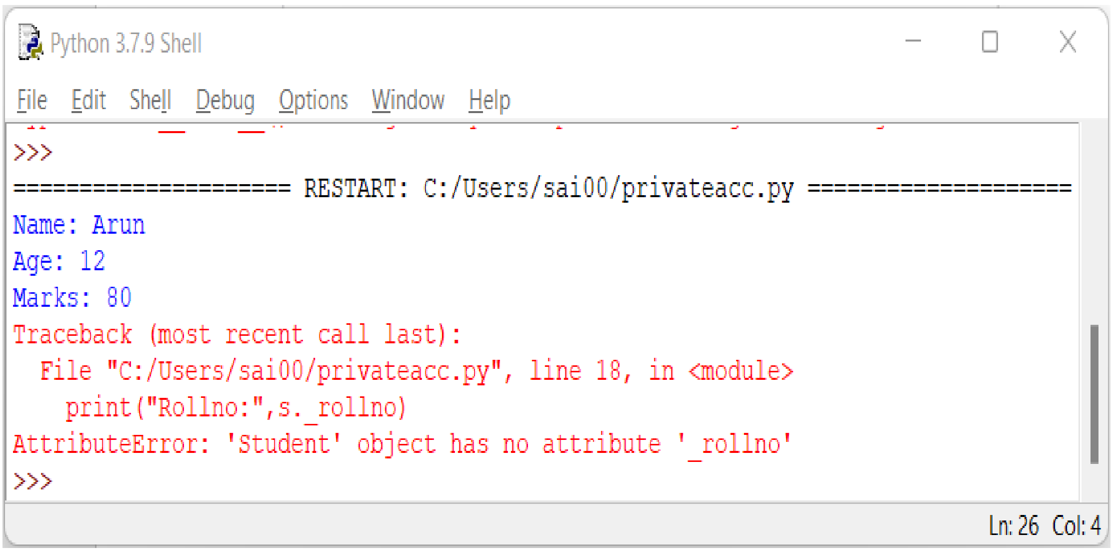
Those variables and methods that are preceded by a double underscore (__) are considered private and cannot be accessed by anybody outside of the class in which they are contained. An illustration will help us better comprehend it.

Example:


privateacc.py - C:/Users/sai00/privateacc.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 18 Col: 26

```
# python code to demonstrate private access specifier

class Student:
   __rollno = "S100"
   def __init__(self, rollno,name, age,marks):
      self.rollno = rollno
      self.name = name
      self.age = age
      self.marks= marks

   def display(self):
      print("Name:", self.name)
      print("Age:", self.age)
      print("Marks:", self.marks)

s = Student("S101","Arun", 12,80)
s.display()
print("Rollno:",s.__rollno)
```

Output:



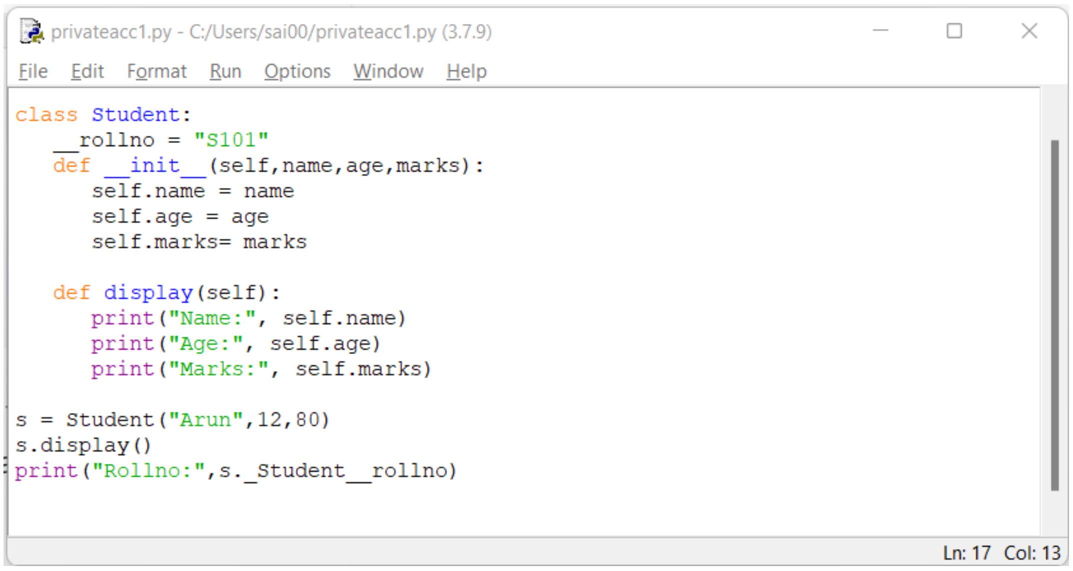
```
>>>
==================== RESTART: C:/Users/sai00/privateacc.py ====================
Name: Arun
Age: 12
Marks: 80
Traceback (most recent call last):
  File "C:/Users/sai00/privateacc.py", line 18, in <module>
    print("Rollno:",s.__rollno)
AttributeError: 'Student' object has no attribute '__rollno'
>>>
```

The code that is presented above is responsible for the creation of the class **'Student'**, as well as the creation of a private variable called **s.__rollno**. Subsequently, we attempted to gain access to the confidential information after we had created the class and given the necessary inputs.

We received an **AttributeError** that stated that the class **Student** does not have an attribute called **__rollno**. This occurred because private data included within a class cannot be accessed by individuals who are not members of the class. But we can access the secret methods and variables that are contained within the class.

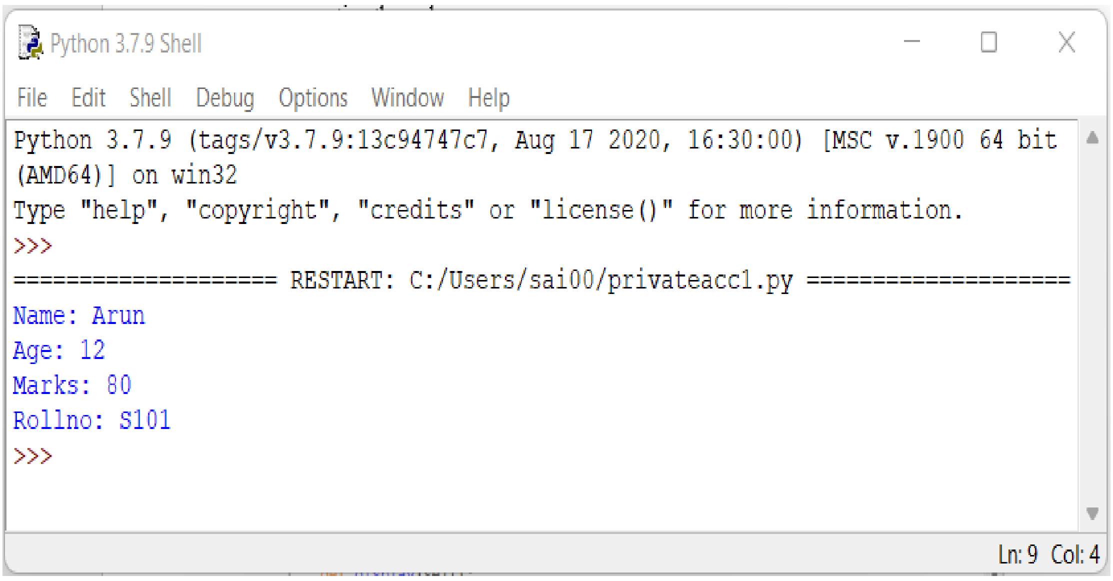
The error generated above program is rectified with the modified code shown in given below:

Example:


privateacc1.py - C:/Users/sai00/privateacc1.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 17 Col: 13

```
class Student:
   __rollno = "S101"
   def __init__(self,name,age,marks):
      self.name = name
      self.age = age
      self.marks= marks

   def display(self):
      print("Name:", self.name)
      print("Age:", self.age)
      print("Marks:", self.marks)

s = Student("Arun",12,80)
s.display()
print("Rollno:",s._Student__rollno)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/privateacc1.py ====================
Name: Arun
Age: 12
Marks: 80
Rollno: S101
>>>
```

With the output shown in above, shows that successfully accessed the all the attributes belongs to public **'name'**, **'age'**, **'marks'** and private variable '**__rollno'** outside the class.

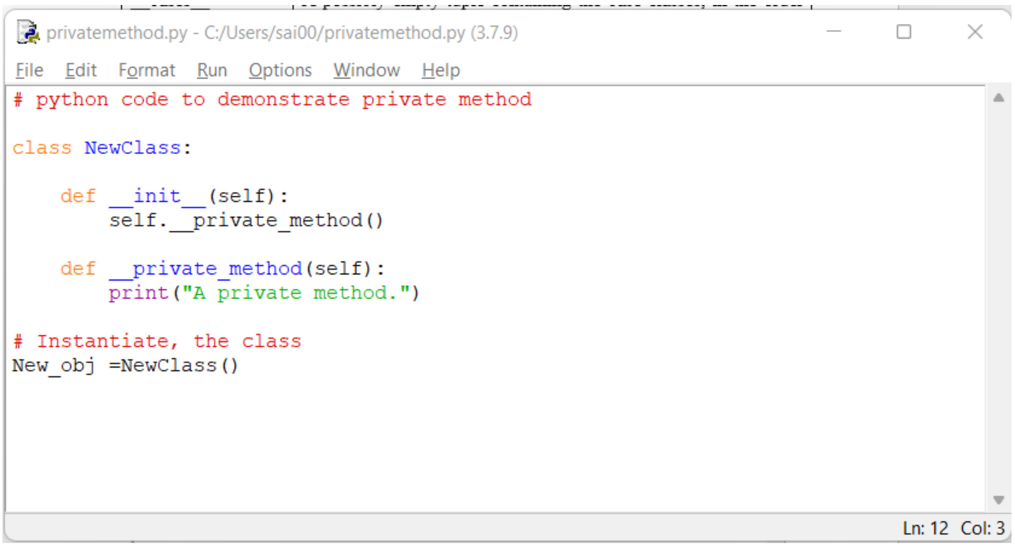
13.5.3 Private Methods

A method is private if it is not meant to be used by any program that is not a member of the class in which it is defined. The implementation of the class's core functionality is accomplished using these methods. It is not intended for these to be utilized by code from the outside. The name of a private method in Python is preceded by a double underscore, which specifies that the method is private.The syntax to define private method shown in below:

Syntax:

```
class Class_Name
      def __init__(self):
            …
            …
              def __private_method(self):  # Private Method
            …
            …
# Instantiate, the class
obj = Class_Name  ()
```

Example:

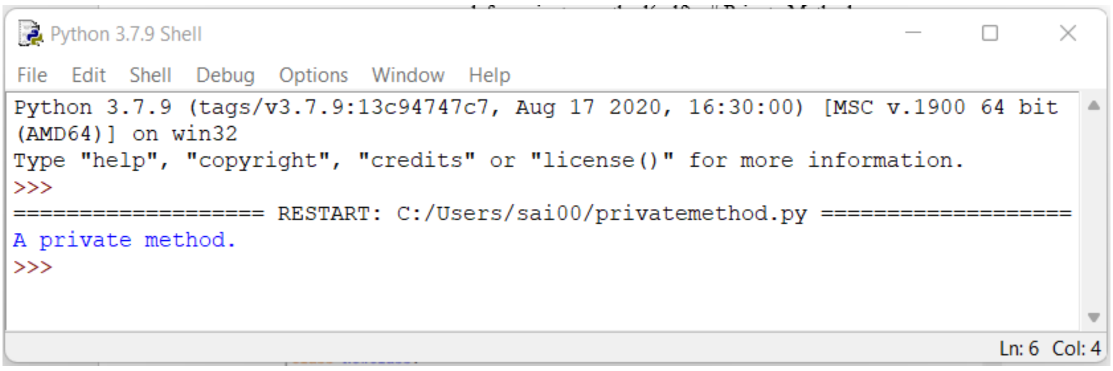


privatemethod.py - C:/Users/sai00/privatemethod.py (3.7.9)

```
# python code to demonstrate private method

class NewClass:

    def __init__(self):
        self.__private_method()

    def __private_method(self):
        print("A private method.")

# Instantiate, the class
New_obj =NewClass()
```

Ln: 12 Col: 3

Output:



Python 3.7.9 Shell

```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/privatemethod.py ===================
A private method.
>>>
```

Ln: 6 Col: 4

In the above code, defined a class that we will refer to as NewClass in the example cited above. __private_method is the name of the private method that it possesses. It is possible to invoke this method by calling the self.__private_method() function from the class constructor (__init__). Since the method is preceded by a double underscore, it is considered private and cannot be accessed by anyone outside of the class.

The advantages and disadvantages of private methods:

1. Private methods enable encapsulation. It is a core notion of object-oriented programming. Making specific methods private allows the programmer to regulate how external code accesses the class's internal capabilities. This increases the class's security and helps to avoid undesired changes to its functionality.

2. Private methods enable code reusability by implementing internal functionality within a class. This can help to reduce code duplication while improving code maintainability.

3. Private methods provide for easier debugging by isolating the class's behavior.

1. Private methods can only be accessed within the class where they are defined. This means that if a programmer wishes to access the functionality of a private method from outside the class, they must define a public method that calls the private method. This can increase the code's complexity.

2. False sense of security: Python's secret methods are not actually private. They can still be accessed outside of the class using the syntax _classname__methodname(). However, this is considered poor practice and should be avoided.

3. Private methods can increase code complexity, making it difficult to comprehend and maintain. This is especially relevant when private methods are inadequately documented.

13.6 BUILT-IN CLASS ATTRIBUTES

The built-in class attributes provide us with information about the class. Using the dot **(.)** operator, we may access the built-in class attributes. The built-in class attributes in python are listed below and shown in Table 13.1:

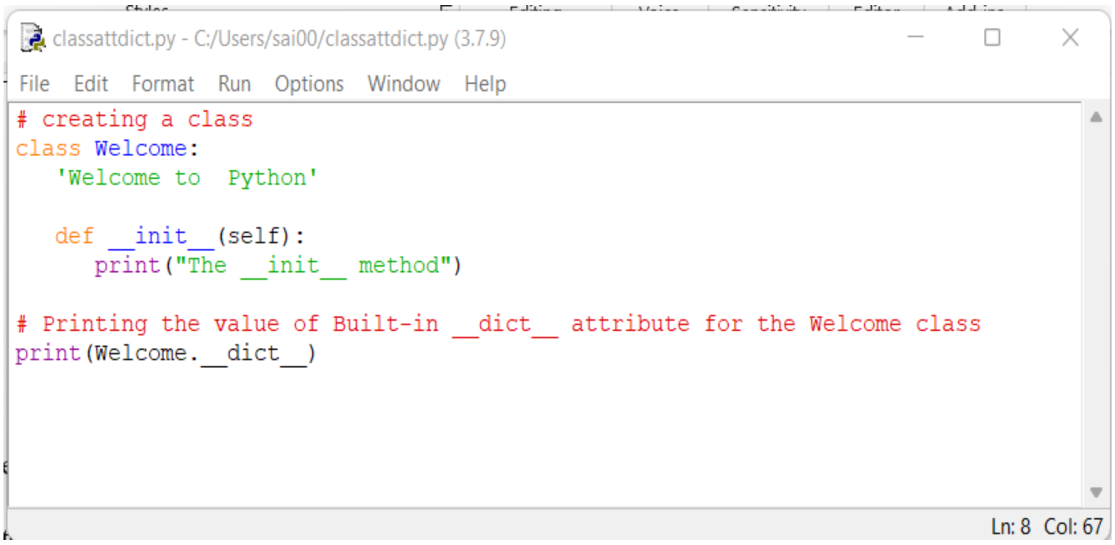
Table 13.1. Built-in Class attributes in Python

Attributes	**Description**
_dict__	Dictionary containing the class namespace
__doc__	If there is a class documentation class, this returns it. Otherwise, None
__name__	Class name.
__module__	Module name in which the class is defined. This attribute is "__main__" in interactive mode.
__bases__	A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

13.6.1 __dict__ Class Attribute

In Python, the __dict__ variable is used to represent a dictionary or other mapping object that is utilized for the purpose of storing the properties of the assigned object. It is also possible to refer to them as mappingproxy objects. To put it another way, each object in Python possesses an attribute that is represented by the symbol __dict.

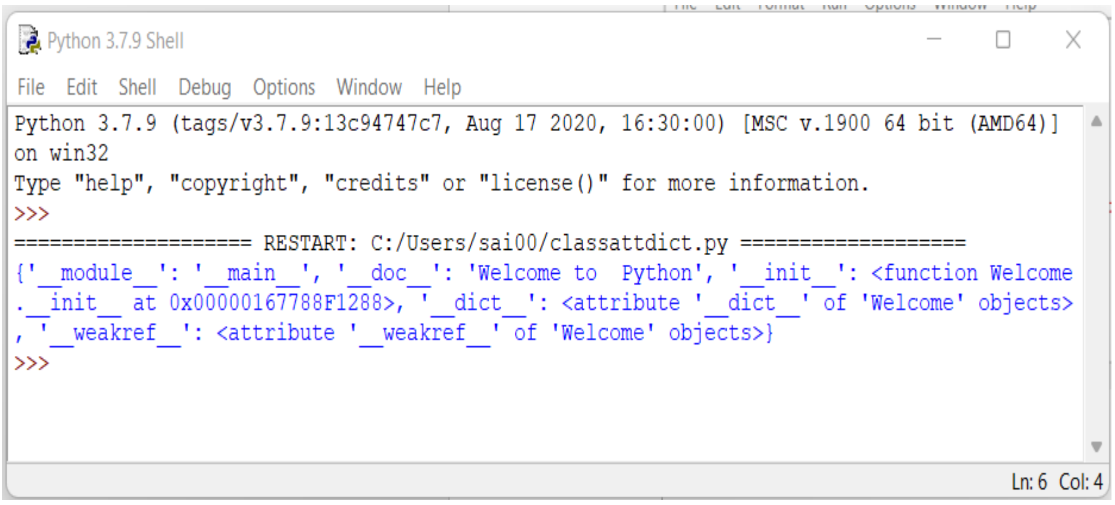
Example:


classattdict.py - C:/Users/sai00/classattdict.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 8 Col: 67

```
# creating a class
class Welcome:
   'Welcome to  Python'

   def __init__(self):
      print("The __init__ method")

# Printing the value of Built-in __dict__ attribute for the Welcome class
print(Welcome.__dict__)
```

In the above code created class **"Welcome"** and initialized self-method and displayed information related to name space with **Welcome._dict_** and result is shown in output.

Output:

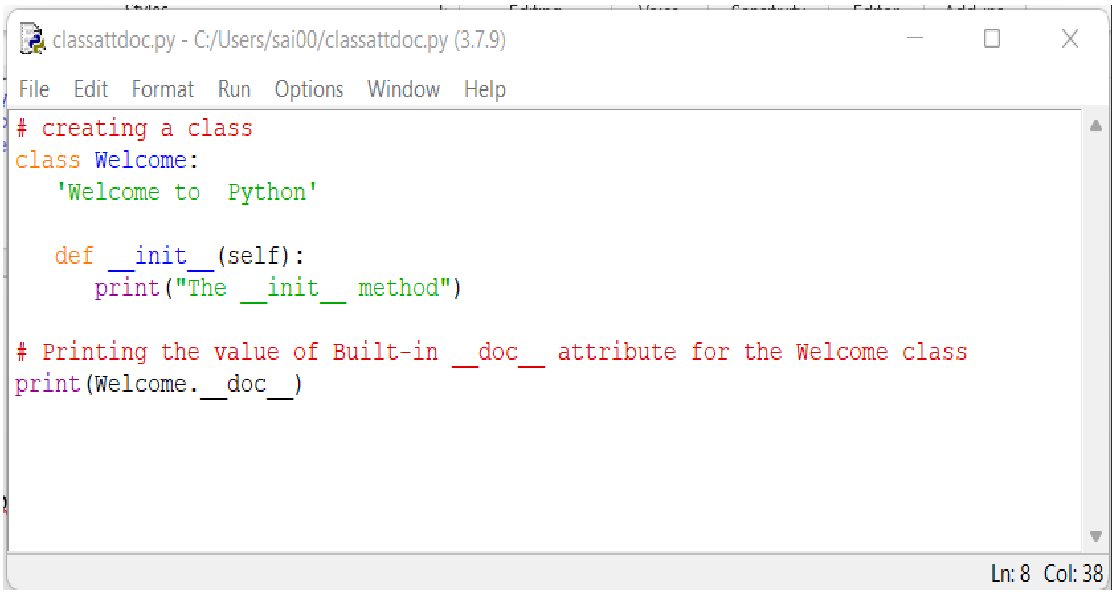


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Users/sai00/classattdict.py ===================
{'__module__': '__main__', '__doc__': 'Welcome to  Python', '__init__': <function Welcome
.__init__ at 0x00000167788F1288>, '__dict__': <attribute '__dict__' of 'Welcome' objects>
, '__weakref__': <attribute '__weakref__' of 'Welcome' objects>}
>>>
```

13.6.2 __doc__ Class Attribute

The documentation string for a module, class, function, or method is stored in the __doc__ property, which is a special attribute in the Python programming language. The documentation string (also known as the documentation string) is a concise description of the object that is frequently utilized in the Python documentation to offer documentation for the object.

Example:

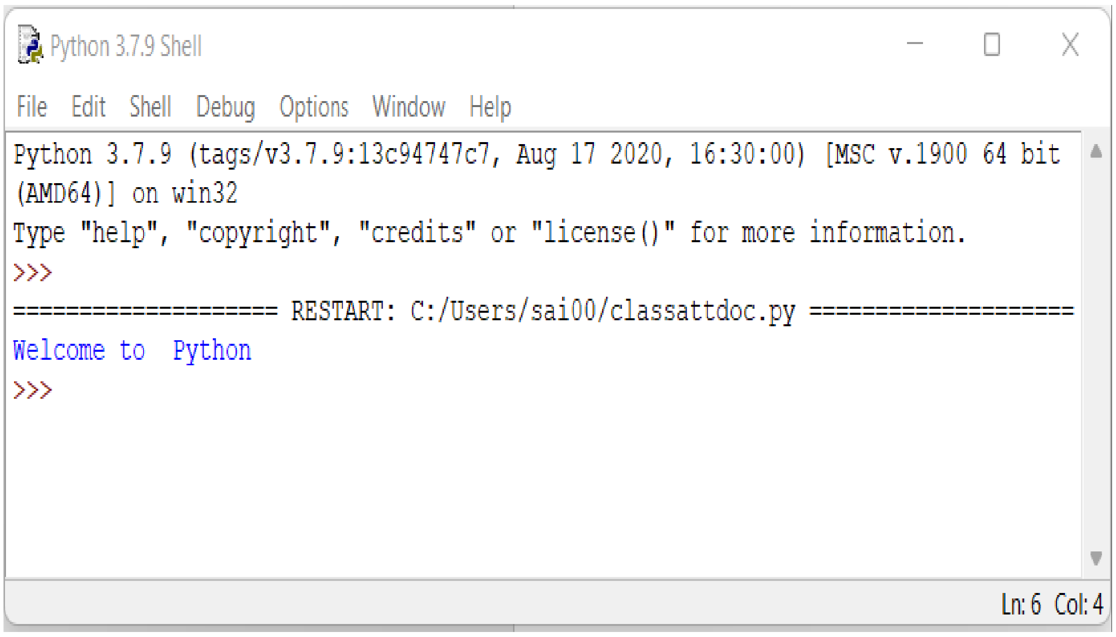


classattdoc.py - C:/Users/sai00/classattdoc.py (3.7.9)

```
# creating a class
class Welcome:
   'Welcome to  Python'

   def __init__(self):
      print("The __init__ method")

# Printing the value of Built-in __doc__ attribute for the Welcome class
print(Welcome.__doc__)
```

In the above code created class **"Welcome"** and initialized self-method and displayed information related to class documenation with **Welcome._doc_** and result is shown in output.

Output:



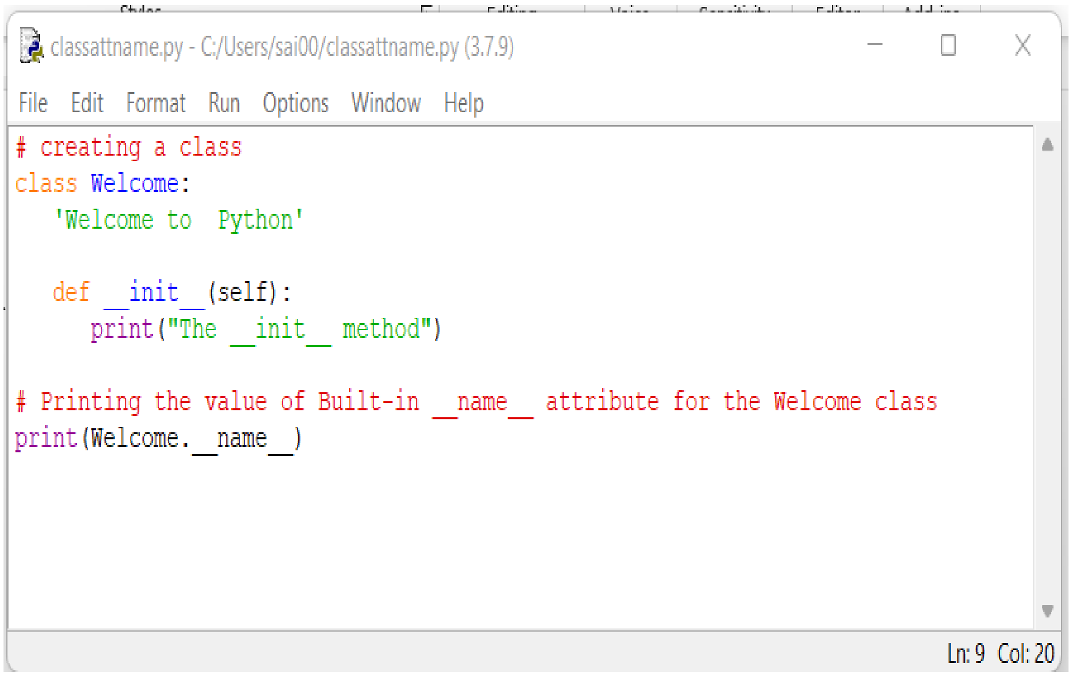
Python 3.7.9 Shell

```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/classattdoc.py ===================
Welcome to  Python
>>>
```

13.6.3 __name__ Class Attribute

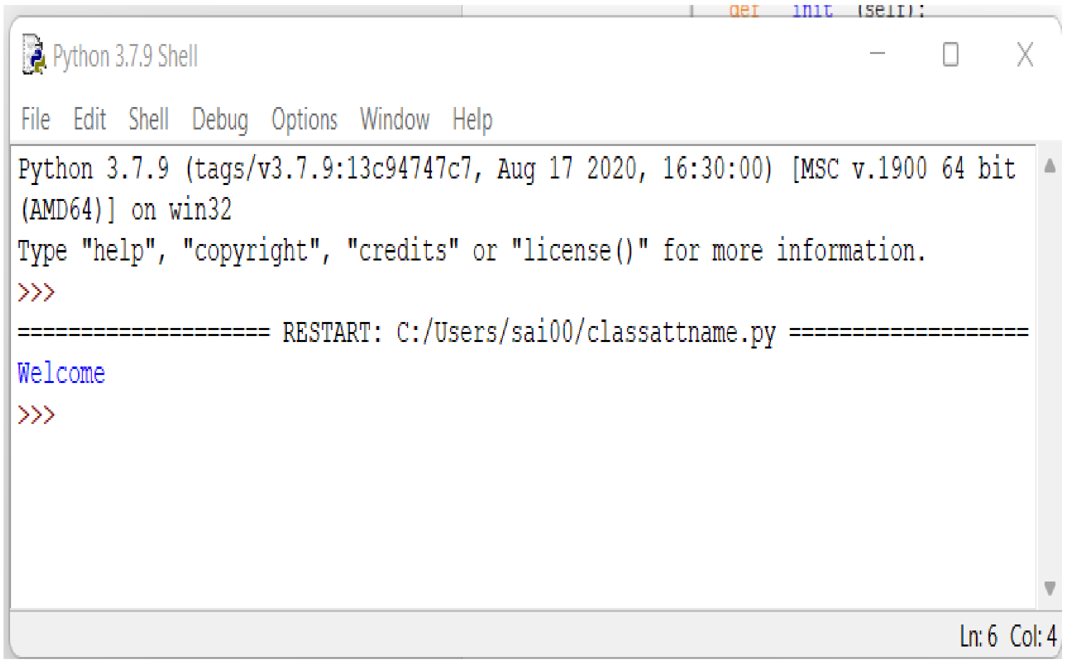
It is the name of the module that is returned by the __name__ attribute. In the default configuration, the value of the __name__attribute is used to determine the name of the file, excluding the extension.py. A great instance of such a special variable is __name__. In the event that the source file is used as the main program, the interpreter will assign the value "__main__" to the __name__ variable. If this file is being imported from another module, the name of the module will be assigned to the __name__ variable.

Example:


classattname.py - C:/Users/sai00/classattname.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 9 Col: 20

```
# creating a class
class Welcome:
   'Welcome to  Python'

   def __init__(self):
      print("The __init__ method")

# Printing the value of Built-in __name__ attribute for the Welcome class
print(Welcome.__name__)
```

Output:



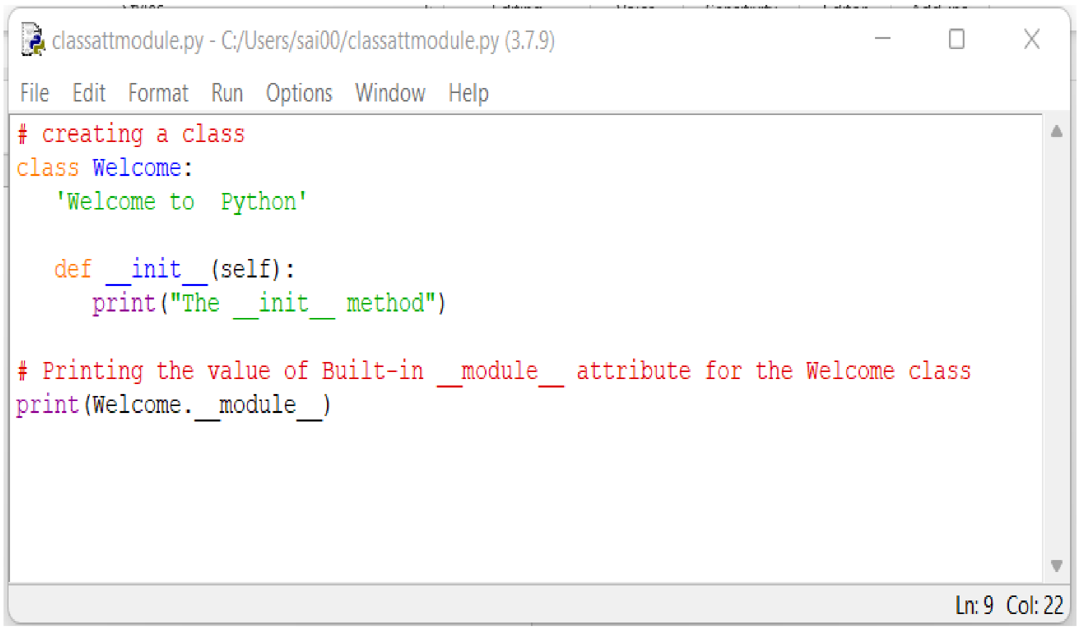
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/classattname.py ==================
Welcome
>>>
```

The reason for this is that the __name__ attribute is defined explicitly as a component of the class specification. In the above example , ‘ Welcome’ is produced as a name attribute returned by the _name_attribute is given to the instance to the class.

13.6.4 __module__ Class Attribute

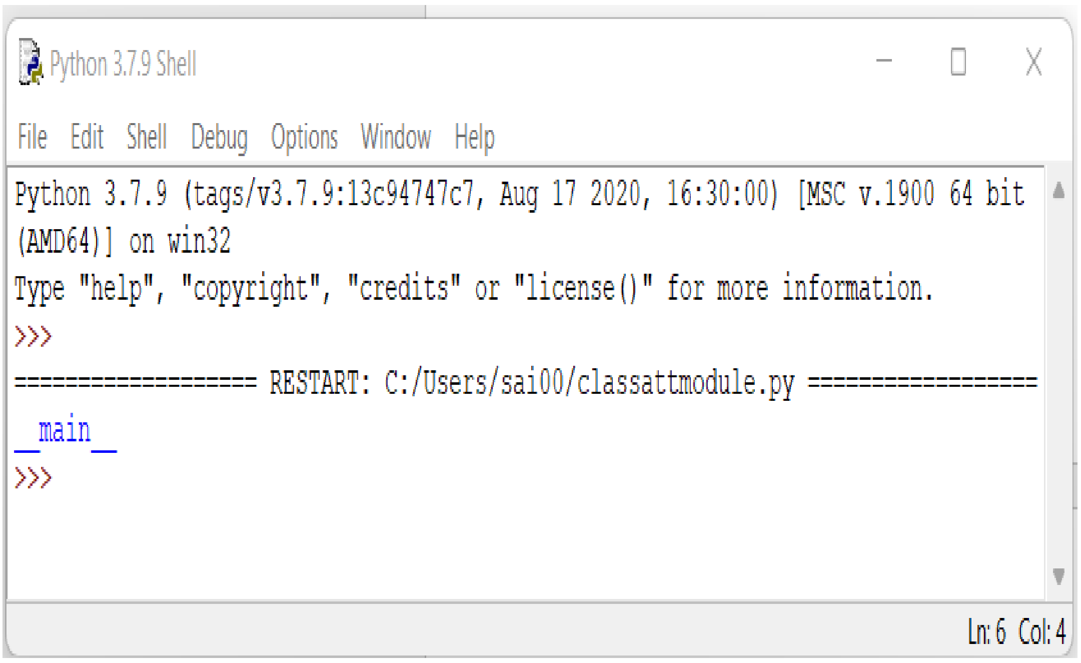
In the Python code below, we print the module of the class using the __module__ class attribute.

Example:


classattmodule.py - C:/Users/sai00/classattmodule.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 9 Col: 22

```
# creating a class
class Welcome:
   'Welcome to  Python'

   def __init__(self):
      print("The __init__ method")

# Printing the value of Built-in __module__ attribute for the Welcome class
print(Welcome.__module__)
```

Output:

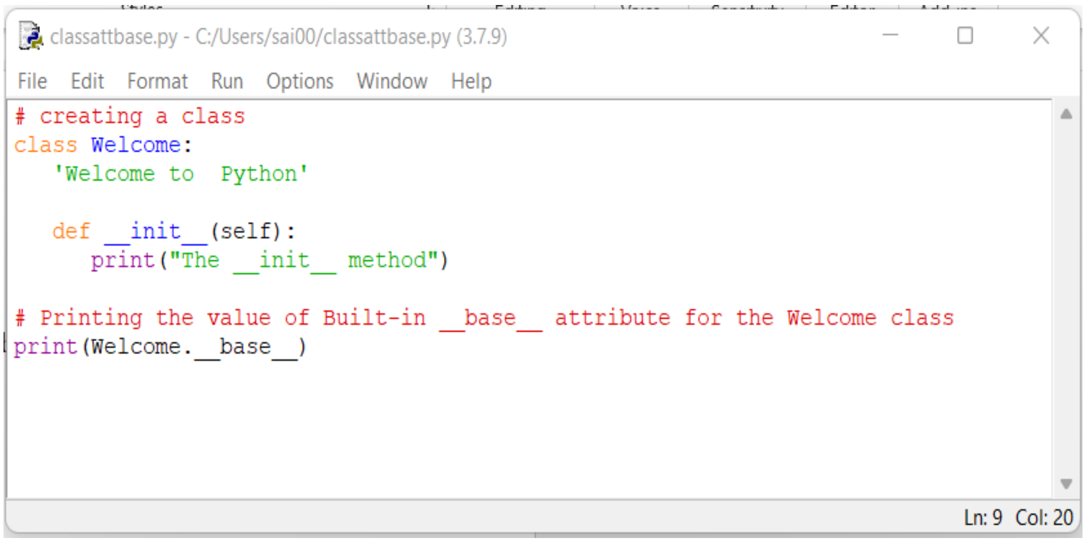


```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/classattmodule.py =================
__main__
>>>
```

13.6.5 __bases__ Class Attribute

The implementation of the __bases__ attribute of a class is carried out by a descriptor that is located in the metaclass or type. On the other hand, you need to exercise a little bit of caution since type, which is one of the fundamental components of the Python object model, is an instance of itself, and hence type. When, it comes to introspection, __bases__ does not perform as you would expect it to.

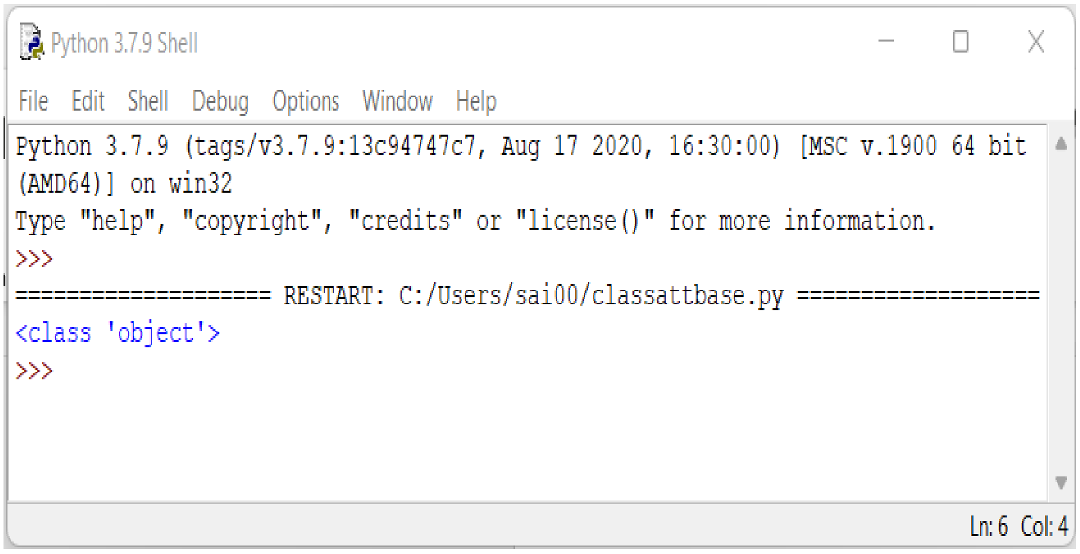
Example:



classattbase.py - C:/Users/sai00/classattbase.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 9 Col: 20

```
# creating a class
class Welcome:
    'Welcome to  Python'

    def __init__(self):
        print("The __init__ method")

# Printing the value of Built-in __base__ attribute for the Welcome class
print(Welcome.__base__)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/classattbase.py ===================
<class 'object'>
>>>
```

In the above code created class **"Welcome"**, the information related to the module and base class information is displayed using the **Welcome._module_ and Welocme._base_** class attributes. The result is **_main_** and **<class 'object'>** shown in output.

13.7 SUMMARY

To work with OOP in Python, classes and methods are extremely significant concepts they contribute to the creation of code that is not only comprehensible but also reusable. By building a class, you can put together a collection of information and capabilities into a single entity that can be utilized in the construction of a variety of things. You can have access to the methods and properties of an object after it has been formed by utilizing the dot notation.

By having a solid understanding of Python's classes and objects, you will be able to create code that is more logical, efficient, and maintainable.

13.8 TECHNICAL TERMS

Class, Object, Class variable, Object Variables, Class Attributes, Public Specifier, Private Specifier, Private Method

13.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. What is private method? Discuss with example.
2. Compare public and private access specifiers.
3. Explain built-in class attributes with example.

Short Notes:

1. Define Class?
2. List advantages of private methods

13.10 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.
2. Reemathareja – Python Programming using problem solving approach, Oxford Publication
3. A Byte of Python by Swaroop C.H.
4. "Python 3 Object-Oriented Programming" by Dusty Phillips

Dr. KAMPA LAVANYA

LESSON- 14

PYTHON STATIC METHOD

AIMS AND OBJECTIVES

In this unit the student can recognize the importance of static methods in python, describe what is static method, how it is created. The advantages and applications. After completing this chapter, you will be able to explain how static method is different from the normal class method. How static method is used in different applications. How to use it properly.

STRUCTURE

14.1 INTRODUCTION

Python is a versatile programming language that effortlessly blends Object-Oriented Programming (OOP) features. While classes and objects are important in OOP, the methods within these classes help a coder be more efficient. One such critical function is Python's static method. A static method is one that is connected to the class itself rather than the class's object.

Static methods complete a task independently of the class because they do not employ implicit arguments like self or cls. As a result, static methods cannot change the state of a class. It is there in a class because the method makes sense to be there. This method facilitates the definition of utility methods by establishing a logical relationship to a class. The method call is made directly on the class object or via a class instance.

14.2 CREATING STATIC METHOD

Python provides two ways to create a static method.

- staticmethod()
- @staticmethod

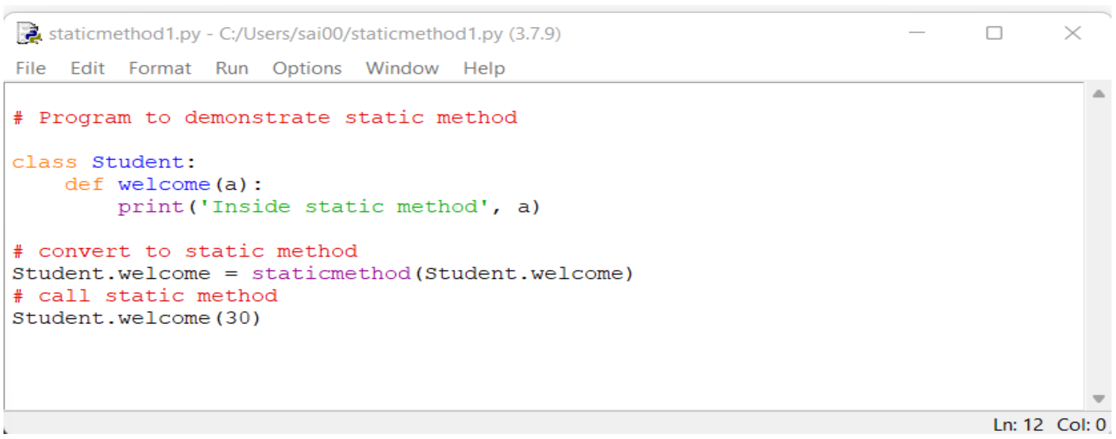
14.2.1 using staticmethod()

When you want to define a static method in a class, use staticmethod(). It is important to note that the method should not contain the self-argument.

Syntax:

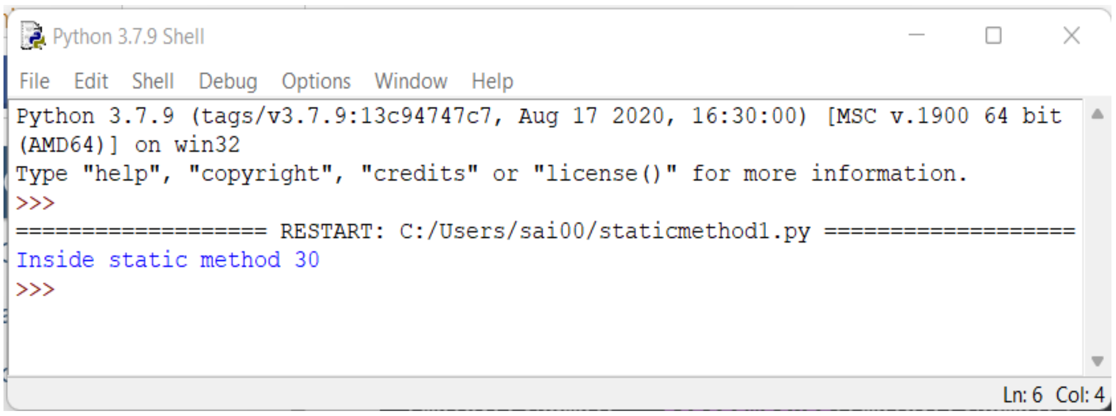
```
class Class_Name:

    def fun_name(arg1, arg2, ...):
     ...
     ...
Class_Name.fun_name = staticmethod(Class_Name.fun_name)

Class_Name.fun_name(value)    #static method calling
```

Example:


staticmethod1.py - C:/Users/sai00/staticmethod1.py (3.7.9)

```
# Program to demonstrate static method

class Student:
    def welcome(a):
        print('Inside static method', a)

# convert to static method
Student.welcome = staticmethod(Student.welcome)
# call static method
Student.welcome(30)
```

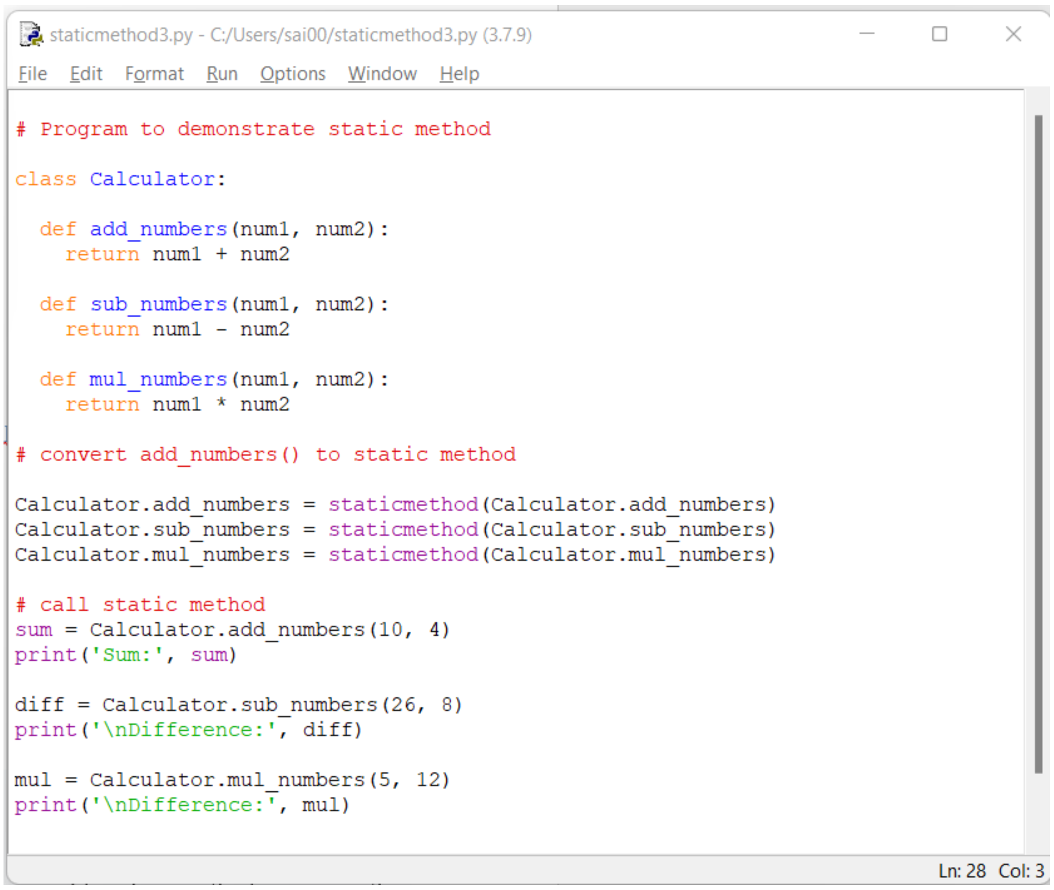
Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/staticmethod1.py ==================
Inside static method 30
>>>
```

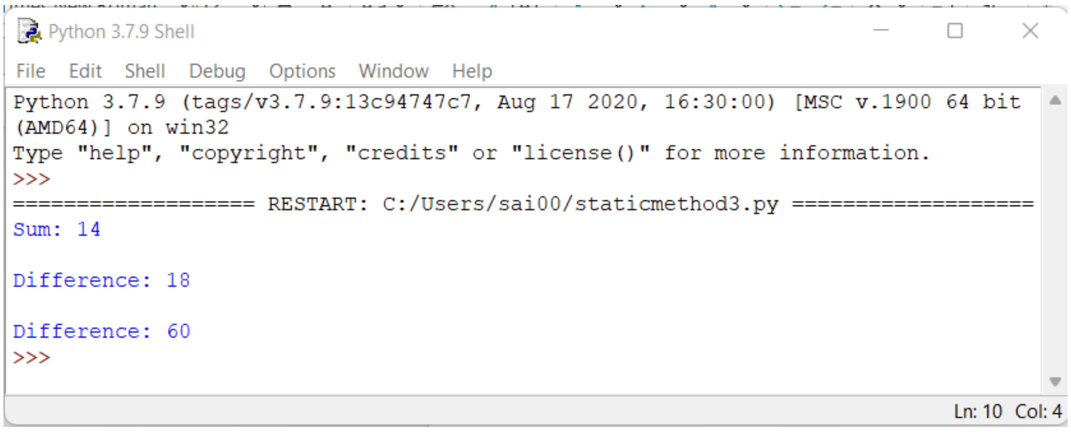
The above program was first designed using the class method, i.e., **welcome ()** inside the class **student**. Later, that procedure is changed to a static method using **staticmethod()** , which accepts the argument **'student.welcome'.** Finally, we called the static method **'student.welcome(30)** ' and produced output as **“ Inside static method 30”** .

Example2:


staticmethod3.py - C:/Users/sai00/staticmethod3.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 28 Col: 3

```
# Program to demonstrate static method

class Calculator:

  def add_numbers(num1, num2):
    return num1 + num2

  def sub_numbers(num1, num2):
    return num1 - num2

  def mul_numbers(num1, num2):
    return num1 * num2

# convert add_numbers() to static method

Calculator.add_numbers = staticmethod(Calculator.add_numbers)
Calculator.sub_numbers = staticmethod(Calculator.sub_numbers)
Calculator.mul_numbers = staticmethod(Calculator.mul_numbers)

# call static method
sum = Calculator.add_numbers(10, 4)
print('Sum:', sum)

diff = Calculator.sub_numbers(26, 8)
print('\nDifference:', diff)

mul = Calculator.mul_numbers(5, 12)
print('\nDifference:', mul)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/staticmethod3.py ===================
Sum: 14

Difference: 18

Difference: 60
>>>
```

The above calculator application contains a number of non-static functions, including **add(), sub(), and mul()**. Later, all were converted to static methods and finally called. The results of each procedure are documented separately.

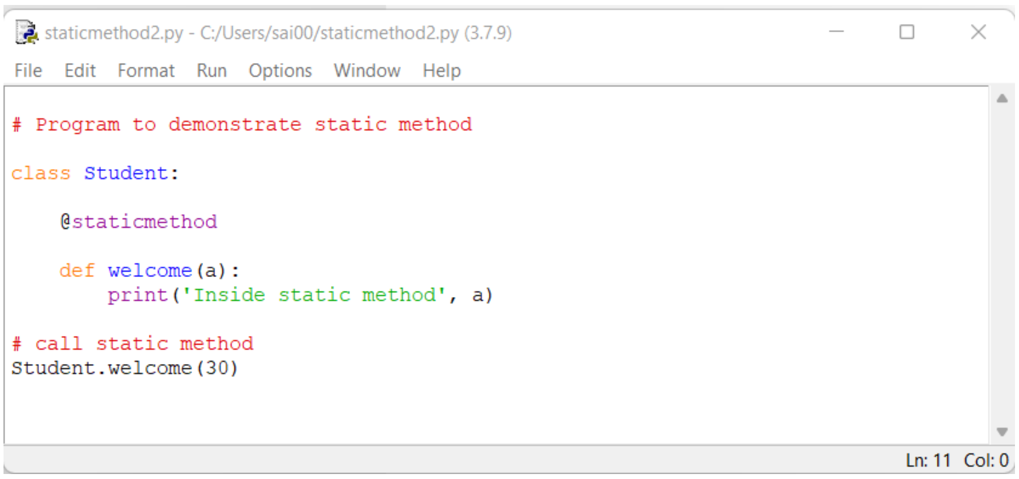
14.2.2 using @staticmethod

This is the recommended technique to build a static method. We only need to mark the method with the @staticmethod decorator.

Syntax:

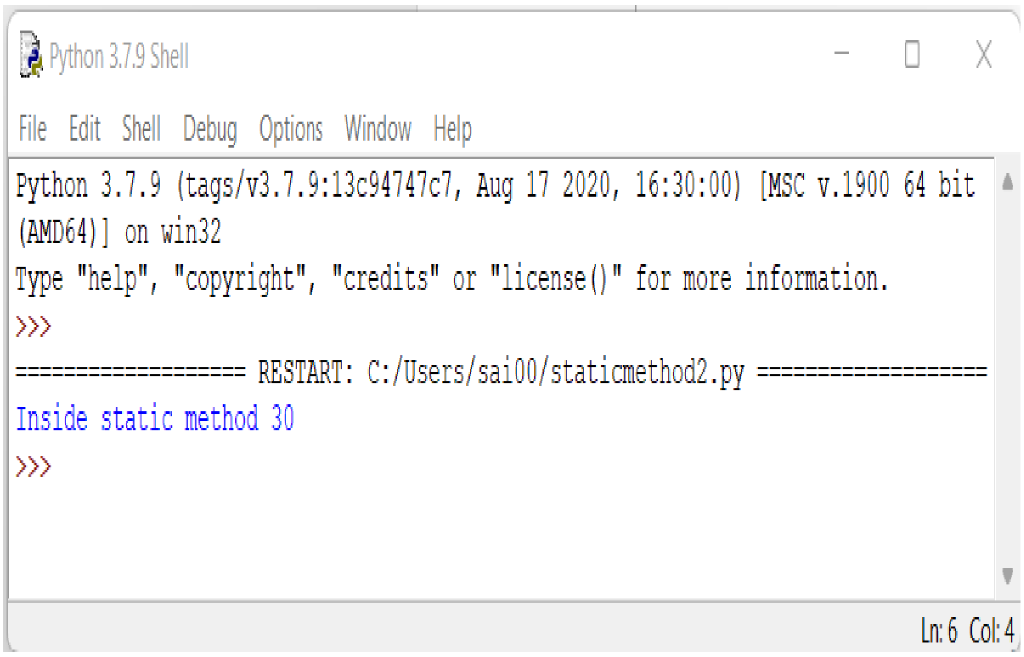
```
class Class_Name:

    @staticmethod
    def fun_name(arg1, arg2, ...):
     ...
     ...
Class_Name.fun_name(value)     #static method calling
```

Example1:


staticmethod2.py - C:/Users/sai00/staticmethod2.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 11 Col: 0

```
# Program to demonstrate static method

class Student:

    @staticmethod

    def welcome(a):
        print('Inside static method', a)

# call static method
Student.welcome(30)
```

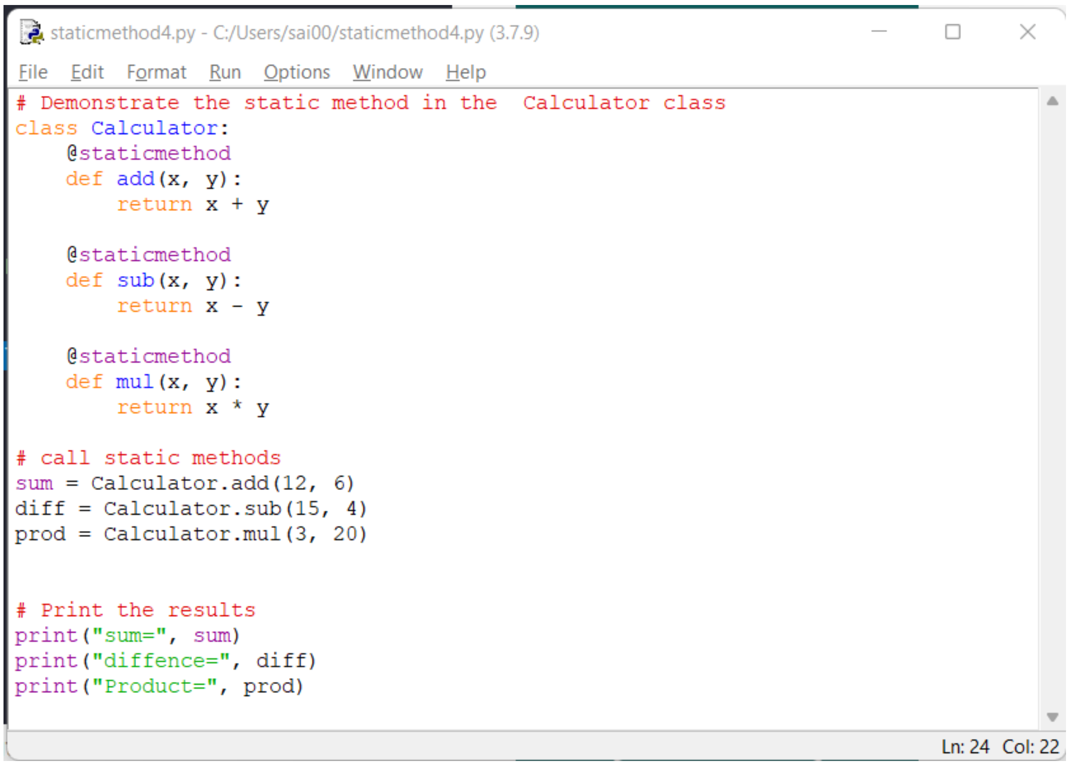
Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/staticmethod2.py ==================
Inside static method 30
>>>
```

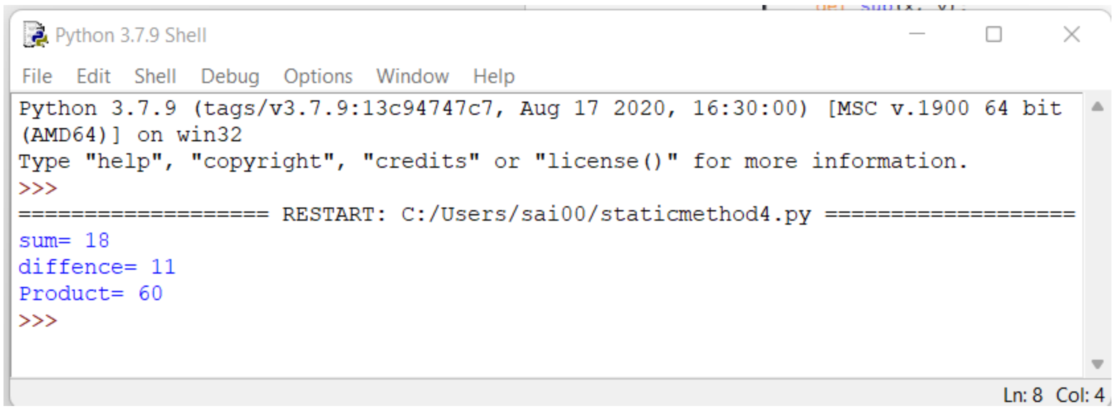
The above program has the first static method, **welcome ()**, which is part of the class student. Later, the method is known as **Stuedet.welcome().** Here there is no need of converting class method to static method so that execution speed.

Example1:


staticmethod4.py - C:/Users/sai00/staticmethod4.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 24 Col: 22

```python
# Demonstrate the static method in the  Calculator class
class Calculator:
    @staticmethod
    def add(x, y):
        return x + y

    @staticmethod
    def sub(x, y):
        return x - y

    @staticmethod
    def mul(x, y):
        return x * y

# call static methods
sum = Calculator.add(12, 6)
diff = Calculator.sub(15, 4)
prod = Calculator.mul(3, 20)


# Print the results
print("sum=", sum)
print("diffence=", diff)
print("Product=", prod)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/staticmethod4.py ===================
sum= 18
diffence= 11
Product= 60
>>>
```

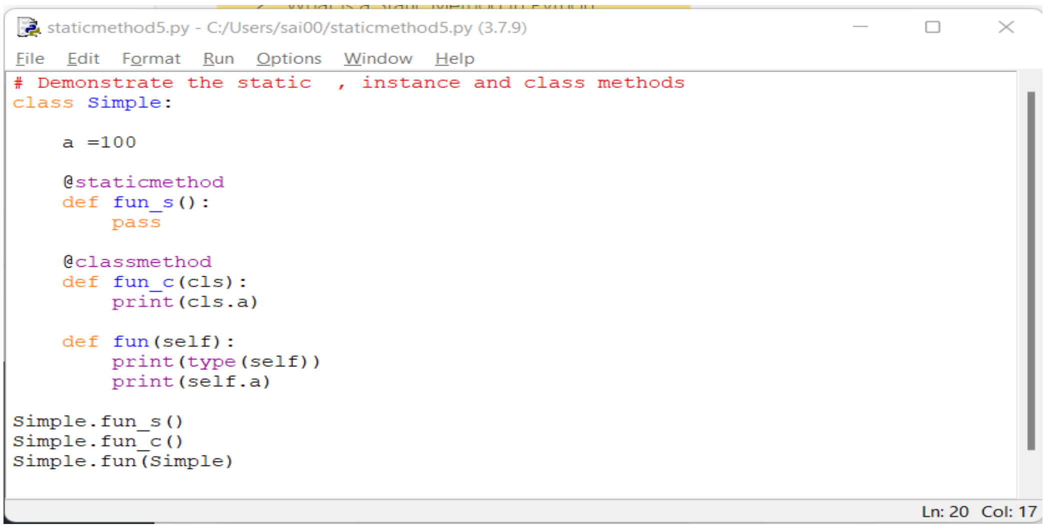
The above calculator program designed a number of static methods i.e., add(), sub() and mul(). The result of each method is displayed as “sum =18” with the x = 12 and y = 6 . Similarly the remain two operations sub() and mul() produced result as ‘diffrence=11’ and ‘Product = 60 ‘ with the given x and y values respectively.

14.3 CLASS METHOD Vs STATIC METHOD

The difference between the Class method and the static method is:

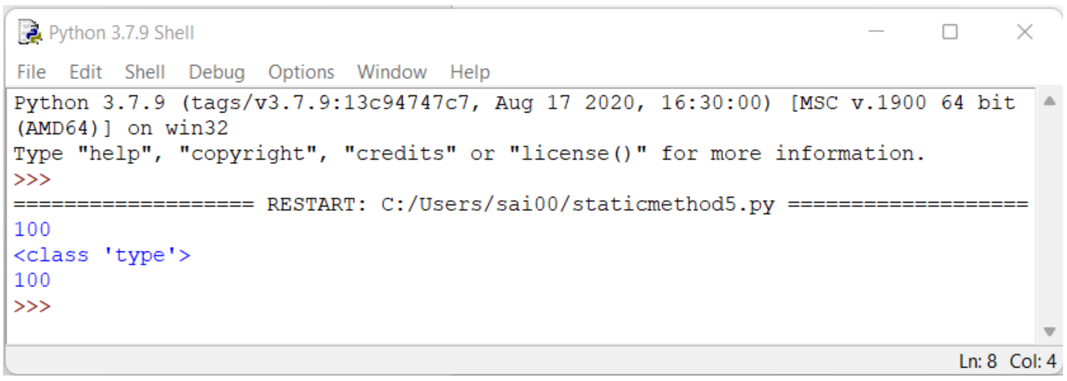
- A class method takes **cls** as the first parameter while a static method needs no specific parameters.
- A class method can access or modify the class state while a static method can’t access or modify it.
- In general, static methods know nothing about the class state. They are utility-type methods that take some parameters and work upon those parameters. On the other hand, class methods must have class as a parameter.
- We use **@classmethod** decorator in python to create a class method and we use **@staticmethod** decorator to create a static method in python.

Example:


staticmethod5.py - C:/Users/sai00/staticmethod5.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 20 Col: 17

```python
# Demonstrate the static  , instance and class methods
class Simple:

    a =100

    @staticmethod
    def fun_s():
        pass

    @classmethod
    def fun_c(cls):
        print(cls.a)

    def fun(self):
        print(type(self))
        print(self.a)

Simple.fun_s()
Simple.fun_c()
Simple.fun(Simple)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/staticmethod5.py ==================
100
<class 'type'>
100
>>>
```

Table 14.1. Instance Method Vs Class Method Vs Static Method

Class Method	Static Method
• Decorator: **@classmethod**	• Decorator: **@staticmethod**
• Initial argument: **cls** (class itself)	• No Initial argument
• Can't change state or properties of class	• Can't change state or properties of class
• Good for class-specific operations	• Good for utility operations not related to class
• class-level variables allowed	• class-level variables restricted

14.4 Advantages of Static Method in Python

There are several benefits to using static methods in Python classes. All static techniques are:

- **Risk-free.** Static methods are unable to alter the class state since they do not have access to the data stored in the class or instance. There is no way for this method to modify the behavior or have an unforeseen impact on a class.
- **Consistent.** The execution of a static method is independent of any class or instance state. A static method always acts in the same predictable way.
- **Adaptable.** One way to make code more readable and to indicate that a function does not depend on an instance is to call its static method on a class.
- **A tool for organizing.** To avoid naming conflicts and improve organization, static methods offer a mechanism to namespace code.

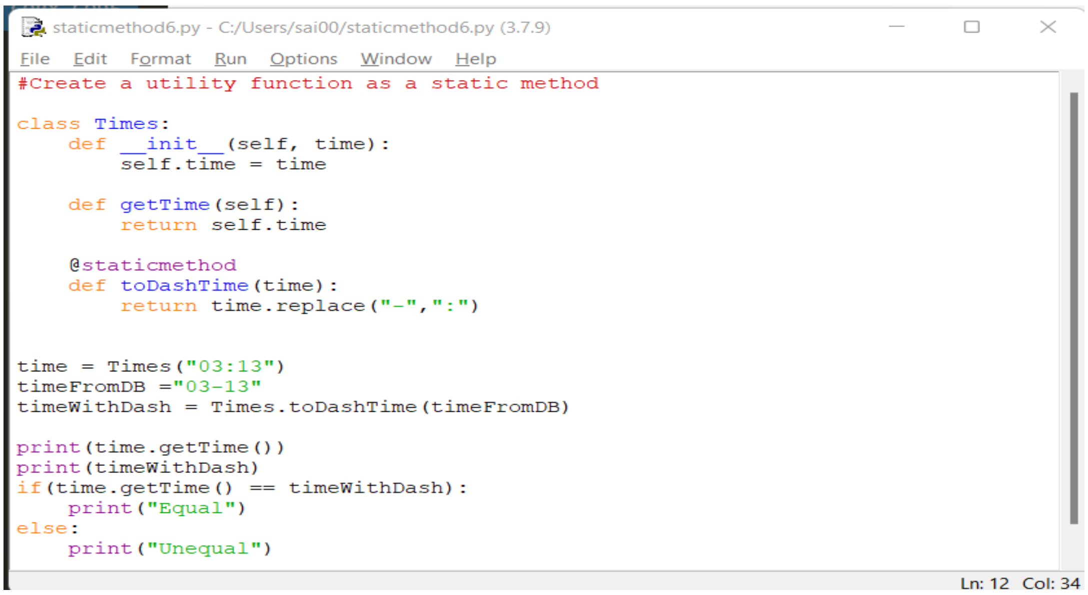
14.5. APPLICATIONS OF STATIC METHODS

Static methods are more important. While working with real-time classes, there are several reasons to use static methods, some of which are explained below:

14.5.1 Use utility function to a class.

Like other methods contained within a class, static methods are incapable of accessing the class's properties, which restricts their intended application. Utilizing static methods has numerous applications, including utility functions. These are methods for performing routine, frequently resorted-to operations that are beneficial for accomplishing typical programming duties. As a consequence, we can declare a static method on a utility function that requires only the parameters and does not require access to any class attributes.

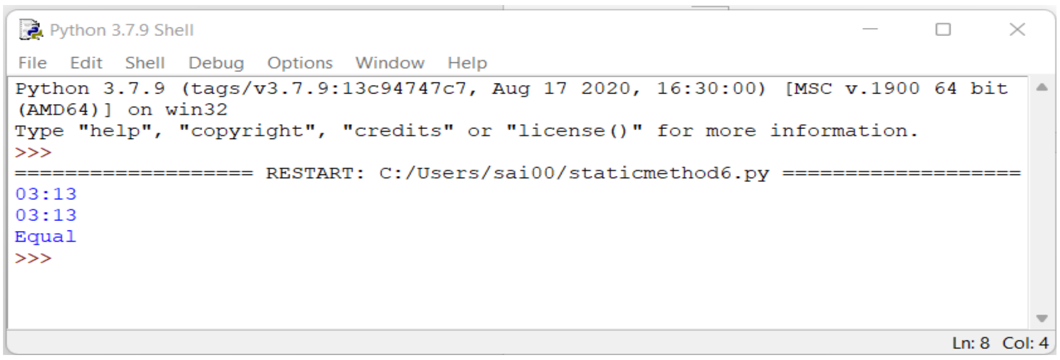
Example:


staticmethod6.py - C:/Users/sai00/staticmethod6.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 12 Col: 34

```
#Create a utility function as a static method

class Times:
    def __init__(self, time):
        self.time = time

    def getTime(self):
        return self.time

    @staticmethod
    def toDashTime(time):
        return time.replace("-",":")


time = Times("03:13")
timeFromDB ="03-13"
timeWithDash = Times.toDashTime(timeFromDB)

print(time.getTime())
print(timeWithDash)
if(time.getTime() == timeWithDash):
    print("Equal")
else:
    print("Unequal")
```

In the above program, a '**Times'** class is present which exclusively operates on Times with a colon. Conversely, in our prior database, every time was enclosed in dashes. To transform colon-times to dash-times, the **toDashTime** utility function has been implemented within Times. The technique is classified as static due to its lack of requirement for accessing any Times properties; it solely accepts the parameters. While it is possible to construct **toDashTime** instances outside the Times class, it is more logical to maintain its functionality within the times class.

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/staticmethod6.py ===================
03:13
03:13
Equal
>>>
```

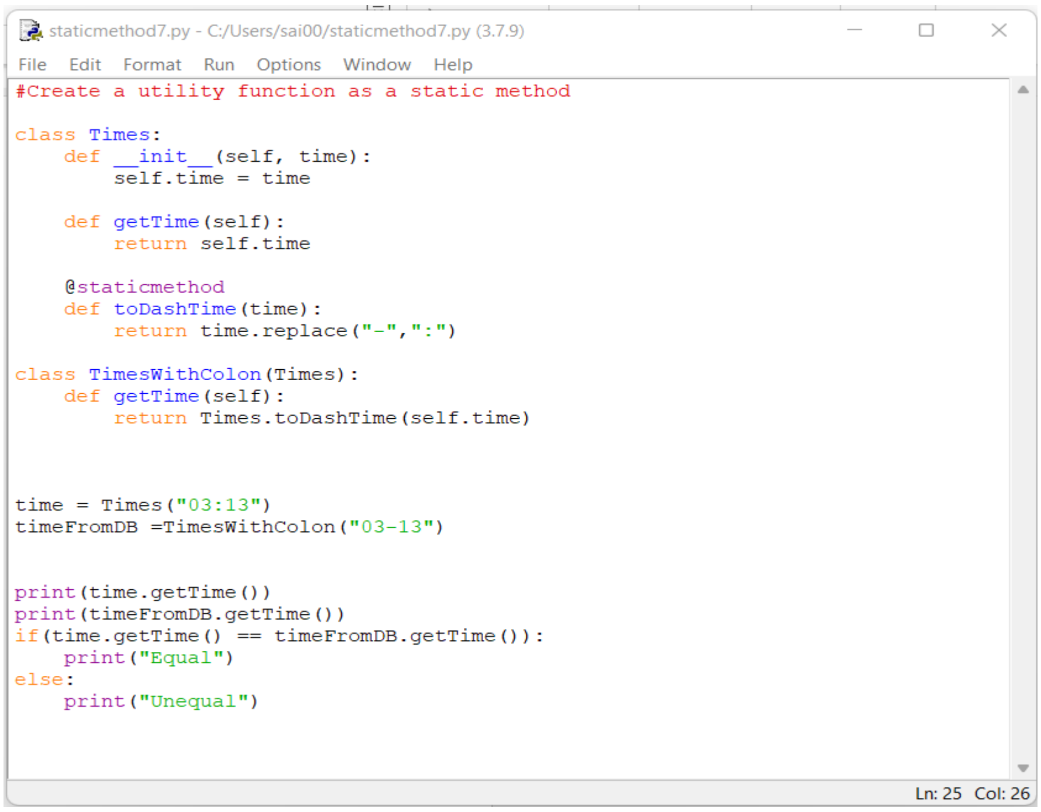
As shown in the output static method successfully converts the 03-13 into the format of 03:13 so the two times are equal. Other than any format input is given to time which is not trained by static method will produces output false.

14.5.2 A single design

It is necessary to make use of static methods in situations where we do not want subclasses of a class to modify or override a particular implementation of a method.
From here created the sub-class **class TimesWithColon(Times)** to the original **class Times** as shown in below**:**

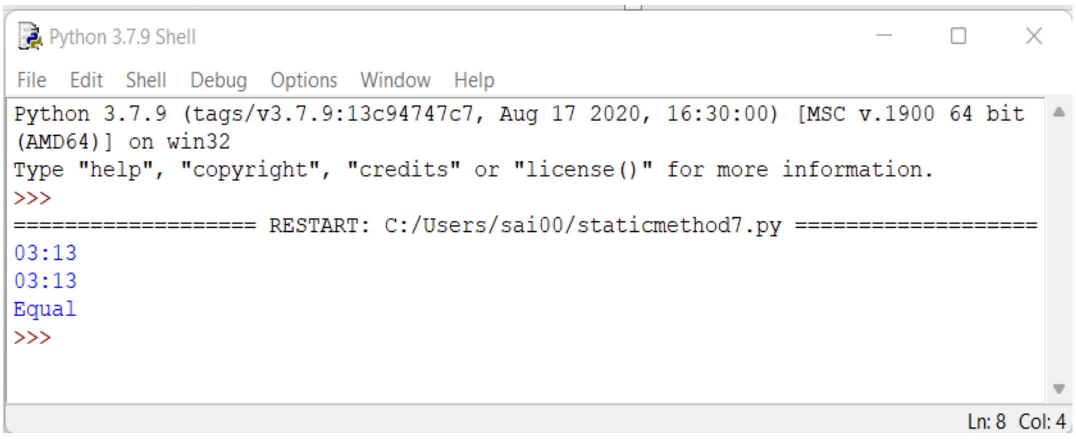
```
class Times:
    def __init__(self, time):
        self.time = time

    def getTime(self):
        return self.time

    @staticmethod
    def toDashTime(time):
        return time.replace("-",":")

class TimesWithColon(Times):
    def getTime(self):
        return Times.toDashTime(self.time)
```

Example:


staticmethod7.py - C:/Users/sai00/staticmethod7.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 25 Col: 26

```
#Create a utility function as a static method

class Times:
    def __init__(self, time):
        self.time = time

    def getTime(self):
        return self.time

    @staticmethod
    def toDashTime(time):
        return time.replace("-",":")

class TimesWithColon(Times):
    def getTime(self):
        return Times.toDashTime(self.time)



time = Times("03:13")
timeFromDB =TimesWithColon("03-13")


print(time.getTime())
print(timeFromDB.getTime())
if(time.getTime() == timeFromDB.getTime()):
    print("Equal")
else:
    print("Unequal")
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/staticmethod7.py ===================
03:13
03:13
Equal
>>>
```

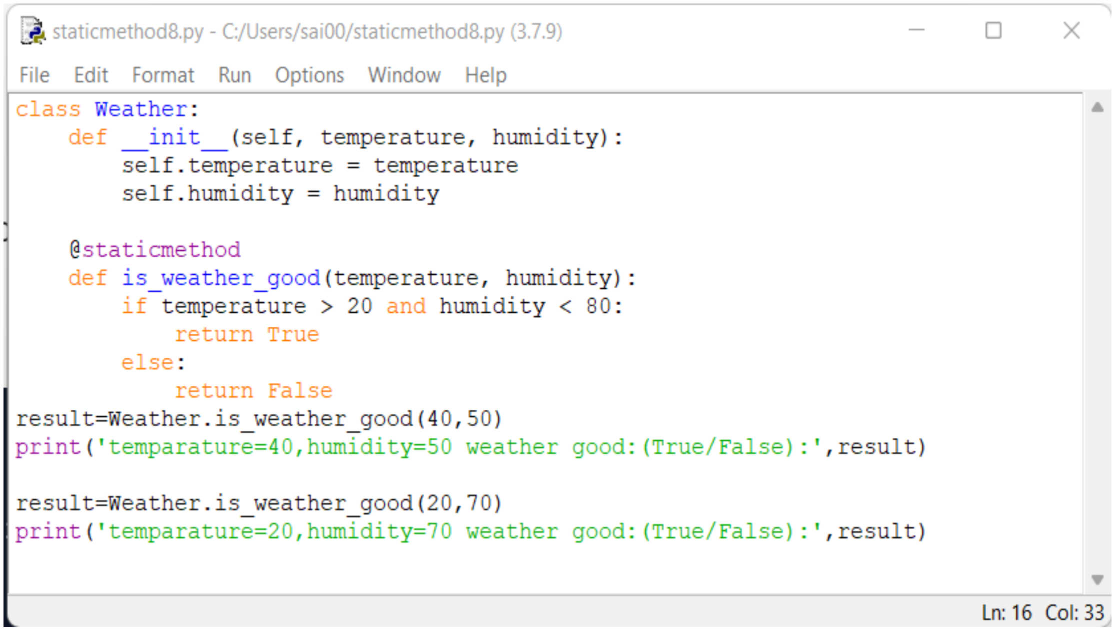
It would be undesirable for the subclass **TimesWithColon** to override the static utility function **toDashTime** in this scenario since the **toDashTime** method is only useful for one thing, which is to alter the time to dash-times.Overriding the **getTime()** function in the subclass in order to make it compatible with the **TimesWithColon** class would allow us to easily take advantage of the static method and use it to our greatest advantage.

14.5.3 Smart Performance

In situations when it is not necessary to access or modify data that is specific to an object, static methods may be marginally faster. In circumstances in which performance is of the utmost importance and method calls are made often, the static approach may provide a speed improvement that is not significant but nonetheless important.

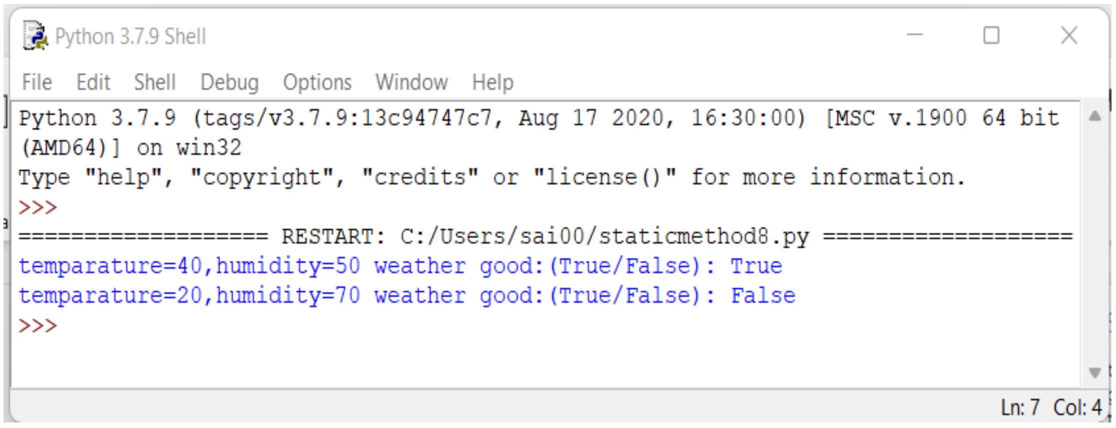
There is no requirement for that **is_weather_good()** method to access any instance state in this particular scenario. For the sake of making this function more efficient, we may make it a static method. It is not necessary to give any additional arguments when you use a static method because the method is looked up directly on the class. Static methods are thus quicker and more efficient than instance methods as a result of this.

Example:


staticmethod8.py - C:/Users/sai00/staticmethod8.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 16 Col: 33

```python
class Weather:
    def __init__(self, temperature, humidity):
        self.temperature = temperature
        self.humidity = humidity

    @staticmethod
    def is_weather_good(temperature, humidity):
        if temperature > 20 and humidity < 80:
            return True
        else:
            return False
result=Weather.is_weather_good(40,50)
print('temparature=40,humidity=50 weather good:(True/False):',result)

result=Weather.is_weather_good(20,70)
print('temparature=20,humidity=70 weather good:(True/False):',result)
```

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/sai00/staticmethod8.py ==================
temparature=40,humidity=50 weather good:(True/False): True
temparature=20,humidity=70 weather good:(True/False): False
>>>
```

14.5.4 Stateless Operations

Utilizing a static method is the best option in situations where a method does not need to access or modify data that is specific to a class or instance being used. The purity of the method is preserved because of this, as it guarantees that no extraneous object state is referenced.

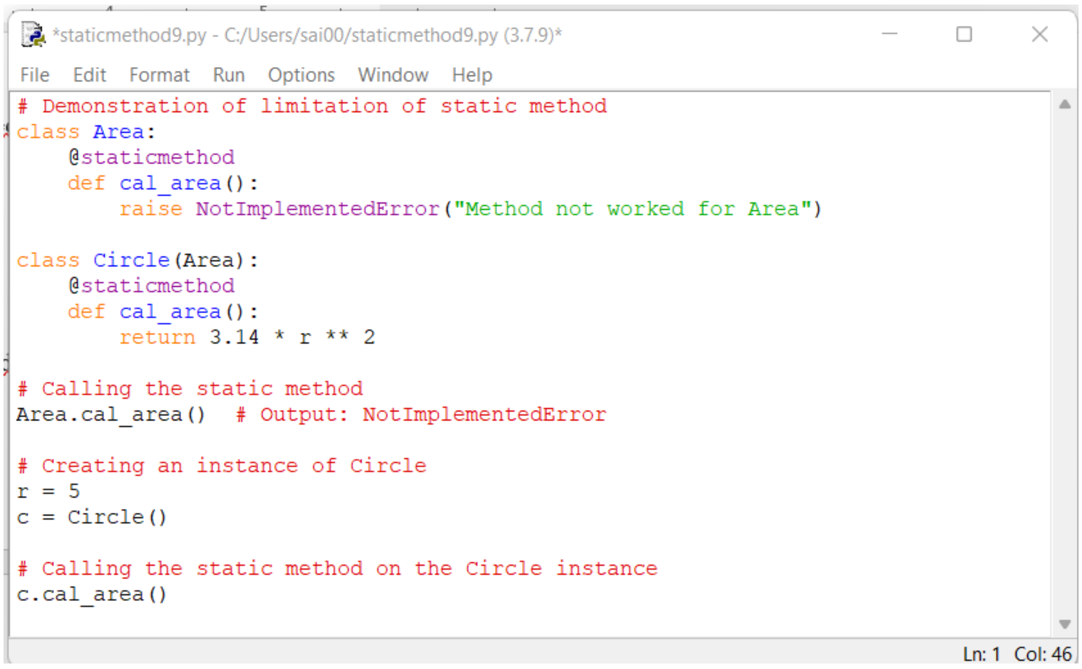
14.5.5 Cohesion Class

When a method does not need to access or modify data that is specific to a class or instance that is being used, the best solution is to use a static method. This is because static methods are not accessible to other methods. This ensures that no superfluous object state is referenced, which leads to the preservation of the method's purity. Consequently, the method is not compromised.

14.6. LIMITATIONS OF STATIC METHOD

- It is not possible for the method to access the instance or any of its attributes. Consider utilizing instance methods or class methods if you discover that you require access to data that is exclusive to an instance.
- It is not possible to make use of the approach. In the event that you desire polymorphic behavior, you should think about utilizing instance methods or class method.
- It is not possible for the technique to make use of inheritance. Using class methods is something to think about if you need to inherit features or characteristics.
- Excessive use of the procedure may result in a reduction in the modularity of the code. When it is suitable, you should think about using instance methods or class methods to improve the organization and maintainability of the code.
- When it comes to selecting between static methods, instance methods, and class methods, understanding these constraints will assist you in making good decisions in Real-time implementations.

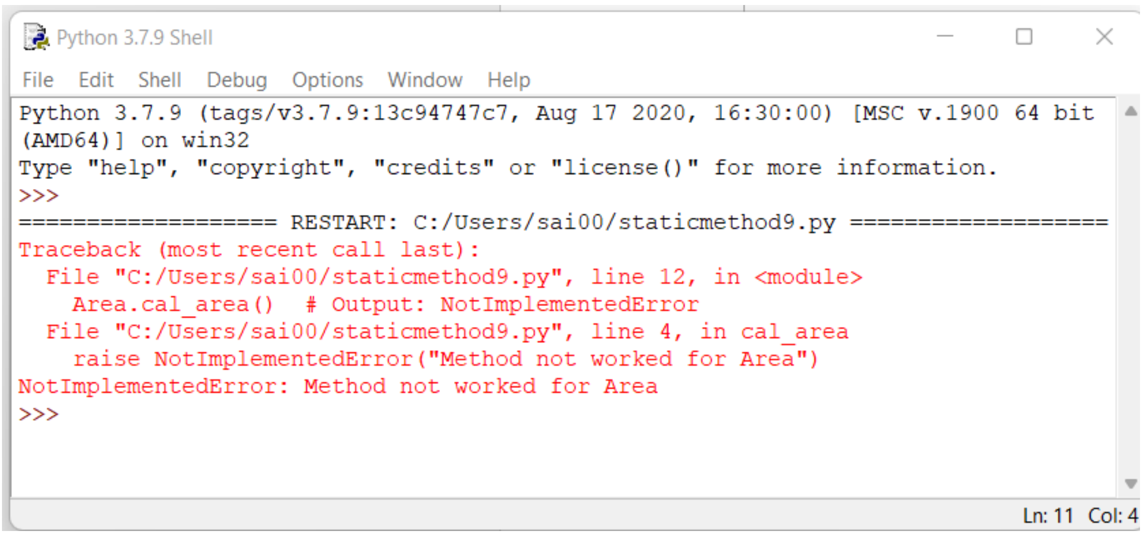
Example:


staticmethod9.py - C:/Users/sai00/staticmethod9.py (3.7.9)
File Edit Format Run Options Window Help
Ln: 1 Col: 46

```
# Demonstration of limitation of static method
class Area:
    @staticmethod
    def cal_area():
        raise NotImplementedError("Method not worked for Area")

class Circle(Area):
    @staticmethod
    def cal_area():
        return 3.14 * r ** 2

# Calling the static method
Area.cal_area()  # Output: NotImplementedError

# Creating an instance of Circle
r = 5
c = Circle()

# Calling the static method on the Circle instance
c.cal_area()
```

Calculations can be performed with static methods; however, these methods have restrictions when it comes to polymorphism and inheritance on the other hand.

Output:



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Users/sai00/staticmethod9.py ===================
Traceback (most recent call last):
  File "C:/Users/sai00/staticmethod9.py", line 12, in <module>
    Area.cal_area()  # Output: NotImplementedError
  File "C:/Users/sai00/staticmethod9.py", line 4, in cal_area
    raise NotImplementedError("Method not worked for Area")
NotImplementedError: Method not worked for Area
>>>
```

'NotImplementedError' was raised in the output that was shown above since the method was overridden. On the other hand, because static methods cannot be overridden, the same error is generated even when they are invoked on a **Circle** instance. The relevance of selecting the appropriate method type based on the behavior that is intended is brought to light by this limitation.

14.7 SUMMARY

When it comes to Python programming, static methods are extremely significant since they contribute to the creation of code that is not only comprehensible but also reusable, encapsulated, quicker, and simpler to test. The addition of static functions to your Python applications makes them simpler to manage and allows them to run more quickly. A comprehensive examination of the Python static method is presented in this chapter. Topics covered include its distinctions from other types of methods, its benefits, and its applications.

14.8 TECHNICAL TERMS

Static Method, Class Method, Instance Method, Inheritance, Polymorphism, Less Coherent, Overriding, Utility

14.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. What is static method? Discuss various applications.
2. Compare instance, class and static method.
3. Explain types of methods to create static method.

Short Notes:

1. Define Static Method.
2. List advantages of static methods
3. Limitations of static methods

14.10 SUGGESTED READINGS

1. Steven cooper – Data Science from Scratch, Kindle edition.

2. Reemathareja – Python Programming using problem solving approach, Oxford Publication

3. A Byte of Python by Swaroop C.H.

4. Python Programming for Beginners: 2 Books in 1—The Ultimate Step-by-Step Guide to Learn Python Programming Quickly with Practical Exercises by Mark Reed

5. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming by Eric Matthes

Dr. KAMPA LAVANYA