DEPARTMENT OF COMPUTER APPICATIONS
III B.com-CECS -V92023-3024)
DATA SCIENCE USING PYTHON MATERIAL
UNIT-1

INTRODUCTION TO DATA SCIENCE

Data Science: Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from data in various forms, both Structured and Unstructured data...

Data scientist: A data scientist is an analytics professional who is responsible for collecting, analyzing, and interpreting data to help drive decision-making in an organization. In businesses, data scientists typically mine data for information that can be used to predict customer behavior and identify new revenue opportunities.

Data science and its importance

Data Science is the study of extracting insights from massive amounts of data using various scientific approaches, processes, and algorithms. The development of big data, data analysis, and quantitative statistics has given rise to the term "data science." Data science is now more important than ever. The reason for this is data transformation. In the past, the data was in a structured format, was compact, and could be processed by straightforward BI tools.

However, most data nowadays are either semi-structured or unstructured, meaning it takes the form of multimedia such as photos, audio, and videos. Therefore, sophisticated analytical methods that can handle vast volumes of such heterogeneous data are needed for this data.

The need for a data scientist is expanding along with the value of data. They are increasingly becoming necessary components of goods, professions, governmental institutions, and nonprofit organizations. An information and computer scientist, database and software programmer, curator, and knowledgeable annotator are all examples of data scientists.

They are all crucial for the administration of digital data collection to be successful. The data scientist puts a lot of effort into sifting through a mountain of data to find relevant information and identify patterns and designs that can be utilized to pinpoint future goals and objectives. This demonstrates why data science matters and data scientists are becoming more well-known and significant.

Importance of Data Science in Business

Data science's goal is to assist organizations in comprehending the patterns of variance in data, including client information, business growth rates, data volume, or any measurable quantity. In data science, you use statistical and probabilistic models to analyze changes and improvements in historical or current data. Data science has transformed operations for businesses all over the world.

Importance of Data Science in Healthcare

The healthcare sector produces enormous datasets of valuable data on patient demographics, treatment plans, outcomes of medical exams, insurance, etc. The vast amounts of dispersed, structured, and unstructured data generated by healthcare systems can be processed, analyzed,

assimilated, and managed with the help of data science. This data needs to be managed and analyzed effectively to obtain true findings.

Importance of Data Science in the Future Companies today have access to massive databases due to documenting every aspect of client engagement. Data science plays a crucial role in analyzing and developing these data-driven machine-learning models. As the industry grows, more jobs should become accessible because analysis needs more data scientists. A bright future is expected for those who are interested in a career in data science.

Artificial intelligence is a key component in the future of data science. In the future, AI will likely be the most powerful tool that data scientists will have to work with. Artificial intelligence is already being used by businesses to make decisions and run their operations. Artificial intelligence will be used in real-world scenarios to use automated solutions to screen through massive volumes of data to find patterns that help present firms make better decisions.

Advantages of 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. Some key benefits include:

- **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%.
- **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.
- **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.

The process of data science:

The Data Science Process is a systematic approach to solving data-related problems and consists of the following steps:

Data Collection – After formulating any problem statement the main task is to calculate data that can help us in our analysis and manipulation. Sometimes data is collected by performing some kind of survey and there are times when it is done by performing scrapping.

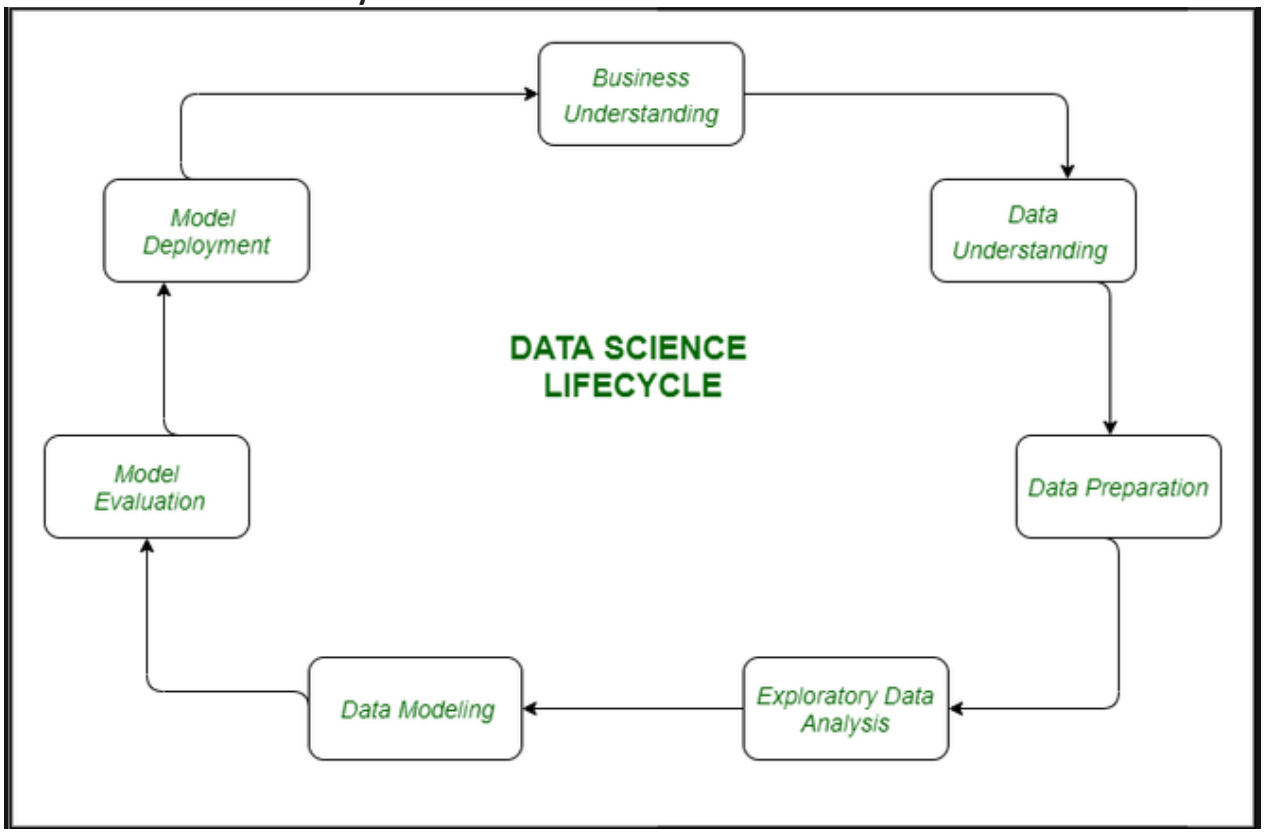
Data Cleaning – Most of the real-world data is not structured and requires cleaning and conversion into structured data before it can be used for any analysis or modeling.

Exploratory Data Analysis – This is the step in which we try to find the hidden patterns in the data at hand. Also, we try to analyze different factors which affect the target variable and the extent to which it does so. How the independent features are related to each other and what can be done to achieve the desired results all these answers can be extracted from this process as well. This also gives us a direction in which we should work to get started with the modeling process.

Model Building – Different types of machine learning algorithms as well as techniques have been developed which can easily identify complex patterns in the data which will be a very tedious task to be done by a human.

Model Deployment – After a model is developed and gives better results on the holdout or the real-world dataset then we deploy it and monitor its performance. This is the main part where we use our learning from the data to be applied in real-world applications and use cases.

Data Science Process Life Cycle



Responsibilities of a data scientist:

A data scientist's job is to gather a large amount of data, analyze it, separate out the essential information, and then utilize tools like SAS, R programming, Python, etc. to extract insights that may be used to increase the productivity and efficiency of the business. Depending on an organization's needs, data scientists have a wide range of roles and responsibilities. The following is a list of some of the data scientist roles and responsibilities:

- Collect data and identify data sources.
- Analyze huge amounts of data, both structured and unstructured
- Create solutions and strategies to business problems.
- Work with team members and leaders to develop data strategy.
- To discover trends and patterns, combine various algorithms and modules.
- Present data using various data visualization techniques and tools • Investigate additional technologies and tools for developing innovative data strategies.
- Create comprehensive analytical solutions, from data gathering to display; assist in the construction of data engineering pipelines.
- Supporting the data scientists, BI developers, and analysts' team as needed for their projects Working with the sales and pre-sales team on cost reduction, effort estimation, and cost optimization.
- To boost general effectiveness and performance, stay current with the newest tools, trends, and technologies.
- collaborating with the product team and partners to provide data-driven solutions created with original concepts.
- Create analytics solutions for businesses by combining various tools, applied statistics, and machine learning.
- Architect, implement, and monitor data pipelines, as well as conduct knowledge sharing sessions with peers to ensure effective data use.

Qualifications of a Data Scientist

Becoming a data scientist generally requires a very strong background in mathematics and computer science, as well as experience working with large amounts of data. In addition, it is often helpful to have experience with machine learning and statistical modeling. While there is no one specific path to becoming a data scientist, here are some helpful prerequisites or experiences that can improve your chances of success:

- A strong background in mathematics and computer science: As a data scientist, you will be working with large amounts of data daily. Therefore, it is essential that you have a strong foundation in mathematics and computer science. You should be comfortable with statistical methods and algorithms.
- Experience working with large amounts of data: Data scientists must be able to effectively manipulate and analyze large data sets. Therefore, it is important to have some experience working with large data sets before becoming a data scientist.
- Experience with machine learning and statistical modeling: Machine learning and statistical modeling are powerful tools that data scientists use to make predictions from data. Therefore, experience with these techniques is essential for anyone interested in becoming a data scientist.
- Strong communication and visualization skills: Data scientists must be able to effectively communicate their findings to others. Therefore, strong communication and visualization skills are essential for anyone interested in becoming a data scientist.

Why use Python for Data Science?

Python is an open source, interpreted, high level language and provides a great approach for object-oriented programming. It is one of the best languages used by data scientists for various data science projects/applications. Pythons provide great functionality to deal with mathematics, statistics, and scientific function. It provides great libraries to deal with data science applications. Allows developer to run the code anywhere, including Windows, Mac OS X, UNIX, and Linux.

Most Commonly used libraries for data science: NumPy, pandas, SciPy , Matplotlib , Seaborn

Python has been in demand for the past few years and the recent survey also suggested the same, Python leads the chart among the top programming languages in both the TIOBE index & PYPL Index. However, to support this, there are 5 concrete reasons behind this,

1. Easy to Learn: Being an open-source platform, Python has a simple and intuitive syntax that is easy to learn and read. This makes it a great language for beginners to learn data science.

2. Cross-Platform: Being a developer, you don't need to worry about the data types. The reason is, Python allows developers to run the code on Windows, Mac OS X, UNIX, and Linux.

3. Portable: Being an easy & beginner's friendly programming language, Python is highly portable in nature which means that a developer can run their code on different machines without making any further changes.

4. Extensive Library: Python has several powerful libraries that make data analysis and visualization easy. Pandas is a library for data manipulation and analysis, NumPy is a library for numerical computation, and Matplotlib is a library for data visualization.

5. Community Support: Python has a large and active community that supports and contributes to the development of various libraries and tools for data science. This community has created many useful libraries, including Pandas, NumPy, matplotlib, and SciPy, which are widely used in data science.

DEPARTMENT OF COMPUTER APPICATIONS
III B.com-CECS -V92023-3024)
DATA SCIENCE USING PYTHON MATERIAL
UNIT-2

INTRODUCTION TO PYTHON

What is python

Python is a high-level, interpreted programming language that was first released in 1991. It is designed to be easy to read and write, and it is used for a wide variety of applications, from web development and scientific computing to machine learning and data analysis.

One of the strengths of Python is its simplicity and readability. It has a relatively easy-to understand syntax and a vast array of libraries that make it useful for a wide range of tasks. Additionally, Python is an interpreted language, meaning that code can be executed without the need for compilation, making it easy to test and debug code.

Python is an open-source language, which means that it is free to use and distribute. It has a large community of users and developers who contribute to its development and maintenance, ensuring that it remains relevant and up to date.

Features of Python:

Python has a variety of features that make it a popular programming language for many different applications. Here are some of its key features:

1. Free and Open Source

Python language is freely available at the official website, and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open source, this means that source code is also available to the public. So, you can download it, use it as well as share it.

2. Easy to code

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, JavaScript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

3. Easy to Read

As you will see, learning Python is quite simple. As was already established, Python's syntax is straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

4. Object-Oriented Language

One of the key features of Python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

5. GUI Programming Support

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, python, or Tk in Python. PyQt5 is the most popular option for creating graphical apps with Python.

6. High-Level Language

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

7. Large Community Support

Python has gained popularity over the years. Our questions are constantly answered by the enormous Stack Overflow community. These websites have already provided answers to many questions about Python, so Python users can consult them as needed.

8. Easy to Debug

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program's issues once you understand how to interpret Python's error traces. Simply by glancing at the code, you can determine what it is designed to perform.

9. Python is a Portable language

Python language is also a portable language. For example, if we have Python code for Windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

10. Python is an integrated language

Python is also an integrated language because we can easily integrate Python with other languages like C, C++, etc.

11. Interpreted Language:

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called bytecode.

12. Large Standard Library

Python has a large standard library that provides a rich set of modules and functions, so you do not have to write your own code for every single thing. There are many libraries present in Python such as regular expressions, unit-testing, web browsers, etc.

13. Dynamically Typed Language

Python is a dynamically typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

14. Frontend and backend development

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like JavaScript. Backend is the strong forte of Python it's extensively used for this work because of its frameworks like Django and Flask.

15. Allocating Memory Dynamically

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

History of Python:

Python is a widely used general-purpose, high-level programming language. It was initially designed by **Guido van Rossum in 1991** and developed by **Python Software Foundation**. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

In the late 1980s, history was about to be written. It was that time when working on Python started. Soon after that, Guido Van Rossum began doing its application-based work in December of **1989 at Centrum Wickenden & Informatica (CWI)** which is situated in the Netherlands.

It was started firstly as a hobby project because he was looking for an interesting project to keep him occupied during Christmas. The programming language in which Python is said to have succeeded is ABC Programming Language, which had interfacing with the **Amoeba Operating System** and had the feature of exception handling. He had already helped to create ABC earlier in his career and he had seen some issues with ABC but liked most of the features.

After that what he did was very clever. He had taken the syntax of ABC, and some of its good features. It came with a lot of complaints too, so he fixed those issues completely and had created a good scripting language that had removed all the flaws. The inspiration for the name came from BBC's TV Show – 'Monty Python's Flying Circus', as he was a big fan of the TV show and, he wanted a short, unique, and slightly mysterious name for his invention and hence he named it Python! He was the **"Benevolent dictator for life" (BDFL)** until he stepped down from the position as the leader on **12th July 2018**. For quite some time he used to work for Google, but currently, he is working at Dropbox.

The language was finally released in 1991. When it was released, it used a lot fewer codes to express the concepts, when we compare it with Java, C++ & C. Its design philosophy was quite good too. Its main objective is to provide code readability and advanced developer productivity.

Execute the Python program:

Python programmers must know every possible way to run the Python scripts or code. This is the only way to verify whether code is working as we want. Python interpreter is responsible for executing the Python scripts. Python interpreter is a piece of software which works between the Python program and computer hardware.

The Python interactive mode: To run the Python code, we can use the Python interactive session. We need to start Python interactive session, just open a command-line or terminal in start menu, then type in python, and press enter key.

Here is the example of how to run Python code using interactive shell. It allows us to check every piece of code, and this facility makes it an awesome development tool. But once we close the session it will lose all code that we have written.

```
if True:
  print ("Answer")
  print ("True")
else:
  print ("Answer")
  print ("False")
```

Below are the few options to exit the interactive mode.

- Type built-in functions quit () or exit(). Or
- Type the enter ctrl+ Z key combination to end the current session of Python interactive shel

Python Program basic syntax:

Let us execute a Python program to print "Hello, World!" in two different modes of Python Programming.
(a) Interactive Mode Programming
(b) Script Mode Programming.

Python - Interactive Mode Programming

We can invoke a Python interpreter from command line by typing python at the command prompt as following −

```
>>> print ("Hello, World!")
```

If you are running older version of Python, like Python 2.4.x, then you would need to use print statement without parenthesis as in print "Hello, World!". However in Python version 3.x, this produces the following result −

Output: Hello, World!

Script Mode Programming

We can invoke the Python interpreter with a script parameter which begins the execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

```
>>>print ("Hello, World!")
```

We assume that you have Python interpreter path set in PATH variable. Now, let's try to run this program as follows −

```
$ python3 test.py
```

This produces the following result −

```
Hello, World!
```

Variable:

A variable in Python is a reserved memory location to store values. In Python, we need.
not declare a variable with some specific data type.
For example: >>> n = 300.

Datatypes in Python:

Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. Variables can hold values, and every value has a datatype. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type. There are different types of data types in Python. Some built-in Python data types are:

- Numeric data types: int, float, complex
- String data types: str
- Sequence types: list, tuple, range
- Binary types: bytes, byte array, memory view
- Mapping data type: dict

- Boolean type: bool

Numeric Data type

In Python, numeric data type is used to hold numeric values.

- int - holds signed integers of non-limited length.
- float - holds floating decimal points and it's accurate up to 15 decimal places.
- complex - holds complex numbers.

String Data Type

String is a sequence of characters represented by either single or double quotes. " ..... "

List Data Type

List is an ordered collection of similar or different types of items separated by commas and enclosed within brackets.

Tuple Data Type Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified. In Python, we use the parentheses () to store items of a tuple.

Dictionary Data Type

Python dictionary is an ordered collection of items. It stores elements in key/value pairs. Here, keys are unique identifiers that are associated with each value.

Keyword:

Keywords are predefined, reserved words used in Python programming that have special. meanings to the compiler. We cannot use a keyword as a variable name, function name, or any other identifier. They are used to define the syntax and structure of the Python language. All the keywords except True, False and None are in lowercase.

Data type:

Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. Variables can hold values, and every value has a datatype. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.
Example: >>> a=5

Operator:

In Python, operators are special symbols that designate that some sort of computation should. be performed. The values that an operator acts on are called operands.
Here is an example:

```
>>> a = 10
>>> b = 20
>>> a + b
```

Result = 30 In this case, the **+** operator adds the operands **a** and **b** together.

Operators in Python:

Python Operators: Operators are used to perform operations on variables and values. Python operators are the constructs which can manipulate the value of operands. These are symbols used for the purpose of logical, arithmetic, and various other operations. Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator. Now we will study different types of Python operators.

Types of Python Operators:

Python language supports the following types of operators.

Arithmetic Operators (+, -, * , / , % ,**, //)

Comparison (Relational)

Operators (= =, != , < , > , <= , =>)

Assignment Operators (+=, -=, *= , /= , %= ,**=, //=)

Logical Operators (AND , OR , NOT)

Bitwise Operators (& , | , ^ , >> , <);

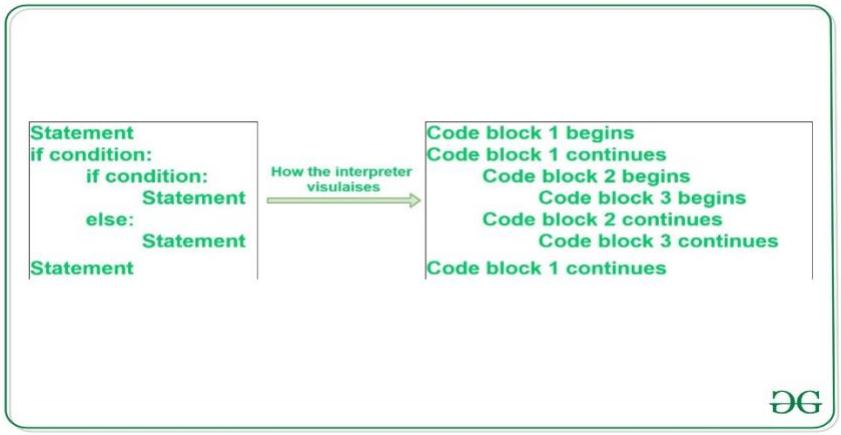
Membership Operators (in , not in)

Identity Operators (is , is not)

Here's an overview of each of these types of operators:

- **Arithmetic Operators: These** operators are used to perform arithmetic operations on numeric values. The arithmetic operators in Python are + (addition), - (subtraction), * (multiplication), / (division), // (floor division), % (modulus), and ** (exponentiation).
- **Comparison Operators:** These operators are used to compare two values and return a Boolean value (True or False). The comparison operators in Python are == (equal to), != (not equal to), < (less than), > (greater than), <= (less than or equal to), and >= (greater than or equal to).
- **Logical Operators:** These operators are used to perform logical operations on Boolean values. The logical operators in Python are and, or, and not.
- **Bitwise Operators:** These operators are used to perform bitwise operations on integers. The bitwise operators in Python are & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT), << (left shift), and >> (right shift). • Assignment Operators: These operators are used to assign values to variables. The assignment operators in Python are =, +=, -=, *=, /=, //=, %= and **=.
- **Membership Operators:** These operators are used to test if a value is a member of a sequence. The membership operators in Python are in and not in.
- **Identity Operators:** These operators are used to test if two variables refer to the same object. The identity operators in Python are and is not. Overall, understanding and using these operators effectively can help you perform a wide variety of operations in Python programming.

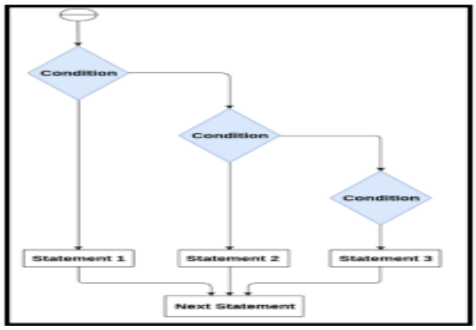
Indentation(:) Python indentation refers to adding white space before a statement to a particular block of code. In another word, all the statements with the same space to the right, belong to the same code block.



Conditional statements in python:

A conditional statement as the name suggests itself, is used to handle conditions in your program. These statements guide the program while making decisions based on the conditions encountered by the program. Conditional Statements in Python:

- if statement
- if-else statement
- if-elif-else statement
- Nested – if statement



The if statement:

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be.
any valid logical expression which can be either evaluated to true or false.

Syntax: if expression:
statement

Example:
```
num = int (input ("enter the number:"))
    if num%2 == 0:
    print ("Number is even")
```

Output: enter the number:10
Number is even.

The if-else statement:

The if-else statement provides an else block combined withthe if statement which is executed in the false case of the condition.
If the condition is true, then the if-block is executed. Otherwise, the else block is executed.

Syntax:

```
if condition:
#block of statements
else:
#another block of statements (else-block)
```

Example: Program to check whether a person is eligible to vote or not.

```
age = int (input("Enter your age? "))
if age>=18:
        print("You are eligible to vote !!");
else:
        print("Sorry! you have to wait !!");
```

Output:

```
Enter your age? 90
You are eligible to vote!!
```

The elif statement:

The elif statement enables us to check multiple conditions and execute. the specific block of statements depending upon the true condition. among them. We can have any number of elif statements in our program. depending upon our need.

Syntax:

```
if expression 1:
 # block of statements
 elif expression 2:
 # block of statements
elif expression 3:
 # block of statements
else:
 # block of statements
```

Example:

```
number = int(input("Enter the number?"))
if number==10:
        print("number is equals to 10")
elif number==50:
        print("number is equal to 50");
elif number==100:
        print("number is equal to 100");
else:
        print("number is not equal to 10, 50 or 100");
```

Output: Enter the number?15
number is not equal to 10, 50 or 100

Looping statements in python

In Python, a looping statement is used to execute a block of code repeatedly. There are two main types of looping statements in Python:

- for loops
- while loops

for loop:

A **for** loop is used to iterate over a sequence of values, such as a list, tuple, or string. The general syntax for a **for** loop is as follows the process of traversing a sequence is known as iteration.

```
for variable in sequence:
#code to be executed
```

In this syntax,
the **variable** takes on each value in the **sequence** and the code block is.
executed for each value of the variable. the variable value is used to hold the value of every. item present in the sequence before the iteration begins until this iteration is. completed. Loop iterates until the final item of the sequence are reached.

Code:

```
# Python program to show how the for loop works
# Creating a sequence which is a tuple of numbers
numbers = [4, 2, 6, 7, 3, 5, 8, 10, 6, 1, 9, 2]
# variable to store the square of the number
square = 0
# Creating an empty list
squares = []
# Creating a for loop
for value in numbers:
 square = value ** 2
 squares. Append(square)
print ("The list of squares is", squares)
```

Output:

The list of squares is[16,4,36,49,9,25,64,100,36,1,81,4]

while loop:

A **while** loop is used to execute a block of code repeatedly while certain. condition is true. The general syntax for a while loop is as follows It is also called a pre-tested loop.

In Python, the **while** loop executes the statement or group of statements repeatedly while the given condition is True. And when the condition becomes false, the loop ends and moves to the next statement after the loop.

```
While condition:
    #code to be executed
```

In this syntax, the condition is a Boolean expression that is evaluated before each iteration of

the loop. If the condition is true, the code block is executed, and then the condition is evaluated again.

Input:

```
Count=0
While(count<5):
        Count=count+1
        Print("flexible")
```

Output:

flexible
flexible
flexible
flexible
flexible

The loop prints 'Flexiple' till the value of count becomes 5 and the condition is False.

Python break Statement

The break statement is used to terminate the loop immediately when it is encountered.
The syntax of the break statement is:

break

```
for val in sequence:
  # code
  if condition:
    break

  # code
```

```
while condition:
  # code
  if condition:
    break

  # code
```

```
# program to find first 5 multiples of 6

i = 1

while i <= 10:
    print('6 * ',(i), '=',6 * i)
```

```
    if i >= 5:
        break

    i = i + 1
```

Output

```
6 *  1 = 6
6 *  2 = 12
6 *  3 = 18
6 *  4 = 24
6 *  5 = 30
```

In the above example, we have used the `while` loop to find the first **5** multiples of **6**. Here notice the line,

```
if i >= 5:
    break
```

This means when `i` is greater than or equal to **5**, the `while` loop is terminated.

Python continue Statement

The continue statement is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.
The syntax of the continue statement is:
continue

```
for val in sequence:
    # code
    if condition:
        continue

    # code
```

```
while condition:
    # code
    if condition:
        continue

    # code
```

```
# program to print odd numbers from 1 to 10

num = 0

while num < 10:
    num += 1

    if (num % 2) == 0:
        continue

    print(num)
```

Output

```
1
3
5
7
9
```

In the above example, we have used the `while` loop to print the odd numbers between **1** to **10**.

Notice the line,

```
if (num % 2) == 0:
    continue
```

Here, when the number is even, the continue statement skips the current iteration and starts the next iteration.

Pass:

In Python programming, the pass statement is a null statement which can be used as a placeholder for future code.
Suppose we have a loop or a function that is not implemented yet, but we want to implement. it in the future. In such cases, we can use the pass statement.

DEPARTMENT OF COMPUTER APPICATIONS
III B.com-CECS -V92023-3024)
DATA SCIENCE USING PYTHON MATERIAL
UNIT-3

CONTROL STRUCTURES AND STRINGS

Strings:

A String is a data structure in Python that represents a sequence of characters. It is an immutable data type, meaning that once you have created a string, you cannot change it. Strings are used widely in many different applications, such as storing and manipulating text data, representing names, addresses, and other types of data that can be represented as text.

String in Python:

Python does not have a character data type, a single character is simply a string with a length of 1.

Example: "Geeksforgeeks" or 'Geeksforgeeks' or "a"

```
print("A Computer Science portal for geeks")
print('A')
```

output: A Computer Science portal for geeks
A

Creating a String in Python

Strings in Python can be created using single quotes or double quotes or even triple quotes. Let us see how we can define a string in Python.

Example:

In this example, we will demonstrate different ways to create a Python String. We will create a string using single quotes (' '), double quotes (" "), and triple double quotes (""" """). The triple quotes can be used to declare multiline strings in Python.

```
# Python Program for
# Creation of String

# Creating a String
```

```
# with single Quotes
String1 = 'Welcome to the Geeks World'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String
# with double Quotes
String1 = "I'm a Geek"
print("\nString with the use of Double Quotes: ")
print(String1)
```

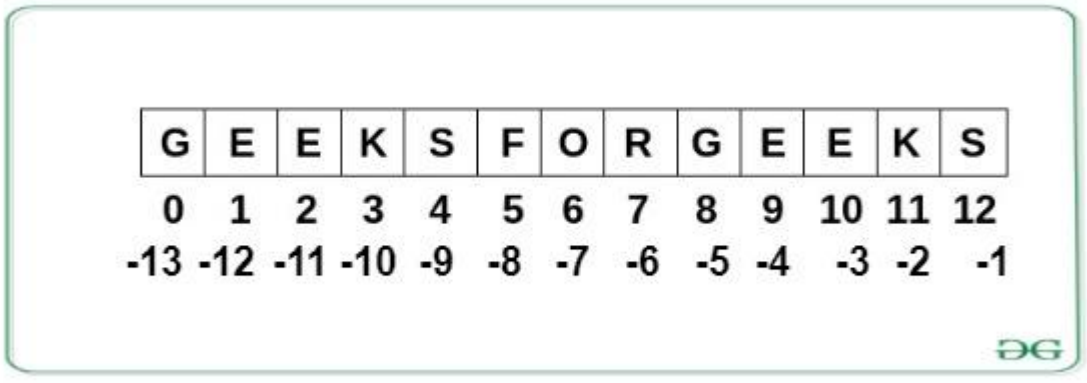
output:

String with the use of Single Quotes:
Welcome to the Geeks World
String with the use of Double Quotes:
I'm a Geek

Accessing characters in Python String

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.

While accessing an index out of the range will cause an **IndexError**. Only Integers are allowed to be passed as an index, float or other types that will cause a **TypeError**.



Python String indexing

Example:

In this example, we will define a string in Python and access its characters using positive and negative indexing. The 0th element will be the first character of the string whereas the -1th element is the last character of the string.

- Python3

```
# Python Program to Access

# characters of String

String1 = "GeeksForGeeks"

print("Initial String: ")

print(String1)

# Printing First character

print("\nFirst character of String is: ")

print(String1[0])

# Printing Last character

print("\nLast character of String is: ")

print(String1[-1])
```

Output:

Initial String:
GeeksForGeeks
First character of String is:
G
Last cha racter of String is:
s

String Slicing

In Python, the String Slicing method is used to access a range of characters in the String. Slicing in a String is done by using a Slicing operator, i.e., a colon (:). One thing to keep in mind while using this method is that the string returned after slicing includes the character at the start index but not the character at the last index.

Example:

In this example, we will use the string-slicing method to extract a substring of the original string. The [3:12] indicates that the string slicing will start from the 3rd index of the string to the 12th index, (12th character not including). We can also use negative indexing in string slicing.

- Python3

```
# Python Program to

# demonstrate String slicing

# Creating a String

String1 = "GeeksForGeeks"

print("Initial String: ")

print(String1)

# Printing 3rd to 12th character

print("\nSlicing characters from 3-12: ")

print(String1[3:12])

# Printing characters between

# 3rd and 2nd last character

print("\nSlicing characters between " +
```

```
    "3rd and 2nd last character: ")

print(String1[3:-2])
```

Output:

Initial String:
GeeksForGeeks
Slicing characters from 3-12:
ksForGeek
Slicing characters between 3rd and 2nd last character:
ksForGee

Python basic List operations:

In Python, lists are used to store multiple data at once.

Suppose we need to record the ages of **5** students. Instead of creating **5** separate variables, we can simply create a list.

Create a List

We create a list by placing elements inside [], separated by commas. For example,

```
ages =  [19, 26, 23]

print(ages)

# Output: [19, 26, 23]
```

Run Code

Here, we have created a list named ages with **3** integer items.

A list can

- store elements of different types (integer, float, string, etc.)
- store duplicate elements

```
# list with elements of different data types
list1 = [1, "Hello", 3.4]

# list with duplicate elements
list1 = [1, "Hello", 3.4, "Hello", 1]

# empty list
list3 = []
```

Note: We can also create a list using the list() constructor.

Access List Elements

In Python, lists are **ordered** and each item in a list is associated with a number. The number is known as a list **index**.

The index of the first element is **0**, second element is **1** and so on. For example,

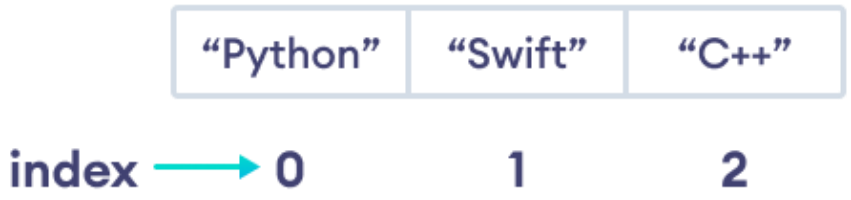
```
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[0])  # Python

# access item at index 2
print(languages[2])  # C++
```

Run Code

In the above example, we have created a list named `languages`.



List Indexing in Python

Here, we can see each list item is associated with the index number. And we have used the index number to access the items.

> **Remember:** The list index always starts with **0**. Hence, the first element of a list is present at index **0**, not **1**.

Slicing of a List

In Python, it is possible to access a portion of a list using the slicing operator `:`. For example,

```
# List slicing in Python

my_list = ['p','r','o','g','r','a','m','i','z']

# items from index 2 to index 4
print(my_list[2:5])

# items from index 5 to end
print(my_list[5:])

# items beginning to end
print(my_list[:])
```

Run Code

Output

```
['o', 'g', 'r']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

Here,

- `my_list[2:5]` returns a list with items from index **2** to index **4**.
- `my_list[5:]` returns a list with items from index **5** to the end.
- `my_list[:]` returns all list items

Change List Items

Python lists are mutable. Meaning lists are changeable. And we can change items of a list by assigning new values using the `=` operator. For example,

```
languages = ['Python', 'Swift', 'C++']
```

```
# changing the third item to 'C'
languages[2] = 'C'

print(languages)  # ['Python', 'Swift', 'C']
```
Run Code

Here, initially the value at index **3** is `'C++'`. We then changed the value to `'C'` using

```
languages[2] = 'C'
```

- **Modifying a list:**

Lists are mutable, which means you can change their contents after they are created. You can add elements to a list using the append () method, remove elements using the remove () method, or modify elements using indexing.

```
my_list.append (6) #Add a new element to the end of the list
my list.remove('four) #remove the specified element from the list
my_list[0]='one' # Modify the first element of the list
```

- **Combining Lists:** You can combine two lists into a single list using the + operator

```
my_list1=[1,2,3]
my_list2=[4,5,6]
combined_ list = my list1+my_list2
print(combined_list) #output) ]1,2,3,4,5,6]
```

- **Sorting a list:** You can sort the elements of a list using the sort () method. By default, the elements are sorted in ascending order.

```
my_list=[3,1,4,1,5,9,2,6,5]
my_list.sort()
print(my_list) #output: [1,1,2,3,4,5,5,6,9]
```

Functions in Lists:

In Python, you can use a **list** function which creates a collection that can be manipulated for your analysis. This collection of data is called a list object.

While all methods are functions in Python, not all functions are methods. There is a key difference between functions and methods in Python. Functions take objects as inputs. Methods in contrast act on objects.

Python offers the following list functions:

- **sort()**: Sorts the list in ascending order.
- **type(list)**: It returns the class type of an object.
- **append()**: Adds a single element to a list.
- **extend()**: Adds multiple elements to a list.
- **index()**: Returns the first appearance of the specified value.
- **max(list)**: It returns an item from the list with max value.
- **min(list)**: It returns an item from the list with min value.
- **len(list)**: It gives the total length of the list.
- **list(seq)**: Converts a tuple into a list.
- **cmp(list1, list2)**: It compares elements of both lists list1 and list2.
- **filter(fun,list)**: filter the list using the Python function.
- Python List Functions & Methods

METHODS:

Python sort list method

The `sort()` method is a built-in Python method that, by default, sorts the list in ascending order. However, you can modify the order from ascending to descending by specifying the sorting criteria.

```
prices = [238.11, 237.81, 238.91]

prices.sort()

print(prices)
```

output:

```
[237.81, 238.11, 238.91]
```

Python list append method

The `append()` method will add certain content you enter to the end of the elements you select.

```
months = ['January', 'February', 'March'

months.append('April')

print(months)
```

output:

```
['January', 'February', 'March', 'April']
```

Python list extend method

The `extend()` method increases the length of the list by the number of elements that are provided to the method, so if you want to add multiple elements to the list, you can use this method.

Example

```
x = [1, 2, 3]


x.extend([4, 5])


x


output:
[1, 2, 3, 4, 5]
```

FUNCTIONS:

Python list type() function

For the `type()` function, it returns the class type of an object
The sort() method is a built-in Python method that, by default, sorts the list in ascending order.

Python list max function

The `max()` function will return the highest value of the inputted values.

Example

In this example, we will look to use the `max()` function to find the maximum price in the list named `price`.

```
# Find the maximum price in the list price
prices = [159.54, 37.13, 71.17]
price_max = max(prices)
print(price_max)


output:
159.54
```

Python list min function

The `min()` function will return the lowest value of the inputted values.

Example

In this example, you will find the month with the smallest consumer price index (CPI).

To identify the month with the smallest consumer price index, you first apply the `min()` function on prices to identify the `min_price`. Next, you can use the index method to find the index location of the `min_price`. Using this indexed location on months, you can identify the month with the smallest consumer price index.

```
months = ['January', 'February', 'March']


prices = [238.11, 237.81, 238.91]
```

Python list len function

The `len()` function shows the number of elements in a list. In the below example, we will look at stock price data again using integers.

Example

```
stock_price_1 = [50.23]
stock_price_2 = [75.14, 85.64, 11.28]


print('stock_price_1 length is ', len(stock_price_1))
print('stock_price_2 length is ', len(stock_price_2))
```

Python list() function

The `list()` function takes an iterable construct and turns it into a list.

Syntax

```
list([iterable])
```

Python cmp function

For the `cmp()` function, it takes two values and compares them against one another. It will then return a negative, zero, or positive value based on what was inputted.

Example

In the example below, we have two stock prices, and we will compare the integer values to see which one is larger:

```
stock_price_1 = [50.23]
stock_price_2 = [75.14]


print(cmp(stock_price_1, stock_price_2))
print(cmp(stock_price_1, stock_price_1))
print(cmp(stock_price_2, stock_price_1))
```

Python Tuple

A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses `()`, separated by commas. The parentheses are optional, however, it is a good practice to use them.
A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

```
# Different types of tuples

# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)
```

Output

```
()
(1, 2, 3)
```

1. Tuples are immutable, which means that once you create a tuple, you cannot modify its contents. In contrast, you can modify the contents of a list.
2. Tuples use less memory than lists, and can be faster to access in some cases.

Accessing a Tuple: To access the elements of a tuple, you can use indexing, just like you would with a list:

Methods in Tuple: However, there are a few methods that can be used with tuples:

- count (): This method returns the number of occurrences of a specified value in a tuple.

• index (): This method returns the index of the first occurrence of a specified value in a tuple. Note that because tuples are immutable, you cannot modify their elements individually. However, you can replace the entire tuple with a new one that has different value.

Dictionaries in python:

In Python, a dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable.

Creating a Dictionary: To create a dictionary in Python, you can use curly braces {} and separate the key-value pairs with colons. Here's an example: In this example, my_dict is a dictionary with three items, where "apple", "banana", and "orange" are the keys, and 2, 3, and 1 are the corresponding values.

```
my_dict={"apple":2,"banana":3,"orange":1}
```

Accessing a Dictionary: You can access the values in a dictionary by specifying the corresponding key in square brackets. For example: There is also a method called get() that will also help in accessing the element from a dictionary. This method accepts key as argument and returns the value.

```
print(my_dict=["apple"])
#output:2
```

Deleting a Dictionary: To delete an item from a dictionary, you can use the del keyword:

```
del my_dict['banana']
```

Adding an element to Dictionary: You can also add new key-value pairs to a dictionary by assigning a value to a new key

```
my_dict=['kiwi']=1
```

DEPARTMENT OF COMPUTER APPICATIONS
III B.com-CECS -V92023-3024)
DATA SCIENCE USING PYTHON MATERIAL
UNIT-4

Functions and Methods:

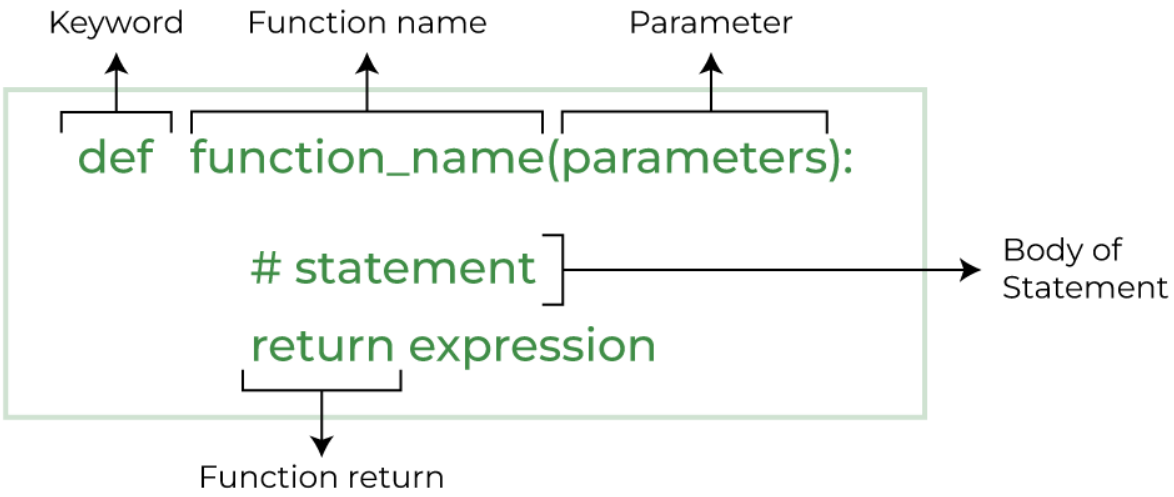
Functions: A Function is a block of small program (small piece of code) which consists of a code to compute some calculation. We write functions especially for code reusability. Once we write a function, it can be reused as and when required in any number of times. So, functions are also called as reusable code. This code is used to perform some action or task.

Some Benefits of Using Functions

- Increase Code Readability
- Increase Code Reusability

Python Function Declaration

The syntax to declare a function is:



Syntax of Python Function Declaration

Types of Functions in Python

There are mainly two types of functions in Python.

Built-in library function: These are Standard functions in Python that are available to use.

User-defined function: We can create our own functions based on our requirements.

Creating a Function in Python

We can create a user-defined function in Python, using the def keyword. We can add any type of functionalities and properties to it as we require.

A simple Python functions

```
def fun():
  print("Welcome to GFG")
```

Calling a Python Function

After creating a function in Python we can call it by using the name of the function followed by parenthesis containing parameters of that particular function.

```
# A simple Python function
def fun():
    print("Welcome to GFG")


# Driver code to call a function
fun()
```

Output:

```
Welcome to GFG
```

Python Function with Parameters

If you have experience in C/C++ or Java then you must be thinking about the return type of the function and data type of arguments. That is possible in Python as well (specifically for Python 3.5 and above).

Defining and calling a function with parameters

```
 def function name(parameter: datatype) -> return type:
    """Docstring"""
    # body of the function
    return expression
```

The following example uses arguments and parameters that you will learn later in this article so you can come back to it again if not understood.

```
def add(num1: int, num2: int) -> int:
    """Add two numbers"""
    num3 = num1 + num2

    return num3

# Driver code
num1, num2 = 5, 15
ans = add(num1, num2)
print(f"The addition of {num1} and {num2} results {ans}.")
```

Output:

The addition of 5 and 15 results 20.

Python Function Arguments:

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.
In this example, we will create a simple function in Python to check whether the number passed as an argument to the function is even or odd.

```
# A simple Python function to check
# whether x is even or odd
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")


# Driver code to call the function
evenOdd(2)
evenOdd(3)
```

output:
even
odd

Types of Python Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following 4 types of function arguments.

- **Default argument**
- **Keyword arguments (named arguments)**
- **Positional arguments**
- **Arbitrary arguments** (variable-length arguments *args and **kwargs)

Let's discuss each type in detail.

Default Arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

```
# Python program to demonstrate
# default arguments
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)

# Driver code (We call myFun() with only
# argument)
myFun(10)
```

Output:
x: 10
y: 50
Like C++ default arguments, any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

Keyword Arguments

The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.

```
## Python program to demonstrate Keyword Arguments

def student(firstname, lastname):
    print(firstname, lastname)


# Keyword arguments
student(firstname='ISHA', lastname='SHEIK')
```

Output:
ISHA SHEIK

Positional Arguments:

We used the Position argument during the function call so that the first argument (or value) is assigned to name and the second argument (or value) is assigned to age. By changing the position, or if you forget the order of the positions, the values can be used in the wrong places, as shown in the Case-2 example below, where 27 is assigned to the name and Suraj is assigned to the age.

```
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is ", age)


# You will get correct output because
# argument is given in order
print("Case-1:")
nameAge("Suraj", 27)
# You will get incorrect output because
# argument is not in order
print("\nCase-2:")
nameAge(27, "Suraj")
```

Output:
Case-1:
Hi, I am Suraj
My age is 27

Case-2:
Hi, I am 27
My age is Suraj

Global and Local Variables in Python

Python Global variables are those which are not defined inside any function and have a global scope whereas Python **local variables** are those which are defined inside a function and their scope is limited to that function only. In other words, we can say that local variables are accessible only inside the function in which it was initialized whereas the global variables are accessible throughout the program and inside every function.

Python Local Variables

Local variables in Python are those which are initialized inside a function and belong only to that function. It cannot be accessed anywhere outside the function. Let's see how to create a local variable.
Creating local variables in Python
Defining and accessing local variables

```
def f():

    # local variable
    s = "I love Geeksforgeeks"
    print(s)


# Driver code
f()
```

Python Global Variables
These are those which are defined outside any function and which are accessible throughout the program, i.e., inside and outside of every function. Let's see how to create a Python global variable.

Create a global variable in Python
Defining and accessing Python global variables.

```
# This function uses global variable s
def f():
    print("Inside Function", s)

# Global scope
s = "I love PYTHON"
f()
print("Outside Function", s)
```

Output

Inside Function I love PYTHON
Outside Function I love PYTHON
The variable s is defined as the global variable and is used both inside the function as well as outside the function.

Recursive Function in Python

A function is called recursive when it is called by itself. Let's understand this with an example. Let's consider a function which calculates the factorial of a number. It can be written as a recursive function as explained below.

Example: Factorial using the recursive function in python (Demo33.py)

```
def factorial(n):
if n ==0:
result=1
else:
result = n * factorial(n-1)
return result
x= factorial(4)
print("Factorial of 4 is: ",x)
```

Output: Factorial of 4 is: 24

Lambda Function (Anonymous Functions) in Python

The name of the function is the mandatory item in the function definition as discussed earlier. But, in python, we have a keyword called 'lambda' with which we can define a simple function in a single line without naming it. Such functions are called lambda functions.

Syntax: lambda argument_list: expression

Example: Lambda Function in Python (Demo35.py)

```
s = lambda a: a*a
x=s(4)
print(x)
```

Output: 16

Python Random Module

Python Random module is an in-built module of Python that is used to generate random numbers in Python. These are pseudo-random numbers means they are not truly random. This module can be used to perform random actions such as generating random numbers, printing random a value for a list or string, etc.

List of all the functions Python Random Module

Function Name	Description
seed()	Initialize the random number generator
getstate()	Returns an object with the current internal state of the random number generator
setstate()	Used to restore the state of the random number generator back to the specified state
getrandbits()	Return an integer with a specified number of bits
randrange()	Returns a random number within the range
randint()	Returns a random integer within the range
choice()	Returns a random item from a list, tuple, or string
choices()	Returns multiple random elements from the list with replacement
sample()	Returns a particular length list of items chosen from the sequence
random()	Generate random floating numbers
uniform()	Return a random floating number between two numbers both inclusive
triangular()	Return a random floating point number within a range with a bias towards one extreme
betavariate()	Return a random floating point number with beta distribution
expovariate()	Return a random floating point number with exponential distribution
gammavariate()	Return a random floating point number with a gamma distribution

Function Name	Description
gauss()	Return a random floating point number with Gaussian distribution
lognormvariate()	Return a random floating point number with a log-normal distribution
normalvariate()	Return a random floating point number with normal distribution
vonmisesvariate()	Return a random floating point number with von Mises distribution or circular normal distribution
paretovariate()	Return a random floating point number with a Pareto distribution
weibullvariate()	Return a random floating point number with Weibull distribution

Python Math Module

Sometimes when working with some kind of financial or scientific projects it becomes necessary to implement mathematical calculations in the project. Python provides the math module to deal with such calculations. Math module provides functions to deal with both basic operations such as
addition(+)
subtraction(-),
multiplication(*),
division(/)
and advance operations like
trigonometric,
logarithmic,
exponential functions.
In this article, we learn about the math module from basics to advance using the help of a huge dataset containing functions explained with the help of good examples.

Constants provided by the math module

Math module provides various the value of various constants like pi, tau. Having such constants saves the time of writing the value of each constant every time we want to use it and that too with great precision. Constants provided by the math module are –

- Euler's Number
- Pi
- Tau
- Infinity
- Not a Number (NaN)

DEPARTMENT OF COMPUTER APPICATIONS
III B.com-CECS -V92023-3024)
DATA SCIENCE USING PYTHON MATERIAL
UNIT-5

Classes & Objects:

Class:

In Python, a class is a blueprint for creating objects. It defines a set of attributes and methods that objects of that class will have. Attributes are variables that hold values and describe the characteristics of the object, while methods are functions that define the behaviour of the object. To define a class in Python, you use the class keyword, followed by the name of the class and a colon.

```
Class ClassName:
   #class definition
```

Object: An object is called an instance of a class. For example, suppose ” Bike “ is a class then we can create objects like bike1, bike2, etc from the class.

Here's the syntax to create an object.

```
objectName = ClassName()
```

Class method in Python: In Python, a class method is a method that is bound to the class rather than an instance of the class. This means that it can be called the class itself, rather than an instance of the class. To create a class method in Python, you need to use the @classmethod decorator. The first argument of a class method is conventionally named cls, and it represents the class itself. You can then use this cls parameter to access class-level attributes and methods. Here's an example of how to create a class method in Python:

```
Class Myclass:
   Class_variable=42
      @classmethod

      Def class method(cls)

      Print(“class method called”)

      Print(“class variable:” cls.class_variable)
```

In this example, we have defined a class called MyClass, which has a class variable called class variable. We have also defined a class method called class_method, which takes the cls parameter and prints a message along with the value of the class_variable. To call the class method, you can do so on the class itself, rather than an instance of the class:

```
MyClass.class_method()
```

This will output:

Class method called
Class variable;42

Self arguments in Python:

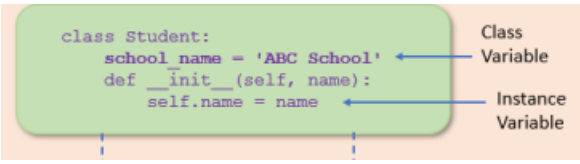
Self represents the instance of the class. By using the "self" we can access the attributes and methods of the class in python. It binds the attributes with the given arguments. The reason you need to use self. is because. Python decided to do methods in a way that makes the instance to which the method belongs be passed automatically, but not received automatically: the first parameter of methods is the instance the method is called on. (self)

```
Class Person
   def..init..(sekf,name,age):
     self.name=name
     self.age=age

def say_hello(self):
  prinmt("hello,my name is,self.name)
```

Class variables & Object variables in Python

Class Variables: A class variable is a variable that is declared inside of class, but outside of any instance method or __init__() method. Class variables are shared by all instances of a class. Unlike instance variable, the value of a class variable is not varied from object to object



Create Class Variables: A class variable is declared inside of class, but outside of any instance method or __init__() method. By convention, typically it is placed right below the class header and before the constructor method and other methods.

Object variables: Object variables are also called as Instance variables owned by each individual object/instance of the class. In this case, each object has its own copy of the field i.e. they are not shared and are not related in any way to the field by the same name in a different instance of the same class.

- If the value of a variable varies from object to object, then such variables are called instance variables. Instance variables in a class: these are called fields or attributes of an object
- When we create classes in Python, instance methods are used regularly. we need to create an object to execute the block of code or action defined in the instance method.
- Instance variables are used within the instance method. We use the instance method to perform a set of actions on the data/value provided by the instance variable.
- We can access the instance variable using the object and dot (.) operator. Example: In the following example, we are creating two instance variable name and age in the Student class

Public & Private Data members in python:

In Python, you can have two types of data members in a class: public and private. Public data members are accessible from outside the class, whereas private data members are only accessible within the class.

PUBLIC DATA MEMBERS are attributes of a class that can be accessed from outside the class. These attributes do not require any special syntax to declare, and they can be accessed using the dot notation. For example:

```
Class Myclass:
  def..init..(self):
```

```
    self.public_attribute=1
obj=myclass()
print(obj.public_attribute) #output=1
```

In this example public_attribute is a public data member of the MyClass class. It is assigned a value of 1 in the constructor, and it can be accessed from outside the class using the

Syntax: obj.public_attribute Public data members are often used to represent properties or characteristics of an object that are relevant to its interaction with other objects in the program. For example, a Person class might have public data members for the person's name, age, and address.

PRIVATE DATA MEMBERS To define a private data member in Python, you need to prefix the attribute name with double underscores (e.g., __attribute). This makes the attribute name "name-mangled" to prevent conflicts with other attributes. For example: In the above example, public_attribute is accessible from outside the class, whereas __private_attribute is not. If you try to access __private_attribute from outside the class, you will get an AttributeError

Private methods in Python:

A private method is an access modifier used in a class that can only be called from inside the class where it is defined. It means that you cannot access or call the methods defined under private class from outside.

Consider a real-life example as a car engine. The car engine is made up of various parts like valves, sparks, pistons, etc. No user can make use of these parts by themselves. The same situation is with a private method which hides the inner functionality of the class from the outside world.

Remember that even the base class cannot access the methods of private class. In python programming, there are no private methods that cannot be accessed except inside the class.

To define the private method, you have to prefix the member name with a double underscore (_ _)

```
class Fruit:

    name = ''
    quantity = 0

    def __init__(self, n, a):

        self.name = n
        self.quantity = a
```