Python Programming Language

Python is a widely used programming language that offers several unique features and advantages compared to languages like **Java** and **C++.** Our Python tutorial thoroughly explains Python basics and advanced concepts, starting with installation, conditional statements, loops, built-in data structures, Object-Oriented Programming, Generators, Exception Handling, Python RegEx, and many other concepts. This tutorial is designed for beginners and working professionals.

In the late 1980s, **Guido van Rossum** dreamed of developing Python. The first version of **Python 0.9.0 was released in 1991**. Since its release, Python started gaining popularity. According to reports, Python is now the most popular programming language among developers because of its high demands in the tech realm.

What is Python

Python is a general-purpose, dynamically typed, high-level, compiled and interpreted, garbage-collected, and purely object-oriented programming language that supports procedural, object-oriented, and functional programming.

Features of Python:

- **Easy to use and Read -** Python's syntax is clear and easy to read, making it an ideal language for both beginners and experienced programmers. This simplicity can lead to faster development and reduce the chances of errors.
- **Dynamically Typed** - The data types of variables are determined during run-time. We do not need to specify the data type of a variable during writing codes.
- **High-level** - High-level language means human readable code.
- **Compiled and Interpreted** - Python code first gets compiled into bytecode, and then interpreted line by line. When we download the Python in our system form org we download the default implement of Python known as CPython. CPython is considered to be Complied and Interpreted both.
- **Garbage Collected** - Memory allocation and de-allocation are automatically managed. Programmers do not specifically need to manage the memory.
- **Purely Object-Oriented** - It refers to everything as an object, including numbers and strings.
- **Cross-platform Compatibility** - Python can be easily installed on Windows, macOS, and various Linux distributions, allowing developers to create software that runs across different operating systems.

- **Rich Standard Library** - Python comes with several standard libraries that provide ready-to-use modules and functions for various tasks, ranging from **web development** and **data manipulation** to **machine learning** and **networking**.
- **Open Source** - Python is an open-source, cost-free programming language. It is utilized in several sectors and disciplines as a result.

Python has many *web-based assets*, *open-source projects*, and *a vibrant community*. Learning the language, working together on projects, and contributing to the Python ecosystem are all made very easy for developers.

Because of its straightforward language framework, Python is easier to understand and write code in. This makes it a fantastic programming language for novices. Additionally, it assists seasoned programmers in writing clear and error-free code.

Python has many third-party libraries that can be used to make its functionality easier. These libraries cover many domains, for example, web development, scientific computing, data analysis, and more.

Java vs. Python

Python is an excellent choice for rapid development and scripting tasks. Whereas Java emphasizes a strong type system and object-oriented programming.

Here are some basic programs that illustrates key differences between them.

Printing 'Hello World'

Python Code:

```
1. print("Hello World!")
```

Test it Now

Output:

Hello, World!

In Python, it is one line of code. It requires simple syntax to print 'Hello World'

Java Code:

```
1. public class HelloWorld {
2.     public static void main(String[] args) {
3.         System.out.println("Hello, World!");
4.     }
5. }
```

Output:

Hello, World!
In Java, we need to declare classes, method structures many other things.

While both programs give the same output, we can notice the syntax difference in the print statement.

- In Python, it is easy to learn and write code. While in Java, it requires more code to perform certain tasks.
- Python is dynamically typed, meaning we do not need to declare the variable Whereas Java is statistically typed, meaning we need to declare the variable type.
- Python is suitable for various domains such as Data Science, Machine Learning, Web development, and more. Whereas Java is suitable for web development, mobile app development (Android), and more.

Python Basic Syntax

There is no use of curly braces or semicolons in Python programming language. It is an English-like language. But Python uses indentation to define a block of code. Indentation is nothing but adding whitespace before the statement when it is needed.

For example -

```
def func():
    statement 1
    statement 2
    ……….
    ……….
     statement N
```

In the above example, the statements that are the same level to the right belong to the function. Generally, we can use four whitespaces to define indentation.

Instead of Semicolon as used in other languages, Python ends its statements with a NewLine character.

Python is a case-sensitive language, which means that uppercase and lowercase letters are treated differently. For example, 'name' and 'Name' are two different variables in Python.

In Python, comments can be added using the '#' symbol. Any text written after the '#' symbol is considered a comment and is ignored by the interpreter. This trick is useful for adding notes to the code or temporarily disabling a code block. It also helps in understanding the code better by some other developers.

'If', 'otherwise', 'for', 'while', 'try', 'except', and 'finally' are a few reserved keywords in Python that cannot be used as variable names. These terms are used in the language for particular reasons and have fixed meanings. If you use these keywords, your code may include errors, or the interpreter may reject them as potential new Variables.

History of Python

Python was created by Guido van Rossum. In the late 1980s, Guido van Rossum, a Dutch programmer, began working on Python while at the Centrum Wiskunde & Informatica (CWI) in the Netherlands. He wanted to create a successor to the **ABC programming language** that would be easy to read and efficient.

In February 1991, the first public version of Python, version 0.9.0, was released. This marked the official birth of **Python as an open-source project**. The language was named after the British comedy series "**Monty Python's Flying Circus**".

Python development has gone through several stages. **In January 1994, Python 1.0 was released as a usable and stable programming language.** This version included many of the features that are still present in Python today.

From the 1990s to the 2000s, Python gained popularity for its simplicity, readability, and versatility. **In October 2000, Python 2.0 was released**. Python 2.0 introduced list comprehensions, garbage collection, and support for Unicode.

In December 2008, Python 3.0 was released. Python 3.0 introduced several backward-incompatible changes to improve code readability and maintainability.

Throughout 2010s, Python's popularity increased, particularly in fields like data science, machine learning, and web development. Its rich ecosystem of libraries and frameworks made it a favourite among developers.

The **Python Software Foundation (PSF) was established in 2001** to promote, protect, and advance the Python programming language and its community.

Why learn Python?

Python provides many useful features to the programmer. These features make it the most popular and widely used language. We have listed below few-essential features of Python.

- **Easy to use and Learn:** Python has a simple and easy-to-understand syntax, unlike traditional languages like C, C++, Java, etc., making it easy for beginners to learn.
- **Expressive Language:** It allows programmers to express complex concepts in just a few lines of code or reduces Developer's Time.

- **Interpreted Language:** Python does not require compilation, allowing rapid development and testing. It uses Interpreter instead of Compiler.
- **Object-Oriented Language**: It supports object-oriented programming, making writing reusable and modular code easy.
- **Open-Source Language:** Python is open-source and free to use, distribute and modify.
- **Extensible:** Python can be extended with modules written in C, C++, or other languages.
- **Learn Standard Library:** Python's standard library contains many modules and functions that can be used for various tasks, such as string manipulation, web programming, and more.
- **GUI Programming Support:** Python provides several GUI frameworks, such as Tkinter and PyQt, allowing developers to create desktop applications easily.
- **Integrated:** Python can easily integrate with other languages and technologies, such as C/C++, Java, and . NET.
- **Embeddable:** Python code can be embedded into other applications as a scripting language.
- **Dynamic Memory Allocation:** Python automatically manages memory allocation, making it easier for developers to write complex programs without worrying about memory management.
- **Wide Range of Libraries and Frameworks:** Python has a vast collection of libraries and frameworks, such as NumPy, Pandas, Django, and Flask, that can be used to solve a wide range of problems.
- **Versatility:** Python is a universal language in various domains such as web development, machine learning, data analysis, scientific computing, and more.
- **Large Community:** Python has a vast and active community of developers contributing to its development and offering support. This makes it easy for beginners to get help and learn from experienced developers.
- **Career Opportunities:** Python is a highly popular language in the job market. Learning Python can open up several career opportunities in data science, artificial intelligence, web development, and more.
- **High Demand:** With the growing demand for automation and digital transformation, the need for Python developers is rising. Many industries seek skilled Python developers to help build their digital infrastructure.
- **Increased Productivity:** Python has a simple syntax and powerful libraries that can help developers write code faster and more efficiently. This can increase productivity and save time for developers and organizations.
- **Big Data and Machine Learning:** Python has become the go-to language for big data and machine learning. Python has become popular among data scientists and machine learning engineers with libraries like NumPy, Pandas, Scikit-learn, TensorFlow, and more.

Where is Python used?

Python is a general-purpose, popular programming language, and it is used in almost every technical field. The various areas of Python use are given below.

- **Data Science:** Data Science is a vast field, and Python is an important language for this field because of its simplicity, ease of use, and availability of powerful data analysis and visualization libraries like NumPy, Pandas, and Matplotlib.
- **Desktop Applications:** PyQt and Tkinter are useful libraries that can be used in GUI - Graphical User Interface-based Desktop Applications. There are better languages for this field, but it can be used with other languages for making Applications.
- **Console-based Applications:** Python is also commonly used to create command-line or console-based applications because of its ease of use and support for advanced features such as input/output redirection and piping.
- **Mobile Applications:** While Python is not commonly used for creating mobile applications, it can still be combined with frameworks like Kivy or BeeWare to create cross-platform mobile applications.
- **Software Development:** Python is considered one of the best software-making languages. Python is easily compatible with both from Small Scale to Large Scale software.
- **Artificial Intelligence**: AI is an emerging Technology, and Python is a perfect language for artificial intelligence and machine learning because of the availability of powerful libraries such as TensorFlow, Keras, and PyTorch.
- **Web Applications:** Python is commonly used in web development on the backend with frameworks like Django and Flask and on the front end with tools like JavaScript HTML and CSS.
- **Enterprise Applications:** Python can be used to develop large-scale enterprise applications with features such as distributed computing, networking, and parallel processing.
- **3D CAD Applications:** Python can be used for 3D computer-aided design (CAD) applications through libraries such as Blender.
- **Machine Learning:** Python is widely used for machine learning due to its simplicity, ease of use, and availability of powerful machine learning libraries.
- **Computer Vision or Image Processing Applications:** Python can be used for computer vision and image processing applications through powerful libraries such as OpenCV and Scikit-image.
- **Speech Recognition:** Python can be used for speech recognition applications through libraries such as SpeechRecognition and PyAudio.
- **Scientific computing:** Libraries like NumPy, SciPy, and Pandas provide advanced numerical computing capabilities for tasks like data analysis, machine learning, and more.

- **Education:** Python's easy-to-learn syntax and availability of many resources make it an ideal language for teaching programming to beginners.
- **Testing:** Python is used for writing automated tests, providing frameworks like unit tests and pytest that help write test cases and generate reports.
- **Gaming:** Python has libraries like Pygame, which provide a platform for developing games using Python.
- **IoT:** Python is used in IoT for developing scripts and applications for devices like Raspberry Pi, Arduino, and others.
- **Networking:** Python is used in networking for developing scripts and applications for network automation, monitoring, and management.
- **DevOps**: Python is widely used in DevOps for automation and scripting of infrastructure management, configuration management, and deployment processes.
- **Finance:** Python has libraries like Pandas, Scikit-learn, and Statsmodels for financial modeling and analysis.
- **Audio and Music:** Python has libraries like Pyaudio, which is used for audio processing, synthesis, and analysis, and Music21, which is used for music analysis and generation.
- **Writing scripts:** Python is used for writing utility scripts to automate tasks like file operations, web scraping, and data processing.

Python Popular Frameworks and Libraries

Python has wide range of libraries and frameworks widely used in various fields such as machine learning, artificial intelligence, web applications, etc. We define some popular frameworks and libraries of Python as follows.

- **Web development (Server-side)** - **Django Flask**, **Pyramid**, **CherryPy**
- **GUIs based applications** - **Tkinter**, **PyGTK**, **PyQt**, **PyJs**, etc.
- **Machine Learning** - **TensorFlow**, **PyTorch**, **Scikit-learn**, **Matplotlib**, **Scipy**, etc.
- **Mathematics** - **NumPy**, **Pandas**, etc.
- **BeautifulSoup:** a library for web scraping and parsing HTML and XML
- **Requests**: a library for making HTTP requests
- **SQLAlchemy**: a library for working with SQL databases
- **Kivy:** a framework for building multi-touch applications
- **Pygame:** a library for game development
- **Pytest:** a testing framework for Python Django
- **REST framework**: a toolkit for building RESTful APIs
- **FastAPI**: a modern, fast web framework for building APIs
- **Streamlit**: a library for building interactive web apps for machine learning and data science
- **NLTK**: a library for natural language processing

Python print() Function

Python print() function is used to display output to the console or terminal. It allows us to display text, variables and other data in a human readable format.

Syntax:

print(object(s), sep=separator, end=end, file=file, flush=flush)

It takes one or more arguments separated by comma(,) and adds a 'newline' at the end by default.

Parameters:

- *object(s) - As many as you want data to display, will first converted into string and printed to the console.*
- *sep - Separates the objects by a separator passed, default value = " ".*
- *end - Ends a line with a newline character*
- *file - a file object with write method, default value = sys.stdout*

Example:

```
# Displaying a string
print("Hello, World!")

# Displaying multiple values
name = "Aman"
age = 21
print("Name:", name, "Age:", age)

# Printing variables and literals
x = 5
y = 7
print("x =", x, "y =", y, "Sum =", x + y)

# Printing with formatting
percentage = 85.75
print("Score: {:.2f}%".format(percentage))
```

Output:

Hello, World!
Name: Aman Age: 21
X = 5 y = 7 Sum = 12

Score: 85.75%
In this example, the print statement is used to print string, integer, and float values in a human readable format.

The print statement can be used for debugging, logging and to provide information to the user.

Python Conditional Statements

Conditional statements help us to execute a particular block for a particular condition. In this tutorial, we will learn how to use conditional expression to execute a different block of statements. Python provides if and else keywords to set up logical conditions. The **elif** keyword is also used as a conditional statement.

Example code for if..else statement

```
x = 10
y = 5

if x > y:
    print("x is greater than y")
else:
    print("y is greater than or equal to x")
```

Output:

x is greater than y
In the above code, we have two variables, x, and y, with 10 and 5, respectively. Then we used an if..else statement to check if x is greater than y or vice versa. If the first condition is true, the statement "x is greater than y" is printed. If the first condition is false, the statement "y is greater than or equal to x" is printed instead.

The if keyword checks the condition is true and executes the code block inside it. The code inside the else block is executed if the condition is false. This way, the if..else statement helps us to execute different blocks of code based on a condition.

We will learn about this in more detail in the further article for the Python tutorial.

Python Loops

Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several times. For this purpose, the programming languages provide various loops capable of repeating some specific code several times. Consider the following tutorial to understand the statements in detail.

Python For Loop

```
1. fruits = ["apple", "banana", "cherry"]
2. for x in fruits:
3.     print(x, end=" ")
```

Test it Now

Output:

apple banana cherry

Python While Loop

```
1. i = 1
2. while i<5:
3.     print(i, end=" ")
4.     i += 1
```

Output:

1 2 3 4

In the above example code, we have demonstrated using two types of loops in Python - For loop and While loop.

The For loop is used to iterate over a sequence of items, such as a list, tuple, or string. In the example, we defined a list of fruits and used a for loop to print each fruit, but it can also be used to print a range of numbers.

The While loop repeats a code block if the specified condition is true. In the example, we have initialized a variable i to 1 and used a while loop to print the value of i until it becomes greater than or equal to 6. The i += 1 statement is used to increment the value of i in each iteration.

We will learn about them in the tutorial in detail.

Python Data Structures

Python offers four built-in data structures: lists, tuples, sets, and dictionaries that allow us to store data in an efficient way. Below are the commonly used data structures in Python, along with example code:

1. Lists

- Lists are **ordered collections** of data elements of different data types.
- Lists are **mutable** meaning a list can be modified anytime.

- Elements can be **accessed using indices**.
- They are defined using square bracket '**[]**'.

Example:

```
# Create a list
fruits = ['apple', 'banana', 'cherry']
print("fuirts[1] =", fruits[1])

# Modify list
fruits.append('orange')
print("fruits =", fruits)

num_list = [1, 2, 3, 4, 5]
# Calculate sum
sum_nums = sum(num_list)
print("sum_nums =", sum_nums)
```

Output:

fuirts[1] = banana
fruits = ['apple', 'banana', 'cherry', 'orange']
sum_nums = 15

2. Tuples

- Tuples are also **ordered collections** of data elements of different data types, similar to Lists.
- Elements can be **accessed using indices**.
- Tuples are **immutable** meaning Tuples can't be modified once created.
- They are defined using open bracket '**()**'.

Example:

```
# Create a tuple
point = (3, 4)
x, y = point
print("(x, y) =", x, y)

# Create another tuple
tuple_ = ('apple', 'banana', 'cherry', 'orange')
print("Tuple =", tuple_)
```

Output:

(x, y) = 3 4
Tuple = ('apple', 'banana', 'cherry', 'orange')

3. Sets

- Sets are **unordered** collections of immutable data elements of different data types.
- Sets are **mutable**.
- Elements can't be accessed using indices.
- Sets **do not contain duplicate elements**.
- They are defined using curly braces '**{}**'

Example:

```
# Create a set
set1 = {1, 2, 2, 1, 3, 4}
print("set1 =", set1)

# Create another set
set2 = {'apple', 'banana', 'cherry', 'apple', 'orange'}
print("set2 =", set2)
```

Output:

set1 = {1, 2, 3, 4}
set2 = {'apple', 'cherry', 'orange', 'banana'}

4. Dictionaries

- Dictionary are **key-value pairs** that allow you to associate values with unique keys.
- They are defined using curly braces '**{}**' with key-value pairs **separated by colons ':'**.
- Dictionaries are **mutable**.
- Elements can be accessed using keys.

Example:

```
# Create a dictionary
person = {'name': 'Umesh', 'age': 25, 'city': 'Noida'}
print("person =", person)
print(person['name'])

# Modify Dictionary
person['age'] = 27
print("person =", person)
```

Output:

person = {'name': 'Umesh', 'age': 25, 'city': 'Noida'}
Umesh
person = {'name': 'Umesh', 'age': 27, 'city': 'Noida'}
These are just a few examples of Python's built-in data structures. Each data structure has its own characteristics and use cases.

Python Functional Programming

This section of the Python tutorial defines some important tools related to functional programming, such as lambda and recursive functions. These functions are very efficient in accomplishing complex tasks. We define a few important functions, such as reduce, map, and filter. Python provides the functools module that includes various functional programming tools. Visit the following tutorial to learn more about functional programming.

Recent versions of Python have introduced features that make functional programming more concise and expressive. For example, the "walrus operator":= allows for inline variable assignment in expressions, which can be useful when working with nested function calls or list comprehensions.

Python Function

1. **Lambda Function** - A lambda function is a small, **anonymous function** that can take any number of arguments but can only have one expression. Lambda functions are often used in functional programming to create functions "on the fly" without defining a named function.
2. **Recursive Function** - A recursive function is a function that calls itself to solve a problem. Recursive functions are often used in functional programming to perform complex computations or to traverse complex data structures.
3. **Map Function** - The map() function applies a given function to each item of an iterable and returns a new iterable with the results. The input iterable can be a list, tuple, or other.
4. **Filter Function** - The filter() function returns an iterator from an iterable for which the function passed as the first argument returns True. It filters out the items from an iterable that do not meet the given condition.
5. **Reduce Function** - The reduce() function applies a function of two arguments cumulatively to the items of an iterable from left to right to reduce it to a single value.
6. **functools Module** - The functools module in Python provides higher-order functions that operate on other functions, such as partial() and reduce().
7. **Currying Function** - A currying function is a function that takes multiple arguments and returns a sequence of functions that each take a single argument.
8. **Memoization Function** - Memoization is a technique used in functional programming to cache the results of expensive function calls and return the cached Result when the same inputs occur again.
9. **Threading Function** - Threading is a technique used in functional programming to run multiple tasks simultaneously to make the code more efficient and faster.

Python Modules

Python modules are the program files that contain Python code or functions. Python has two types of modules - User-defined modules and built-in modules. A module the user defines, or our Python code saved with .py extension, is treated as a user-define module.

Built-in modules are predefined modules of Python. To use the functionality of the modules, we need to import them into our current working program.

Python modules are essential to the language's ecosystem since they offer reusable code and functionality that can be imported into any Python program. Here are a few examples of several Python modules, along with a brief description of each:

Math: Gives users access to mathematical constants and pi and trigonometric functions.

Datetime: Provides classes for a simpler way of manipulating dates, times, and periods.

OS: Enables interaction with the base operating system, including administration of processes and file system activities.

Random: The random function offers tools for generating random integers and picking random items from a list.

JSON: JSON is a data structure that can be encoded and decoded and is frequently used in online APIs and data exchange. This module allows dealing with JSON.
Re: Supports regular expressions, a potent text-search and text-manipulation tool.

Collections: Provides alternative data structures such as sorted dictionaries, default dictionaries, and named tuples.

NumPy: NumPy is a core toolkit for scientific computing that supports numerical operations on arrays and matrices.

Pandas: It provides high-level data structures and operations for dealing with time series and other structured data types.

Requests: Offers a simple user interface for web APIs and performs HTTP requests.

Python File I/O

Files are used to store data in a computer disk. In this tutorial, we explain the built-in file object of Python. We can open a file using Python script and perform various operations such as writing, reading, and appending. There are various ways of opening a file. We are explained with the relevant example. We will also learn to perform read/write operations on binary files.

Python's file input/output (I/O) system offers programs to communicate with files stored on a disc. Python's built-in methods for the file object let us carry out actions like reading, writing, and adding data to files.

The **open()** method in Python makes a file object when working with files. The name of the file to be opened and the mode in which the file is to be opened are the two parameters required by this function. The mode can be used according to work that needs to be done with the file, such as "**r**" for reading, "**w**" for writing, or "**a**" for attaching.

After successfully creating an object, different methods can be used according to our work. If we want to write in the file, we can use the write() functions, and if you want to read and write both, then we can use the append() function and, in cases where we only want to read the content of the file we can use read() function. Binary files containing data in a binary rather than a text format may also be worked with using Python. Binary files are written in a manner that humans cannot directly understand. The **rb** and **wb** modes can read and write binary data in binary files.

Python Exceptions

An exception can be defined as an unusual condition in a program resulting in an interruption in the flow of the program.

Whenever an exception occurs, the program stops the execution, and thus the other code is not executed. Therefore, an exception is the run-time errors that are unable to handle to Python script. An exception is a Python object that represents an error.

Python Exceptions are an important aspect of error handling in Python programming. When a program encounters an unexpected situation or error, it may raise an exception, which can interrupt the normal flow of the program.

In Python, exceptions are represented as objects containing information about the error, including its type and message. The most common type of Exception in Python is the Exception class, a base class for all other built-in exceptions.

To handle exceptions in Python, we use the **try** and **except** statements. The **try** statement is used to enclose the code that may raise an exception, while the **except** statement is used to define a block of code that should be executed when an exception occurs.

For example, consider the following code:

```
1. try:
2.     x = int ( input ("Enter a number: "))
3.     y = 10 / x
4.     print ("Result:", y)
```

```
except ZeroDivisionError:
    print ("Error: Division by zero")
except ValueError:
    print ("Error: Invalid input")
```

Output:

Enter a number: 0
Error: Division by zero
In this code, we use the try statement to attempt to perform a division operation. If either of these operations raises an exception, the matching except block is executed.

Python also provides many built-in exceptions that can be raised in similar situations. Some common built-in exceptions include **IndexError, TypeError**, and **NameError**. Also, we can define our custom exceptions by creating a new class that inherits from the Exception class.

Python CSV

A CSV stands for "comma separated values", which is defined as a simple file format that uses specific structuring to arrange tabular data. It stores tabular data such as spreadsheets or databases in plain text and has a common format for data interchange. A CSV file opens into the Excel sheet, and the rows and columns data define the standard format.

We can use the CSV.reader function to read a CSV file. This function returns a reader object that we can use to repeat over the rows in the CSV file. Each row is returned as a list of values, where each value corresponds to a column in the CSV file.

For example, consider the following code:

```
import csv

with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Here, we open the file data.csv in read mode and create a **csv.reader** object using the **csv.reader()** function. We then iterate over the rows in the CSV file using a for loop and print each row to the console.

We can use the **CSV.writer()** function to write data to a CSV file. It returns a writer object we can use to write rows to the CSV file. We can write rows by calling the **writer ()** method on the writer object.

For example, consider the following code:

```
import csv

data = [    ['Name', 'Age', 'Country'],
    ['Alice', '25', 'USA'],
    ['Bob', '30', 'Canada'],
    ['Charlie', '35', 'Australia']
]

with open('data.csv', 'w') as file:
    writer = csv.writer(file)
    for row in data:
        writer.writerow(row)
```

In this program, we create a list of lists called data, where each inner list represents a row of data. We then open the file data.csv in write mode and create a **CSV.writer** object using the CSV.writer function. We then iterate over the rows in data using a for loop and write each row to the CSV file using the writer method.

Python Sending Mail

We can send or read a mail using the Python script. Python's standard library modules are useful for handling various protocols such as PoP3 and IMAP. Python provides the smtplib module for sending emails using SMTP (Simple Mail Transfer Protocol). We will learn how to send mail with the popular email service SMTP from a Python script.

Python Magic Methods

The Python magic method is the special method that adds "magic" to a class. It starts and ends with double underscores, for example, **_init_** or **_str_**.

The built-in classes define many magic methods. The **dir()** function can be used to see the number of magic methods inherited by a class. It has two prefixes and suffix underscores in the method name.

- Python magic methods are also known as **dunder methods**, short for "double underscore" methods because their names start and end with a double underscore.
- **Magic methods** are automatically invoked by the Python interpreter in certain situations, such as when an object is created, compared to another object, or printed.
- Magic methods can be used to customize the behavior of classes, such as defining how objects are compared, converted to strings, or accessed as containers.

- Some commonly used magic methods include **init** for initializing an object, str for converting an object to a string, **eq** for comparing two objects for equality, and **getitem** and **setitem** for accessing items in a container object.

For example, the **str** magic method can define how an object should be represented as a string. Here's an example

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name} ({self.age})"

person = Person('Vikas', 22)
print(person)
```

Output:

Vikas (22)

In this example, the str method is defined to return a formatted string representation of the Person object with the person's name and age.

Another commonly used magic method is **eq**, which defines how objects should be compared for equality. Here's an example:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

point1 = Point(2, 3)
point2 = Point(3, 4)
point3 = Point(2, 3)

print(point1 == point2)
print(point1 == point3)
```

Output:

False

True

In this example, the **eq** method is defined to return True if two Point objects have the same x and y coordinates and False otherwise.

Python Oops Concepts

Everything in Python is treated as an object, including integer values, floats, functions, classes, and none. Apart from that, Python supports all oriented concepts. Below is a brief introduction to the Oops concepts of Python.

- **Classes and Objects** - Python classes are the blueprints of the Object. An object is a collection of data and methods that act on the data.
- **Inheritance** - An inheritance is a technique where one class inherits the properties of other classes.
- **Constructor** - Python provides a special method __init__() which is known as a constructor. This method is automatically called when an object is instantiated.
- **Data Member** - A variable that holds data associated with a class and its objects.
- **Polymorphism** - Polymorphism is a concept where an object can take many forms. In Python, polymorphism can be achieved through method overloading and method overriding.
- **Method Overloading** - In Python, method overloading is achieved through default arguments, where a method can be defined with multiple parameters. The default values are used if some parameters are not passed while calling the method.
- **Method Overriding** - Method overriding is a concept where a subclass implements a method already defined in its superclass.
- **Encapsulation** - Encapsulation is wrapping data and methods into a single unit. In Python, encapsulation is achieved through access modifiers, such as public, private, and protected. However, Python does not strictly enforce access modifiers, and the naming convention indicates the access level.
- **Data Abstraction**: A technique to hide the complexity of data and show only essential features to the user. It provides an interface to interact with the data. Data abstraction reduces complexity and makes code more modular, allowing developers to focus on the program's essential features.

To read the Oops concept in detail, visit the following resources.

- Python Oops Concepts - In Python, the object-oriented paradigm is to design the program using classes and objects. The object is related to real-word entities such as book, house, pencil, etc. and the class defines its properties and behaviours.
- Python Objects and classes - In Python, objects are instances of classes and classes are blueprints that defines structure and behaviour of data.

- Python Constructor - A constructor is a special method in a class that is used to initialize the object's attributes when the object is created.
- Python Inheritance - Inheritance is a mechanism in which new class (subclass or child class) inherits the properties and behaviours of an existing class (super class or parent class).
- Python Polymorphism - Polymorphism allows objects of different classes to be treated as objects of a common superclass, enabling different classes to be used interchangeably through a common interface.

Python Advance Topics

Python includes many advances and useful concepts that help the programmer solve complex tasks. These concepts are given below.

Python Iterator

An iterator is simply an object that can be iterated upon. It returns one Object at a time. It can be implemented using the two special methods, **__iter__()** and __next__().

Iterators in Python are objects that allow iteration over a collection of data. They process each collection element individually without loading the entire collection into memory.

For example, let's create an iterator that returns the squares of numbers up to a given limit:

```
1.  def __init__(self, limit):
2.      self.limit = limit
3.      self.n = 0
4.
5.  def __iter__(self):
6.      return self
7.
8.  def __next__(self):
9.      if self.n <= self.limit:
10.         square = self.n ** 2
11.         self.n += 1
12.         return square
13.     else:
14.         raise StopIteration
15.
16. numbers = Squares(5)
17. for n in numbers:
18.     print(n)
```

Output:

0
1
4
9
16
25

In this example, we have created a class Squares that acts as an iterator by implementing the __iter__() and __next__() methods. The __iter__() method returns the Object itself, and the __next__() method returns the next square of the number until the limit is reached.

To learn more about the iterators, visit our Python Iterators tutorial.

Python Generators

Python generators produce a sequence of values **using a yield statement** rather than a return since they are functions that return iterators. Generators terminate the function's execution while keeping the local state. It picks up right where it left off when it is restarted. Because we don't have to implement the iterator protocol thanks to this feature, writing iterators is made simpler. Here is an illustration of a straightforward generator function that produces squares of numbers:

```
# Generator Function
def square_numbers(n):
    for i in range(n):
        yield i**2

# Create a generator object
generator = square_numbers(5)

# Print the values generated by the generator
for num in generator:
    print(num)
```

Output:

0
1
4
9
16

Python Modifiers

Python Decorators are functions used to modify the behaviour of another function. They allow adding functionality to an existing function without modifying its code directly. Decorators are defined using the **@** symbol followed by the name of the decorator function. They can be used for logging, timing, caching, etc.

Here's an example of a decorator function that adds timing functionality to another function:

```
import time
from math import factorial

# Decorator to calculate time taken by
# the function
def time_it(func):
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"{func.__name__} took {end-start:.5f} seconds to run.")
        return result
    return wrapper

@time_it
def my_function(n):
    time.sleep(2)
    print(f"Factorial of {n} = {factorial(n)}")

my_function(25)
```

Output:

In the above example, the time_it decorator function takes another function as an argument and returns a wrapper function. The wrapper function calculates the time to execute the original function and prints it to the console. The @time_it decorator is used to apply the time_it function to the my_function function. When my_function is called, the decorator is executed, and the timing functionality is added.

Python MySQL

Python MySQL is a powerful relational database management system. We must set up the environment and establish a connection to use MySQL with Python. We can create a new database and tables using SQL commands in Python.

- **Environment Setup**: Installing and configuring MySQL Connector/Python to use Python with MySQL.
- **Database Connection**: Establishing a connection between Python and MySQL database using MySQL Connector/Python.
- **Creating New Database**: Creating a new database in MySQL using Python.
- **Creating Tables**: Creating tables in the MySQL database with Python using SQL commands.
- **Insert Operation**: Insert data into MySQL tables using Python and SQL commands.
- **Read Operation**: Reading data from MySQL tables using Python and SQL commands.
- **Update Operation**: Updating data in MySQL tables using Python and SQL commands.
- **Join Operation**: Joining two or more tables in MySQL using Python and SQL commands.
- **Performing Transactions**: Performing a group of SQL queries as a single unit of work in MySQL using Python.

Other relative points include handling errors, creating indexes, and using stored procedures and functions in MySQL with Python.

Python MongoDB

Python MongoDB is a popular NoSQL database that stores data in JSON-like documents. It is schemaless and provides high scalability and flexibility for data storage. We can use MongoDB with Python using the PyMongo library, which provides a simple and intuitive interface for interacting with MongoDB.

Here are some common tasks when working with MongoDB in Python:

1. **Environment Setup**: Install and configure MongoDB and PyMongo library on your system.
2. **Database Connection**: Connect to a MongoDB server using the MongoClient class from PyMongo.
3. **Creating a new database**: Use the MongoClient Object to create a new database.
4. **Creating collections**: Create collections within a database to store documents.
5. **Inserting documents**: Insert new documents into a collection using the insert_one() or insert_many() methods.
6. **Querying documents**: Retrieve documents from a collection using various query methods like find_one(), find(), etc.
7. **Updating documents**: Modify existing documents in a collection using update_one() or update_many() methods.
8. **Deleting documents**: Remove documents from a collection using the delete_one() or delete_many() methods.
9. **Aggregation**: Perform aggregation operations like grouping, counting, etc., using the aggregation pipeline framework.
10. **Indexing:** Improve query performance by creating indexes on fields in collections.

There are many more advanced topics in MongoDB, such as data sharding, replication, and more, but these tasks cover the basics of working with MongoDB in Python.

Python SQLite

Relational databases are built and maintained using Python SQLite, a compact, serverless, self-contained database engine. Its mobility and simplicity make it a popular option for local or small-scale applications. Python has a built-in module for connecting to SQLite databases called SQLite3, enabling developers to work with SQLite databases without difficulties.

Various API methods are available through the SQLite3 library that may be used to run SQL queries, insert, select, update, and remove data, as well as get data from tables. Additionally, it allows transactions, allowing programmers to undo changes in case of a problem. Python SQLite is a fantastic option for creating programs that need an embedded database system, including desktop, mobile, and modest-sized web programs. SQLite has become popular among developers for lightweight apps with database functionality thanks to its ease of use, portability, and smooth connection with Python.

Python CGI

Python CGI is a technology for running scripts through web servers to produce dynamic online content. It offers a communication channel and a dynamic content generation interface for external CGI scripts and the web server. Python CGI scripts may create HTML web pages, handle form input, and communicate with databases. Python CGI enables the server to carry out Python scripts and provide the results to the client, offering a quick and effective approach to creating dynamic online applications.

Python CGI scripts may be used for many things, including creating dynamic web pages, processing forms, and interacting with databases. Since Python, a potent and popular programming language, can be utilized to create scripts, it enables a more customized and flexible approach to web creation. Scalable, safe, and maintainable online applications may be created with Python CGI. Python CGI is a handy tool for web developers building dynamic and interactive online applications.

Asynchronous Programming in Python

Asynchronous programming is a paradigm for computer programming that enables independent and concurrent operation of activities. It is frequently used in applications like web servers, database software, and network programming, where several tasks or requests must be handled concurrently.

Python has asyncio, Twisted, and Tornado among its libraries and frameworks for asynchronous programming. Asyncio, one of these, offers a simple interface for asynchronous programming and is the official asynchronous programming library in Python.

Coroutines are functions that may be halted and restarted at specific locations in the code and are utilized by asyncio. This enables numerous coroutines to operate simultaneously without interfering with one another. For constructing and maintaining coroutines, the library offers several classes and methods,
including **asyncio.gather(), asyncio.wait(),** and **asyncio.create_task().**

Event loops, which are in charge of planning and operating coroutines, are another feature of asyncio. By cycling between coroutines in a non-blocking way, the event loop controls the execution of coroutines and ensures that no coroutine blocks another. Additionally, it supports timers and scheduling callbacks, which may be helpful when activities must be completed at specified times or intervals.

Python Concurrency

The term "**concurrency**" describes a program's capacity to carry out several tasks at once, enhancing the program's efficiency. Python offers several modules and concurrency-related methods, including asynchronous programming, multiprocessing, and multithreading. While multiprocessing involves running many processes simultaneously on a system, multithreading involves running numerous threads concurrently inside a single process.

The **threading module** in Python enables programmers to build multithreading. It offers classes and operations for establishing and controlling threads. Conversely, the multiprocessing module allows developers to design and control processes. Python's asyncio module provides asynchronous programming support, allowing developers to write non-blocking code that can handle multiple tasks concurrently. Using these techniques, developers can write highperformance, scalable programs that can handle multiple tasks concurrently.

Python's threading module enables the concurrent execution of several threads within a single process, which is helpful for I/O-bound activities.

For CPU-intensive operations like image processing or data analysis, multiprocessing modules make it possible to execute numerous processes concurrently across multiple CPU cores.

The asyncio module supports asynchronous I/O and permits the creation of single-threaded concurrent code using coroutines for high-concurrency network applications.

With libraries like Dask, PySpark, and MPI, Python may also be used for parallel computing. These libraries allow workloads to be distributed across numerous nodes or clusters for better performance.

Web Scrapping using Python

The process of web scraping is used to retrieve data from websites automatically. Various tools and libraries extract data from HTML and other online formats. Python is among the most widely used programming languages for web scraping because of its ease of use, adaptability, and variety of libraries.

We must take a few steps to accomplish web scraping using Python. We must first decide which website to scrape and what information to gather. Then, we can submit a request to the website and receive the HTML content using Python's requests package. Once we have the HTML text, we can extract the needed data using a variety of parsing packages, like **Beautiful Soup and lxml**.

We can employ several strategies, like slowing requests, employing user agents, and using proxies, to prevent overburdening the website's server. It is also crucial to abide by the terms of service for the website and respect its robots.txt file.

Data mining, lead creation, pricing tracking, and many more uses are possible for web scraping. However, as unauthorized web scraping may be against the law and unethical, it is essential to utilize it professionally and ethically.

Natural Language Processing (NLP) using Python

A branch of artificial intelligence (AI) called "natural language processing" (NLP) studies how computers and human language interact. Thanks to NLP, computers can now understand, interpret, and produce human language. Due to its simplicity, versatility, and strong libraries like NLTK (Natural Language Toolkit) and spaCy, Python is a well-known programming language for NLP.

For NLP tasks, including tokenization, stemming, lemmatization, part-of-speech tagging, named entity identification, sentiment analysis, and others, NLTK provides a complete library. It has a variety of corpora (big, organized text collections) for developing and evaluating NLP models. Another well-liked library for NLP tasks is spaCy, which offers quick and effective processing of enormous amounts of text. It enables simple modification and expansion and comes with pre-trained models for various NLP workloads.

NLP may be used in Python for various practical purposes, including chatbots, sentiment analysis, text categorization, machine translation, and more. NLP is used, for instance, by chatbots to comprehend and reply to user inquiries in a natural language style. Sentiment analysis, which may be helpful for brand monitoring, customer feedback analysis, and other purposes, employs NLP to categorize text sentiment (positive, negative, or neutral). Text documents are categorized using natural language processing (NLP) into pre-established categories for spam detection, news categorization, and other purposes.

Python is a strong and useful tool when analyzing and processing human language. Developers may carry out various NLP activities and create useful apps that can communicate with consumers in natural language with libraries like NLTK and spaCy.