TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Python

JAN - 2025

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Index

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 1

Python - Fundamentals of python language

Introduction of Python

Python is a high-level, interpreted, and general-purpose programming language, known for its simplicity, readability, and versatility. It was created by **Guido van Rossum** and first released in **1991**. Python has since become one of the most popular programming languages, used in various fields, such as web development, data science, artificial intelligence, machine learning, automation, and more.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Features of Python

Easy to Learn and Use: Python's syntax is clear and closely resembles the English language, making it beginner-friendly.

Interpreted Language: Python code is executed line by line, which simplifies debugging and allows for quick testing of code snippets.

Dynamically Typed: Variables in Python do not require explicit type declarations. The type of a variable is determined at runtime.

Object-Oriented: Python supports object-oriented programming (OOP), enabling developers to create reusable code through classes and objects.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Features of Python

Extensive Standard Library: Python has a rich set of built-in libraries that support tasks such as file I/O, regular expressions, and networking, among others.

Cross-platform: Python is platform-independent, meaning it can run on various operating systems, such as Windows, macOS, and Linux.

Open Source: Python is free to use, distribute, and modify, which has led to a large and active community of developers contributing to its growth.

Extensibility: Python can integrate with other programming languages, such as C, C++, and Java, to optimize performance-critical sections of code.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Advantages of Python

1. Easy to Learn and Use

- **Readable Syntax**: Python’s syntax is simple and similar to English, which makes it easy to read and understand. This is especially beneficial for beginners.
- **Minimalist Syntax**: Python requires fewer lines of code compared to many other languages, making it easier to write and maintain.

2. Cross-Platform Compatibility

- Python is **platform-independent**, which means code written in Python can run on any operating system (Windows, Linux, macOS) without modification.

3. Extensive Libraries and Frameworks

- Python offers a wide range of **pre-built libraries** and frameworks for various tasks, including:
    - **Web development**: Django, Flask
    - **Data Science & Machine Learning**: Pandas, NumPy, TensorFlow, Scikit-learn
    - **GUI development**: Tkinter, PyQt
    - **Automation**: Selenium, OpenPyXL

4. Large Community Support

- Python has a **vast and active community** that provides support through forums, tutorials, and open-source contributions. This makes it easier to find solutions to problems and learn from others.

5. Integration Capabilities

- Python integrates easily with other programming languages like C, C++, and Java.
- It also supports integration with technologies such as **REST APIs** and databases.

6. Versatile Applications

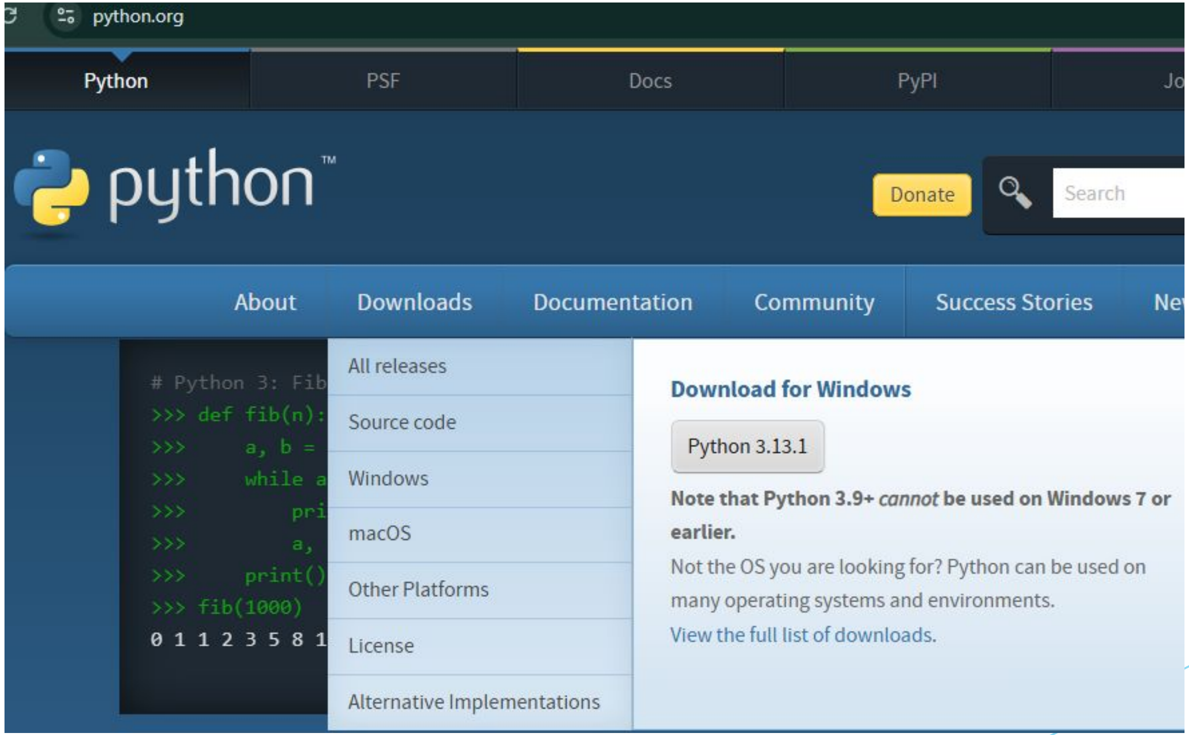
- Python can be used for a wide range of applications, including:
  - **Web development**
  - **Data analysis and visualization**
  - **Machine learning and artificial intelligence**
  - **Scripting and automation**
  - **Game development**

7. Great for Prototyping and Development Speed

- Python is known for its **rapid prototyping** capabilities, which allows developers to create and test applications faster.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Download and Install python software



```
# Python 3: Fib
>>> def fib(n):
>>>     a, b =
>>>     while a
>>>         pri
>>>         a,
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 1
```

Download for Windows

Note that Python 3.9+ *cannot* be used on Windows 7 or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Print() function

Print is an inbuilt function which is used to display appropriate message on console screen (output screen).

Syntax :

print(object , sep = " " , end = " ", ..)

Here, sep and end are optional arguments

Click here for practicals

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Escape Sequences

Escape Sequence	use
V	Single quote
\"	Double quote
\\	backslash
\n	New line
\t	tab
\b	backspace

Click here for practicals

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Programming Style

PEP 8 – Python Style Guide

Python's official style guide, **PEP 8** (Python Enhancement Proposal 8), defines conventions for writing clean and consistent Python code. It covers several aspects such as naming conventions, indentation, line length, and more.

- **Indentation**: Use **4 spaces** per indentation level (avoid using tabs).
- **Maximum Line Length**: Limit all lines to a maximum of **79 characters** (72 for docstrings).
- **Naming Conventions**

  Python has specific conventions for naming variables, functions, classes, and modules. These conventions help make code more readable and consistent.

- **Classes:** Use CamelCase (first letter capitalized, no underscores)

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Data types

- Datatype which is represent different types of values.

- **Numeric Data type :**
  - **int** (Integer): Represents whole numbers, positive or negative, without decimals.
  - **float** (Floating-point): Represents real numbers or numbers with a fractional part.
  - **complex**: Represents complex numbers, which have a real and imaginary part.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Data types

- **Sequence Data type :**
  - **str** (String): Represents a sequence of characters enclosed in single, double, or triple quotes
  - **list**: Represents an ordered, mutable collection of items. Lists can hold different types of data (integers, strings, other lists, etc.)..
  - **tuple**: Represents an ordered, immutable collection of items. Once created, its elements cannot be changed.
  - **range**: Represents an immutable sequence of numbers, often used in loops.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Data types

- **Map Data type :**
  - **dict** (Dictionary): Represents a collection of key-value pairs. Each key is unique, and the associated value can be any data type.
- **Set Data type :**
  - **set**: Represents an unordered collection of unique elements. Sets do not allow duplicate values.
- **Boolean Data type :**
  - **bool**: Represents boolean values, either **True** or **False**. Often used in conditional statements
  - Practicals

Difference between Python2 and Python3

One difference between Python 2 and Python 3 is the print statement. In Python 2, the "print" statement is not a function, and therefore can be invoked without a parenthesis. However, in Python 3, it is a function, and must be invoked with parentheses.

Comments

A comment is a programmer- readable explanation or

annotation in the source code of a computer program. They are added with the purpose of

making the source code easier for humans to understand, and are generally ignored by

compilers and interpreters.

Types of comments

Single line :

```
# this is comment
```

Multiline :

```
“”” this
      Is comment
“””
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Variable : Variable is a name which is contain some value.

Variable Rules:

Variable Names Must Begin with a Letter or Underscore:

The first character of a variable name must be a letter (a-z, A-Z) or an underscore (_).

For example, age, name, _value, and score1 are valid variable names.

Variable Names Can Contain Letters, Numbers, and Underscores:

After the first character, the variable name can include letters (a-z, A-Z), digits (0-9), and underscores (_).

For example, age_1, student_score, and value_2 are valid variable names.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Variable Names Are Case-Sensitive:

- Python treats uppercase and lowercase letters as different. For example, age and Age are two different variables.
- age, Age, and AGE are considered distinct.

Variable Names Cannot Be Reserved Keywords:

- Python has a set of reserved keywords (like class, for, if, True, False, def, etc.) that cannot be used as variable names.
- For example, if, for, and while are not allowed as variable names.

Variable Names Should Be Meaningful:

- While not a strict rule, it's good practice to choose meaningful names that convey the purpose of the variable. For example, age for a person's age, total_amount for a total cost, etc.

Click here for example

Memory Management

Python uses an automatic memory management system that involves dynamic typing, reference counting, and garbage collection. These components work together to efficiently allocate and deallocate memory during the program's execution.

Heap Memory: Python objects (like integers, strings, lists, etc.) are stored in the heap (dynamic memory).

Stack Memory: Local variables (such as function arguments) and function calls are managed in the stack.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

type()

- Python have a built-in method called as type which generally come in
handy while figuring out the type of variable used in the program in the runtime.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Type - Casting

Convert one data type value into another data type.

In python Casting done with function such as int() or float() or str() .

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

input() :

This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list.

If the input provided is not correct then either syntax error or exception is raised by python.

String input example

Int input example

split() :

Using of split() function at some point, need to break a large string down into smaller chunks, or strings.

Click here for example

format() :

format() function in Python is used for string formatting. It allows you to embed variables or expressions inside a string and control how they are displayed. It provides a more powerful and flexible alternative to traditional string concatenation.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Operators in Python

To perform specific operations we need to use some symbols ..
that symbols are operator

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Arithmetic Operators

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Assignment Operators

Operator	Example
=	a=10
+=	a+=10
*=	a*=10
/=	a/=10
%=	a%=10
**=	a**=10
//=	a//=10

Logical Operators

Operator	name
and	And operator
or	Or operator
not	Not operator

Comparison Operators

Operator	Name
==	Equal
! =	Not equal
>	Greater than
<	Less than
> =	Greater than or equal to
< =	Less than or equal to

Identity Operators

Operator	Example
is	A is B
is not	A is not B

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Membership Operators

Operator	Example
in	A in student_list
Not in	A not in student_list

Conditional Statements

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome.

Types of conditional Statements

1) If statement
2) If..else statement
3) Elif statement
4) Nested if

Click here for examples

Looping Statements

A loop statement allows us to execute a statement or group of statements multiple times.

1) For loop : ***click her for example***
2) While loop : ***click here for example***

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

For loop

The for loop iterates over a sequence (like a list, tuple, or string) and executes the block of code for each item in the sequence.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

while loop

The while loop repeats a block of code as long as the given condition is true - while loop is an entry controlled loop

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Control statements

Used to alter the flow of control in a loop or a function

There are 3 statements

1) break
2) continue
3) pass

break statement

The break statement is used to terminate the current loop entirely, even if the loop condition is still true. When break is executed, the loop is exited, and the program continues with the next statement after the loop.

Click here for example

continue statement

The continue statement is used to skip the rest of the current iteration of the loop and move directly to the next iteration.

How it works:

- When continue is encountered, the loop skips the remaining code in the current iteration and goes to the next iteration of the loop.
- Unlike break, continue does not terminate the loop, it just skips to the next cycle.

Click here for example

pass statement

The pass statement is a null operation in Python. It does nothing when executed. It is used as a placeholder where syntax requires a statement but you don't want to execute any code.

How it works:

- It is often used in places where code is syntactically required but you don't want to implement anything yet (like in a function or loop that you want to leave empty).
- It can be used for stubbing out functions, classes, loops, or conditionals.

Click here for example

String

string is a sequence of characters enclosed in quotes. Strings are one of the most commonly used data types in Python and can represent text data. Python has powerful built-in methods for manipulating and working with strings.

1. `len()`

- **Purpose**: Returns the length of a string (number of characters in the string).
- **Syntax**: `len(string)`

2. `lower()`

- **Purpose**: Converts all characters in the string to lowercase.
- **Syntax**: `string.lower()`

3. `upper()`

- **Purpose**: Converts all characters in the string to uppercase.
- **Syntax**: `string.upper()`

4. `capitalize()`

- **Purpose**: Converts the first character of the string to uppercase and the rest to lowercase.
- **Syntax**: `string.capitalize()`

5. `title()`

- **Purpose**: Converts the first character of each word in the string to uppercase.
- **Syntax**: `string.title()`

6. `strip()`

- **Purpose**: Removes any leading (spaces at the beginning) and trailing (spaces at the end) whitespace characters from the string.
- **Syntax**: `string.strip()`

7. `replace()`

- **Purpose**: Replaces a substring within the string with another substring.
- **Syntax**: `string.replace(old, new, count)`
    - `old`: The substring to be replaced.
    - `new`: The substring to replace with.
    - `count` (optional): The maximum number of occurrences to replace (default is all occurrences).

8. `find()`

- **Purpose**: Searches for a substring and returns the index (position) of the first occurrence. If the substring is not found, it returns `-1`.
- **Syntax**: `string.find(substring)`

9. `startswith()`

- **Purpose**: Checks if the string starts with the specified substring.
- **Syntax**: `string.startswith(prefix)`

10. `endswith()`

- **Purpose**: Checks if the string ends with the specified substring.
- **Syntax**: `string.endswith(suffix)`

11. `split()`

- **Purpose**: Splits the string into a list of substrings based on a specified delimiter (default is whitespace).
- **Syntax**: `string.split(separator, maxsplit)`
    - `separator` (optional): The delimiter to split the string by.
    - `maxsplit` (optional): The maximum number of splits to make.

12. `join()`

- **Purpose**: Joins a list of strings into a single string, with a specified separator between each element.
- **Syntax**: `separator.join(iterable)`

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

13. `isalpha()`

- **Purpose**: Returns `True` if all characters in the string are alphabetic (i.e., letters), and `False` otherwise.
- **Syntax**: `string.isalpha()`

14. `isdigit()`

- **Purpose**: Returns `True` if all characters in the string are digits (numbers), and `False` otherwise.
- **Syntax**: `string.isdigit()`

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

15. `isalnum()`

- **Purpose**: Returns `True` if all characters in the string are alphanumeric (letters and numbers), and `False` otherwise.
- **Syntax**: `string.isalnum()`

16. `zfill()`

- **Purpose**: Pads the string with zeros (0) on the left until the string reaches a specified length.
- **Syntax**: `string.zfill(width)`

string functions are essential for handling and manipulating text data in Python

Click here for string examples

String Slicing

String slicing allows you to extract a portion (substring) from a string. This operation is powerful and flexible, as it enables you to access specific characters or parts of a string by specifying a range of indices.

Syntax of String Slicing:

```
String[start:end:step]
```

- `start` (optional): The index where the slice begins. The character at this index is included in the result.
- `end` (optional): The index where the slice ends. The character at this index is **not included** in the result.
- `step` (optional): The step value determines how many characters to skip. The default is `1` (which means no skipping).

Click here for string examples

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 2

Collection, Function And Modules

Collection :

collections are special types of containers that allow you to store and organize multiple items together. These collections can hold different types of data, such as numbers, strings, or even other collections. Python provides several ways to store groups of data, and they are called collection types

Types of collection

List:

- A list is like a **box** where you can keep a collection of items (numbers, strings, etc.). You can change the items in the list later. (mutable)
- Example: my_list = [1, 2, 3, "apple"]

Tuple:

- A tuple is similar to a list, but once you put items inside a tuple, you **can't change them**. (immutable)
- Example: my_tuple = (1, 2, 3, "apple")

Types of collection

Dictionary:

- A dictionary is like a **phone book** where each item has a name (key) and a value (phone number).
- Example: my_dict = {"name": "Alice", "age": 25}

Set:

- A set is like a **collection of unique items**, meaning it doesn't allow duplicates.
- Example: my_set = {1, 2, 3, 4}

List

A **list** in Python is one of the most versatile and widely used data structures. It is an **ordered collection** of items, which can be of **any data type** (such as numbers, strings, or even other lists). Lists are **mutable**, meaning you can change their contents after they are created.

Ordered: The order of elements is maintained.

Mutable: You can modify, add, or remove elements after the list is created.

Heterogeneous: A list can contain elements of different types (integers, strings, floats, etc.).

Indexed: You can access elements using their index, with support for negative indexing.

Iterable: You can loop through the elements of a list using a for loop.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Common List Methods

- append(): Add an element to the end of the list.
- insert(): Insert an element at a specific position.
- extend(): Add multiple elements to the end of the list.
- remove(): Remove the first occurrence of a value.
- pop(): Remove and return an element at a specific position.
- clear(): Remove all elements from the list.
- index(): Find the index of the first occurrence of an element.
- count(): Count how many times an element appears in the list.
- sort(): Sort the list in ascending order.
- reverse(): Reverse the order of elements in the list.

Click here for example

List Comprehension

List comprehension is a concise and readable way to create new lists by performing operations or filtering elements from an existing iterable (like a list, tuple, or string). It allows you to replace loops with a single line of code, making your code shorter and more Pythonic

Syntax : [expression for item in iterable]

expression: This is the value or operation that is applied to each item in the iterable.

item: Represents each individual element from the iterable.

iterable: The collection (like a list, tuple, or string) that you're iterating over.

Click here for example

Tuple

A tuple is a collection type in Python that is similar to a list but with an important difference: tuples are immutable. This means once a tuple is created, you cannot modify its elements, unlike lists, which are mutable (you can change the content of lists).

Tuples are used to store multiple items in a single variable and are typically used when you want to ensure the integrity of the data (i.e., you don’t want the data to be accidentally modified).

Characteristics of Tuples

1. **Immutable**: Once created, elements of a tuple cannot be changed, added, or removed. This makes tuples **hashable** and useful as keys in dictionaries, unlike lists which are not hashable.
2. **Ordered**: Tuples maintain the order of elements. This means that the order in which you add elements to the tuple is preserved when you access them.
3. **Allow Duplicates**: Tuples can contain duplicate elements. Each element is treated as a separate item, even if it's identical to another one.
4. **Heterogeneous**: Tuples can store elements of different types. For example, you can have integers, strings, lists, and even other tuples in the same tuple.

Click here for tuple example

Tuple Methods

`count()`

- The `count()` method returns the number of times a specified element appears in the tuple.

`index()`

- The `index()` method returns the index of the first occurrence of a specified element in the tuple. If the element is not found, it raises a `ValueError`.

Syntax:

```
tuple.index(element, start=0, end=len(tuple))
```

- `start` (optional): The index to start searching from.
- `end` (optional): The index to stop searching at.

Tuple Functions

`len()`

- The `len()` function returns the number of elements in the tuple

`min()`

- The `min()` function returns the smallest element in the tuple.

`max()`

- The `max()` function returns the largest element in the tuple.

`sum()`

- The `sum()` function returns the sum of all the elements in the tuple (only works for numeric tuples).

`sorted()`

- The `sorted()` function returns a sorted list of the elements in the tuple (does not modify the tuple itself).

Dictionary

A **dictionary** is a collection of **key-value pairs**, where each key is unique, and each key is associated with a value. Dictionaries are unordered, mutable, and indexed, which makes them incredibly useful for situations where you need to map or associate one piece of data (the key) with another (the value).

Creating a Dictionary

In Python, dictionaries are created using curly braces {} with keys and values separated by colons :. Each key-value pair is separated by a comma.

Syntax:

```
    my_dict = {key1: value1, key2: value2, key3: value3, ...}
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Dictionary Methods

clear() : Removes all items from the dictionary, leaving it empty.

copy() : Returns a shallow copy of the dictionary. Modifying the copy does not affect the original dictionary.

get()

- Returns the value for a specified key. If the key is not found, it returns None (or a default value if provided).

items()

- Returns a view object that displays a list of the dictionary's key-value tuple pairs.

keys()

- Returns a view object that displays a list of all the dictionary's keys.

pop()

- Removes the item with the specified key and returns its value. If the key is not found, it raises a KeyError (unless a default value is provided).

popitem()

- Removes and returns the last inserted key-value pair as a tuple. (Introduced in Python 3.7, dictionaries are ordered in Python 3.7+).

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

update()

- Updates the dictionary with key-value pairs from another dictionary or iterable of key-value pairs. If the key already exists, its value is updated.

Syntax:

```
dict.update(other_dict)
```

setdefault()

- Returns the value of a key if it exists; if not, inserts the key with the specified value and returns the value.

  Syntax :

  dict.setdefault(key, default=None)

Dictionary Function

del: Deletes a key-value pair from the dictionary.

len(): Returns the number of items in the dictionary.

sorted(): Returns a sorted list of dictionary keys.

dict(): Creates a dictionary from an iterable or key-value pairs.

Click here for example

Set

set is an unordered collection of unique elements. It is a built-in data type that allows you to store multiple items in a single variable

Unordered: The items in a set are not stored in any specific order.

Unique Elements: Sets cannot contain duplicate values. If you try to add a duplicate element, it will be ignored.

Mutable: Sets are mutable, meaning you can add or remove elements from a set after its creation.

No Indexing: Sets do not support indexing, slicing, or other sequence-like behavior

Click here for practical

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Function

A **function** in Python is a block of code that only runs when it is called. Functions allow us to reuse code, make the code more modular, and help break down complex problems into smaller, manageable tasks. Functions in Python can take inputs (called **parameters**) and can return a value.

Creating Function

To create a function in Python, we use the def keyword, followed by the function name and parentheses (which may include parameters).

Syntax:

```
def function_name(parameters):

    # Code block

    return value
```

- def is the keyword that starts the definition of a function.
- function_name is the name of the function.
- parameters are optional values passed to the function. (Can be zero or more).
- return is used to return a value from the function (optional). Without return, the function will return None.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Functions with Parameters

Functions can accept parameters to allow more flexible behavior. Parameters are placeholders that accept values when the function is called.

Returning a Value from a Function

Functions can return a value using the return keyword. Once a function encounters a return statement, it exits the function and sends the returned value back to the caller.

Default Parameters

You can also define default values for parameters. This is useful when you want to provide an optional argument that will be used if the caller doesn't specify one.

Keyword Arguments

You can also pass arguments by explicitly specifying the parameter names when calling the function. This is known as **keyword arguments**.

Arbitrary Arguments (*args and **kwargs)

Sometimes you don't know in advance how many arguments a function will need. In these cases, you can use *args for a variable number of positional arguments and **kwargs for a variable number of keyword arguments.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Lambda Functions (anonymous functions)

Lambda functions are short functions that can have any number of arguments but only one expression. They are often used when a small, throwaway function is needed.

Syntax:

```
lambda arguments: expression
```

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Local Variables:

A **local variable** is defined inside a function and can only be accessed within that function. Once the function execution ends, the local variable is destroyed and cannot be accessed outside the function.

Global Variables:

A **global variable** is defined outside any function, typically at the top of the script. It can be accessed anywhere in the program, including inside functions.

However, to modify a global variable from inside a function, you must declare it as global within the function. Otherwise, Python treats it as a local variable and won't modify the global variable.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Recursion Function

Recursion is a technique in programming where a function calls itself in order to solve a problem. A recursive function has two main parts:

1. **Base Case**: This is the condition where the function stops calling itself. Without a base case, the function would call itself infinitely.
2. **Recursive Case**: This is the part where the function calls itself, reducing the problem size or moving toward the base case

Click here for example

Map()

map() function is a built-in function that allows you to apply a specified function to all the items in an iterable (like a list, tuple, or any other iterable) and return a new iterable (usually a map object) with the results.

Syntax :

```
map(function, iterable)
```

The result of map() is an iterator, which means it will not immediately display the results until you iterate over it. To get a list or other data structure, you often need to explicitly convert it (e.g., using list()).

map() applies the function to each element in the iterable and returns the result.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

filter()

`filter()` function in Python is used to filter elements from an iterable (like a list, tuple, or string) based on a condition specified in a function. The `filter()` function constructs an iterator from elements of the iterable for which the function returns `True`

Syntax:

```
filter(function, iterable)
```

The filter() function returns an **iterator**, so to display or manipulate the results, you usually need to convert it into a list, tuple, or other collection types.

If the function returns True for an element, that element is included in the output; otherwise, it is excluded.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

reduce()

reduce an iterable to a single value by applying a **binary function** (a function that takes two arguments) cumulatively to the items in the iterable.

It **does not produce a one-to-one mapping** but rather applies the function across all elements until a single result is obtained.

reduce() is generally used when you want to **combine** elements, like **summation, multiplication, or finding a maximum**.

Syntax:

```
from functools import reduce

reduce(function, iterable, [initializer])
```

function: A function of two arguments that will be applied to the elements of the iterable.

iterable: The sequence (like a list or tuple) that you want to reduce.

initializer (optional): If provided, it is used as the initial value for the reduction. If not provided, the first item in the iterable is used as the initial value.

Click here for practicals

Generators

generators are a type of iterable, like lists or tuples, but unlike lists, they do not store all the values in memory at once. Instead, a generator produces items one at a time, only when they are needed. This makes generators more memory-efficient when working with large datasets or streams of data.

Generators can be created using:

1. **Generator functions** – A function that uses the yield keyword.
2. **Generator expressions** – A concise syntax for creating a generator

1. Generator Functions

A **generator function** is a function that contains one or more yield statements. When called, it returns a generator object but does not execute the function immediately. Each time the generator is iterated over, the function runs until it hits the next yield statement.

Syntax:

```
def generator_function():
    yield value1
    yield value2
    # More yield statements can follow…
```

Generators do not generate all items at once and do not store them in memory. This makes them more memory-efficient for large datasets.

Real world example

2. Generator Expressions

A generator expression is similar to a list comprehension, but instead of creating a list, it creates a generator object. The syntax is almost identical to list comprehensions, but with parentheses () instead of square brackets [].

Syntax : (expression for item in iterable)

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Using next() with Generators

Generators support the next() function, which can be used to manually retrieve the next value from the generator. Every time next() is called, the generator resumes execution from the last yield statement until it reaches the next one.

If the generator is exhausted (i.e., there are no more values to yield), calling next() will raise the StopIteration exception.

Click here for example

Iterators

An **iterator** is an object that allows you to traverse through all the elements of a collection, such as a list, tuple, or dictionary, one by one. Python provides a mechanism that makes iterating over objects easier.

Key Concepts of an Iterator:

- **Iterator Object**: An object that implements two core methods:
  - **__iter__()**: Returns the iterator object itself. It is used to initialize the iterator.
  - **__next__()**: Returns the next item from the collection. If there are no more items left, it raises the StopIteration exception.

Iterators

- **Iteration Process**: You create an iterator from a collection using iter(). Then, you can use the next() function to retrieve items one by one

  Click here for example

iterator in a for Loop:

A for loop automatically creates an iterator for any iterable (like a list or tuple) and calls next() on it internally until all items are consumed.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Modules

modules are files containing Python code that define functions, classes, and variables, as well as executable code. They help organize code into reusable components and avoid redundancy. Python has both **standard modules** (built-in with Python) and **third-party modules** (which can be installed separately).

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Types of modules

1) User defined module
2) Standard module (in-built)
3) 3rd party module

Creating a Module

A module is simply a Python file with a `.py` extension containing definitions and statements.

For example, a module `math_functions.py` might look like this:

```
def sum(a, b):
    return a + b
def sub(a, b):
    return a - b
```

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Using a module

use a module by importing it into another script using the import statement

```
import math_functions


result = math_functions.add(5, 3)

print(result)  # Output: 8
```

[Click here for practical]

Importing Specific Functions or Variables:

```
from math_functions import function_name1,function_name2
```

Click here from practical

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Aliasing a Module:

```
import math_functions as mf
```

Click here from practical

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Standard Library Modules:

Python comes with many built-in modules that provide useful functionality.

- **math**: Provides mathematical functions like math.sqrt(), math.sin().
- **os**: Interacts with the operating system (e.g., file handling).
- **sys**: Provides access to system-specific parameters and functions.
- **datetime**: Handles date and time operations.
- **random**: Generates random numbers and performs random operations.

Click here for practical

Third-Party Modules:

numpy: For numerical computations.

pandas: For data analysis and manipulation.

requests: For HTTP requests.

flask: For web development.

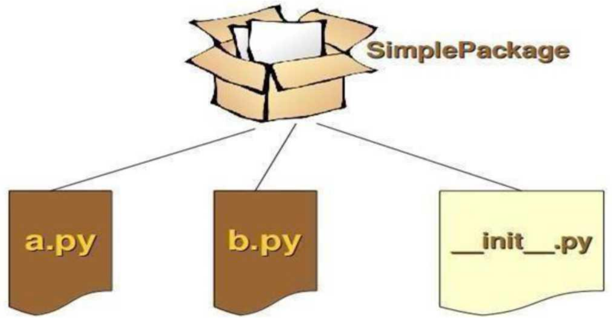
matplotlib: For data visualization.

Note : 3rd party module install using of pip

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Package

package in Python is a way of organizing related modules and files in a structured manner. It allows you to group multiple Python files (modules) together to form a larger program or library. A package can contain multiple modules, sub-packages, and other resources such as data files.



__init__.py file

__init__.py file is an essential part of Python packages. It is a special file that Python uses to recognize a directory as a package. Without this file, Python will not treat the directory as a package, and you won't be able to import any modules from it using the package syntax

Structure of a Package: A Python package is essentially a directory containing an __init__.py file and other Python files (modules). The __init__.py file indicates that the directory should be treated as a package. This file can be empty, but it is necessary for the directory to be recognized as a package.

Example directory structure for a package:

```
mypackage/
    __init__.py
    module1.py
    module2.py
```

Using a Package: After creating a package, you can import and use its modules in your Python code.

For example, to use module1.py from the mypackage package:

```
from mypackage import module1
```

Click here for practical

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 3

Advance Python Programming

File Handling

File handling in Python refers to the ability to read from and write to files stored on your computer.

This allows Python programs to interact with text files, binary files, or other file formats. File handling involves opening files, reading and writing data to them, and then closing them when done.

Open File

Before work with a file in Python (whether reading or writing), you first need to **open** it using the open() function. This function requires two arguments:

- **File path**: The location of the file on your computer.
- **Mode**: The mode determines the type of operation you want to perform on the file (e.g., reading, writing, appending).

```
file = open('example.txt', 'r') # 'r' is the mode for reading a file
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Modes for Opening Files

The **mode** tells Python what you want to do with the file.

some common modes:

- 'r': Read mode – This opens the file for reading. The file must exist.
- 'w': Write mode – This opens the file for writing. If the file already exists, it will be overwritten. If it doesn't exist, a new file will be created.
- 'a': Append mode – This opens the file for writing, but it won't overwrite the existing content. Instead, new data will be added at the end of the file.
- 'rb': Read mode in binary – For reading binary files (e.g., images, audio files).
- 'wb': Write mode in binary – For writing binary files.

read(): Reads the entire content of the file as a single string

Click here for practical

readline(): Reads the next line from the file. If called multiple times, it will return the next line each time.

Click here for practical

readlines(): Reads all lines from the file and returns them as a list of strings (each line is an element in the list).

Click here for practical

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Writing to a File

If you open a file in write mode ('w') or append mode ('a'), you can write content to the file

`write()`: Writes a string to the file.

Click here for example

Closing the File

Once you're done reading from or writing to a file, you should close the file to free up resources. This is done using the close() method

`writelines()`:

If you have multiple lines to write at once, you can use the writelines() method.

This method takes a list of strings and writes each string to the file without adding line breaks automatically. You need to ensure that each string ends with \n if you want a new line.

Click here for example

with

The with statement is recommended for handling files because it automatically handles file opening and closing. It ensures that the file is properly closed even if an error occurs while writing.

Use the 'with' statement to automatically handle file opening and closing

Click here for example

append()

To append a string to a file in Python, you can use the open() function with the 'a' (append) mode. This mode allows you to add new content to the end of an existing file without modifying its current content

Click here for example

JSON

JSON file from user input in Python, you can use the json module. This module allows you to easily convert Python data structures (like dictionaries, lists) into JSON format and save them into a file.

Click here for example

Click here for multiple input example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Exception

exception is an event or condition that disrupts the normal flow of execution of a program.

It usually happens when the program encounters an error or an unexpected situation during execution. Exceptions can arise from various situations.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Exception Handling

Exception handling is the process of anticipating and managing exceptions in a program so that it does not crash.

Instead of letting the program stop when an error occurs, you can handle the exception gracefully, allowing the program to either recover from the error, skip over it, or show an appropriate error message to the user.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Key Concepts of Exception Handling:

- **try Block**: You write the code that might cause an exception inside a try block. This is the section where you anticipate something could go wrong.
- **except Block**: If an exception occurs within the try block, the code in the except block is executed. You specify the type of exception you want to catch (e.g., ZeroDivisionError, ValueError).
- **else Block**: The else block runs only if no exception occurred in the try block. It's used for code that should only execute when no errors are raised.
- **finally Block**: The finally block is executed no matter what, whether an exception occurs or not. It's typically used for cleanup tasks, such as closing files or releasing resources.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

ZeroDivisionError

This error occurs when you try to divide a number by zero.

Click here for example

ValueError

This error occurs when you pass an argument of the right type but an inappropriate value. For example, when trying to convert a string that doesn't represent a valid number into an integer.

Click here for example

IndexError

This error occurs when you try to access an element in a list using an index that is out of range.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

`KeyError`

This error occurs when you try to access a dictionary with a key that doesn't exist.

Click here for example

`FileNotFoundError`

This error occurs when you try to open a file that does not exist.

Click here for example

`TypeError`

This error occurs when you perform an operation on an object of inappropriate type.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

`AttributeError`

This error occurs when you try to access an attribute or method that does not exist for a particular object.

Click here for example

`NameError`

This error occurs when you try to use a variable or function that has not been defined.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

else block

The else block is executed only if no exception is raised in the try block. It is useful for code that should only run when no errors occur in the try block

finally block

The finally block is always executed, no matter what happens, whether an exception occurs or not. It is typically used for clean-up operations, such as closing files, releasing resources, or committing changes to a database.

The finally block runs even if an exception is raised in the try block, and it also runs if the exception is caught by an except block. It is guaranteed to execute before the program exits the try-except structure.

Click here for example

Custom Exception

create your own exceptions by defining a custom exception class. Custom exceptions allow you to define specific error conditions that are relevant to your program, making your code more readable and easier to maintain.

To create a custom exception, you define a new class that inherits from the built-in Exception class (or any of its subclasses). You can also add custom attributes or methods to your exception class if needed.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Decorators

decorators are a powerful and elegant way to modify or extend the behavior of functions or methods without changing their actual code.

A decorator is essentially a **function** that takes another function as an argument, adds some functionality to it, and then returns a new function with the added functionality.

Basic Syntax of a Decorator:

```
def decorator_function(original_function):

    def wrapper_function():

        print("Wrapper executed before {}".format(original_function.__name__))

        return original_function()

    return wrapper_function
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Why use Decorators :

Decorators are often used in Python for a variety of tasks:

- **Logging**: Add automatic logging to functions.
- **Access Control:** Implement access control or authentication (e.g., for web applications).
- **Memoization:** Cache the results of function calls for performance improvements (like functools.lru_cache).
- **Validation:** Check inputs to functions or methods before they execute.

Click here for practicals

OOPS

Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around data, or objects, rather than functions and logic. Python is an object-oriented language, which means it supports all the main concepts of OOP, including **classes**, **objects**, **inheritance**, **encapsulation**, **polymorphism**, and **abstraction**.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Class and Object

Class: A class is a blueprint for creating objects (a particular data structure), defining initial states (attributes), and behaviors (methods).

Object: An object is an instance of a class. It is the actual entity created based on the class blueprint.

Concepts in Python Classes

Class Definition: A class is defined using the class keyword, followed by the class name (in CamelCase style by convention).

Attributes: Attributes are variables that hold data for the objects created from the class. These can be instance attributes (specific to an object) or class attributes (shared by all instances of the class).

Methods: Methods are functions that define the behaviors of the class. They typically operate on the attributes of the object or class and can be used to modify or interact with the object's data.

Constructor (__init__): The constructor is a special method that is called when an object is created from the class. It initializes the object's state (attributes).

Instance vs. Class Attributes:

Instance attributes are specific to each object created from the class.

Class attributes are shared by all instances of the class.

Instance Methods: Methods that operate on instances of the class, using the instance's attributes.

Class Methods: Methods that operate on the class itself rather than individual instances. These methods are defined with the `@classmethod` decorator.

Static Methods: Methods that do not operate on an instance or the class itself but behave like normal functions. These methods are defined with the `@staticmethod` decorator

Syntax

```
class ClassName:

    class_variable = "This is a class attribute"  # Class attribute


    def __init__(self, instance_variable):

        self.instance_variable = instance_variable  # Instance attribute


    def instance_method(self):

        return f"Instance method, value: {self.instance_variable}"
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Syntax

```
@classmethod

 def class_method(cls):

     return f"Class method, value: {cls.class_variable}"


 @staticmethod

 def static_method():

     return "Static method, no access to instance or class.”
```

Click here for example

Instance Attributes : These are specific to each object created from the class.

Click here for example

Class Attributes : These are shared by all instances of the class.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

`Self :` self parameter in methods refers to the current object being operated on. This allows each instance of the class to store its own data. You don't pass it explicitly when calling a method on an object (it’s passed automatically).

Instance Methods :

These methods operate on individual objects (instances) and have access to instance attributes through self.

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Class Methods :

Class methods are methods that operate on the class itself rather than on instances. They take cls as their first parameter (instead of self), which refers to the class, not the instance. These methods can access class-level attributes and modify them

They are defined using the @classmethod decorator.

Click here for practical

Static Methods :

Static methods do not depend on the instance or the class. They are simply functions that belong to the class's namespace but do not have access to instance or class-specific data. Static methods do not take **self** or **cls** as parameters. They are often used for utility functions that perform tasks unrelated to the instance or the class

They are defined using the @staticmethod decorator.

They do not take self or cls as the first parameter.

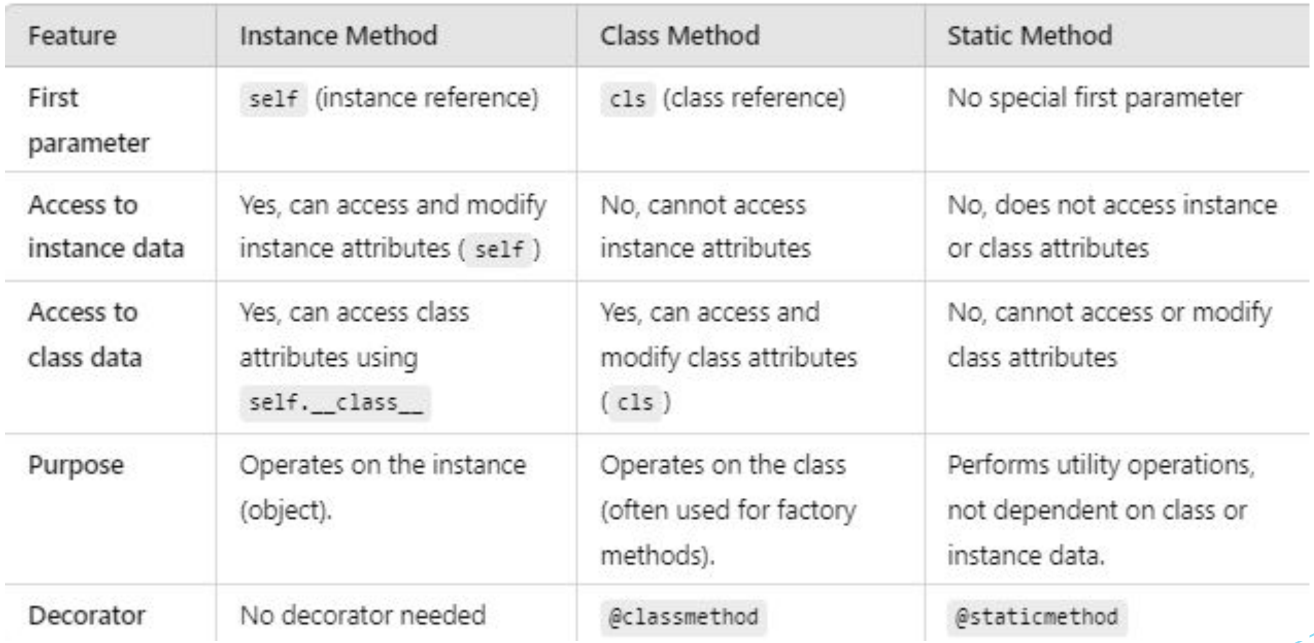
They cannot access or modify instance-specific or class-specific data.

Syntax :

```
class MyClass:
    @staticmethod
    def some_utility_function(arg1, arg2):
        return arg1 + arg2
```

Click here for practicals

Difference between instance , class and static methods



Feature	Instance Method	Class Method	Static Method
First parameter	`self` (instance reference)	`cls` (class reference)	No special first parameter
Access to instance data	Yes, can access and modify instance attributes (`self`)	No, cannot access instance attributes	No, does not access instance or class attributes
Access to class data	Yes, can access class attributes using `self.__class__`	Yes, can access and modify class attributes (`cls`)	No, cannot access or modify class attributes
Purpose	Operates on the instance (object).	Operates on the class (often used for factory methods).	Performs utility operations, not dependent on class or instance data.
Decorator	No decorator needed	`@classmethod`	`@staticmethod`

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Object

Objects are instances of classes, which are templates or blueprints for creating objects. An object represents a collection of attributes (data) and methods (functions) that operate on the data. When a class is instantiated, an object is created, and it holds a specific set of data.

Memory Management: Objects are dynamically allocated in memory. When an object is no longer in use (i.e., no references point to it), it is eligible for **garbage collection**, which is a process where Python automatically frees up memory by removing unused objects.

Garbage Collection: Python uses a garbage collector to manage memory and remove objects that are no longer being used.

Encapsulation

Encapsulation in Python refers to the concept of restricting access to certain details of an object and exposing only necessary components. This concept is a fundamental principle of object-oriented programming (OOP), where the internal state of an object is hidden, and the object interacts with the outside world only through a well-defined interface (methods or functions).

Encapsulation is a way to bundle the data (attributes) and methods (functions) that work on the data into a single unit, i.e., a class, and restrict access to some of the object's components.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Private Attributes and Methods

By default, all attributes and methods in Python are public, which means they can be accessed and modified from outside the class. However, we can make attributes and methods private by using a specific naming convention.

Private attributes and methods are intended to be accessed only within the class itself. To make an attribute or method private, we prefix the attribute/method name with two underscores (__).

Click here for practical

Public Attributes and Methods:

Public attributes and methods are those that can be accessed from outside the class. By default, all attributes and methods are public unless specifically made private.

Getter and Setter Methods:

Getters and setters are methods used to get and set the value of private attributes. They provide controlled access to the private attributes, allowing us to apply validation or other logic before updating or retrieving the data.

Click here for practical

Inheritance

Inheritance in Python is an object-oriented programming concept where a class (child class) can inherit attributes and methods from another class (parent class). This allows code reusability and helps to create a hierarchical relationship between classes.

Parent Class: The class whose properties and methods are inherited by another class.

Child Class: The class that inherits the properties and methods from the parent class.

Method Overriding: The ability of a child class to provide a specific implementation of a method that is already defined in its parent class.

Inheritance

Syntax :

```
    class ParentClass:

        # Parent class code here

        pass


    class ChildClass(ParentClass):

        # Child class code here

        pass
```

Click here for Practical

Types of Inheritance

Single Inheritance: One child class inherits from one parent class.

Multiple Inheritance: One child class inherits from more than one parent class.

Multilevel Inheritance: A class inherits from another class, which itself inherits from a third class.

Hierarchical Inheritance: Multiple child classes inherit from a single parent class.

Hybrid Inheritance: A combination of multiple types of inheritance.

Click here for practical

Abstraction

Abstraction in Python is one of the fundamental concepts of Object-Oriented Programming (OOP). It refers to the process of hiding the complex implementation details of a system and exposing only the essential features or interfaces to the user. This helps to reduce complexity and makes the code more readable and maintainable.

Key Points of Abstraction:

1. **Hiding Implementation Details:** The idea is to separate what the object does from how it does it.
2. **Interfaces:** An abstraction allows users to interact with an object through a clear interface without needing to understand its underlying complexities.
3. **Simplifying Code:** By abstracting complex logic, you can create code that is easier to manage and extend.

Abstraction

How Abstraction Works in Python:

In Python, abstraction is achieved through:

1. **Abstract Classes**
2. **Abstract Methods**

Abstract Classes

An **abstract class** is a class that cannot be instantiated on its own. It serves as a blueprint for other classes. Abstract classes are defined using the abc (Abstract Base Class) module.

Abstraction

Abstract Methods

An **abstract method** is a method that is declared in an abstract class but does not have any implementation. The subclasses of this abstract class are required to provide their own implementation of the abstract methods.

Click here for practical

Constructor Overloading

Python allows only one constructor method, which is the __init__ method. However, you can achieve the effect of constructor overloading by using default arguments or variable-length argument lists (*args, **kwargs).

Click here for practical

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Dunder Method

Dunder methods (short for **"double underscore" methods**) are special methods in Python that begin and end with double underscores (__).

These methods are also known as **magic methods** or **special methods**. They are used to implement and customize behavior for Python's built-in operations on objects, such as addition, subtraction, string representation, and more.

Dunder methods enable you to define how objects of a class behave with built-in functions or operators. For example, they allow you to specify how objects of your class are added, multiplied, or compared, as well as how they are represented as strings.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Dunder Method

__init__(self, ...):

- Called when an object is created (constructor).
- Used to initialize the object's attributes.

__str__(self):

- Defines how an object should be represented as a string (e.g., for print()).
- Should return a string

__add__(self, other):

- Allows you to define the behavior of the + operator for your objects.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Dunder Method

__eq__(self, other):

- Defines the behavior of the equality operator == for your objects

__len__(self):

- Defines the behavior of the len() function on your objects.

__getitem__(self, key):

- Defines the behavior when indexing an object (using obj[key]).

__setitem__(self, key, value):

- Defines the behavior when setting an item in an object (using obj[key] = value).

Click here for example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Polymorphism

Polymorphism is a fundamental concept in Object-Oriented Programming (OOP) that allows objects of different classes to be treated as objects of a common superclass. It means "many forms" in Greek, and in programming, it refers to the ability to call the same method on different objects and each object responds in its own way, depending on its class.

In simpler terms, polymorphism allows different classes to define methods that have the same name but may behave differently based on the specific class or instance they are called on.

Types Of Polymorphism

There are two main types of polymorphism in Python:

1. **Compile-time Polymorphism (Static Binding)**
   - This is also known as method overloading in some languages, though Python does not support it directly due to its dynamic nature.
   - In languages that support compile-time polymorphism, you can have multiple methods with the same name but different parameters (method overloading).
2. **Runtime Polymorphism (Dynamic Binding)**
   - This is the type of polymorphism that Python supports, where the method that gets executed is determined at runtime based on the object's actual type.
   - The key mechanism for runtime polymorphism in Python is method overriding.

Click here for example

Aggregation

Aggregation is a type of association in object-oriented programming (OOP) where one object (called the *whole*) contains or is composed of other objects (called *parts*), but the lifecycle of the contained objects is independent of the lifecycle of the whole object. In other words, the contained objects can exist independently outside the context of the whole object.

In simple terms, **aggregation** represents a "Has-A" relationship between two objects, where one object "has" or "owns" another object, but both objects can exist independently

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Why Use Aggregation?

1. **Flexibility**: Aggregation allows flexibility in the design. The parts can exist independently of the whole and can be shared across different objects.
2. **Reduced Coupling**: It reduces the coupling between the whole object and the part objects, making the system more modular.
3. **Reusability**: Aggregation allows objects to be reused in multiple contexts. For instance, a Book can belong to multiple libraries or even no library at all.

Click here for practical

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Regular Expressions

Regular Expressions (regex) are sequences of characters that form search patterns. They are primarily used for string matching and manipulation. In Python, the re module provides support for working with regular expressions, which can help you search, match, replace, and split strings based on patterns.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Re Module

- re.match(pattern, string): Checks if the pattern matches at the beginning of the string.
- re.search(pattern, string): Searches for the pattern anywhere in the string.
- re.findall(pattern, string): Returns a list of all matches of the pattern in the string.
- re.finditer(pattern, string): Returns an iterator yielding match objects.
- re.sub(pattern, repl, string): Replaces occurrences of the pattern with the replacement string.
- re.split(pattern, string): Splits the string by occurrences of the pattern.

Re Module

Metacharacters and Their Usage

Dot (.): Matches any character except newline (\n).

- Example: a.b will match "aab", "acb", "a3b", but not "ab".

Caret (^): Matches the start of the string.

- Example: ^Hello will match if the string starts with "Hello".

Dollar Sign ($): Matches the end of the string.

- Example: end$ will match if the string ends with "end".

Asterisk (*): Matches zero or more repetitions of the preceding element.

Example: lo*l will match "ll", "lol", "lool", etc.

Plus (+): Matches one or more repetitions of the preceding element.

Example: a+b will match "ab", "aab", "aaab", etc.

Question Mark (?): Matches zero or one occurrence of the preceding element.

Example: colou?r will match both "color" and "colour".

Square Brackets [ ]: Matches any one of the characters inside the brackets.

- Example: [aeiou] will match any vowel.

Hyphen (-): Defines a range inside square brackets.

- Example: [0-9] matches any digit, and [a-z] matches any lowercase letter.

Parentheses ( ): Groups expressions and captures matched parts.

- Example: (abc)+ will match one or more repetitions of "abc".

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Special Sequences

`\d`: Matches any digit (equivalent to `[0-9]`).

`\D`: Matches any non-digit.

`\w`: Matches any word character (alphanumeric + underscore).

`\W`: Matches any non-word character.

`\s`: Matches any whitespace character (spaces, tabs, newlines).

`\S`: Matches any non-whitespace character.

`\b`: Matches a word boundary.

`\B`: Matches a non-word boundary.

`\n`: Matches a newline character.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Quantifiers

Quantifiers define the number of times an element should appear in the pattern. Common quantifiers are:

1. *: Zero or more times.
   - Example: a* matches "", "a", "aa", "aaa", etc.
2. +: One or more times.
   - Example: a+ matches "a", "aa", "aaa", etc., but not "".
3. **{n}**: Exactly n times.
   - Example: a{2} matches "aa".
4. **{n,}**: At least n times.
   - Example: a{2,} matches "aa", "aaa", "aaaa", etc.
5. **{n,m}**: Between n and m times.
   - Example: a{2,4} matches "aa", "aaa", or "aaaa".

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Anchors

Anchors specify positions in a string:

1. ^: Asserts the position at the start of the string.
   - Example: ^abc matches "abc" at the beginning of the string.
2. $: Asserts the position at the end of the string.
   - Example: abc$ matches "abc" at the end of the string.

Click here for practical

Tkinter

Tkinter is the standard Python interface to the **Tk GUI (Graphical User Interface)** toolkit. Tk is a popular toolkit for building desktop applications, and Tkinter provides a way to create windows, dialogs, buttons, labels, and other GUI elements that make your Python programs interactive.

Tkinter is a wrapper around the Tk GUI toolkit and provides an easy way to create desktop applications in Python. It's simple to use and comes pre-installed with Python, so you don't need to install it separately.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Basic Components in Tkinter

1. **Window**: The main GUI container.
2. **Widgets**: Elements like buttons, labels, entry fields, etc., that you place inside a window.
3. **Geometry**: Controls the layout of the widgets.
4. **Event handling**: Lets you specify what happens when users interact with the application (e.g., clicking a button).

Click here for practical

Layout Management Methods

1. **pack()**:
   - Automatically arranges widgets in the available space.
   - Example: widget.pack()
2. **grid()**:
   - Organizes widgets in a table-like structure (rows and columns).
   - Example: widget.grid(row=0, column=0)
3. **place()**:
   - Positions widgets at an absolute location (x, y).
   - Example: widget.place(x=100, y=200)

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Create your own exe file

To convert a **Tkinter Python file (.py)** into a **Windows executable (.exe)**, we can use tools like **PyInstaller** or **cx_Freeze**.

Click here for practical

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Pymysql

PyMySQL is a pure-Python MySQL client library that allows Python applications to interact with MySQL databases. It is one of the popular libraries used for connecting to MySQL databases in Python because of its simplicity and ease of use.

Installation of Pymysql

```
pip install pymysql
```

Importing PyMySQL

```
import pymysql
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Establishing a Connection to MySQL Database

```
import pymysql

# Establishing the connection

connection = pymysql.connect(

    host='localhost',      # MySQL server address

    user='your_user',      # MySQL username

    password='your_pass', # MySQL password

    database='your_db',    # Database to connect to

)
```

Cursor Object

The cursor interacts with the MySQL database and fetches the results of queries.

```
cursor = connection.cursor()
```

Click here for practical

Executing a SELECT Query

A SELECT query retrieves data from the database. Use the cursor to execute the query and fetch the results.

```
# SQL SELECT query

query = "SELECT * FROM users"

cursor.execute(query)



# Fetch all results

results = cursor.fetchall()



# Iterate through the results

for row in results:

    print(row)
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Executing an INSERT Query

To insert data into the database, use an `INSERT INTO` statement:

```
query = "INSERT INTO users (name, age) VALUES (%s, %s)"

values = ("John Doe", 28)


cursor.execute(query, values)
connection.commit()  # Commit the transaction
```

Executing an UPDATE Query

You can update data in the database using an `UPDATE` query:

```
query = "UPDATE users SET age = %s WHERE name = %s"

values = (30, "John Doe")


cursor.execute(query, values)

connection.commit()
```

Executing a DELETE Query

To delete records from the database:

```
query = "DELETE FROM users WHERE name = %s"

values = ("John Doe",)


cursor.execute(query, values)
connection.commit()
```

Click here for practical

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 4

DB and Framework

HTML

- HTML Introduction
- HTML Getting Started
- HTML Elements
- HTML Attributes
- HTML Basic Tags
- HTML - Images
- HTML - Tables
- HTML - Lists
- HTML - Form

HTML

- **Introduction to HTML**
- **What is HTML?**

  HTML (HyperText Markup Language) is the standard language used to create and design web pages. It structures the content of websites using elements and tags.

HTML

Key Features of HTML:

- **Markup Language**: Uses tags to structure and format content.
- **Platform Independent**: Runs on any browser.
- **Building Block of Websites**: Forms the foundation for web development, combined with CSS and JavaScript.
- **Hyperlinks**: Connects multiple pages through links.

- **HTML <form> Tag**
- The <form> tag in HTML is used to create an interactive form for collecting user inputs. Forms are essential for tasks such as user registration, login, searching, and data submission.

Key Attributes of the <form> Tag:

1. **action**: Specifies the URL where the form data will be sent after submission.
   Example: action="submit.php"
2. **method**: Defines how the form data is sent to the server. Common methods:
   - GET: Appends data to the URL. Suitable for non-sensitive data.
   - POST: Sends data in the request body. Used for sensitive or large data.
3. **enctype**: Specifies how form data should be encoded when submitting it.
   - Commonly used with POST method (e.g., for file uploads).
4. **target**: Defines where to display the response after form submission.
   Example: _blank (new tab), _self (same tab).

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

HTML

Commonly Used Form Elements:

- <input>: For text, passwords, emails, etc.
- <textarea>: For multiline text input.
- <button>: For submitting or resetting forms.
- <select>: Dropdown list.
- <label>: Describes form elements.

CSS

- **Introduction to CSS**
- **What is CSS?**
  CSS (Cascading Style Sheets) is a stylesheet language used to style and layout web pages. It allows developers to control the appearance of HTML elements, including colors, fonts, spacing, positioning, and overall design.

Key Features of CSS:

1. **Separation of Content and Design**: HTML defines structure, while CSS handles styling.
2. **Reusable**: One CSS file can style multiple HTML pages.
3. **Flexibility**: Enables responsive designs for various devices and screen sizes.
4. **Customizable**: Provides control over every aspect of a webpage's appearance.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Types Of Css

Inline CSS: Applied directly within an HTML element using the style attribute.

```
<p style="color: blue;">This is blue text.</p>
```

Internal CSS: Defined within a <style> tag inside the <head> of an HTML document.

```
<style>
  p {
    color: green;
  }
</style>
```

External CSS: Written in a separate .css file and linked to the HTML file.
Example:

```
<link rel="stylesheet" href="styles.css">
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

CSS

Advantages of CSS:

1. Saves time: Allows consistent styling across multiple pages.
2. Enhances design: Enables complex layouts and animations.
3. Device adaptability: Facilitates responsive designs for mobile, tablet, and desktop.
4. Easy maintenance: Styling changes can be made in a single location.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

CSS

Limitations of CSS:

1. Browser compatibility issues may arise.
2. Complex designs can require a deeper understanding of CSS rules.
3. Not a programming language, so dynamic behavior needs JavaScript.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

JavaScript

- JavaScript Introduction
- Types Of JS
- Implement JS in HTML
- Client Side Validation in JS

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

JavaScript

- **Introduction to JavaScript**
- **What is JavaScript?**
  JavaScript is a powerful, lightweight, and versatile programming language primarily used to add interactivity, dynamic behavior, and logic to web pages. It is a client-side language, but it can also be used server-side (e.g., with Node.js).

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

JavaScript

Key Features of JavaScript:

1. **Dynamic and Interactive:** Adds interactivity to web pages, such as form validation, animations, and real-time updates.
2. **Lightweight:** Designed to run efficiently in web browsers.
3. **Cross-Platform:** Works across all major browsers and platforms.
4. **Event-Driven:** Responds to user actions like clicks, mouse movements, or keyboard inputs.
5. **Integration:** Seamlessly integrates with HTML and CSS.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

JavaScript

Applications of JavaScript:

- Form validation (e.g., checking required fields).
- Interactive content like slideshows and popups.
- Dynamic updates without reloading the page (using AJAX).
- Backend development (e.g., Node.js).
- Game development and animations.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

JavaScript

Basic Syntax:

1. **Including JavaScript:**

Inline:

```
<button onclick="alert('Hello!')">Click Me</button>
```

Internal:

```
<script>

  console.log('JavaScript is running!');

</script>
```

External:

```
<script src="script.js"></script>
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

JavaScript

- **Variables:**
- let name = 'John'; // Block-scoped variable
- const age = 25;   // Constant value
- var city = 'Paris';

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

JavaScript

Functions:

```
function greet() {
        console.log('Hello, World!');
}
greet();
```

JavaScript

Events:

```
<button id="btn">Click Me</button>
<script>
  document.getElementById('btn').onclick = function () {
    alert('Button clicked!');
  };
</script>
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

JavaScript

Advantages of JavaScript:

1. Enhances user experience with dynamic content.
2. Executes directly in the browser, reducing server load.
3. Rich ecosystem of libraries and frameworks (e.g., React, Angular).
4. Used for both front-end and back-end development.

Limitations of JavaScript:

1. Security risks like Cross-Site Scripting (XSS) if not handled properly.
2. Runs differently on various browsers, requiring testing and debugging.
3. Can become complex in large applications without proper structure.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Django Framework Introduction

Django is a high-level, open-source web framework written in **Python**. It is designed to help developers build robust, secure, and scalable web applications quickly and efficiently. It follows the **Model-View-Controller (MVC)** architectural pattern, though Django refers to it as **Model-View-Template (MVT)**.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Django Framework Introduction

Key Features of Django

1. **Rapid Development**: Django provides ready-to-use components for handling common web development tasks, speeding up the development process.
2. **Scalability**: It is capable of handling applications with heavy traffic and complex requirements.
3. **Security**: Django helps developers avoid common security issues such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
4. **Versatility**: Suitable for various types of projects, from small websites to large-scale enterprise applications.
5. **Extensibility**: Offers a large ecosystem of reusable apps and plugins.

Django Framework Introduction

Components of Django

1. **Model**: Represents the database schema and data structure. It provides an abstraction layer over the database.
2. **View**: Contains the business logic and determines what data to present and how to handle user requests.
3. **Template**: Manages the presentation layer, focusing on how the data is displayed to the user.
4. **URL Dispatcher**: Maps URLs to views, enabling a clean and organized URL structure.
5. **ORM (Object-Relational Mapping)**: Django's built-in ORM simplifies database interactions by allowing developers to interact with databases using Python objects instead of raw SQL.
6. **Admin Interface**: Automatically generated interface for managing application data, significantly reducing the need for custom backend tools.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Django Framework Introduction

Advantages

- Large and active community support.
- Comprehensive documentation.
- Pre-built tools for handling authentication, session management, and more.
- Portable across multiple platforms and databases.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

What is virtualenv

virtualenv is a tool used to create isolated Python environments. In the context of Django (or any other Python project), it allows you to create a separate environment with its own Python interpreter and dependencies, independent of the system's global Python environment. This is useful because:

- It helps manage project-specific dependencies (e.g., specific versions of Django or other libraries).
- It avoids conflicts between packages required by different projects.
- It prevents potential issues that arise from using global packages that could be updated or modified.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Why Use virtualenv in Django?

- **Isolate project dependencies**: Different Django projects may require different versions of Django or other dependencies. Virtual environments allow each project to have its own set of dependencies, ensuring that one project's requirements don't affect another project.

- **Avoid permission issues**: When you install packages globally, you may run into permission issues. Virtual environments let you install packages locally within the environment without requiring elevated privileges (e.g., sudo).

- **Cleaner project setup**: Each project gets a clean, isolated environment, making it easier to replicate the environment for others, such as colleagues or when deploying to a production server.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Django setup steps

Install virtualenv (if not already installed):

```
#pip install virtualenv
```

Create a virtual environment:

```
#virtualenv -m venv myenv
# cd myenv
```

Activate the virtual environment:

On windows :

```
#Scripts\Activate
```

On Mac :

```
#source /bin/activate
```

Install Django:

```
#pip install django
```

Create a Django Project

```
#django-admin startproject myproject
```

Navigate into your project directory:

```
#cd myproject
```

Create a Django App:

```
#python manage.py startapp myapp
```

Run the Development Server:

```
#python manage.py runserver
```

Models

- models are the Python classes that represent the structure of your database tables. They are an essential part of Django’s Object-Relational Mapping (ORM) system, which allows you to interact with your database using Python objects instead of raw SQL queries. A model in Django defines the fields and behaviors of the data you want to store in the database

Key Points about Django Models:

- Django Models = Database Tables: Each model class corresponds to a database table, and each model instance corresponds to a row in that table.

- Fields: The attributes of a model class define the columns in the database table. Each field in a model corresponds to a specific type of data that will be stored in that column (e.g., strings, dates, numbers).

- Automatic Database Handling: Django automatically handles the creation of the corresponding database tables and operations (like inserts, updates, deletes) based on your model definitions.

Types of Fields in Django Models

Django provides a wide range of field types for different data types. Some of the commonly used fields are:

- Text Fields:
  - CharField: For short strings, like names, titles, or short descriptions.
  - TextField: For longer text data (e.g., paragraphs).
- Date/Time Fields:
  - DateField: For date values (e.g., YYYY-MM-DD).
  - TimeField: For time values.
  - DateTimeField: For both date and time values.
- Numeric Fields:
  - IntegerField: For storing integer values.
  - FloatField: For floating-point numbers.
  - DecimalField: For fixed-point decimal numbers (often used for financial data).

Boolean Fields:

    ◦ `BooleanField`: For True/False values.

- File Fields:
    - `FileField`: For uploading files.
    - `ImageField:` A specialized FileField for image uploads.

- Foreign Keys, Many-to-Many, and One-to-One Relationships:
    - `ForeignKey:` A one-to-many relationship (e.g., a book can have one author, but an author can have many books).
    - `ManyToManyField:` A many-to-many relationship (e.g., a student can enroll in multiple courses, and a course can have multiple students).
    - `OneToOneField:` A one-to-one relationship (e.g., each user has a unique profile).

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

ORM

ORM (Object-Relational Mapping) is a programming technique that allows you to interact with a relational database using an object-oriented paradigm. In simpler terms, it provides a way for developers to interact with databases using the programming language's objects, instead of writing raw SQL queries.

Django, as a web framework, uses its own ORM to map the database schema to Python classes and allows developers to interact with the database using Python objects, rather than writing SQL directly. This makes it easier to manage database operations like querying, updating, inserting, and deleting records.

ORM Methods

`all()`: Retrieves all records from the table.

`filter()`: Filters the records based on a condition (similar to a `WHERE` clause in SQL).

`exclude()`: Excludes records that match a condition.

`get()`: Retrieves a single record (raises `DoesNotExist` if no record is found).

`create()`: Creates and saves a new record in the database.

`update()`: Updates fields of the records.

`delete()`: Deletes records from the table.

`order_by()`: Orders the query results.

`distinct()` : Returns only distinct (unique) records from the QuerySet.

Migrations

Migrations allow you to version control database changes

makemigrations in Django

makemigrations is a Django management command that is used to create new migration files based on changes you've made to your models. When you modify or add new models, fields, or relationships in your Django app's models.py file, you need to generate a migration to reflect those changes in the database.

What does makemigrations do?

- It inspects the changes made to the models in your Django app.
- It generates migration files that represent those changes.
- These migration files are stored in the migrations/ directory of your app.

#python manage.py makemigrations

Django will analyze your models, detect the changes, and generate migration files in the migrations/ directory.

migrate in Django

Once you have generated migration files using makemigrations, you need to apply those changes to the actual database. This is done using the migrate command.

What does migrate do?

- It applies the migrations to your database, ensuring that the schema is updated according to the migration files.
- It runs the migration files in sequence, making the necessary changes to the database.
- It keeps track of which migrations have been applied by creating a table in the database called django_migrations.

#python manage.py migrate

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

views

Views are functions or classes that receive web requests and return web responses. A view is responsible for processing user requests, interacting with models (if necessary), and returning an appropriate response, such as HTML content, JSON data, or redirects.

Types of Views in Django

Django provides two main types of views:

1. Function-based views (FBV)
2. Class-based views (CBV)

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

CSRF token

(Cross-Site Request Forgery token) is a security feature designed to protect web applications from CSRF attacks. Django automatically generates and validates CSRF tokens for forms that submit data to the server, ensuring that requests are legitimate and originated from the actual user rather than a malicious third party.

Click here for practical

Redirects and HTTP Responses

Django views often need to return more than just plain text or HTML. They may need to redirect users or return other types of responses:

- **HttpResponse:** A basic response that can contain text, HTML, JSON, or other content types.

- **Redirect:** Used to redirect the user to a different URL.

- **HttpResponseRedirect:** Similar to Redirect, but explicitly indicates the use of a response for redirection.

HttpResponse in Django

HttpResponse is the most basic response type in Django. It allows you to send any kind of data as a response, such as plain text, HTML, JSON, or other content types.

Basic Structure of HttpResponse

```
from django.http import HttpResponse

def my_view(request):

    return HttpResponse("Hello, World!")
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Redirect in Django

Redirect is used to send an HTTP redirect response to the client.

It tells the browser to navigate to a different URL.

When you want to redirect users to another page, you can use the Redirect class or the HttpResponseRedirect class in Django

```
from django.shortcuts import redirect

def redirect_to_homepage(request):
    return redirect('homepage')
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Redirect with Arguments

You can also redirect to a URL that includes dynamic arguments. For example, if you want to redirect to a URL that includes the ID of an object, you can pass the ID as an argument:

```
from django.shortcuts import redirect

def redirect_to_book(request, book_id):
    return redirect('book_detail', pk=book_id)
```

HttpResponseRedirect

HttpResponseRedirect is used when you want to redirect the user to a different URL. It is typically used after handling a form submission or some other action that requires the user to be redirected to another page.

```
from django.http import HttpResponseRedirect

from django.urls import reverse


def redirect_view(request):

    return HttpResponseRedirect(reverse('homepage'))
```

the view redirects the user to the URL named `'homepage'`.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Using reverse() with Arguments

If you need to generate a URL based on a named pattern and pass arguments dynamically, you can use the reverse() function. This is often used with HttpResponseRedirect.

```
from django.http import HttpResponseRedirect
from django.urls import reverse

def redirect_to_book_detail(request, book_id):
    url = reverse('book_detail', args=[book_id])
    return HttpResponseRedirect(url)
```

URLs

Uniform Resource Locators define how users access different parts of your web application.

They are mapped to specific views, which are Python functions or classes that process requests and return responses.

Django provides a flexible way to manage URLs through the URL dispatcher, which is defined in the urls.py file of Django project or application.

Object and a QuerySet

Object: An object is an instance of a model. It represents a single record (or row) in the database.

Example: A specific Faculty record such as “Anjali patel”.

QuerySet: A QuerySet is a collection of model instances (objects). It represents a collection of records (or rows) from the database.

Example: A collection of all students in the database or a collection of students filtered by a particular studentname.

Lazy Evaluation of QuerySets

A QuerySet is not executed until you actually access it (for example, when you iterate over it, convert it to a list, or use it in a template).

This allows Django to optimize database queries by combining them and reducing redundant database hits.

```
# The database is not queried yet
students = Student.objects.filter(subject="Python")

# The database is queried when the result is accessed
for student in students:
    print(student.fname)
```

QuerySet Caching

Once a QuerySet is evaluated, it is cached.

This means that if you reuse the same QuerySet (without modifying it), Django will not hit the database again but will return the cached results.

```
student= Student.objects.all()  # Query is executed here
student1= Student.objects.all()  # This will not hit the database; it
uses the cached result
```

Middleware

middleware is a way to process requests and responses globally before they reach the view (for requests) or before they are sent to the user's browser (for responses).

Think of middleware as a series of **filters** that sit between the request from the user and the response that goes back to the user. These filters can modify the request before it reaches the view, or modify the response before it is sent to the user.

PROJECTS : CLICK HERE

API

API (Application Programming Interface) in Django refers to a set of rules and protocols that allow different software applications to communicate with each other. In the context of Django, an API usually means an HTTP-based web service that allows clients (like web browsers, mobile apps, or other servers) to interact with a Django web application.

Django provides various ways to create APIs, and the most common approach for building APIs in Django is by using Django REST Framework (DRF), which provides tools for handling HTTP requests and responses, and for serializing data into JSON, XML, etc.

Django Rest Framework

Django REST Framework (DRF) is a powerful toolkit for building Web APIs in Django. It provides various ways to handle HTTP requests, including Function-Based Views (FBVs) and Class-Based Views (CBVs).

While DRF offers several ways to define views, Function-Based Views (FBVs) are one of the most common approaches for simpler and more explicit API handling.

Function Based Views

A Function-Based View (FBV) is simply a Python function that takes an HTTP request as an argument and returns an HTTP response. In Django REST Framework, these views are decorated with the @api_view decorator, which makes them capable of handling HTTP methods (such as GET, POST, PUT, DELETE).

Key Components:

- Request: The incoming HTTP request object.
- Response: The outgoing HTTP response object, usually in JSON format for APIs.
- Decorator: @api_view to specify the allowed HTTP methods.

DRF Setup

Install Django REST Framework:

```
#pip install djangorestframework
```

Add DRF to Installed Apps:

```
INSTALLED_APPS =
[
    # Other apps
    'rest_framework',
]
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

What is Serializer

A **serializer** is a class that allows complex data types (such as Django models or querysets) to be converted into native Python data types that can be easily rendered into JSON, XML, or other content types. Serializers also handle converting incoming data back into complex types (like Django models), and validating that data.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Types of Serializers in DRF

There are two main types of serializers in DRF:

1. Serializer class (manual serializer): A base class for creating custom serializers.

2. ModelSerializer class (automated serializer): A subclass of Serializer that automatically generates fields based on Django models.

Serializer class Example

```
from rest_framework import serializers


class BookSerializer(serializers.Serializer):
    title = serializers.CharField(max_length=100)
    author = serializers.CharField(max_length=100)
    published_date = serializers.DateField()
    price = serializers.DecimalField(max_digits=5, decimal_places=2)
```

ModelSerializer Example

```
from django.db import models

from rest_framework import serializers


# ModelSerializer for Book

class BookSerializer(serializers.ModelSerializer):

    class Meta:

        model = Book

        fields = ['id', 'title', 'author', 'published_date', 'price']
```

Click here for Example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

THANK YOU !!!