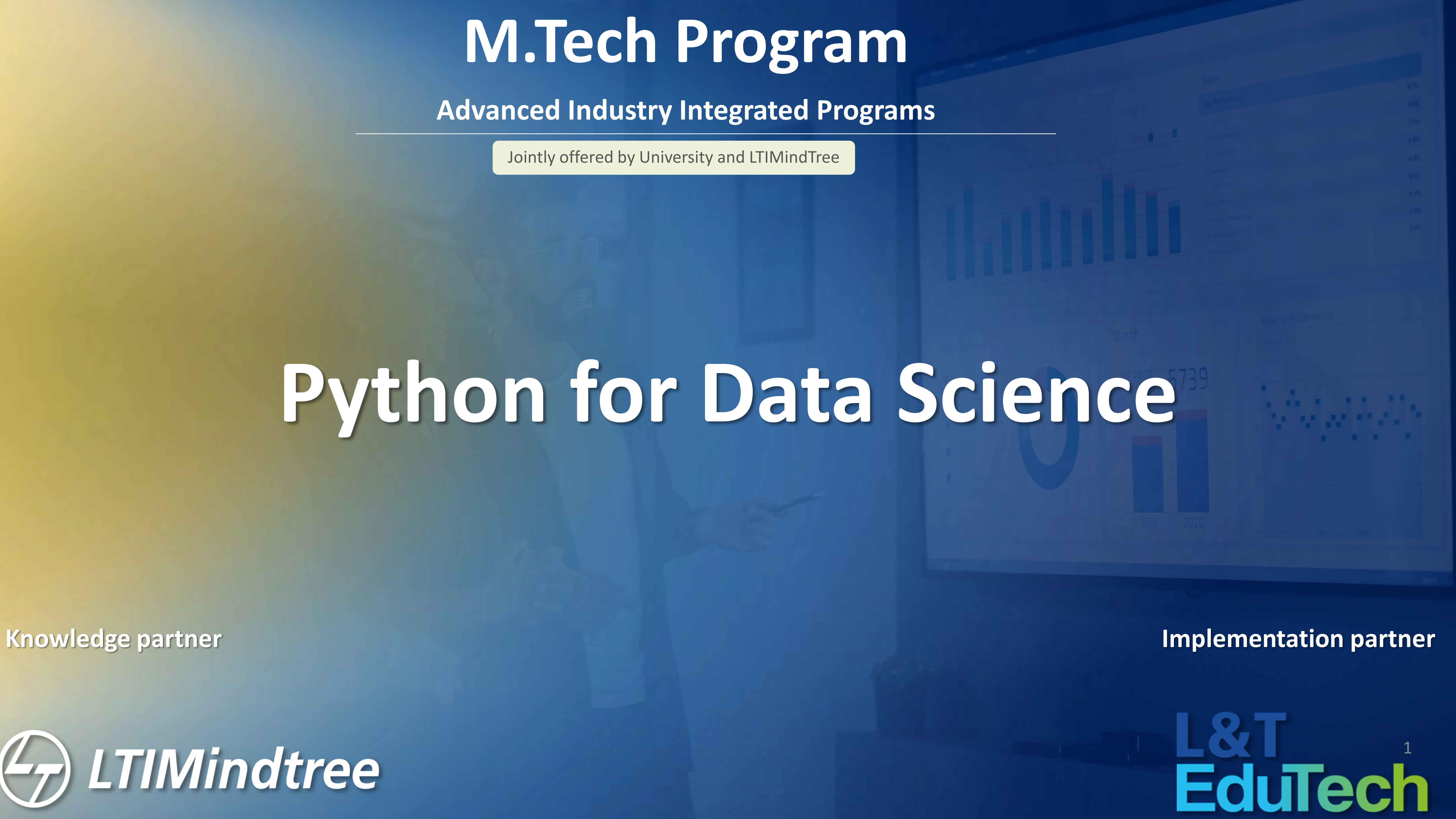
M.Tech Program

Advanced Industry Integrated Programs

Jointly offered by University and LTIMindTree



Python for Data Science

Knowledge partner

LTIMindtree

Implementation partner

L&T EduTech

Course Objective

- Understand Python data structures and OOP principles.
- Utilize NumPy and Pandas for data manipulation.
- Create visualizations using various Python libraries.
- Implement regression and classification algorithms effectively.
- Apply advanced unsupervised machine learning techniques.

Modules to cover

1. **Python - Data Structures, OOPS & Modules**
2. Python - Numpy, Pandas & DS Libraries
3. Visualization
4. Regression and Classification
5. Unsupervised and Advanced Machine Learning

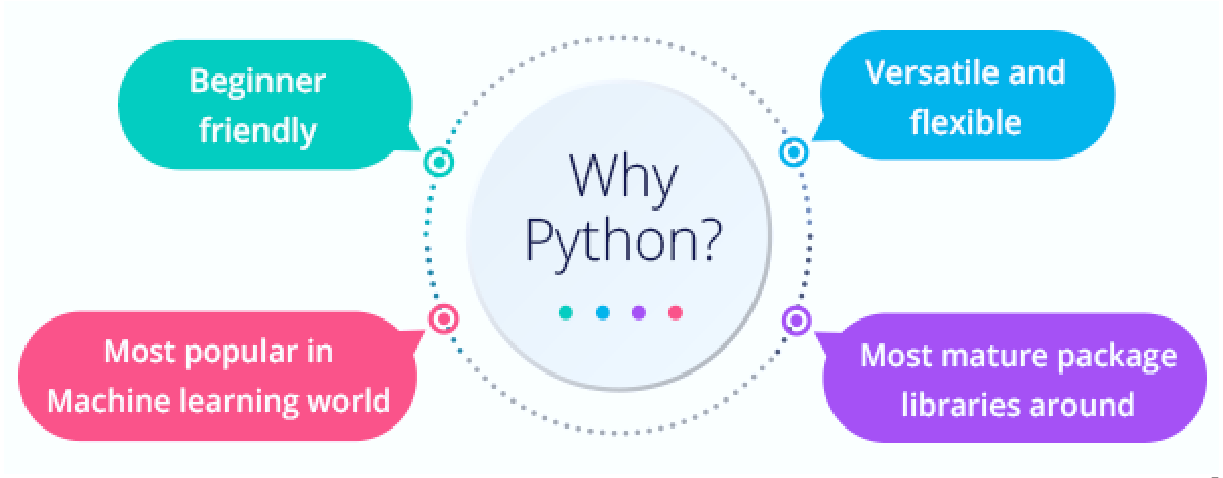
Python -Data Structures, OOPS & Modules

Introduction for Python

Python -Data Structures, OOPS & Modules

About Python

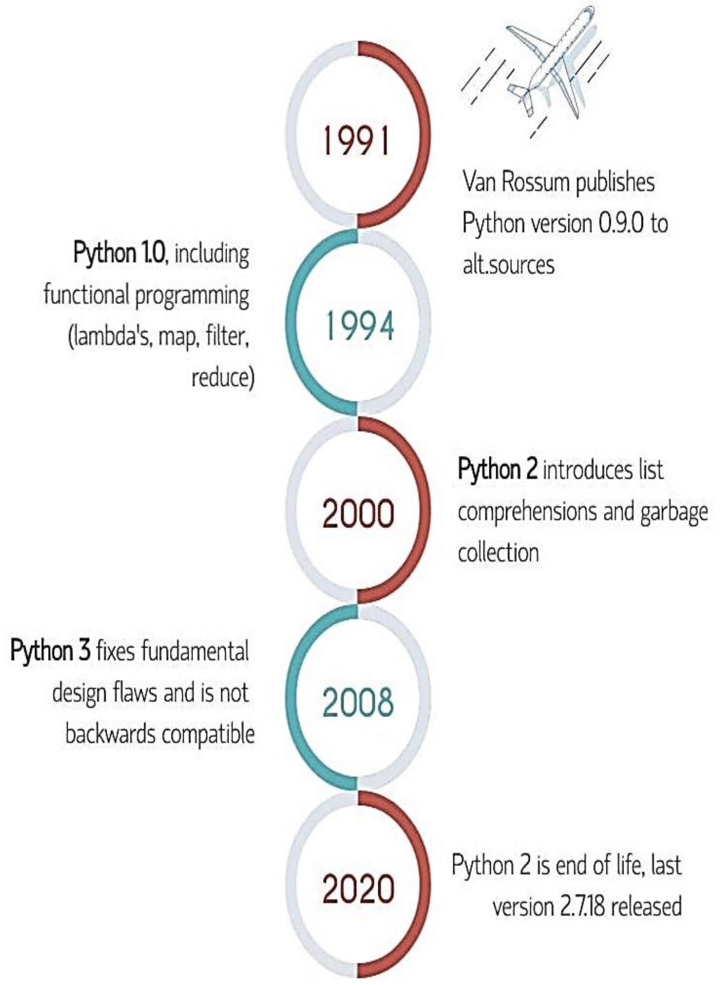
- High-level, interpreted programming language.
    - Designed to be easy to read and simple to implement.
        - Developed by Guido van Rossum and first released in 1991.


Why Python?
Beginner friendly
Versatile and flexible
Most popular in Machine learning world
Most mature package libraries around

Python -Data Structures, OOPS & Modules

History of Python

- **1980s:** Guido van Rossum started developing Python at Centrum Wiskunde & Informatica (CWI) in the Netherlands.
- **1991:** First released as Python 0.9.0.
- **2000:** Python 2.0 introduced new features like list comprehensions and garbage collection.
- **2008:** Python 3.0 released to rectify fundamental design flaws. Not backward compatible with Python 2.x.
- **2010s:** Python 2 was phased out, and Python 3 became the standard. Python's popularity surged with its use in web development, data science, artificial intelligence, and scientific computing.
- **2020:** Python 2 officially retired; ongoing development focused on Python 3.


1991
Van Rossum publishes Python version 0.9.0 to alt.sources
Python 1.0, including functional programming (lambda's, map, filter, reduce)
1994
2000
Python 2 introduces list comprehensions and garbage collection
Python 3 fixes fundamental design flaws and is not backwards compatible
2008
2020
Python 2 is end of life, last version 2.7.18 released

Python -Data Structures, OOPS & Modules

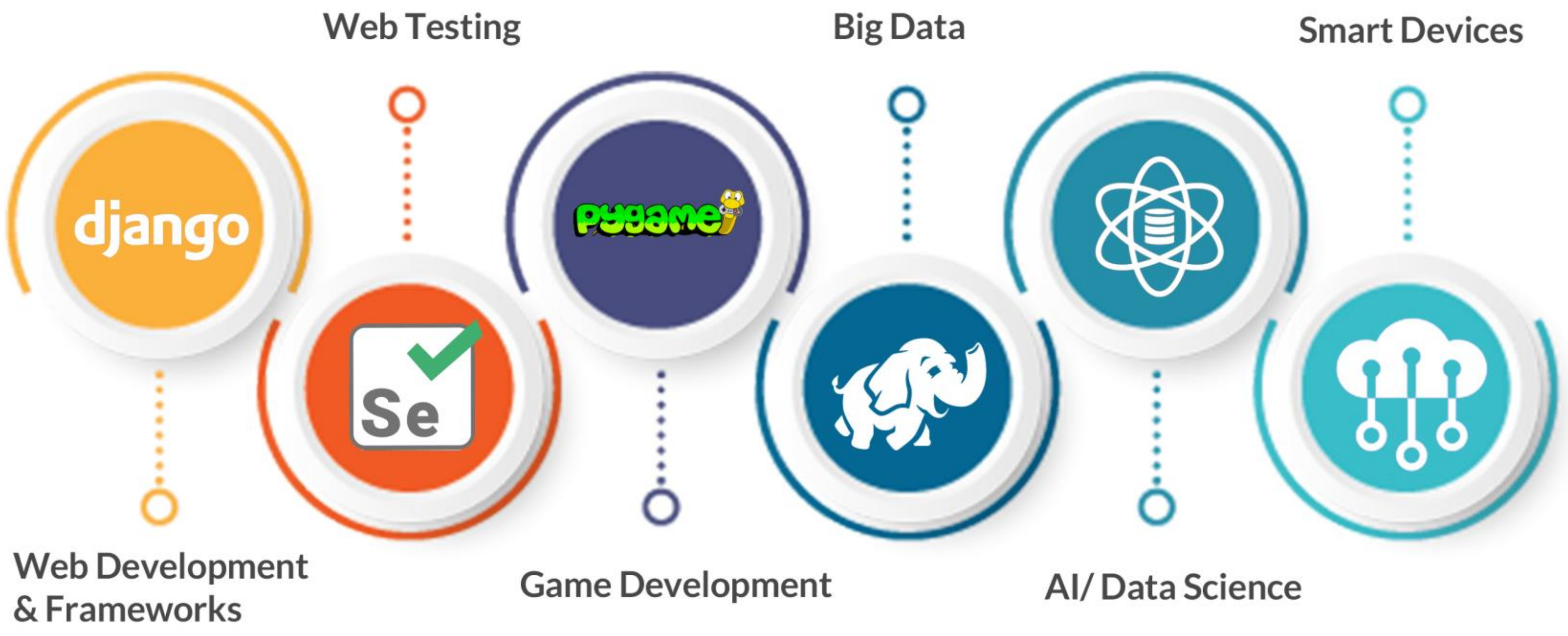
Key Features

• no compiling or linking	• rapid development cycle
• no type declarations	• simpler, shorter, more flexible
• automatic memory management	• garbage collection
• high-level data types and operations	• fast development
• object-oriented programming	• code structuring and reuse, C++
• embedding and extending in C	• mixed language systems
• classes, modules, exceptions	• "programming-in-the-large" support
• dynamic loading of C modules	• simplified extensions, smaller binaries

Python -Data Structures, OOPS & Modules

Python Framework

- Python provides extensive library support to create various applications.



Python -Data Structures, OOPS & Modules

Applications using Python

- Python supports to develop different applications in various fields.

- Web Development
- Game Development
- Machine Learning and Artificial Intelligence
- Data Science and Data Visualization
- Web Scraping Applications
- Desktop GUI Application

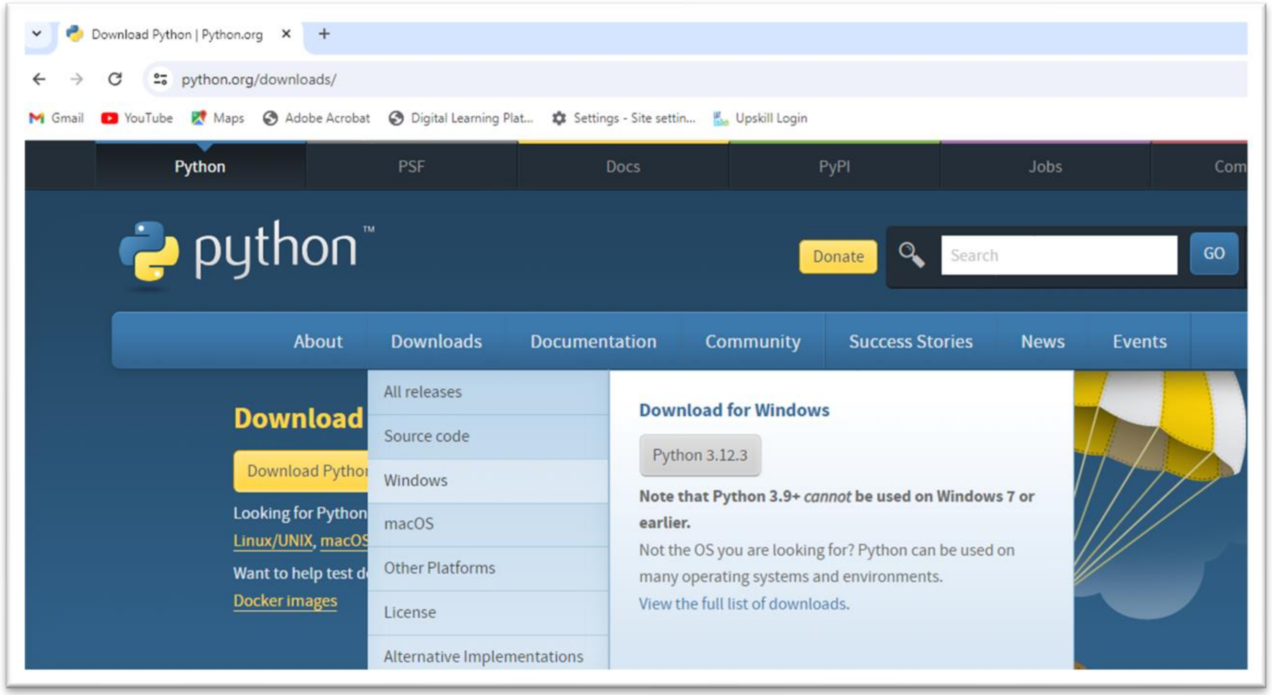
Dropbox
Instagram
Reddit
Spotify
Uber
Pinterest

Python Installation using Windows/Linux/Anaconda

Python -Data Structures, OOPS & Modules

Python Installation

- **Step 1: Download Python:**
  - Visit the official Python website at python.org.
  - You'll find the latest Python version available for download.
- **Step 2:Download Installer:**
  - Click on the download link to get the Python installer for your operating system (Windows, macOS, or Linux).
- **Step 3:Run the Installer:**
  - After downloading, locate the installer file and run it.
  - Follow the installation wizard's instructions.
  - Be sure to check the box that says "Add Python X.X to PATH" during installation.
- **Step 4:Install code Editor(Optional)**
  - While Python includes the IDLE development environment, many developers prefer using code editors like Visual Studio Code, PyCharm, or Jupyter Notebook for a more robust coding experience.
  - Download and install a code editor of your choice.



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.

Python Fundamentals

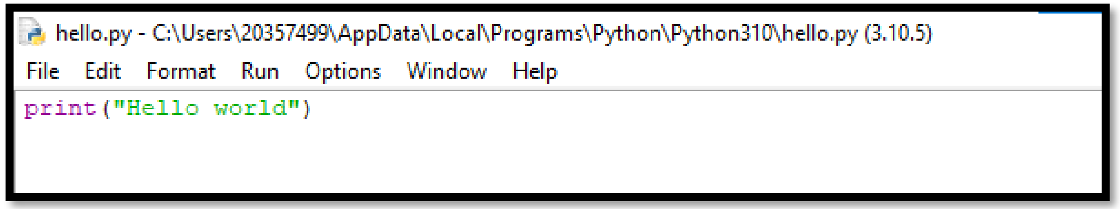
Python -Data Structures, OOPS & Modules

Python Fundamentals - Hello World Program

- **Step 1: Open Your Code Editor**:-
  - Open your chosen code editor
- **Step 2:Create New python File:-**
  - Create a new file and save it with a .py extension, e.g., hello_world.py
- **Step 3:Write & Save the Code**:-
  - print("Hello, World!")
- **Step 4: Run the program :-**
  - Python hello.py

How to display the print message?

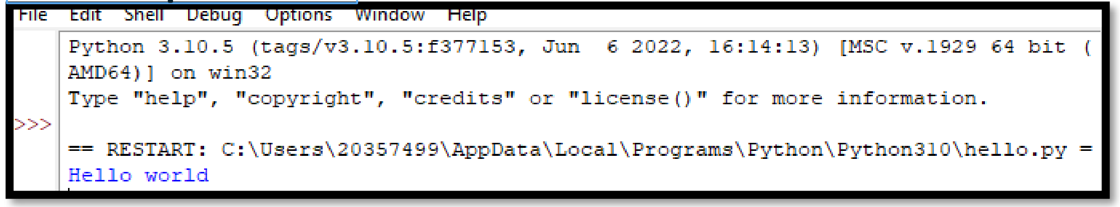
- hello.py



hello.py - C:\Users\20357499\AppData\Local\Programs\Python\Python310\hello.py (3.10.5)
File Edit Format Run Options Window Help

```
print("Hello world")
```

- output



```
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:\Users\20357499\AppData\Local\Programs\Python\Python310\hello.py =
Hello world
```

Python -Data Structures, OOPS & Modules

Variable Declaration

- Variables are storage area used to store data value.

- **Variable Declaration:-**
  - Ex:-

    Coursename="Python" , Number=10

    - **Variable is:-**
      - Coursename , Number
    - **The value is literals:-**
      - python,10

- **Variable Rules:-**
  - A-Z & a-z and numbers.
  - Special characters not allowed.
  - _ is allowed.
  - Keywords are not allowed.
  - White space not allowed.
  - Case Sensitive

Python -Data Structures, OOPS & Modules

Reserved Keywords

- Keywords are reserved words which has predefined meaning to do some actions.

and	exec	not
as	finally	or
assert	for	pass
break	from	print
class	global	raise
continue	if	return
def	import	try
del	in	while
elif	is	with
else	lambda	yield
except		

Python -Data Structures, OOPS & Modules

Data Types

- A data type is a classification that specifies the type of data a variable can hold.

Name	Type	Description
Integer	int	Whole numbers:- a=200, b=456
Floating Point	float	Decimal numbers:- Number=123.456
Complex	complex	Number with imaginary value:- number=12+j
String	str	Sequence of characters- Example:- name="John"
Boolean	bool	Logical value indicating True / False
Lists	list	Sequence of ordered items- numbers=[2,5,3,9,8]
Sets	set	Collection of unordered unique items:- numbers={2,4,5,1,7}
Tuples	tup	Ordered immutable sequence items- numbers=(1,4,2,5,8)
Dictionaries	dict	Key value pair list:- mydict={"name"="john", "age"=12}

Python -Data Structures, OOPS & Modules

Input Function

- Input():- This function will help to get input from the user.

- **Default input value is string.**
  - **Ex:- Name=input(“Enter the name”)**
    - **Taking input from the user as integer:**
      - **Ex:- Number=int(input(“Enter the number”))**
        - **Taking input from the user as float:**
          - **Ex:- Number=float(input(“Enter the number”))**
            - **Taking input from the user as complex:**
              - **Ex:- Number=complex(input(“Enter the number”))**

Python -Data Structures, OOPS & Modules

Operators

- Operators are mathematical symbols to perform Math/logical operations.

Arithmetic Operators

+,-,*,/,%,++,--

Ex:-
a=10+20;
a=a++ or a=100%2

Logical Operators

and, or, not

Ex:-
a=10;
a=a>10 || a<100

Assignment Operators

=,=+,=-,=*,=/,=%

Ex:-
a=10;
a=a+10 or a+=10

Comparison Operators

>=,==,<=,!=,>,<

Ex:-
a=10;
a=a>10 ; a<=10

Bitwise Operators

&,^,|,>>,<<,!

Ex:-
a=10;
a=a>>10 ; 10&20;

Python -Data Structures, OOPS & Modules

Knowledge Check

Python is the ……………………… Programming language

a) General Purpose

b) High level

c) Interpreted

d) All of the above

Answer:- d)

Python is the general purpose high level interpreted programming language

Python -Data Structures, OOPS & Modules

Knowledge Check

Which of the following is the correct syntax to print a message in Python?

a) print "Hello, World!“

b) echo "Hello, World!“

c) print("Hello, World!")

d) printf("Hello, World!")

Answer:- c)

print("Hello, World!")

Conditional & Branching Statements

Python -Data Structures, OOPS & Modules

Conditional & Looping Statements

Conditional Statements

- **Simple-If, If –Else,If-Else if –Else**
- **Nested If**

Conditional statements are used to instruct the computer to make the decision using given conditions.

Looping Statements

- **While**
- **For**
- **Break & Continue**

Looping statements are executed sequentially until a certain condition is reached.

Python -Data Structures, OOPS & Modules

Conditional Statements

- **Simple If**

```
if  condition
expression:
      statement(s)
```

```
num = 10

if  num > 0:
      print (“Positive number)

print (“This statement is outside condition)
```

- **Simple If else**

```
if  condition expression:
      statement(s) if condition
success
else:
      statement(s) if condition fails
```

```
num = 10

if  num > 0:
      print (“Positive number)
else:
      print(“Not a positive number)

print (“This statement is outside condition)
```

Python -Data Structures, OOPS & Modules

Conditional Statements

- **If…elif…else statement**

```
if condition1:
      # code block
 elif condition2:
      # another code block
else:
      # alternative code block
```

```
num = int("Please enter #:")

if  num > 0:
      print (“Positive number”)
elif num == 0
      print(“zero”)
else:
      print(“Negative number”)

print (“This statement is outside condition)
```

Python -Data Structures, OOPS & Modules

Iterative Statements

- ❑ **for loop**: Iterate over a sequence (like a list, tuple, or string)

```
for item in sequence:
    # code block
```

Note:- Range function is to generate sequence of numbers

- ❑ **while loop**: Execute a block of code as long as a condition is true.

```
while condition:
    # code block
```

# Example of for loop

```
for i in range(5):
    print(i)
```

# Example of while loop

```
count = 0
while count < 5:
    print(count)
    count += 1
```

Python -Data Structures, OOPS & Modules

Iterative Statements

- **break**: Exit the loop prematurely

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

- **continue**: Skip the rest of the code inside the loop for the current iteration

```
for item in sequence:
    if condition:
        continue
```

Python -Data Structures, OOPS & Modules

Knowledge Check Activity

- **Get a movie name and movie rating (0-5)**

Movie Rating	**Print the below**
Above 4.5	Super Hit
3.5 to 4.5	Good Movie
2.5 to 3.5	Average One time watch
Below 2.5	Flop

- **Get the purchase value and calculate the price after discount**

Amount	**Discount %**
Below 5000	5%
5001 to 10000	10%
10001 to 20000	15%
Above 20000	25%

Types of Data structure in Python

Python -Data Structures, OOPS & Modules

Built in Data Structures: Lists

- Lists are ordered sequence of item to perform list related operations.

- Lists are used to store multiple items in a single variable.
  - lists can be heterogeneous
    - a = ['spam', 'eggs', 100, 1234, 2*2]
  - Lists can be indexed and sliced:
    - a[0] → spam
    - a[:2] → ['spam', 'eggs']
  - Lists can be manipulated
    - a[2] = a[2] + 23
    - a[0:2] = [1,2]
    - a[0:0] = []
    - len(a) → 5

Python -Data Structures, OOPS & Modules

Built in Data Structures: List Method

Methods	Description
append()	Add an element to the end of the list
extend()	Add all elements of a list to another list
insert()	Insert an item at the defined index
remove()	Removes an item from the list
pop()	Removes and returns an element at the given index
clear()	Removes all items from the list
index()	Returns the index of the first matched item
count()	Returns the count of number of items passed as an argument
sort()	Sort items in a list in ascending order
reverse()	Reverse the order of items in the list
copy()	Returns a shallow copy of the list

Python -Data Structures, OOPS & Modules

Built in Data Structures: Tuples

- A tuple is a sequence of immutable objects like lists.

- The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.
- tup1 = ('physics', 'chemistry', 1997, 2000)
- tup2 = (1, 2, 3, 4, 5)
- tup3 = "a", "b", "c", "d"

Python -Data Structures, OOPS & Modules

Built in Data Structures: Dictionary

- Dictionaries are used to store data values in key: value pairs.

- A dictionary is a collection which is ordered ,changeable and do not allow duplicates.
- {} is used
- student = {

```
        “name”: “Rupesh”,
        “course”: “Python”,
        “mark”:95
}
print (student[“name”], student[“course”])
print(student.get(“age”))
```

Python -Data Structures, OOPS & Modules

Built in Data Structures: Set

- Python sets are unordered and unindexed items stored in single variable.

- Set can be heterogeneous
  - `a = {'spam', 'eggs', 100, 1234, 2*2}`
- Set can be unindexed:-
  - The set value is keep on changing while executing.
- Set creation:-
  - Myset=set() **# set is the keyword which is used to create new set.**
  - {}-Empty set.

Python -Data Structures, OOPS & Modules

Built in Data Structures: Set

Methods	Description
add()	Add an element to the set.
update()	Add all elements of a list/set into set.
remove()	Removes an item from the list
pop()	Removes and returns an element.
clear()	Removes all items from the list
discard()	Remove an item. Won't raise an error if item not exist.
count()	Returns the count of number of items passed as an argument
sort()	Sort items in ascending order
reverse()	Reverse the order of items in the list
copy()	Returns a shallow copy of the list

User Defined Data Structures in Stack, Queue, Priority Queue

Python -Data Structures, OOPS & Modules

User Defined Data Structures- Stack

- **Stack**: A linear data structure that follows the Last In, First Out (LIFO) principle.

- **Use Cases:**
  - **Function Call Management:** Managing function calls in recursion.
  - **Undo Mechanisms:** Implementing undo features in applications.
  - **Expression Evaluation:** Evaluating postfix, prefix expressions.

- **Implementation:**

```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop() if self.items
     else:
        None
```

Python -Data Structures, OOPS & Modules

User Defined Data Structures- Queue

- **Queue**: A linear data structure that follows the First In, First Out (FIFO) principle.

- **Use Cases:**
  - **Order Processing:** Managing tasks in multi-threaded environments.
  - **Breadth-First Search (BFS):** Implementing BFS in graph algorithms.
  - **Print Spooling:** Handling print jobs in a printer queue.

- **Implementation:**

```
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        return self.items.pop(0) if self.items else
None
```

Python -Data Structures, OOPS & Modules

User Defined Data Structures- Priority Queue

- **Priority Queue**:-
  - A data structure where each element has a priority.
  - Elements with higher priority are dequeued before elements with lower priority.

- **Implementation:**

```
import heapq

class PriorityQueue:
    def __init__(self):
        self.queue = []

    def push(self, item, priority):
        heapq.heappush(self.queue, (priority, 
item))

    def pop(self):
        return heapq.heappop(self.queue)[1]

    def is_empty(self):
        return len(self.queue) == 0
```

Python Strings and its method

Python -Data Structures, OOPS & Modules

Python Strings

- String is sequence or collection of characters. It specifies using within double quotes.

- **String Slicing:-**
  - Return range of string using index to create substring
    - Name[start: end]
    - 'Hello'[2] → 'l'
    - slice: 'Hello'[1:2] → 'el'
    - word[-1] → last character
    - Word[::] →All the characters
    - Word[-1::] →Reverse Characters

Python -Data Structures, OOPS & Modules

Python Strings and Its methods

- Python string methods is a collection of in-built Python functions that used for string manipulations.

Method	Description	Example
capitalize()	Capitalizes the first character of the string.	"hello world".capitalize() -> 'Hello world'
lower()	Converts all characters to lowercase.	"HELLO".lower() -> 'hello'
isalnum()	Returns True if all characters are alphanumeric.	"hello123".isalnum() -> True
isalpha()	Returns True if all characters are alphabetic.	"hello".isalpha() -> True
isascii()	Returns True if all characters are ASCII.	"hello".isascii() -> True
isdecimal()	Returns True if all characters are decimal.	"123".isdecimal() -> True
islower()	Returns True if all characters are lowercase.	"hello".islower() -> True
isnumeric()	Returns True if all characters are numeric.	"123".isnumeric() -> True
isupper()	Returns True if all characters are uppercase.	"HELLO".isupper() -> True

Python Function

Python -Data Structures, OOPS & Modules

Custom Function

- **Python Function** is a block of statements that return the specific task.

Function Declaration:-

```
def function_name(parameters):
        statement(s)
```

Example:-

```
def grt(name):
    """This function greets to the person passed in as parameter"""
print("Hello, " + name + ". Good morning!")


>>>grt("John")
Hello, John. Good morning!
```

Python -Data Structures, OOPS & Modules

Built in Functions – Using Math Module

- **import math:-** It allows us to perform mathematical tasks.

Methods	Description	Example
math.sqrt()	It returns square root of given number	Ex:- math.sqrt(25)

math.pi	It will give the pi value (3.141592653589793)	X=math.pi()
		Area=x*r^2

math.floor()	It returns rounds a number downwards to its nearest integer	math.floor(1.5)
math.factorial()	It returns a factorial number	math.factorial(5)
Math.fabs()	It returns absolute value of a number	math.fabs(-20)
Math.fmod()	It returns remainder value of x/y	math.fmod(10/4)

Python -Data Structures, OOPS & Modules

Built-in Functions -Datetime Module

- **Import datetime:-** It allows us to perform datetime activities

Methods	Description	Example
datetime.datetime.now()	It will display the current date	x = datetime.datetime.now()
datetime.datetime(2020, 5, 17)	It generates datetime object	X=datetime.datetime(2020, 5, 10)
strftime(%B), strftime(%b)	Month name, full version & Month name short version	x = datetime.datetime(2018, 6, 1)
strftime(%A), strftime(%a)	Week day, full version & week day, short version	x = datetime.datetime(2018, 6, 1)
Datatime.year()	It returns year for given year	x.Year()
strftime(%c)	It returns local version of data & time	x = datetime.datetime.now()

Python -Data Structures, OOPS & Modules

Built-in Functions- JSON Module

- **Import json:-** It allows us to perform tasks in Javascript object notation documents

Methods	Description	Example

json.loads()	Parsing the data using load method	x = '{ "name":"John", "age":20, "city":“India"}’
		y=json.loads(x)

json.dumps()	convert it into a JSON string by using the json.dumps() method	json.dumps()
json.dumps(x, indent)	Use the indent parameter to define the numbers of indents	json.dumps(x, indent=4)

Python File Handling & Exception Handling

Python -Data Structures, OOPS & Modules

File Handling

- `>>> f = open("test.txt")    # open file in current directory`
- `>>> f = open("C:/Python3/README.txt")  # specifying full path`

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

Python -Data Structures, OOPS & Modules

Exception Handling

Exception handling is a programming construct that allows a program to respond to and manage runtime errors, ensuring the program can continue to operate or terminate gracefully.

Instead of crashing abruptly, the program catches exceptions (errors) and executes alternative code to address the problem.

Python -Data Structures, OOPS & Modules

Print Exception in Python

Basic Exception Handling Use try and except blocks to handle exceptions.

Basic structure:

```
try:
    # Code that may raise an exception
except ExceptionType:
    # Code that runs if the exception occurs
```

Basic structure:

try: result = 10 / 0 except ZeroDivisionError as e: print(f"Caught an exception: {e}")

Example:-

```
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print(f"Caught an exception: {e}")
```

Python -Data Structures, OOPS & Modules

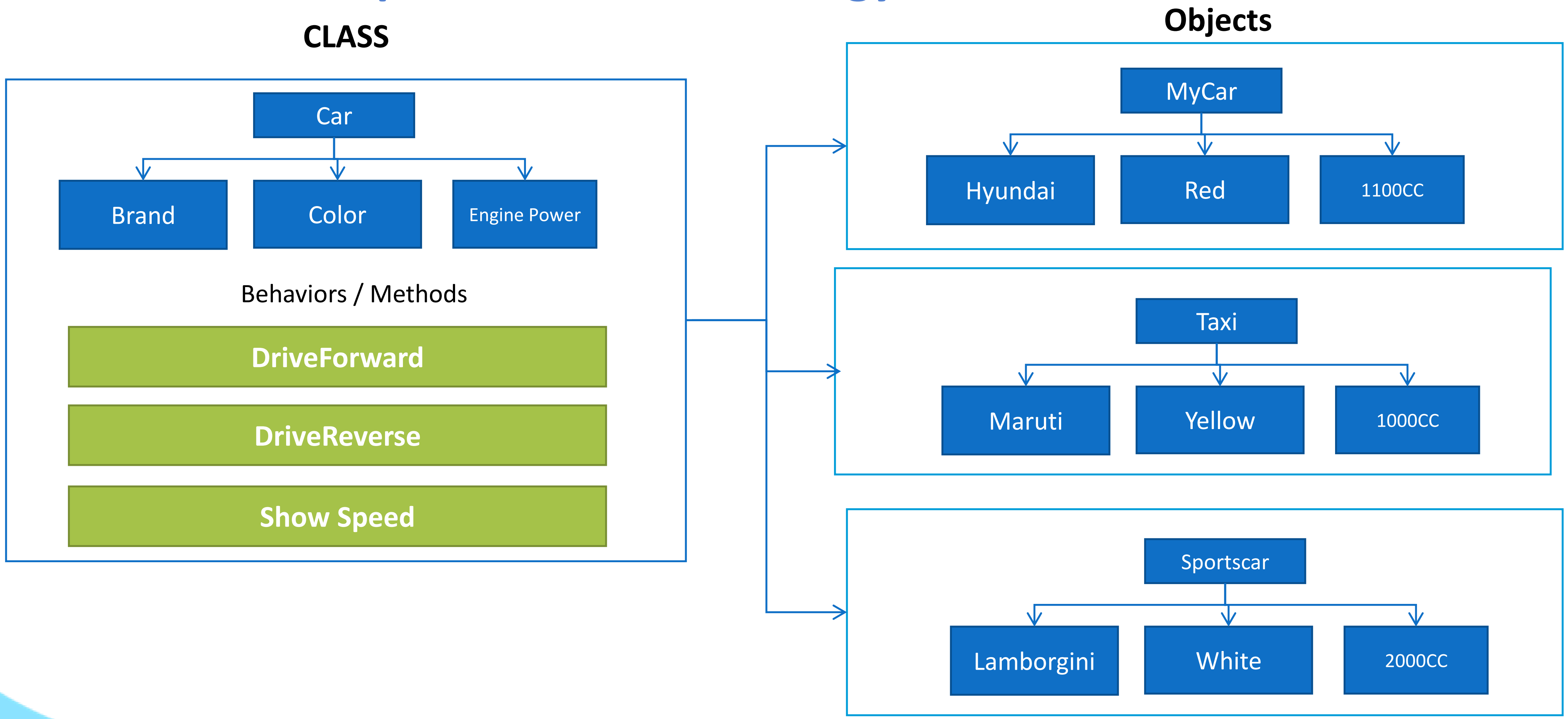
Object Oriented Programming

- **Object-oriented programming** (**OOP**) is a software programming model.
  - It allows us to think of the data in our program in terms of real-world **objects**, with both properties and behaviors.
  - These objects can be passed around throughout our program.

- **An object has two characteristics:**
  - Attributes / Properties define the state of the object.
  - This is the data that the object stores. This data can be a built-in type like int, or even our own custom types we'll create later.
  - Behaviors are the actions our object can take. Oftentimes, this involves using or modifying the properties of our object.

Python -Data Structures, OOPS & Modules

Define OO core concept and its Methodology



Python -Data Structures, OOPS & Modules

Objects and Classes

- **Create a class:-**
  - To create a class, use keyword **class**
  - **Syntax-** Class NewClass #class name
  - **Example:-**
    - class NewClass:
      x = 15
- **Create object:-**
  - p1 = NewClass()
    print(p1.x)
- **Object Methods:-**
  - Methods in objects are functions that belong to the object.

- **Example Program**

Class NewClass: **#"Here we created class"**

```
        X=25
   def myfunc(self) # function
        print("Hello")
   p1 = NewClass()
   print(p1.x)
   P1.x=40
   Print(p1.x)
   P1.myfunc()   #object methods
```

Python -Data Structures, OOPS & Modules

Polymorphism

- **Operator Overloading:** Customizing the behavior of Python's built-in operators for user-defined objects.
- **Purpose:** Extend the functionality of operators to work with user-defined data types.

Key Points:

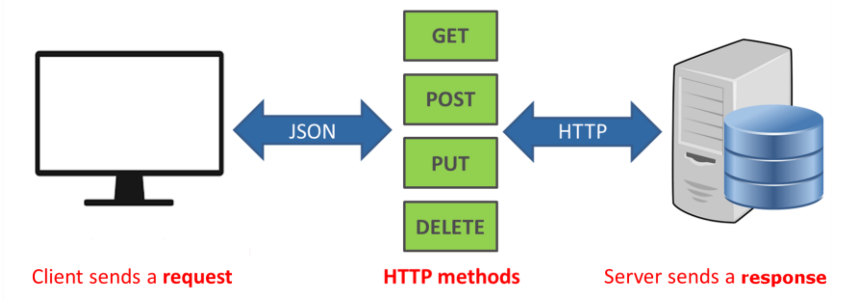
- Operators are linked to special methods (magic methods) in Python.
- Examples of magic methods: __add__, __sub__, __mul__, etc.
- Overriding these methods in custom classes enables specific behavior for operators.

Magic Method	Description
__init__(self, ...)	Constructor method, initializes a new instance of the class.
__str__(self)	Returns a string representation of the object, used by print() and str().
__repr__(self)	Returns an "official" string representation of the object, used by repr() and for debugging.
__add__(self, other)	Defines the behavior of the + operator.
__sub__(self, other)	Defines the behavior of the - operator.
__mul__(self, other)	Defines the behavior of the * operator.
__truediv__(self, other)	Defines the behavior of the / operator.

Python -Data Structures, OOPS & Modules

APIs and Data Collection

- **APIs (Application Programming Interfaces)**
  - **Definition**: Interfaces for building and interacting with software applications.
  - **Data Collection**: APIs enable the gathering of data from various sources programmatically.
- **Simple API – REST APIs & HTTP Requests**
  - **REST (Representational State Transfer)**: A standard architecture for designing networked applications.
  - **HTTP Requests**: GET, POST, PUT, DELETE are common HTTP methods used in REST APIs to perform CRUD operations.



Python -Data Structures, OOPS & Modules

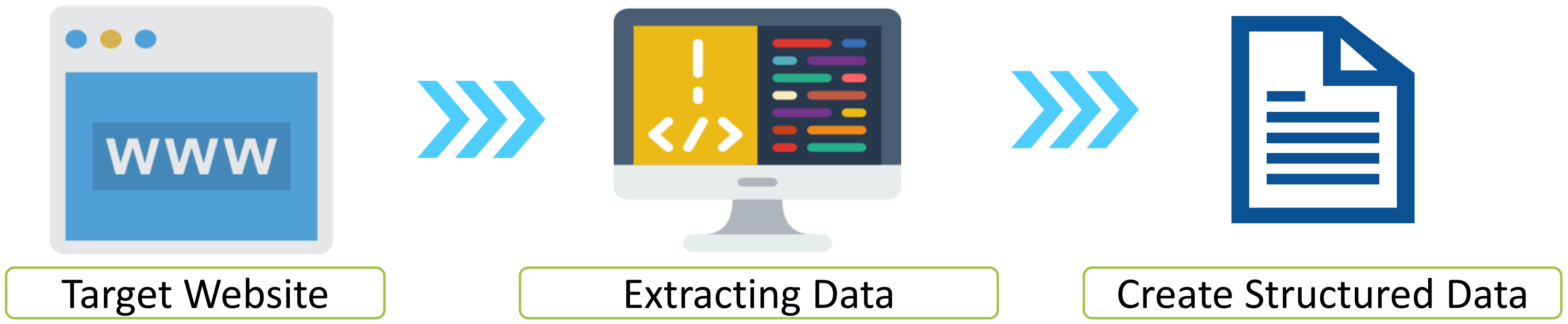
HTML for Web Scraping

- Web scraping is the automated process of extracting data from websites. This involves retrieving the HTML content of a web page and then parsing it to obtain the required information.
- HTML, or Hypertext Markup Language, is the standard markup language used to create and structure web pages. Understanding HTML is crucial for web scraping, as it allows you to navigate and extract data from web pages.

Python -Data Structures, OOPS & Modules

Web Scraping

- **About Web Scraping:-**
  - Scraping is simply a process of extracting (from various means), copying and screening of data.
  - Web scraping which is also known as web data extraction or web harvesting is the extraction of data from web.



Python -Data Structures, OOPS & Modules

Working with different file formats

- Common Libraries for File Handling:-
- Overview of key Python libraries: pandas, openpyxl, json, csv, xml.etree. ElementTree, etc.

File Format	Library/Module	Reading Method	Writing Method
CSV	pandas	pandas.read_csv('file.csv')	pandas.to_csv('file.csv', index=False)
	csv	csv.reader(open('file.csv'))	csv.writer(open('file.csv', 'w'))
Excel	pandas	pandas.read_excel('file.xlsx')	pandas.to_excel('file.xlsx', index=False)
	openpyxl	openpyxl.load_workbook('file.xlsx')	openpyxl.Workbook().save('file.xlsx')
JSON	json	json.load(open('file.json'))	json.dump(data, open('file.json', 'w'))
	pandas	pandas.read_json('file.json')	pandas.to_json('file.json')
XML	xml.etree.ElementTree	xml.etree.ElementTree.parse('file.xml')	xml.etree.ElementTree.ElementTree(root).write('file.xml')
Text	Built-in	open('file.txt').read()	open('file.txt', 'w').write(data)

Python -Data Structures, OOPS & Modules

Working with different file formats

File Handling CSV File

```
import pandas as pd
# Reading CSV
df = pd.read_csv('file.csv')
# Writing CSV
df.to_csv('file.csv', index=False)
```

```
import csv
# Reading CSV
with open('file.csv', newline='') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        print(row)

# Writing CSV
with open('file.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['Column1', 'Column2'])
    writer.writerow(['Value1', 'Value2'])
```

Python -Data Structures, OOPS & Modules

Web Framework

- Python web frameworks simplify web development by providing built-in features to handle aspects like routing, templating, database interactions, and security.

Python -Data Structures, OOPS & Modules

Web Framework

- **Django**
  Description: A high-level framework that encourages rapid development and clean, pragmatic design.
  Use Cases: Large-scale web applications, content management systems, e-commerce sites.

- **Flask**
  Description: A lightweight and flexible framework, ideal for small to medium applications.
  Use Cases: Microservices, APIs, small web applications.

- **FastAPI**
  Description: Modern, fast (high-performance) web framework for building APIs based on standard type hints.
  Use Cases: High-performance APIs, data science and machine learning applications.

Python -Data Structures, OOPS & Modules

Django

- **High-Level Framework**: Django is a high-level Python web framework that promotes rapid development and clean, pragmatic design.
- **Open Source**: It is open source and maintained by the Django Software Foundation (DSF).

- **Key Features**:-
  - **MVC Pattern**: Implements the Model-View-Controller (MVC) architectural pattern, though it refers to it as Model-View-Template (MVT).
  - **Admin Interface**: Comes with a powerful admin interface to manage application data.
  - **ORM**: Includes a robust Object-Relational Mapping (ORM) system to interact with databases.
  - **Security**: Offers built-in security features to protect against common web threats.

Python -Data Structures, OOPS & Modules

Hash Tables

- **Hash Tables**: Data structure that maps keys to values using a hash function.

- **Characteristics**:
  - **Fast Lookups**:
    - Average O(1) time complexity for insertions, deletions, and lookups.
  - **Collision Handling**:
    - Methods include chaining and open addressing.
- **Usage:**
  - Efficiently handles associative arrays, databases, and caches.

- **Implementation:**

```
# Hash table example in Python
hash_table = {}
hash_table['key'] = 'value'
print(hash_table['key’])


 # Output: value
```

Python -Data Structures, OOPS & Modules

Array Data Structures

- **Arrays**: A collection of items stored at contiguous memory locations.
- **Types**:
  - **One-Dimensional Array**: List of elements.
  - **Multi-Dimensional Array**: Array of arrays, like matrices.
- **Characteristics**:
  - **Fixed Size**: Predefined size and type of elements.
  - **Index-Based Access**: O(1) time complexity for accessing elements by index.

- **Implementation:**

```
# Array example in Python

array = [1, 2, 3, 4, 5]
print(array[2])

# Output: 3
```

Python -Data Structures, OOPS & Modules

Records

- **Definition**: Composite data type that groups together related data.

- **Characteristics**:
  - **Fields**: Named fields to store data of different types.
  - **Use Case**: Representing complex data structures like objects.
- **Example**:
  - **Python**: Uses classes or dictionaries to create records.

- **Implementation:**

```
# Record example using dictionary in Python
student = {"name": "Alice", "age": 20, "grade": "A"}
 print(student["name"])

# Output: Alice
```

Python -Data Structures, OOPS & Modules

Structs

- **Definition**: User-defined data type that groups fields of different types.

- **Characteristics**:
  - **Fixed Layout**: Fields of different data types grouped together.
  - **Efficient**: Allows memory-efficient storage of related data.
- **Use Case:**
  - Used for defining simple data structures in low-level programming.

- **Implementation:**

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

student1 = Student("Alice", 20, "A")
print(student1.name)

 # Output: Alice
```

Python -Data Structures, OOPS & Modules

Data Transfer Objects

- **Definition:**
  - Data Transfer Objects (DTOs) are objects that carry data between processes. They are used to encapsulate data and transfer it from one subsystem of an application to another.
- **Purpose:**
  - Reduce the number of method calls
  - Simplify complex data structures
  - Improve performance by aggregating multiple values into a single object

Python -Data Structures, OOPS & Modules

Data Transfer Objects

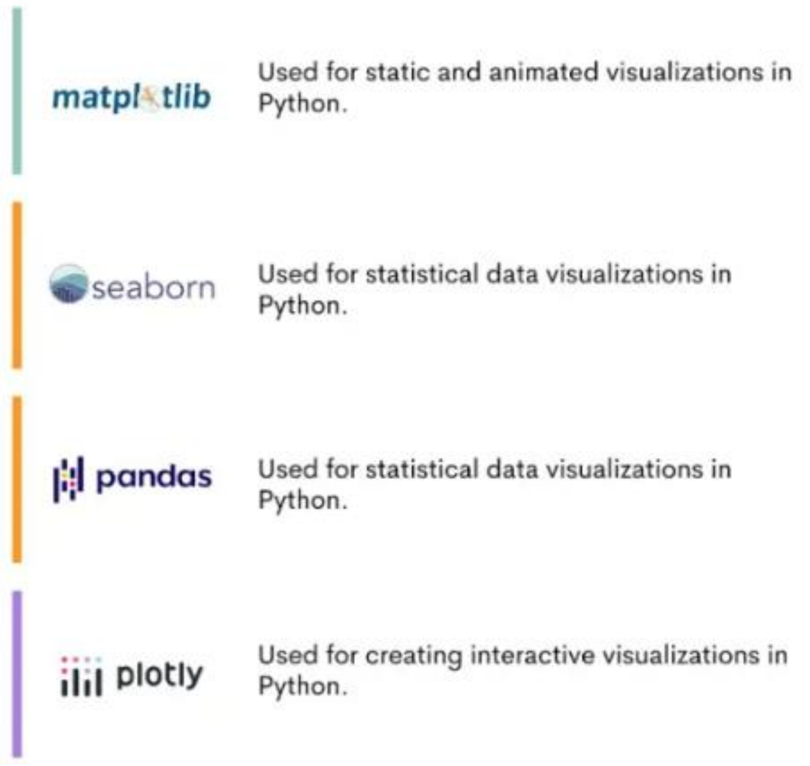
Example of Python -DTO

```
class UserDTO:
    def __init__(self, user_id, name, email):
        self.user_id = user_id
        self.name = name
        self.email = email

# Creating a DTO instance
            user_dto = UserDTO(user_id=1, name='John Doe', email='john.doe@example.com')

# Accessing DTO attributes

    print(user_dto.user_id)
    print(user_dto.name)
    print(user_dto.email)
```

Python -Data Structures, OOPS & Modules

Python Libraries for EDA

- **Pandas**: Data manipulation and analysis.
- **NumPy**: Numerical computing.
- **Matplotlib**: Basic plotting and visualization.
- **Seaborn**: Advanced visualization and statistical plots.
- **Plotly:** Creating interactive visualizations.
- **SciPy**: Scientific computing with Python.



matplotlib	Used for static and animated visualizations in Python.
seaborn	Used for statistical data visualizations in Python.
pandas	Used for statistical data visualizations in Python.
plotly	Used for creating interactive visualizations in Python.

Python -Data Structures, OOPS & Modules

Chatbot in Python

- **Purpose**:
  - Automate interactions using natural language processing and machine learning.
- **Key Steps**:
  - **Data Collection**: Gather conversational data.
  - **Preprocessing**: Clean and prepare text data.
  - **Model Training**: Use machine learning models to understand and generate responses.
  - **Deployment**: Integrate the chatbot into applications or services.

Python -Data Structures, OOPS & Modules

Chatbot in Python

Tools and Libraries:

- **NLTK**: Natural Language Toolkit for text processing.
- **SpaCy**: Industrial-strength NLP library.
- **Rasa**: Open-source framework for building conversational AI.
- **Transformers**: Library by Hugging Face for advanced language models

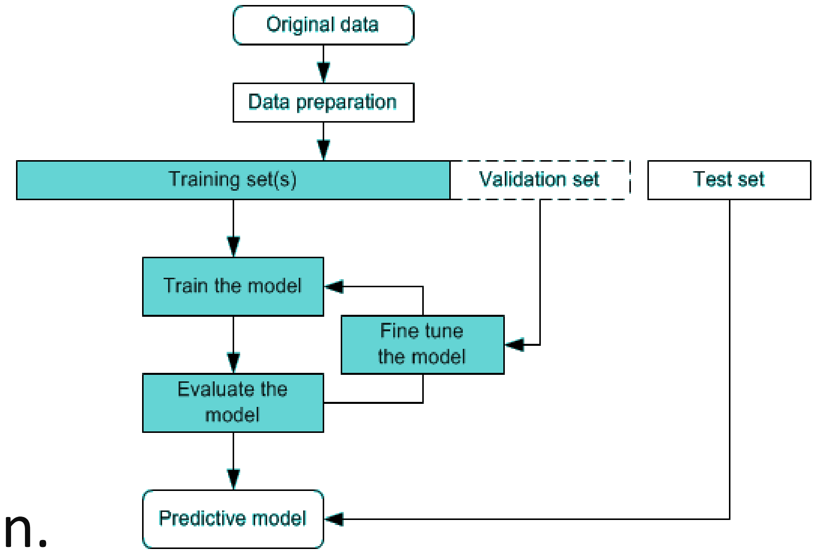
Python -Data Structures, OOPS & Modules

Machine Learning (ML) using Python:

A subset of artificial intelligence where systems learn from data to make decisions or predictions.

Python Libraries for ML

- **Scikit-Learn**: Comprehensive ML library for classical algorithms (classification, regression, clustering).
- **TensorFlow**: Open-source framework for deep learning.
- **Keras**: High-level neural networks API, running on top of TensorFlow.
- **PyTorch**: Deep learning library with strong GPU acceleration.



Python -Data Structures, OOPS & Modules

Exploratory Data Analysis in Python

- A process of analyzing datasets to summarize their main characteristics, often using visual methods.

Key Steps in EDA:-

1. **Data Collection**: Gather data from various sources.
2. **Data Cleaning**: Handle missing values, remove duplicates, and correct inconsistencies.
3. **Data Transformation**: Convert data types, create new features, normalize or scale data.
4. **Data Visualization**: Use plots and charts to visualize data distribution and relationships.
5. **Summary Statistics**: Calculate measures such as mean, median, standard deviation, and quantiles.

Python -Data Structures, OOPS & Modules

OpenCV Library in Python

OpenCV (Open Source Computer Vision Library): An open-source library that provides a comprehensive set of tools for computer vision and image processing.

Python -Data Structures, OOPS & Modules

OpenCV Library in Python

Key Features

- **Image Processing**: Functions for filtering, edge detection, and geometric transformations.
- **Video Analysis**: Tools for capturing, processing, and analyzing video streams.
- **Object Detection**: Pre-trained models and methods for detecting faces, eyes, and other objects.
- **Machine Learning**: Implements machine learning algorithms for computer vision tasks.
- **Real-Time Applications**: Optimized for real-time performance, leveraging hardware acceleration.

Python -Data Structures, OOPS & Modules

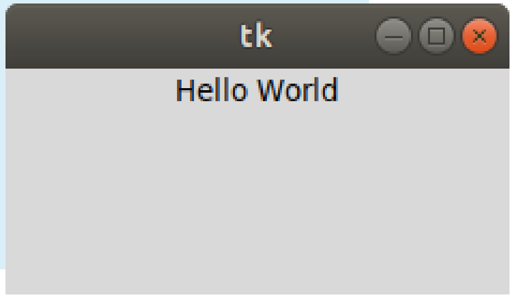
Tkinter – Pythons Turtle Module

Tkinter: Standard Python library for creating graphical user interfaces (GUIs).

Features:

- Widgets: Buttons, labels, text boxes, etc.
- Layout Management: Pack, grid, and place geometry managers.
- Event Handling: Respond to user actions
  - (e.g., clicks, key presses).

```
import tkinter as tk

def greet():
    print("Hello, World!")

root = tk.Tk()
root.title("My Tkinter App")

greet_button = tk.Button(root, text="Greet",
command=greet)
greet_button.pack()

root.mainloop()
```



Python -Data Structures, OOPS & Modules

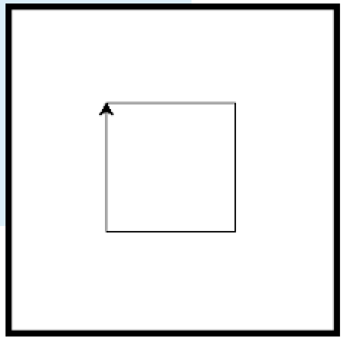
Tkinter – Pythons Turtle Module

Turtle: Module for drawing simple graphics and teaching programming concepts.

Features:

- Drawing: Move the turtle to draw shapes and patterns.
- Simple Commands: Forward, backward, left, right, etc.

```
import turtle

screen = turtle.Screen()
my_turtle = turtle.Turtle()

for _ in 4:
    my_turtle.forward(100)
    my_turtle.right(90)

screen.mainloop()
```



Python -Data Structures, OOPS & Modules

Summary:-

- A versatile, high-level programming language known for its readability and broad applications.
- Implementing conditional statements (if-else) and iterative statements (for, while loops) to control program flow.
- Implementing stacks, queues, and priority queues for custom data handling needs.
- Using REST APIs for data collection and web scraping techniques with HTML parsing to extract information from websites.
- Utilizing NumPy, Pandas, Matplotlib, Seaborn, and SciPy for data manipulation, statistical analysis, and visualization.
- Understanding machine learning models for predictive analysis using libraries.

Thank You

Happy Learning!