Python Basics:

**What is Python, and what are some of its key features that make it popular among developers? Provide examples of use cases where Python is particularly effective.**

**What is Python**

Python is a deciphered at an undeniable level broadly useful programming language. Python develops just as its methodology would mean to assist software engineers with composing, sensible code for little and huge scope projects. Python programming language was developed by Guido van Rossum during 1985–90. (Jaiswal & Dwivedi, 2021) .

**Features of python**

**Easy**

When we say python is easy, there are multiple contexts in which we can apply that. Some of which are: Easy to Code Python is very easy to code as compared to other languages like Java and C++, for example, if we write a hello world code in java vs. python [5]. Easy to Read: Python is also quite easy to read; and by reading, you can understand what this program is supposed to do (Jaiswal & Dwivedi, 2021).

**Open Source**

Python language is an open-source language. Because python is an open-source programming language; that means the source code is accessible to all. So anyone around the world can download and use it as well and can create new functions and contribute to the python community (Jaiswal & Dwivedi, 2021).

**Portability**

Python is a portable language. Therefore, python code can be operated evenly on cross-platforms like Windows, Linux, Macintosh, etc. Python is also highly flexible. That means if you have executed a python program on MAC, then that code can be executed through Windows, Linux, and vice versa (Jaiswal & Dwivedi, 2021) .

**Interpreted**Unlike other programming languages where the user has to compile the code and then run it, python offers execution of the code line by line. This makes it easy to debug any python code. The source code of python is converted to a direct type named byte-code. The benefit of a translated language is that it makes debugging portable and straightforward. So, python can operate on a vast array of hardware platforms and contains the same interface on all platforms (Jaiswal & Dwivedi, 2021).

**Object Oriented**

Python supports the object-oriented and functional programming concept which is one of the key features of the python programming language. It also supports concept of classes, object encapsulation; inheritance, polymorphism, and encapsulation. It also supports multiple inheritances, unlike Java (Jaiswal & Dwivedi, 2021).

**Examples of use cases where Python is particularly effective**

**Web Development**

When it comes to web development, python is the utmost choice of developers. It is because python offers various options for developing a web-based system, for example, Django, Pyramid, Flask, and Bottle for developing web frameworks. These frameworks also provide security, scalability, convenience while developing any web-based system (Jaiswal & Dwivedi, 2021) .

**Game Development**

Python is very common in the development of games. There are libraries, for example, PySoy which is a 3D game motor supporting Python 3, PyGame which gives usefulness, and a library for game turn of events. Other than game turn of events, fashioners can likewise utilize python for creating apparatuses to improve on explicit activities, for example, new level plan or new discourse tree creation, and even use that tool to export the task in formats that can be used by the primary game engine (Jaiswal & Dwivedi, 2021).

**Desktop GUI**

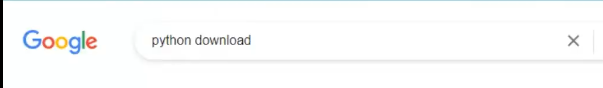
Python also offers many GUI toolkits and frameworks that help us make desktop application development a breeze. PyQt, PyGtk, Kivy, Tkinter, WxPython, PyGUI, and PySide are a portion of the python-based GUI systems that permit designers to make exceptionally practical Graphical User Interfaces (GUIs) applications (Jaiswal & Dwivedi, 2021).

**Installing Python:**

**Describe the steps to install Python on your operating system (Windows, macOS, or Linux). Include how to verify the installation and set up a virtual environment.**

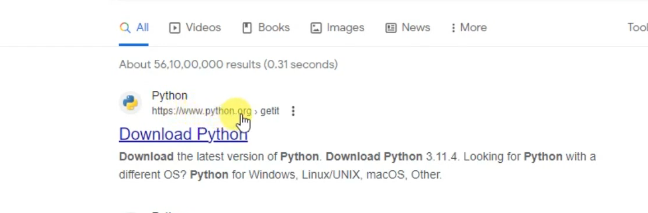
**Step 1:**

**Go to your favourite browser and type python download like so:**

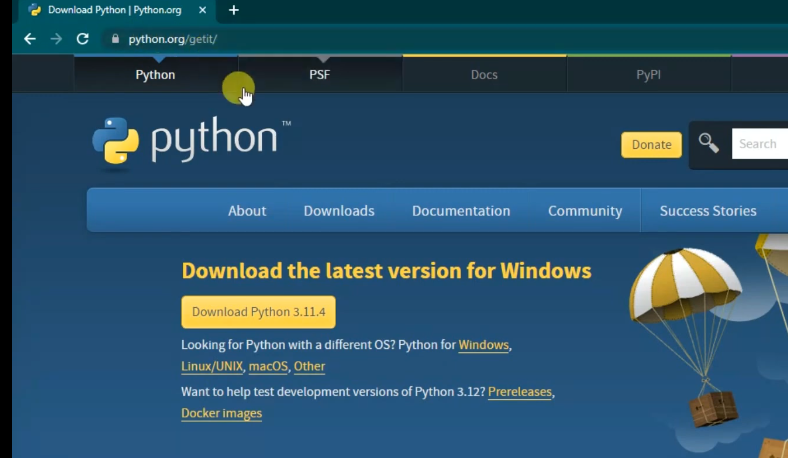


**Step 2**

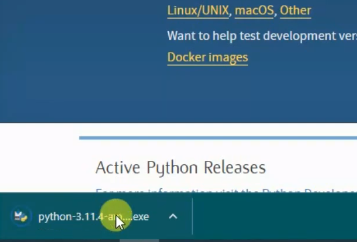
**Click on the first result that will take you to the official python download webpage. You will be landed to python.org page**:



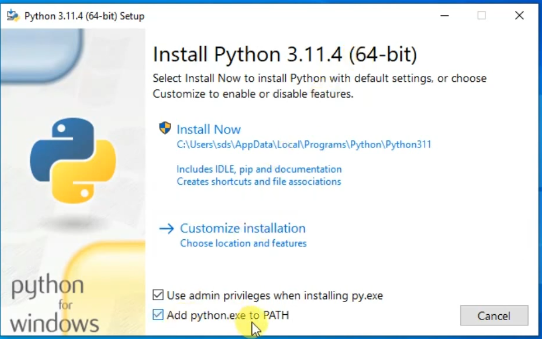
**Step 3: Click on the download button for your operating system. I am using Windows Python executable file download will start shortly after and it will take as long as your internet speed can accommodate.**



**Step 4: Once the download is complete, click on the installer to begin the installation**



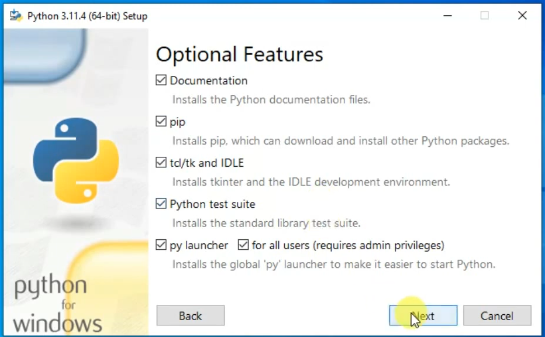
**Step 5: Add python.exe to environment variable once the following window pops up**



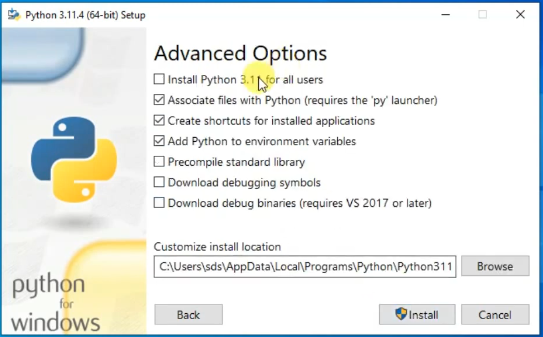
**Step 6: Click custom installation**

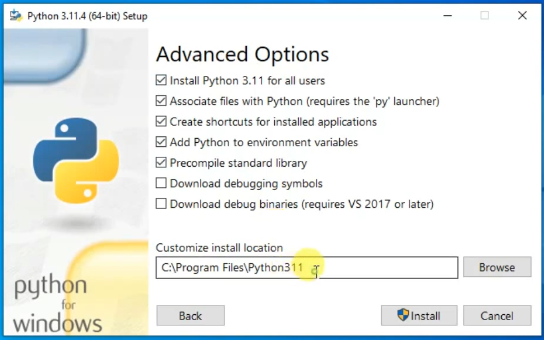


**Step 7: Check all the optional features if none have been checked and click on next**

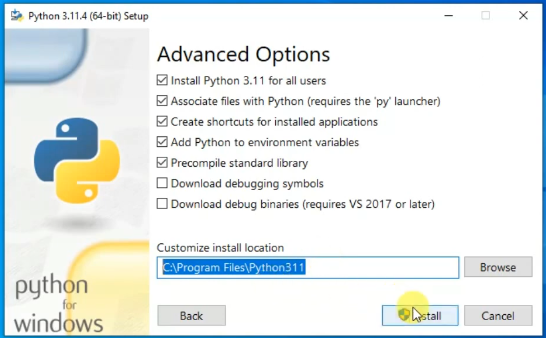


**Step 8: Click install python for all users so that install location is properly defined. Observe next slide for result of this step**

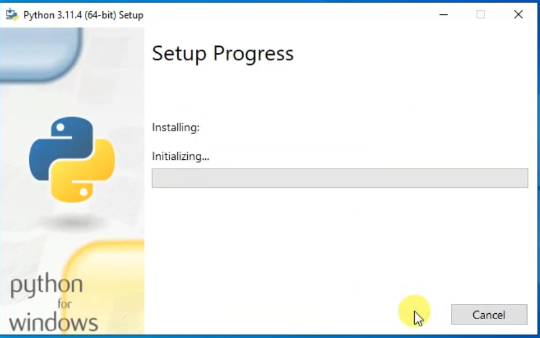




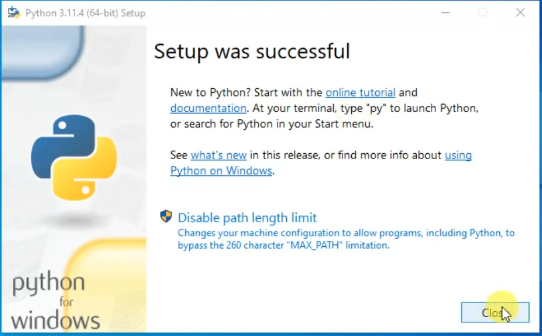
**Step 9: Click the install button to begin installation. At this step, you have properly configured custom settings and are satisfied with them:**



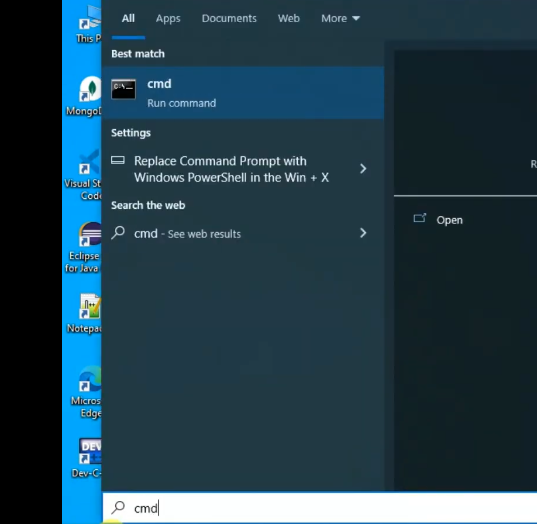
**Step 10: Wait for the installation to complete**



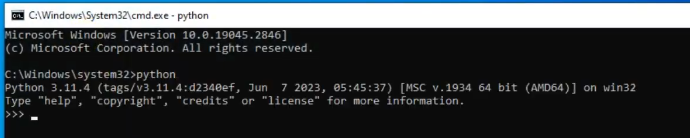
**Step 11: Python has succesfully completed installation. Click on close button to exit the setup**



**Step 12: Check whether python is properly installed into your operating system. Run command prompt**



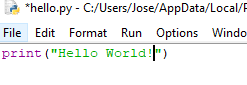
**Step 13: type ‘python’ and hit enter. You will see that command prompt displays the latest installed version of python**



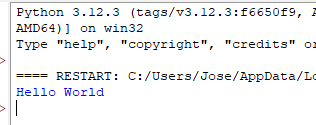
**Python Syntax and Semantics:**

**Write a simple Python program that prints "Hello, World!" to the console. Explain the basic syntax elements used in the program.**

Print(“Hello World!”)



I set “Hello World!” as a parameter on the print(), a python function which takes the text and prints it out to the console. The following is the result:



Data Types and Variables:

List and describe the basic data types in Python. Write a short script that demonstrates how to create and use variables of different data types.

**Integers (int):** Whole numbers without a fractional component. Examples: 5, -3, 42.

**Floating-point numbers (float):** Numbers that contain a decimal point or are in exponential (scientific) notation. Examples: 3.14, -0.001, 2.5e6.

**Strings (str):** Sequences of characters used to represent text. Strings are enclosed in single (') or double (") quotes. Examples: 'hello', "Python".

**Booleans (bool):** Logical values indicating True or False.

**Lists (list):** Ordered, mutable collections of items, which can be of any data type. Lists are defined by square brackets ([]). Examples: [1, 2, 3], ['apple', 'banana', 'cherry'].

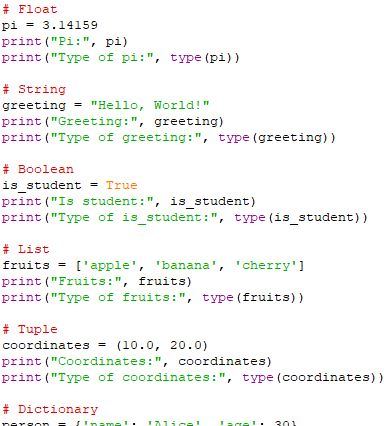
**Tuples (tuple):** Ordered, immutable collections of items. Tuples are defined by parentheses (()). Examples: (1, 2, 3), ('a', 'b', 'c').

**Dictionaries (dict):** Collections of key-value pairs. Keys are unique and typically strings or integers, while values can be of any data type. Dictionaries are defined by curly braces ({}). Examples: {'name': 'Alice', 'age': 25}, {1: 'one', 2: 'two'}.

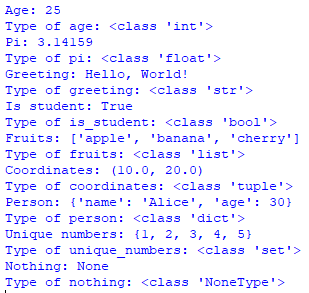
**Sets (set):** Unordered collections of unique items. Sets are defined by curly braces ({}) or the set() function. Examples: {1, 2, 3}, set(['a', 'b', 'c']).

**NoneType (None)**: A special data type that represents the absence of a value. Only one value of this type exists: None.

Script:



Output:



Control Structures:

**Explain the use of conditional statements and loops in Python. Provide examples of an if-else statement and a for loop.**

Conditional statements in Python allow you to execute different blocks of code based on certain conditions. The primary conditional statements in Python are if, elif, and else.

**If-Else Statement**

The if statement evaluates a condition. If the condition is true, the code block following the if statement is executed. The else statement follows an if statement and is executed if the condition in the if statement is false.

Here is an example of an if-else statement:

age = 18

if age >= 18:

print("You are an adult.")

else:

print("You are a minor.")

In this example:

If age is 18 or older, the program prints "You are an adult."

Otherwise, it prints "You are a minor."

**Else if in python**

You can also include multiple conditions using elif (short for else if):

score = 85

if score >= 90:

print("Grade: A")

elif score >= 80:

print("Grade: B")

elif score >= 70:

print("Grade: C")

else:

print("Grade: F")

**For loop in python**

A for loop is used to iterate over a sequence (like a list, tuple, dictionary, set, or string). The loop iterates over each item in the sequence, allowing you to perform operations with each item.

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

**Functions in Python:**

**What are functions in Python, and why are they useful? Write a Python function that takes two arguments and returns their sum. Include an example of how to call this function.**

Functions in Python are blocks of reusable code designed to perform a specific task. They help in organizing code, making it more readable, and reducing redundancy. Functions can take inputs, called arguments, and return outputs, making them versatile tools for a wide range of operations.

**Defining a Function in Python**

To define a function in Python, you use the def keyword followed by the function name and parentheses containing any parameters. Here’s an example:

def add\_numbers(a, b):

"""

This function takes two arguments, a and b,

and returns their sum.

"""

return a + b

**Example of How to Call the Function**

# Calling the function with arguments 3 and 5

result = add\_numbers(3, 5)

print(result) # Output: 8

**Full Example**

Here’s the complete code including the function definition and an example call:

# Defining the function

def add\_numbers(a, b):

"""

This function takes two arguments, a and b,

and returns their sum.

"""

return a + b

# Example of calling the function

result = add\_numbers(3, 5)

print(result) # Output: 8

Why Functions are Useful:

Reusability: Functions allow you to reuse code without rewriting it.

Modularity: Breaking down complex problems into smaller, manageable functions.

Readability: Functions make code easier to read and understand.

Maintainability: Easier to update and maintain code.

**Lists and Dictionaries:**

**Describe the differences between lists and dictionaries in Python. Write a script that creates a list of numbers and a dictionary with some key-value pairs, then demonstrates basic operations on both.**

**Lists:**

Ordered: Elements have a defined order, and this order will not change unless you explicitly modify it.

Indexed: Access elements by their position (index) in the list.

Mutable: You can modify elements, add new elements, and remove elements.

Duplicates: Allow duplicate values.

Syntax: Defined using square brackets [].

**Dictionaries:**

Unordered: Elements are stored as key-value pairs, and the order of elements is not guaranteed.

Keyed: Access elements by their key, not by index.

Mutable: You can modify values, add new key-value pairs, and remove pairs.

Unique Keys: Keys must be unique; however, values can be duplicated.

Syntax: Defined using curly braces {} with key-value pairs separated by colons.

**Script Demonstrating Basic Operations**

# Creating a list of numbers

numbers\_list = [1, 2, 3, 4, 5]

# Creating a dictionary with some key-value pairs

info\_dict = {

"name": "Alice",

"age": 30,

"city": "New York"

}

# Basic operations on the list

print("Original list:", numbers\_list)

numbers\_list.append(6) # Adding an element to the list

print("After appending 6:", numbers\_list)

numbers\_list.remove(3) # Removing an element from the list

print("After removing 3:", numbers\_list)

print("Element at index 2:", numbers\_list[2]) # Accessing an element by index

numbers\_list[1] = 10 # Modifying an element

print("After modifying index 1:", numbers\_list)

# Basic operations on the dictionary

print("\nOriginal dictionary:", info\_dict)

info\_dict["email"] = "alice@example.com" # Adding a new key-value pair

print("After adding email:", info\_dict)

del info\_dict["age"] # Removing a key-value pair

print("After removing age:", info\_dict)

print("Value for key 'city':", info\_dict["city"]) # Accessing a value by key

info\_dict["city"] = "Los Angeles" # Modifying a value

print("After modifying city:", info\_dict)

**Exception Handling:**

**What is exception handling in Python? Provide an example of how to use try, except, and finally blocks to handle errors in a Python script.**

**What is Exception Handling in Python?**

Exception handling in Python is a mechanism that allows you to manage errors that occur during the execution of a program. It helps to handle runtime errors gracefully, ensuring that the program can continue running or fail in a controlled manner without crashing abruptly.

**Key Components of Exception Handling:**

try block: Contains the code that might raise an exception.

except block: Catches and handles the exceptions.

finally block: Contains code that will be executed regardless of whether an exception was raised or not.

**Example of Using try, except, and finally Blocks**

def divide\_numbers(a, b):

try:

# Code that might raise an exception

result = a / b

except ZeroDivisionError as e:

# Handling the specific exception

print(f"Error: {e}. You can't divide by zero.")

result = None

except TypeError as e:

# Handling another specific exception

print(f"Error: {e}. Both inputs must be numbers.")

result = None

except Exception as e:

# Handling any other exceptions

print(f"An unexpected error occurred: {e}")

result = None

finally:

# Code that will run regardless of whether an exception occurred

print("Execution of the try-except block is complete.")

return result

# Example usage

num1 = 10

num2 = 0

print("Attempting to divide:", num1, "by", num2)

result = divide\_numbers(num1, num2)

print("Result:", result)

num1 = 10

num2 = "5"

print("\nAttempting to divide:", num1, "by", num2)

result = divide\_numbers(num1, num2)

print("Result:", result)

num1 = 10

num2 = 2

print("\nAttempting to divide:", num1, "by", num2)

result = divide\_numbers(num1, num2)

print("Result:", result)

**Modules and Packages:**

**Explain the concepts of modules and packages in Python. How can you import and use a module in your script? Provide an example using the math module.**

**Concepts of Modules and Packages in Python**

**Modules:**

A module is a single file (or files) that contains Python code. This code can define functions, classes, and variables that you can reuse across different programs.

Modules help organize code into manageable sections and promote code reuse.

Any Python file can be a module by saving it with a .py extension.

**Packages:**

A package is a collection of modules organized in directories that include a special \_\_init\_\_.py file.

The \_\_init\_\_.py file indicates that the directory should be treated as a package.

Packages can contain sub-packages, providing a hierarchical structure for organizing modules.

**Importing and Using a Module**

You can import and use modules using the import statement. Here are several ways to import a module:

1. Importing the entire module:

import module\_name

1. Importing specific functions, classes, or variables from a module:

from module\_name import name1, name2

1. Importing a module with an alias

import module\_name as alias

**Example Using the math Module**

The math module provides mathematical functions and constants. Here’s an example script that demonstrates how to use it:

# Importing the entire math module

import math

# Using functions from the math module

num = 16

sqrt\_num = math.sqrt(num)

print(f"The square root of {num} is {sqrt\_num}")

# Using a constant from the math module

pi\_value = math.pi

print(f"The value of pi is {pi\_value}")

# Importing specific functions from the math module

from math import factorial, pow

# Using the imported functions

num = 5

fact\_num = factorial(num)

print(f"The factorial of {num} is {fact\_num}")

base = 2

exp = 3

power\_result = pow(base, exp)

print(f"{base} to the power of {exp} is {power\_result}")

# Importing the math module with an alias

import math as m

# Using functions with the alias

cosine\_value = m.cos(m.pi / 4)

print(f"The cosine of pi/4 is {cosine\_value}")

**File I/O:**

**How do you read from and write to files in Python? Write a script that reads the content of a file and prints it to the console, and another script that writes a list of strings to a file.**

**Reading from and Writing to Files in Python**

Python provides built-in functions to read from and write to files. The key functions are open(), read(), write(), and close(). Additionally, the with statement is commonly used for file operations as it ensures that files are properly closed after their suite finishes, even if an exception is raised.

**Reading from a File**

To read the content of a file and print it to the console, you can use the following script:

# Script to read from a file and print its content

# Specify the file path

file\_path = 'example.txt'

# Open the file in read mode

with open(file\_path, 'r') as file:

# Read the content of the file

content = file.read()

# Print the content to the console

print(content)

**Writing to a File**

To write a list of strings to a file, you can use the following script:

# Script to write a list of strings to a file

# List of strings to write to the file

lines\_to\_write = [

"This is the first line.",

"This is the second line.",

"This is the third line."

]

# Specify the file path

file\_path = 'output.txt'

# Open the file in write mode

with open(file\_path, 'w') as file:

# Write each string in the list to the file

for line in lines\_to\_write:

file.write(line + '\n') # Add a newline character after each string

**References**

Jaiswal, A., & Dwivedi, A. (2021). Python: The Versatile Language—Recent Trends in Programming Languages. *RECENT TRENDS IN PROGRAMMING LANGUAGES*.