

## INTRODUCTION TO PYTHON:

**Answer the following questions based on the understanding of python programming**

### QUESTIONS

#### *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 practically effective.**

Python is a popular high-level interpreted programming language that is easy to learn and understand and enables developers to produce logical and straightforward code for both small and large-scale projects.

One of the key features that make Python popular is its simplicity. The language is designed to be easy to read and write, reducing the learning curve for beginners. Python is also highly versatile, supporting multiple programming paradigms, including procedural, object-oriented, and functional programming. Additionally, Python has a vast standard library and an active community, which provides a wealth of modules and frameworks that accelerate development.

Python is highly effective in various practical applications. In web development, frameworks like Django and Flask enable developers to build robust web applications quickly. In data science and machine learning, libraries such as Pandas, NumPy, and TensorFlow facilitate data manipulation and analysis. Python is also used in automation, where scripts can handle repetitive tasks, and in software testing, where tools like pytest help streamline the testing process. Moreover, Python is commonly used in scientific computing and artificial intelligence, demonstrating its wide-ranging utility in the tech industry.

#### *Installing python:*

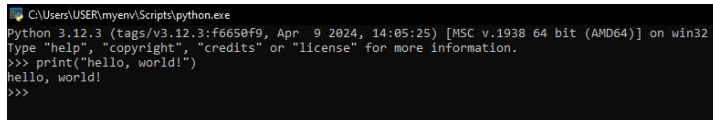
**Describe the steps to install python on your operating system. Include how to verify the installation and set up of a virtual environment.**

To install Python on your operating system, first, download the installer from the official Python website (python.org) that matches your OS (Windows). Run the installer and follow the prompts, ensuring you check the option to add Python to your system PATH. After installation, verify it by opening a command prompt or

terminal and typing `python --version` this should display the installed Python version. To set up a virtual environment, navigate to your project directory in the command prompt or terminal and run `python -m venv env`. Activate the virtual environment by running `.\env\Scripts\activate` on Windows. Your virtual environment is now ready for use, keeping project dependencies isolated.

### ***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.**

A screenshot of a terminal window with a black background and white text. The title bar at the top reads 'C:\Users\USER\myenv\Scripts\python.exe'. The terminal output shows the Python version 'Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32', followed by a prompt '>>>' and the command 'print("hello, world!")'. The output of the command is 'hello, world!' followed by another prompt '>>>'.

### **Function Call: print()**

Print is a built-in Python function used to output text to the console.

### **String: "hello, world!"**

The text within the quotes ("hello, world!") is a string, a sequence of characters.

Strings in Python can be enclosed in either single quotes (') or double quotes (").

### **Parentheses: ()**

The parentheses are used to enclose the arguments passed to the function. Here, "hello, world!" is the argument passed to the print function.

### ***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.**

### **Integer (int):**

Represents whole numbers, positive or negative, without a fractional part. Example: 5, -3.

### **Float (float):**

Represents numbers with a fractional part. Example: 3.14, -0.001.

## String (str):

Represents a sequence of characters. Strings are enclosed in single quotes (') or double quotes ("). Example: "hello", 'world'.

## Boolean (bool):

Represents one of two values: True or False.

```
C:\Users\USER\myenv\Scripts\python.exe
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Integer
>>> age = 21
>>> print("Age:", age)
Age: 21
>>>
>>> # Float
>>> height = 5.4
>>> print("Height:", height)
Height: 5.4
>>>
>>> # String
>>> name = "sharleen"
>>> print("Name:", name)
Name: sharleen
>>>
>>> # Boolean
>>> is_student = True
>>> print("Is student:", is_student)
Is student: True
>>>
```

## 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 most common conditional statements are if, elif, and else. Loops in Python are used to execute a block of code repeatedly. The most common loops are for and while.

```
C:\Users\USER\myenv\Scripts\python.exe
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> age = 21
>>>
>>> if age < 18:
...     print("You are a minor.")
... elif age >= 18 and age < 65:
...     print("You are an adult.")
... else:
...     print("You are a senior.")
>>>
```

```
C:\Windows\py.exe
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 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.**

Python functions are units of reusable code that carry out particular tasks. The function name, parentheses and the def keyword are used to define them (). Code can be better organized and made easier to read and maintain by using functions. They also make it possible to divide complicated issues into smaller, more manageable components, which facilitates modular development and code reuse.

```
C:\Users\USER\myenv\Scripts\python.exe
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Function definition
>>> def add_numbers(a, b):
...     """
...     This function takes two arguments and returns their sum.
...     """
...     return a + b
...
>>> # Calling the function with arguments 3 and 7
>>> result = add_numbers(3, 7)
>>>
>>> # Printing the result
>>> print("The sum is:", result)
The sum is: 10
>>>
```

### ***Lists and dictionaries:***

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

List is an ordered collection of items while dictionaries is an unordered collection of key-value pairs.

Lists maintains order of elements while dictionaries don't maintain order of elements.

Lists are accessed by index while dictionaries are accessed by key.

```
C:\Users\USER\myenv\Scripts\python.exe
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Creating a list of numbers
>>> my_list = [1, 2, 3, 4, 5]
>>>
>>> # Creating a dictionary with initial key-value pairs
>>> my_dict = {
...     "name": "Victor",
...     "age": 21,
...     "city": "Canada"
... }
>>>
>>> # Displaying initial list and dictionary
>>> print("Initial list:", my_list)
Initial list: [1, 2, 3, 4, 5]
>>> print("Initial dictionary:", my_dict)
Initial dictionary: {'name': 'Victor', 'age': 21, 'city': 'Canada'}
>>>
```

### ***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.**

Python's exception handling structure enables programmers to handle unexpected issues or exceptions that can arise while a program is running. Developers can

handle these problems gracefully, keeping the program from crashing and offering alternate actions or error messages, by utilizing try, except and possibly finally blocks.

```
C:\Users\USER\myenv\Scripts\python.exe
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Define a function to perform multiplication
>>> def multiply_numbers(a, b):
...     try:
...         result = a * b # Attempt multiplication
...     except ZeroDivisionError:
...         print("Error: Multiplication by zero is not allowed!")
...     else:
...         print(f"The result of {a} multiplied by {b} is: {result}")
...     finally:
...         print("Execution of multiply_numbers function completed.")
...
>>> # Test the function with different inputs
>>> multiply_numbers(5, 3) # Normal case
The result of 5 multiplied by 3 is: 15
Execution of multiply_numbers function completed.
>>> multiply_numbers(0, 7) # Multiplication by zero case
The result of 0 multiplied by 7 is: 0
Execution of multiply_numbers function completed.
>>> multiply_numbers(8, 4) # Normal case again
The result of 8 multiplied by 4 is: 32
Execution of multiply_numbers function completed.
>>>
```

## ***Modules and packages:***

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

A module is a file that contains declarations, functions and other Python code that may be imported and utilized by other Python scripts. Modules are essential components of code organization that facilitate code maintainability and reusability by separating related functions into distinct files. They give programmers the ability to organize their programs effectively, control dependencies, and add new or customized functionality to expand Python's capabilities. In order to access functions, classes, and constants defined within modules across various sections of a Python project, modules are imported using the import statement.

Packages in Python are directories that bring together similar code and include modules and a `\_\_init\_\_.py` file. They offer a mechanism to manage and arrange Python code into namespaces, facilitating development that is modular and scalable. Packages make it easier to maintain and collaborate across projects by promoting code reuse, avoiding naming conflicts and supporting vertical code structuring.

```
C:\Users\USER\myenv\Scripts\python.exe
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Importing the math module
>>> import math
>>>
>>> # Using functions from the math module with radius 8 and square root of 81
>>> radius = 8
>>> area = math.pi * radius ** 2 # Calculating the area of a circle with radius 8
>>> sqrt_value = math.sqrt(81) # Calculating the square root of 81
>>>
>>> # Displaying results
>>> print(f"Area of the circle with radius {radius} is: {area}")
Area of the circle with radius 8 is: 201.06102982974676
>>> print(f"Square root of 81 is: {sqrt_value}")
Square root of 81 is: 9.0
>>>
```

## *File I/O:*

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

To read from a file in Python, one: **Open the File:** Use the `open ()` function with the file path and mode ('r' for reading). **Read the Content:** Use methods like `read ()` to retrieve the contents of the file. **Close the File:** Always close the file using the `close ()` method to free up system resources.

To write to a file in Python, one: **Open the File:** Use the `open ()` function with the file path and mode ('w' for writing). **Write to the File:** Use methods like `write ()` to write data to the file. **Close the File:** Always close the file using the `close ()` method to save changes and free up system resources.

```
CaUsers\USER\myenv\Scripts\python.exe
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Writing to a file
>>>
>>> # List of strings to write to the file
>>> lines = [
...     "Hello,",
...     "This is a list of strings.",
...     "We are writing them to a file."
... ]
>>>
>>> # File path
>>> file_path = 'output.txt'
>>>
>>> # Step 1: Open the file for writing
>>> try:
...     file = open(file_path, 'w')
...
... File "<stdin>", line 3
...
... ^
SyntaxError: expected 'except' or 'finally' block
>>> # Step 2: Write each string from the list to the file
>>> for line in lines:
...     file "<stdin>", line 1
...     for line in lines:
IndentationError: unexpected indent
>>>         file.write(line + '\n')
...     File "<stdin>", line 1
...         file.write(line + '\n')
IndentationError: unexpected indent
>>>
>>> print(f"Successfully wrote {len(lines)} lines to '{file_path}'.")
...     File "<stdin>", line 1
...     print(f"Successfully wrote {len(lines)} lines to '{file_path}'.")
IndentationError: unexpected indent
>>>
```

```

C:\Users\USER\myenv\Scripts\python.exe
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> # Reading from a file and printing its content
>>>
>>> # Step 1: Open the file for reading
>>> file_path = 'concept.txt'
>>> try:
...     file = open(file_path, 'r')
...
File "<stdin>", line 3
^
SyntaxError: expected 'except' or 'finally' block
>>> # Step 2: Read and print the content
>>>     file_content = file.read()
File "<stdin>", line 1
    file_content = file.read()
IndentationError: unexpected indent
>>>     print("Content of the file:")
File "<stdin>", line 1
    print("Content of the file:")
IndentationError: unexpected indent
>>>     print(file_content)
File "<stdin>", line 1
    print(file_content)
IndentationError: unexpected indent
>>> except FileNotFoundError:
File "<stdin>", line 1
    except FileNotFoundError:
SyntaxError: invalid syntax
>>>     print(f"Error: File '{file_path}' not found.")
File "<stdin>", line 1
    print(f"Error: File '{file_path}' not found.")
IndentationError: unexpected indent
>>> finally:
File "<stdin>", line 1
    finally:
SyntaxError: invalid syntax

```

```

C:\Users\USER\myenv\Scripts\python.exe
^
SyntaxError: expected 'except' or 'finally' block
>>> # Step 2: Read and print the content
>>>     file_content = file.read()
File "<stdin>", line 1
    file_content = file.read()
IndentationError: unexpected indent
>>>     print("Content of the file:")
File "<stdin>", line 1
    print("Content of the file:")
IndentationError: unexpected indent
>>>     print(file_content)
File "<stdin>", line 1
    print(file_content)
IndentationError: unexpected indent
>>> except FileNotFoundError:
File "<stdin>", line 1
    except FileNotFoundError:
SyntaxError: invalid syntax
>>>     print(f"Error: File '{file_path}' not found.")
File "<stdin>", line 1
    print(f"Error: File '{file_path}' not found.")
IndentationError: unexpected indent
>>> finally:
File "<stdin>", line 1
    finally:
SyntaxError: invalid syntax
>>> # Step 3: Close the file
>>>     if 'file' in locals():
File "<stdin>", line 1
    if 'file' in locals():
IndentationError: unexpected indent
>>>         file.close()
File "<stdin>", line 1
    file.close()
IndentationError: unexpected indent
>>>

```

## References:

Introduction to machine learning with python. By Andreas C. Mueller and Sarah Guido published in 2016.

Deep Learning with python By Nikhil Ketkar published in 2017.