SE-Assignment-6

Assignment: Introduction to Python

Instructions:

Answer the following questions based on your understanding of Python programming. Provide detailed explanations and examples where appropriate.

Questions:

1. 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.

- Python is a high-level interpreted programming language

key features include:

- it is easy to learn and use
- it is versatile and powerful
- it has extensive libraries and frameworks
- offers community support

use cases:

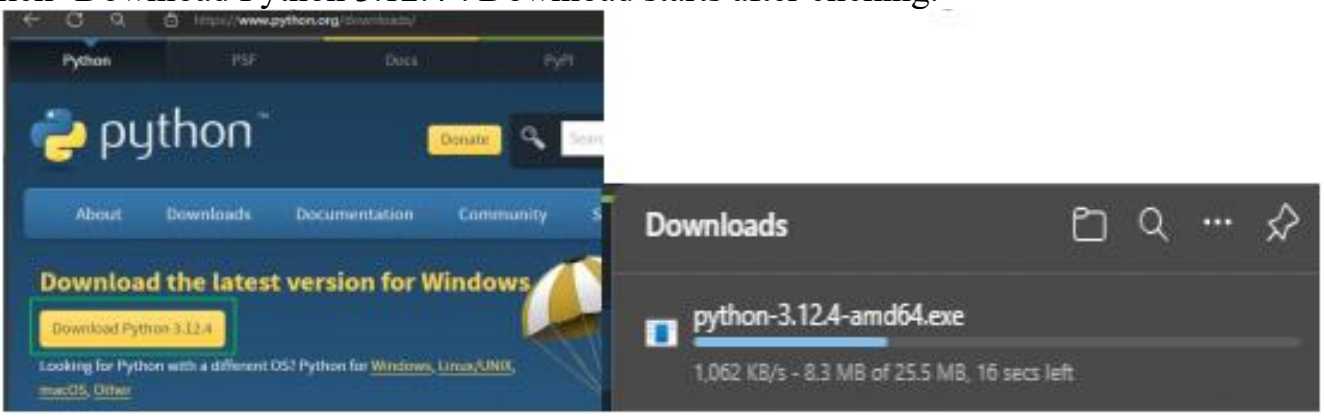
- web development
- data science and analytics
- machine learning and AI
- scientific computing

2. Installing Python:

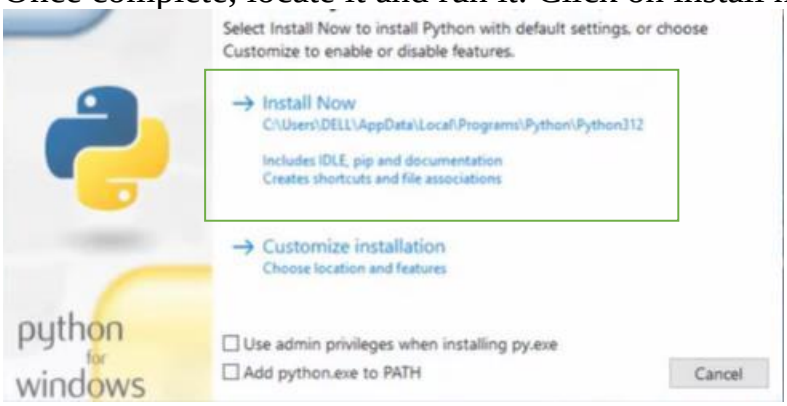
- Describe the steps to install Python on your operating system (Windows, macOS, or Linux). Include how to verify the installation and set up a virtual environment.

Click on this link to direct to Python website. https://www.python.org/downloads/

Click 'Download Python 3.12.4'. Download starts after clicking.

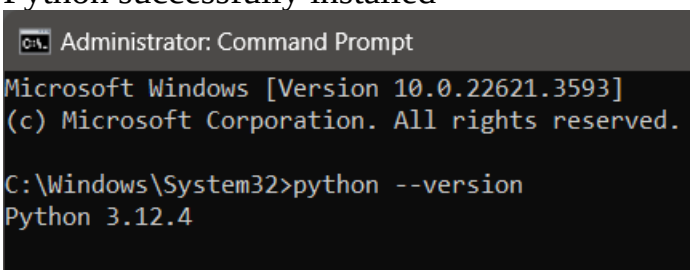


Once complete, locate it and run it. Click on install now after ticking these boxes



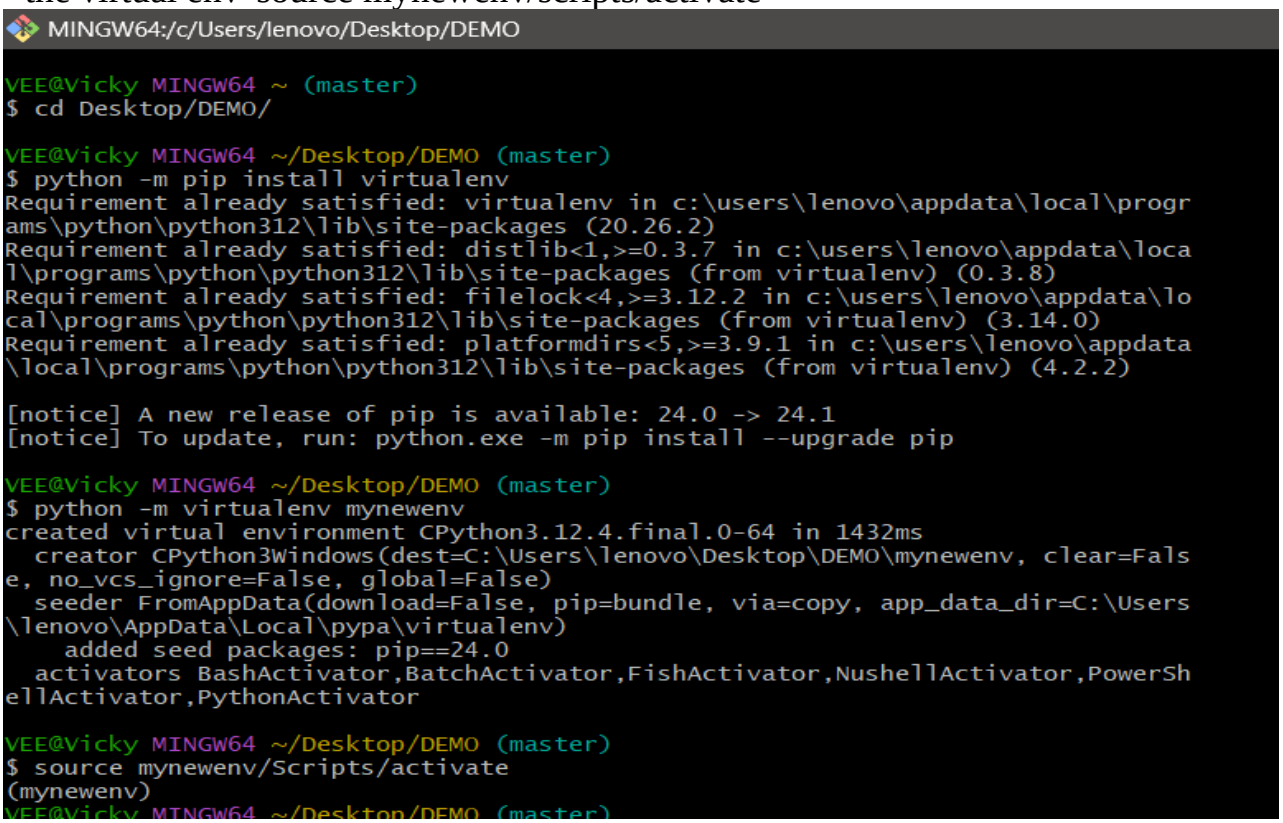
Run the CMD as admin and run this command ‘python --version’

Python successfully installed



setting up virtual environment

- run git bash as administrator
- navigate to a director
- install virtual env 'pip install virtualenv'
- create a virtual env 'python -m virtualenv mynewenv'
- the virtual env 'source mynewenv/scripts/activate'



3. 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.

```python
# Simple Hello, World! program in Python

# Define a variable containing the message
# define a variable 'message' and assign it the value'hello world'
message = "Hello, World!"

# Print the message
# use the 'print()' function to output content of the defined variable 'message'
print(message)
```

```
VEE@Vicky MINGW64 ~ (master)
$ C:/Users/lenovo/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/lenovo/Desktop/
Hello, World!
```

4. 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
- float 'float' - for real numbers with a decimal point
- string 'str' - for a sequence of characters
- boolean 'bool' - for true or false statements
- list 'list' - for an ordered collection off items

5. 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 allow one to execute a block of code based on the condition whether it is true or false. example:

```python
# Define a variable
age = 18

# Conditional statement
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

```
VEE@Vicky MINGW64 ~ (master)
$ C:/Users/lenovo/AppData/Local/Programs/Pythor
Simple Hello, World! program in Python.py"
You are an adult.
```

- loops allow one to execute a block of code repititively. example:

```python
# Define a list
numbers = [1, 2, 3, 4, 5]

# For loop to iterate over the list
for number in numbers:
    print("Number:", number)

```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PO

VEE@Vicky MINGW64 ~ (master)
$ C:/Users/lenovo/AppData/Local/Programs/Python/
Simple Hello, World! program in Python.py"
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

6. 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 are blocks of reusable code that perform a specific task. example:

```python
# Define the function to calculate the sum of two numbers
def add_numbers(a, b):
    """
    This function takes two arguments and returns their sum.
    """
    return a + b

# Example of how to call this function
num1 = 5
num2 = 10
result = add_numbers(num1, num2)

# Print the result
print(f"The sum of {num1} and {num2} is: {result}")

```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

VEE@Vicky MINGW64 ~ (master)
$ C:/Users/lenovo/AppData/Local/Programs/Python/Python312/python.exe "c:/
Simple Hello, World! program in Python.py"
The sum of 5 and 10 is: 15
```

7. 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 maintain the order of elements while dictionaries do not maintain order

```python
# Create a list of numbers
numbers = [1, 2, 3, 4, 5]

# Create a dictionary with key-value pairs
person = {
    'name': 'Alice',
    'age': 30,
    'city': 'New York'
}

# Basic operations on the list

# Append an element to the list
numbers.append(6)
print("List after appending 6:", numbers)

# Remove an element from the list
numbers.remove(3)
print("List after removing 3:", numbers)

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

# Basic operations on the dictionary

# Add a key-value pair to the dictionary
person['profession'] = 'Engineer'
print("Dictionary after adding profession:", person)

# Remove a key-value pair from the dictionary
del person['age']
print("Dictionary after removing age:", person)

# Access a value by key
print("Value for key 'name':", person['name'])

# Get all keys and values
print("Keys in the dictionary:", list(person.keys()))
print("Values in the dictionary:", list(person.values()))
```

```
VEE@Vicky MINGW64 ~ (master)
$ C:/Users/lenovo/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/lenovo/Desktop/DEMO/# Simple Hel
List after appending 6: [1, 2, 3, 4, 5, 6]
List after removing 3: [1, 2, 4, 5, 6]
Element at index 2: 4
Dictionary after adding profession: {'name': 'Alice', 'age': 30, 'city': 'New York', 'profession': 'Engineer'}
Dictionary after removing age: {'name': 'Alice', 'city': 'New York', 'profession': 'Engineer'}
Value for key 'name': Alice
Keys in the dictionary: ['name', 'city', 'profession']
Values in the dictionary: ['Alice', 'New York', 'Engineer']
```

8. 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.

- This is a mechanism that allows one to manage and respond to runtime errors in a controlled manner to prevent the program from crashing.

```python
def divide_numbers(a, b):
    try:
        # Try to perform the division
        result = a / b
    except ZeroDivisionError as e:
        # Handle the division by zero error
        print(f"Error: Cannot divide by zero. {e}")
        result = None
    except TypeError as e:
        # Handle the type error if inputs are not numbers
        print(f"Error: Invalid input type. {e}")
        result = None
    finally:
        # This block will always execute
        print("Execution of the try-except block is complete.")
    return result

# Example usage
num1 = 10
num2 = 0

# Call the function and print the result
print(f"Result of division: {divide_numbers(num1, num2)}")

num1 = 10
num2 = 2

# Call the function and print the result
print(f"Result of division: {divide_numbers(num1, num2)}")

num1 = 10
num2 = "a"

# Call the function and print the result
print(f"Result of division: {divide_numbers(num1, num2)}")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

VEE@Vicky MINGW64 ~ (master)
$ C:/Users/lenovo/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/lenovo
Error: Cannot divide by zero. division by zero
Execution of the try-except block is complete.
Result of division: None
Execution of the try-except block is complete.
Result of division: 5.0
Error: Invalid input type. unsupported operand type(s) for /: 'int' and 'str'
Execution of the try-except block is complete.
Result of division: None
```

9. 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.

- a package is a way of organizing related modules into a single directory
- a module is a single file with definitions and statements allowing organization of code into separate files thus ease of use and management. example:

```python
# Import the math module
import math

# 1. Calculate the square root of a number
num = 25
sqrt_result = math.sqrt(num)
print(f"The square root of {num} is {sqrt_result}")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

VEE@Vicky MINGW64 ~ (master)
$ C:/Users/lenovo/AppData/Local/Programs/Python/Python312/pytho
orld! program in Python.py"
The square root of 25 is 5.0
```

10. 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

to read the file:

- open the file using the open function 'open()'
- read the file contents using 'read()'
- close the file using 'close()'

to write the file
- open the file using the open function 'open()'
- write to the file using 'file.write("string")'
- close the file 'file.close()'

```
# Script to read the content of a file and print it to the console
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            print(content)
    except FileNotFoundError:
        print(f"Error: The file '{file_path}' does not exist.")

# Specify the path to the file you want to read
file_path = 'example.txt'
read_file(file_path)
```

```
# Script to write a list of strings to a file
def write_file(file_path, lines):
    try:
        with open(file_path, 'w') as file:
            for line in lines:
                file.write(line + '\n')
    except IOError as e:
        print(f"Error: Could not write to file '{file_path}'. {e}")

# Specify the path to the file you want to write to
file_path = 'output.txt'
lines_to_write = ["First line", "Second line", "Third line"]
write_file(file_path, lines_to_write)
```