SE-Assignment-6

Assignment: Introduction to Python

Questions:

1. ### Python Basics

What is Python?

Python is a high-level, interpreted programming language known for its readability and simplicity. Created by Guido van Rossum and first released in 1991, Python emphasizes code readability and allows developers to write clear, logical code for both small and large-scale projects.

Key Features of Python:

1. **Readability and Simplicity:** Python's syntax is designed to be readable and straightforward. This makes it easier for developers to understand and write Python code.

2. **Interpreted Language:** Python is an interpreted language, meaning that Python code is executed line by line, which simplifies debugging and allows for rapid prototyping.

3. **Extensive Standard Library:** Python has a rich standard library that supports many common programming tasks such as file I/O, system calls, and even web development.

Use Cases Where Python is Particularly Effective:

1. **Web Development**: Python frameworks such as Django and Flask make web development quick and efficient**.**

2. **Data Science and Machine Learning:** Libraries like NumPy, Pandas, Matplotlib, and scikit-learn make Python a powerful tool for data analysis and machine learning.

3. **Data Science and Machine Learning:** Libraries like NumPy, Pandas, Matplotlib, and scikit-learn make Python a powerful tool for data analysis and machine learning.

2. Installing Python

Windows

1. **Download Python:**
   - Go to the official Python website.
   - Click on the "Download Python" button. Ensure you download the latest version.
2. **Run the Installer:**
   - Open the downloaded installer.
   - Check the box "Add Python to PATH".
   - Click on "Install Now" or customize the installation as needed.
3. **Verify Installation:**
   - Open Command Prompt.
   - Type `python --version` and press Enter. You should see the installed Python version.
4. **Set Up a Virtual Environment:**
   - Open Command Prompt.
   - Navigate to your project directory using `cd path\to\your\project`.
   - Run `python -m venv venv` to create a virtual environment named `venv`.
   - Activate the virtual environment:

     ```
     bash
     Copy code
     venv\Scripts\activate
     ```

3. Python Syntax and Semantics

# This is a comment - anything after the "#" symbol is ignored

`print("Hello, World!")` # This line prints the string "Hello, World!" to the console

Basic syntax elements used in this program:

1. # : This symbol indicates a comment. Anything written after it on the same line is ignored by the interpreter.

2. print() : This is a built-in Python function that prints its argument to the console.

3. "Hello, World!" : This is a string literal, enclosed in double quotes. Strings can contain letters, numbers, and special characters.

4. () : These parentheses are used to enclose the argument(s) passed to the print() function.

4. Data Types and Variables

1. Integer (int)

- **Description:** Represents whole numbers without a fractional part.
- **Example:** `1, -2, 100`

2. Floating-Point Number (float)

- **Description:** Represents numbers with a fractional part, containing a decimal point.
- **Example:** `3.14, -0.001, 2.0`

3. String (str)

- **Description:** Represents a sequence of characters enclosed in single or double quotes.
- **Example:** `"Hello, World!", 'Python'`

4. Boolean (bool)

- **Description:** Represents one of two values: `True` or `False`.
- **Example:** `True, False`

5. List

- **Description:** Represents an ordered collection of items, which can be of different data types. Lists are mutable.

- **Example:** `[1, 2, 3], ["apple", "banana", "cherry"]`

6. Tuple

- **Description:** Represents an ordered collection of items, which can be of different data types. Tuples are immutable.
- **Example:** `(1, 2, 3), ("apple", "banana", "cherry")`

7. Dictionary (dict)

- **Description:** Represents a collection of key-value pairs. Keys must be unique and immutable.
- **Example:** `{"name": "Alice", "age": 25}`

8. Set

- **Description:** Represents an unordered collection of unique items.
- **Example:** `{1, 2, 3}, {"apple", "banana", "cherry"}`

Script Demonstrating Different Data Types

```
python
# Integer
age = 25
print(f"Age: {age}, Type: {type(age)}")

# Floating-Point Number
height = 5.9
print(f"Height: {height}, Type: {type(height)}")

# String
name = "Alice"
print(f"Name: {name}, Type: {type(name)}")

# Boolean
is_student = True
print(f"Is student: {is_student}, Type: {type(is_student)}")

# List
fruits = ["apple", "banana", "cherry"]
print(f"Fruits: {fruits}, Type: {type(fruits)}")

# Tuple
coordinates = (10.0, 20.0)
print(f"Coordinates: {coordinates}, Type: {type(coordinates)}")

# Dictionary
person = {"name": "Alice", "age": 25}
print(f"Person: {person}, Type: {type(person)}")

# Set
unique_numbers = {1, 2, 3, 2, 1}
```

```
print(f"Unique numbers: {unique_numbers}, Type:
{type(unique_numbers)}")
```

5. Control Structures.

Conditional Statements

Conditional statements allow you to execute code based on certain conditions. The most common conditional statement is the if-else statement.

Syntax:

```
if condition:
    # Code to execute if the condition is true
elif another_condition:
    # Code to execute if the another condition is true
else:
    # Code to execute if all conditions are false
```

```
age = 18

if age < 18:
    print("You are a minor.")
elif age == 18:
    print("You just became an adult.")
else:
    print("You are an adult.")
```

Loops

Loops allow you to repeat a block of code multiple times. The most common loops in Python are the `for` loop and the `while` loop.

For Loop

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

Syntax:

```
for item in sequence:
      # Code to execute for each item

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

```
for fruit in fruits:
    print(fruit)
```

6. Functions in Python:

Functions are reusable blocks of code that perform a specific task. They are useful for:

Modularity: Breaking a program into smaller, manageable, and reusable pieces.

Readability: Making the code easier to read and understand.

Avoiding Redundancy: Reducing code duplication by reusing the same block of code.

Maintainability: Simplifying debugging and updating code.

Function: Sum of Two Number

Function Definition:

```
def add_numbers(a, b):
       return a + b
```

Calling of Function:

```
# Calling the function with arguments 5 and 3
result = add_numbers(5, 3)
print(result)  # Outputs: 8
```

7. Lists and Dictionaries

Differences Between Lists and Dictionaries in Python

Lists

- Definition: Lists are ordered collections of items, which can be of any data type. Lists are mutable, meaning their elements can be changed after creation.
- Syntax: Lists are defined using square brackets `[]`.
- Accessing Elements: Elements are accessed by their index, starting from 0.

Dictionaries

- Definition: Dictionaries are collections of key-value pairs. They are unordered and mutable. Keys must be unique and immutable, while values can be of any data type.
- Syntax: Dictionaries are defined using curly braces `{}`.
- Accessing Elements:** Elements are accessed by their keys.

Script

Creating a List and a Dictionary

~~~
```python
 Creating a list of numbers
numbers = [1, 2, 3, 4, 5]

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

Basic Operations on List

1. Accessing elements by index

```
print("First number in the list:", numbers[0])  # Outputs: 1
```

2. Modifying an element

```
numbers[1] = 20
print("Modified list:", numbers)  # Outputs: [1, 20, 3, 4, 5]
```

3. Adding an element

```
numbers.append(6)
print("List after appending 6:", numbers)  # Outputs: [1, 20, 3, 4, 5,
6]
```

4. Removing an element

```
numbers.remove(20)
print("List after removing 20:", numbers)  # Outputs: [1, 3, 4, 5, 6]
```

Basic Operations on Dictionary

1. Accessing values by keys

```
print("Name:", person["name"])  # Outputs: Alice
```

2. Modifying a value

```
person["age"] = 31
```

```
print("Modified dictionary:", person)  # Outputs: {'name':
'Alice','age': 31, 'city': 'New York'}
```

3. Adding a key-value pair

```
person["occupation"] = "Engineer"
print("Dictionary after adding occupation:", person)
# Outputs: {'name': 'Alice', 'age': 31, 'city': 'New York',
'occupation':'Engineer'}
```

4. Removing a key-value pair

```
del person["city"]
print("Dictionary after removing city:", person)
# Outputs: {'name': 'Alice', 'age': 31, 'occupation': 'Engineer'}
```

8. Exception Handling

Exception handling in Python allows you to manage errors and exceptional conditions that occur during the execution of a program. It enables you to run a specific block of code (the try block) and catch and handle exceptions (errors) using the except block. The finally block can be used to execute code regardless of whether an exception was raised or not.

Script:

```
def divide_numbers(a, b):

    try:

        # Attempt to divide two numbers

        result = a / b

    except ZeroDivisionError as e:

        # Handle the specific case where division by zero is
attempted

        print(f"Error: Cannot divide by zero. {e}")

        result = None

    except TypeError as e:

        # Handle the case where non-numeric values are used

        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
```

```
# Test cases
print(divide_numbers(10, 2))  # Outputs: 5.0
print(divide_numbers(10, 0))  # Outputs: Error message and None
print(divide_numbers(10, "a"))  # Outputs: Error message and None
```

9. Modules and Packages

Modules

Definition: A module is a file containing Python definitions and statements. It allows you to organize your code into manageable parts.

Purpose: Modules promote code reuse, improve maintainability, and allow for better organization of your code.

Packages

Definition: A package is a collection of modules organized in directories that include a special __init__.py file. This file distinguishes a directory as a package.

Purpose: Packages provide a hierarchical structuring of the module namespace, making it easier to manage large codebases.

Using the math Module

The math module provides access to mathematical functions and constants.

Import the math Module:

python

import math

Use Functions from the math Module:

python

# Calculate the square root of 16

```
    sqrt_result = math.sqrt(16)

    print(f"Square root of 16: {sqrt_result}")  # Outputs: 4.0
```

# Calculate the sine of pi/2

```
    sine_result = math.sin(math.pi / 2)

    print(f"Sine of pi/2: {sine_result}")  # Outputs: 1.0
```

# Use the constant pi

```
    print(f"Value of pi: {math.pi}")  # Outputs: 3.141592653589793
```

# Calculate the factorial of 5

```
    factorial_result = math.factorial(5)

    print(f"Factorial of 5: {factorial_result}")  # Outputs: 120
```

10. File I\O

Reading from a File

To read from a file, you use the open() function with the mode 'r' (read mode), and then use methods like read(), readline(), or readlines().

Script to Read File Content

```
# Script to read the content of a file and print it to the console

# Open the file in read mode
with open('example.txt', 'r') as file:
    # Read the entire content of the file
    content = file.read()
    # Print the content to the console
    print(content)
```

Writing to a File

To write to a file, you use the open() function with the mode 'w' (write mode) or 'a' (append mode), and then use the write() or writelines() methods.

Script to Write List of Strings to a File

```
# Script to write a list of strings to a file

# List of strings to write to the file
lines = ["Hello, World!", "Python is great.", "File handling in Python."]

# Open the file in write mode
with open('output.txt', 'w') as file:
    # Write each string in the list to the file, followed by a newline character
    for line in lines:
        file.write(line + '\n')
```