SE-ASSIGNMENT 6

1. Python Basics

Python is a high-level, interpreted programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability and a clean, easy-to-understand syntax, which makes it accessible to both beginners and experienced developers.

Key Features of Python

1. Readability and Simplicity: Python's syntax is clear and easy to read, which helps developers write clean and maintainable code. For example, Python uses indentation to define code blocks instead of braces or keywords.
2. Versatility: Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.
3. Extensive Standard Library: Python comes with a rich standard library that provides modules and packages for various tasks, including file I/O, system calls, and even web development.
4. Interpreted Language: Python is an interpreted language, which means you can run code directly without needing a compilation step. This feature facilitates quick development and debugging.
5. Dynamic Typing: Variables in Python do not need explicit declaration of types. The type is determined at runtime, which can make development faster and more flexible.

2) Installing Python

1. Visit https://www.python.org/downloads/
2. Click on the "Download Python" button. This will download the latest version of the Python installer for Windows.
3. Run the Installer:
4. Locate the downloaded installer (e.g., python-3.x.x-amd64.exe) and double-click it to run.
5. Installation Options:
6. Add Python to PATH: Ensure the checkbox for "Add Python to PATH" is checked. This makes Python accessible from the command line.
7. Install Now: Click "Install Now" to use the default settings. This will install Python and pip (Python package installer).
8. Complete Installation:
9. Wait for the installation to complete and then click "Close."
10. 2. Verify the Installation
11. Open Command Prompt:
12. Press Win + R, type cmd, and press Enter to open the Command Prompt.
13. Check Python Version:
14. Type python --version or python -V and press Enter. You should see the Python version number (e.g., Python 3.x.x).
15. Check pip Version:
16. Type pip --version and press Enter. You should see the pip version number, indicating that pip is also installed.
17. Set Up a Virtual Environment
18. Open Command Prompt:
19. Use Win + R, type cmd, and press Enter if it’s not already open.
20. Navigate to Your Project Directory:
21. Use the cd command to navigate to the directory where you want to create your virtual environment using
22. Create a virtual environment using python -m venv myenv.
23. Activate the virtual environment using myenv\Scripts\activate
24. Install packages in the virtual environment using pip install package\_name
25. Deactivate the virtual environment using the command deactivate

3. Python Syntax and Semantics

print ("Hello, World!")

Explanation of Basic Syntax Elements:

print Function:

print() is a built-in Python function used to output text to the console.

The text you want to display is placed inside the parentheses of the print function.

String Literal:

"Hello, World!" is a string literal, which is a sequence of characters enclosed in double quotes.

In Python, string literals can also be enclosed in single quotes (e.g., 'Hello, World!'), but double quotes are used in this example.

Parentheses:

(): The parentheses following the print function are used to pass arguments to the function. In this case, the argument is the string "Hello, World!".

Function Call:

The combination of the function name (print) and the parentheses () is a function call. When the print function is called, it executes the code inside it and produces an output.

Execution Flow:

The Python interpreter reads the print("Hello, World!") line.

It calls the print function with "Hello, World!" as the argument.

The print function outputs the string "Hello, World!" to the console.

4. Data Types and Variables

Basic Data Types in Python

Integer (int):

Represents whole numbers, both positive and negative, without any decimal points.

Example: 42, -7

Floating-Point (float):

Represents numbers with decimal points.

Example: 3.14, -0.001

String (str):

Represents a sequence of characters enclosed in quotes (single, double, or triple quotes).

Example: "Hello, World!", 'Python'

Boolean (bool):

Represents truth values, either True or False.

Example: True, False

List (list):

Represents an ordered, mutable collection of items, which can be of different data types.

Example: [1, 2, 3, 4.5, "hello"]

Tuple (tuple):

Represents an ordered, immutable collection of items, which can be of different data types.

Example: (1, 2, 3, 4.5, "hello")

Dictionary (dict):

Represents a collection of key-value pairs, where keys are unique and values can be of any data type.

Example: {"name": "Alice", "age": 30}

Set (set):

Represents an unordered collection of unique items.

Example: {1, 2, 3, 4}

Script:

# Integer

age = 25

print("Integer:", age)

# Floating-Point

height = 5.9

print("Float:", height)

# String

name = "Alice"

print("String:", name)

# Boolean

is\_student = True

print("Boolean:", is\_student)

# List

colors = ["red", "green", "blue"]

print("List:", colors)

# Tuple

coordinates = (10, 20)

print("Tuple:", coordinates)

# Dictionary

person = {"name": "Bob", "age": 30}

print("Dictionary:", person)

# Set

unique\_numbers = {1, 2, 3, 4, 5}

print("Set:", unique\_numbers)

5. Control Structures

Conditional statements and loops are fundamental control structures in Python that help manage the flow of execution in a program.

Conditional Statements

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

Basic Syntax of Conditional Statements:

if condition:

# code to execute if the condition is true

elif another\_condition:

# code to execute if the other condition is true

else:

# code to execute if none of the above conditions are true

Example of an if-else Statement

age = 18

if age < 18:

print("You are a minor.")

elif age == 18:

print("You are an adult now.")

else:

print("You are an adult.")

Loops

Loops allow you to execute a block of code repeatedly. Python primarily uses two types of loops: for loops and while loops.

for Loop

The for loop iterates over a sequence (such as a list, tuple, string, or range) and executes the block of code for each item in the sequence.

Basic Syntax of a for Loop

for item in sequence:

# code to execute for each item

Example of a for Loop

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

for fruit in fruits:

print(fruit)

While Loop

while Loop

The while loop repeatedly executes a block of code as long as a given condition is true.

Basic Syntax of a while Loop

while condition:

# code to execute while the condition is true

Basic Example of a while Loop

count = 0

while count < 5:

print(count)

count += 1

6. Functions in Pythons

Functions in Python are blocks of reusable code that perform a specific task. They allow you to encapsulate code into a single entity that can be executed when called. Functions are useful because they help:

1. Reduce Code Duplication: By defining a function, you can reuse the same code in multiple places without having to rewrite it.
2. Improve Code Organization: Functions help break down complex problems into smaller, manageable parts.
3. Enhance Code Readability: Well-named functions make code more understandable and maintainable.
4. Facilitate Testing and Debugging: Functions make it easier to test and debug individual components of your code.

Syntax of a Function in Python

def function\_name(parameters):

# code block

return result

Example Function: Sum of Two Numbers

def add\_numbers(a, b):

# Add the two numbers and return the result

return a + b

Example of Calling the Function

# Call the function with arguments 5 and 3

result = add\_numbers(5, 3)

# Print the result

print("The sum is:", result)

Explanation

Function Definition:

def add\_numbers(a, b): defines a function named add\_numbers that takes two parameters, a and b.

return a + b computes the sum of a and b and returns the result.

Function Call:

result = add\_numbers(5, 3) calls the add\_numbers function with arguments 5 and 3.

The result, which is 8, is stored in the variable result.

Output:

print("The sum is:", result) prints the result to the console.

7. Lists and Dictionaries

Differences Between Lists and Dictionaries

Lists:

Ordered: Elements in a list maintain their order. The position of elements is determined by their index, starting from 0.

Mutable: Lists can be modified after creation. You can add, remove, or change elements.

Index-Based: Elements are accessed using integer indices.

Use Case: Ideal for storing sequences of items where the order is important and where you often need to access elements by position.

Dictionaries:

Unordered: Dictionaries are collections of key-value pairs. The order of elements is not guaranteed to be consistent across different versions of Python.

Mutable: Dictionaries can also be modified. You can add, remove, or update key-value pairs.

Key-Based: Elements are accessed using keys, which can be of various immutable types (like strings, numbers, tuples).

Use Case: Useful for storing data that needs to be retrieved based on unique keys, such as database records or mappings.

Script:

# Creating a list of numbers

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

# Basic operations on the list

print("Original list:", numbers)

# Adding an element

numbers.append(6)

print("After appending 6:", numbers)

# Removing an element

numbers.remove(3)

print("After removing 3:", numbers)

# Accessing an element

print("Element at index 2:", numbers[2])

# Creating a dictionary with some key-value pairs

person = {

"name": "Alice",

"age": 30,

"city": "New York"

}

# Basic operations on the dictionary

print("\nOriginal dictionary:", person)

# Adding a new key-value pair

person["occupation"] = "Engineer"

print("After adding occupation:", person)

# Removing a key-value pair

del person["age"]

print("After removing age:", person)

# Accessing a value by key

print("Name of the person:", person["name"])

# Iterating over dictionary items

print("Iterating over dictionary:")

for key, value in person.items():

print(f"{key}: {value}")

# Iterating over list items

print("\nIterating over list:")

for number in numbers:

print(number)

8. Exception Handling

Exception handling in Python is a mechanism that allows a program to deal with runtime errors in a graceful manner without crashing. It involves catching and responding to exceptions (errors) that occur during the execution of code, enabling you to maintain control and provide appropriate feedback or recovery options.

Basic Structure of Exception Handling in Python

Python provides the following blocks for exception handling:

1. try: This block contains code that may potentially raise an exception. It's where you write the code that you want to monitor for errors.
2. except: This block contains code that runs if an exception occurs in the try block. You can specify different types of exceptions to catch specific errors.
3. else (optional): This block contains code that runs if no exceptions occur in the try block.
4. finally: This block contains code that will run regardless of whether an exception occurred or not. It is typically used for cleanup actions like closing files or releasing resources.

Script:

def divide\_numbers(x, y):

try:

# Try to perform division

result = x / y

except ZeroDivisionError:

# Handle the case where division by zero occurs

print("Error: Cannot divide by zero!")

result = None

except TypeError:

# Handle the case where inputs are not numbers

print("Error: Invalid input type. Please provide numbers.")

result = None

else:

# Code to execute if no exception occurs

print("Division successful!")

finally:

# This block will always execute

print("Execution complete.")

return result

# Example usage

num1 = 10

num2 = 0 # Change to a non-zero value to test successful division

result = divide\_numbers(num1, num2)

if result is not None:

print(f"The result is: {result}")

9. Modules and Packages

Concepts of Modules and Packages

Module:

A module is a single file containing Python code. It can include functions, classes, variables, and runnable code.

Modules help organize code into separate files, which makes it easier to manage and maintain.

Modules are imported into other Python scripts to reuse the functions, classes, or variables defined in them.

Package:

A package is a collection of related modules organized in a directory hierarchy. A package is a directory that contains a special file named \_\_init\_\_.py (which can be empty) and one or more module files. Packages allow you to group related modules together, making it easier to manage large projects.

Script to import and use math module

# Import the entire math module

import math

# Use functions from the math module

number = 16

# Calculate the square root

sqrt\_value = math.sqrt(number)

print(f"The square root of {number} is {sqrt\_value}")

# Calculate the factorial of a number

factorial\_value = math.factorial(5)

print(f"The factorial of 5 is {factorial\_value}")

# Calculate the value of pi

pi\_value = math.pi

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

# Import specific functions from the math module

from math import sin, cos, radians

# Use specific functions

angle = 45

angle\_in\_radians = radians(angle)

sin\_value = sin(angle\_in\_radians)

cos\_value = cos(angle\_in\_radians)

print(f"The sine of {angle} degrees is {sin\_value}")

print(f"The cosine of {angle} degrees is {cos\_value}")

10. File I/O

Reading from a File

To read from a file, you use the open() function with the appropriate mode, typically 'r' for reading. After opening the file, you can use methods like .read(), .readline(), or .readlines() to get the content.

Script to read from a file:

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

file\_path = 'example.txt' # Path to the file you want to read from

try:

# Open the file in read mode

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

# Read the entire content of the file

content = file.read()

# Print the content

print("File content:")

print(content)

except FileNotFoundError:

print(f"Error: The file '{file\_path}' was not found.")

except IOError:

print(f"Error: An I/O error occurred while reading the file.")

Script to write to a file:

# Script to write a list of strings to a file

file\_path = 'output.txt' # Path to the file where you want to write the content

# List of strings to be written to the file

lines = [

"First line of text.",

"Second line of text.",

"Third line of text."

]

try:

# 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:

file.write(line + '\n') # Write the string followed by a newline

print(f"Successfully wrote to '{file\_path}'.")

except IOError:

print(f"Error: An I/O error occurred while writing to the file.")