**SE-Assignment-6**

**Assignment: Introduction to Python**

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, general-purpose programming language known for its readability and ease of use. Compared to other languages, it relies on clear, concise syntax that resembles plain English. This makes Python a great starting point for beginners but also a favorite among experienced developers for rapid prototyping and writing efficient code.

**Key Features of Python:**

* Easy to Learn and Read
* Free and Open-Source
* Interpreted Language
* Dynamically Typed
* Extensive Standard Library and Third-Party Packages.

**Use Cases for Python:**

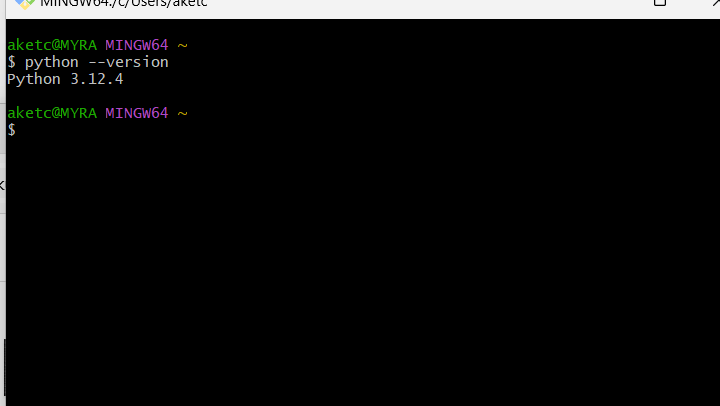
* **Web Development:** Python frameworks like Django and Flask power many popular websites and web applications.
* **Data Science and Machine Learning:** Libraries like pandas, scikit-learn, and TensorFlow make Python a leader in data analysis, machine learning, and artificial intelligence applications.
* **Scripting and Automation:** Python excels in automating repetitive tasks, system administration jobs, and scientific data analysis.
* **Game Development:** With frameworks like Pygame, Python can be used to create simple to medium-complexity games.

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.

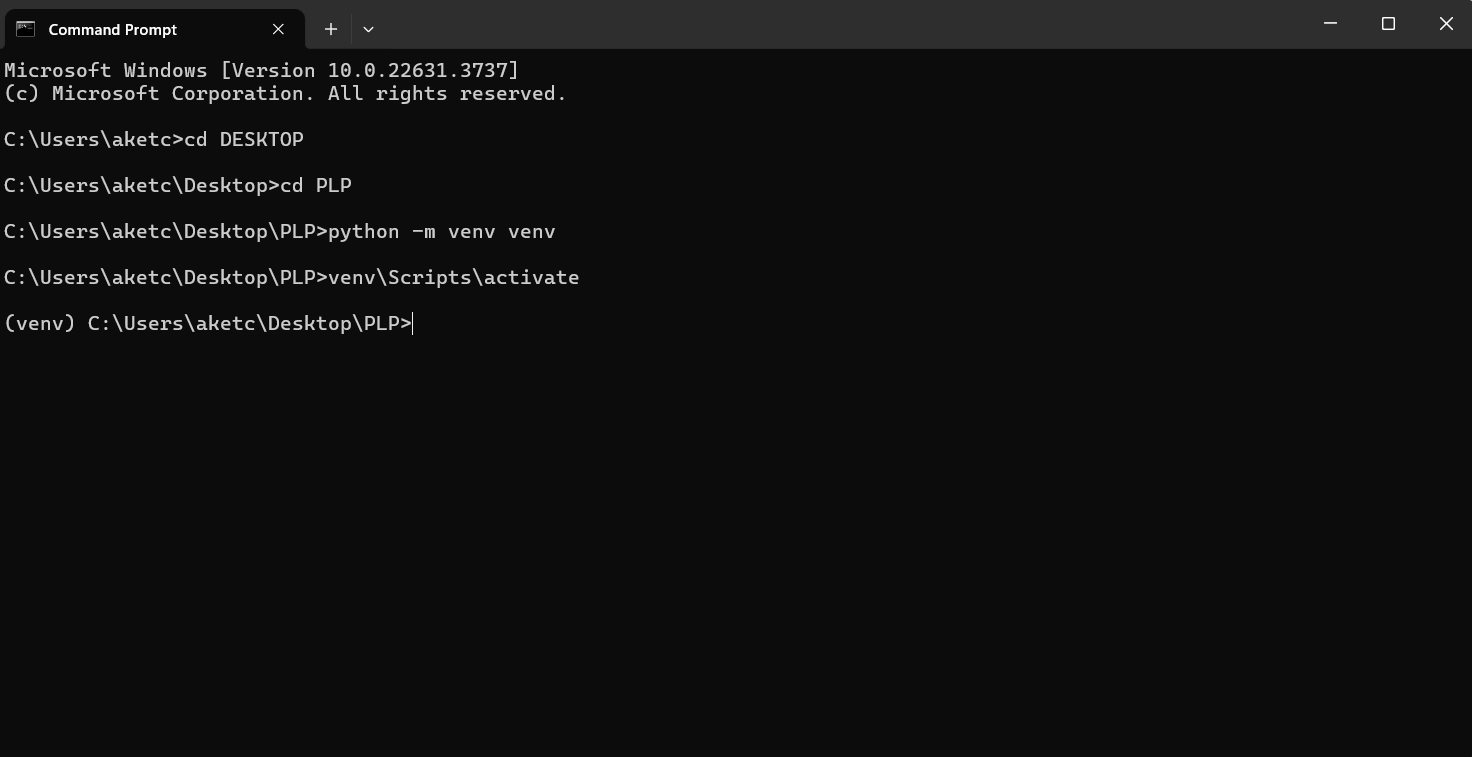
**Installing Python on Windows**

1. **Download the installer:**
   * Head to the official Python download page: [download python windows ON Python.org] and download the latest stable version of Python 3
2. **Run the installer:**
   * Double-click the downloaded executable file (e.g., python-3.x.x-amd64.exe).
   * Check the box that says "Add Python to PATH".
   * Click "Install Now" and follow the on-screen instructions.
3. **Verify the installation (Optional):**
   * Open a command prompt window (search for "cmd" in the Start menu).
   * Type python --version and press Enter. If Python is installed correctly, you should see the installed Python version number.



**Setting 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 with venv\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.

**print("Hello, World!")**

 **print() function**: The print() function is a built-in function in Python that outputs the specified message to the console or other standard output device.

 **Quotation Marks ("")**: The text inside the quotation marks is called a string literal. In Python, strings can be enclosed in single quotes (') or double quotes (").

 **Parentheses ()**:The parentheses () are used to enclose the arguments that are passed to functions. In this example, the string "Hello, World!" is an argument passed to the print() function.

A computer screen with white text

Description automatically generated

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.

* **Numeric Types:**
  + **int (integer):** Represents whole numbers (positive, negative, or zero).
  + **float (floating-point):** Represents numbers with decimals.
  + **complex:** Represents complex numbers (combination of real and imaginary parts).
* **String Type:**
  + **str:** Represents textual data enclosed in single or double quotes.
* **Sequence Types:**
  + **list:** An ordered, mutable collection of elements enclosed in square brackets []. Elements can be of different data types.
  + **tuple:** An ordered, immutable collection of elements enclosed in parentheses (). Elements can be of different data types.
  + **range:** Represents a sequence of numbers within a specified range.
* **Mapping Type:**
  + **dict (dictionary):** An unordered collection of key-value pairs enclosed in curly braces {}. Keys must be unique and immutable, while values can be of any data type.
* **Boolean Type:**
  + **bool:** Represents logical values: True or False.
* **None Type:**
  + **None:** Represents the absence of a value.

**Script Demonstrating Data Types**

# Numeric types

age = 30 # int

pi = 3.14159 # float

complex\_num = 3 + 2j # complex

# String type

name = "Alice" # str

# Sequence types

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

colors = ("red", "green", "blue") # tuple

numbers = range(1, 6) # range

# Mapping type

person = {"name": "Bob", "age": 42, "city": "New York"} # dict

# Boolean type

is\_sunny = True # bool

# None type

empty\_variable = None

# Printing variable types

print("age (int):", type(age))

print("pi (float):", type(pi))

print("complex\_num (complex):", type(complex\_num))

print("name (str):", type(name))

print("fruits (list):", type(fruits))

print("colors (tuple):", type(colors))

print("numbers (range):", type(numbers))

print("person (dict):", type(person))

print("is\_sunny (bool):", type(is\_sunny))

print("empty\_variable (None):", type(empty\_variable))

# Using the variables

print(f"Hello, {name}! You are {age} years old.")

print(f"First three fruits: {fruits[0]}, {fruits[1]}, {fruits[2]}")

print(f"Is it sunny today? {is\_sunny}")

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**

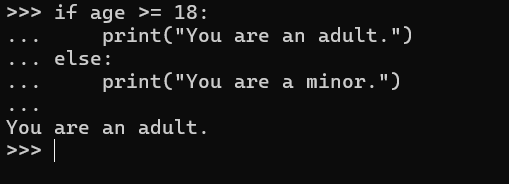
Conditional statements are used to execute code based on certain conditions. In Python, the primary conditional statements are if, elif, and else.

***if statement:*** Executes a code block if a condition is True.

***if-else statement:*** Executes one code block if a condition is True and another block if it's False.

***if-elif-else statement:*** Allows checking multiple conditions sequentially.

Example:



**Loops**

Loops are used to execute a block of code repeatedly as long as a certain condition is met. In Python, the primary loops are for and while.

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

***while loop:*** Continues executing a code block as long as a condition remains True.

Example:

A screen shot of a computer

Description automatically generated

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 reusable blocks of code that perform specific tasks in Python. They encapsulate a set of instructions and can be called from other parts of your program. Functions offer several advantages:

**Reusability:** You can write a function once and use it multiple times throughout your code, saving time and effort.

**Modularity:** Functions break down complex programs into smaller, manageable parts, improving code readability and maintainability.

**Encapsulation:** Functions can hide internal details and expose only what's needed, promoting better code organization.

Example Function

def add\_numbers(a, b):

return a + b

Call the function with example values

result = add\_numbers(10, 5)

Print the result

print(f"The sum of 10 and 5 is: {result}")

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 and dictionaries are both data structures used to store collections of items, but they serve different purposes and have distinct characteristics. A list is an ordered collection of items, which can be of any type, and each item is accessed by its index. Lists are mutable, meaning their contents can be changed after creation, and they allow duplicate elements. Operations such as appending, inserting, and removing elements are common with lists. In contrast, a dictionary is an unordered collection of key-value pairs, where each key must be unique and immutable (like strings, numbers, or tuples), but values can be of any type and are mutable. Dictionaries are optimized for fast retrieval of values based on their keys, making them suitable for lookups and associative arrays, they are unordered collections accessed by unique keys.

***Script Demonstrating Lists and Dictionaries***

# Create a list of numbers

numbers = [1, 5, 8, 3, 2]

# Print the list

print("Numbers list:", numbers)

# Access an element by index (0-based)

first\_number = numbers[0]

print("First element:", first\_number)

# Modify an element in the list

numbers[2] = 10

print("List after modification:", numbers)

# Check if an element exists (using 'in' operator)

if 5 in numbers:

print("Number 5 is present in the list.")

# Create a dictionary with key-value pairs

person = {

"name": "Alice",

"age": 30,

"city": "New York"

}

# Print the dictionary

print("\nPerson dictionary:", person)

# Access a value by key

name = person["name"]

print("Name:", name)

# Add a new key-value pair

person["occupation"] = "Software Engineer"

print("Dictionary after adding occupation:", person)

# Loop through key-value pairs (using 'items' method)

for key, value in person.items():

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

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.

Exception handling is a mechanism in Python to manage errors that may arise during program execution. It allows you to write robust code that can gracefully handle unexpected situations and prevent program crashes.

Here is a basic structure example for exception handling:

try:

# Code that might raise an exception

except ExceptionType:

# Code to handle the exception (optional)

finally:

# Code that always executes (optional)

Example:

def divide(a, b):

try:

result = a / b

except ZeroDivisionError:

print("Error: Division by zero is not allowed.")

result = None

finally:

print("Execution of the divide function is complete.")

return result

# Example usage

result1 = divide(10, 2) # This should succeed

result2 = divide(10, 0) # This should raise a ZeroDivisionError

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.

**Modules**

* A module is a fundamental building block of Python code organization. It's a single .py file containing Python definitions like functions, classes, and variables.
* Modules promote code reusability by allowing you to encapsulate functionalities and import them into your scripts as needed.
* This helps in code modularity, making programs easier to understand, maintain, and extend.

**Packages**

* A package is a collection of related modules organized hierarchically within a directory structure. Think of it as a way to group modules that belong together thematically.
* Each package directory must contain an empty file named \_\_init\_\_.py. This file can also contain initialization code for the package.
* Packages provide a way to avoid naming conflicts between modules from different parts of your codebase or from external libraries.

**Importing and Using Modules**

1. **import Statement:** The import statement is used to import modules into your Python script. There are two main ways to use it:
   * **Importing the entire module:**

Python

import math # Imports all functions, classes, and variables from the math module

* + **Importing specific elements:**

Python

from math import pi, sqrt # Imports only pi and sqrt functions from the math module

1. **Using Imported Elements:** Once a module is imported, you can access its elements using dot notation:

Python

# Using functions from the entire module

area\_of\_circle = math.pi \* radius\*\*2

# Using functions imported specifically

distance = sqrt(x1\*\*2 + y1\*\*2 - x2\*\*2 - y2\*\*2)

**Example Using the math Module**

Python

import math

def calculate\_area\_of\_circle(radius):

"""Calculates the area of a circle using the math.pi constant."""

return math.pi \* radius\*\*2

area = calculate\_area\_of\_circle(5)

print("Area of the circle:", area)

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.

**Reading from Files**

1. **Opening the File:** Use the open() function to open a file in a specific mode. Common modes include:
   * 'r': Open for reading (default)
   * 'w': Open for writing (truncates existing content)
   * 'a': Open for appending (adds content to the end)
2. **Reading the Content:** Once the file is open, use methods like:
   * read(): Reads the entire file as a string.
   * readline(): Reads a single line from the file.
   * readlines(): Reads all lines of the file into a list.
3. **Closing the File:** It's crucial to close the file using the close() method to release resources and ensure data integrity.

**Example: Reading a File**

Python

def read\_file(filename):

"""Reads the contents of a file and prints them to the console."""

try:

with open(filename, 'r') as file:

contents = file.read()

print(contents)

except FileNotFoundError:

print(f"Error: File '{filename}' not found.")

read\_file('my\_file.txt') # Replace 'my\_file.txt' with your actual file path

**Writing to Files**

1. **Opening the File:** Similar to reading, use open() with the appropriate mode:
   * 'w': Open for writing (truncates existing content)
   * 'a': Open for appending (adds content to the end)
2. **Writing the Content:** Use methods like:
   * write(string): Writes the string to the file.
   * writelines(list\_of\_strings): Writes each string from the list to the file, adding newlines by default.
3. **Closing the File:** Close the file using close().

**Example: Writing a List to a File**

Python

def write\_list\_to\_file(filename, data):

"""Writes a list of strings to a file, each string on a new line."""

try:

with open(filename, 'w') as file:

file.writelines(data + ['\n']) # Add newline characters to each item

except (IOError, OSError) as e: # Handle general I/O errors

print(f"Error writing to file '{filename}': {e}")

data\_to\_write = ["Line 1", "Line 2", "Line 3"]

write\_list\_to\_file('my\_data.txt', data\_to\_write)