Solutions 1:

A.What is Python?

Python is a high-level, interpreted programming language known for its simplicity and readability.

1. Key features:

Readability and Simplicity

Interpreted Language

Dynamically Typed

Extensive Standard Library

Community and Ecosystem

Cross-Platform Compatibility

Versatility

1. Use cases:

Frameworks like Django and Flask allow developers to build robust web applications.

Data Science and Machine Learning.

Automation and Scripting

Scientific Computing

Game Development

Solution 2:

1. Python Installation:

### Step 1: Go to [python.org](https://www.python.org/" \t "_new).Navigate to the Downloads section and select "Download Python for Windows".

### Step 2: Run the downloaded installer file.Ensure to check the box "Add Python to PATH".Click "Customize installation" for advanced options, or simply click "Install Now" for a default installation.

### You can select optional features like pip (package manager), IDLE (Python IDE), and others.Choose the installation location if you want to install Python to a specific directory.

### Step 3: Press Win + R, type cmd, and hit Enter.Type python --version or python -V and press Enter.You should see the installed Python version displayed.

### Step 4: Type pip --version in the Command Prompt and press Enter.You should see the pip version displayed. If not, you may need to reinstall Python with the pip option selected.

### Step 5: Use cd path\to\your\project to navigate to your project folder.

### Type python -m venv myenv and press Enter.This creates a virtual environment named myenv.

### On Windows, type myenv\Scripts\activate and press Enter.You should see (myenv) prepended to your command prompt, indicating that the virtual environment is active.

Use pip install package\_name to install any packages you need within this isolated environment.

Type deactivate and press Enter when you're done working in the virtual environment.

1. Python Syntax and Semantics:

x = ‘Hello, World’

print(x)

Where x is the variable name holding the string literal, and print() function is use for output. Finally the tab signifies after the variable declaration is python syntax.

1. Data Types and Variables:

**Integer (int)**: Whole numbers, positive or negative, without decimals. e.g 5, -3, 100

**Float (float)**: Numbers with a decimal point. e.g 3.14, -0.001, 2.0

**String (str)**:Sequence of characters enclosed in single, double, or triple quotes. e.g "Hello", 'Python', """This is a string"""

**Boolean (bool)**:Represents one of two values: True or False. e.g True, False

**List**: Ordered, mutable collection of items, which can be of different types. e.g [1, 2, 3], ['a', 'b', 'c']

**Tuple**: Ordered, immutable collection of items, which can be of different types. e.g (1, 2, 3), ('a', 'b', 'c')

**Dictionary (dict)**: Unordered, mutable collection of key-value pairs. e.g {'name': 'John', 'age': 30}, {1: 'one', 2: 'two'}

**Set**: Unordered collection of unique items. e.g {1, 2, 3}, {'a', 'b', 'c'}

**Python code:**

**Integer**

a = 10print(f"Integer: {a}")

**Float**

b = 3.14print(f"Float: {b}")

**String**

c = "Hello, World!"print(f"String: {c}")

**Boolean**

d = Trueprint(f"Boolean: {d}")

**List**

e = [1, 2, 3, 'a', 'b', 'c']print(f"List: {e}")

**Tuple**

f = (1, 2, 3, 'a', 'b', 'c')print(f"Tuple: {f}")

**Dictionary**

g = {'name': 'Alice', 'age': 28, 'is\_student': True}print(f"Dictionary: {g}")

**Set**

h = {1, 2, 3, 3, 4, 4, 5}print(f"Set: {h}")

**Demonstrating variable usage**

sum\_int\_float = a + b

print(f"Sum of integer and float: {sum\_int\_float}")

list\_element = e[3]

tuple\_element = f[4]

dict\_value = g['name']

set\_size = len(h)

1. Control Structures:

Conditional statements and loops are fundamental control flow tools in Python that allow you to execute code based on certain conditions or to repeat code multiple times.

x = 10 if x > 5:

print("x is greater than 5")

elif x == 5:

print("x is equal to 5")

else:

print("x is less than 5")

A ‘for Loop’

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

for number in numbers:

print(number)

1. Functions in Python:

Functions in Python are blocks of reusable code that perform a specific task. They allow you to encapsulate a piece of logic, making your code more modular, organized, and easier to maintain. Functions can take arguments (input values) and can return a result.

Benefits of Functions:

Functions allow you to reuse code without rewriting it.

Functions help break down complex problems into smaller, more manageable pieces.

Named functions can make your code more readable and understandable.

Functions make it easier to update and manage code.

Function example:

def add\_numbers(a, b):

return a + b

Example of calling the function:

result = add\_numbers(5, 7)

print(f"The sum of 5 and 7 is: {result}")

def add\_numbers(a, b):

return a + b

Example of calling the function

result = add\_numbers(5, 7)

print(f"The sum of 5 and 7 is: {result}")

When you run this code, it will output:

The sum of 5 and 7 is: 12

### Differences Between Lists and Dictionaries in Python

**Lists**:

* **Ordered**: Lists maintain the order of elements.
* **Indexing**: Elements are accessed by their position (index) starting from 0.
* **Mutable**: Elements can be added, removed, or modified.
* **Homogeneous or Heterogeneous**: Lists can contain elements of any data type.

**Dictionaries**:

* **Unordered**: Dictionaries do not maintain any specific order of elements.
* **Key-Value Pairs**: Data is stored as key-value pairs, where each key is unique.
* **Mutable**: Keys and values can be added, removed, or modified.
* **Fast Lookups**: Retrieving values by key is generally faster than searching for elements in a list.

### Example Script

python code

numbers = [10, 20, 30, 40, 50]

print("Original list:", numbers)

numbers.append(60)

print("After appending 60:", numbers)

numbers.remove(20)

print("After removing 20:", numbers)

numbers[2] = 35

print("After modifying the third element:", numbers)

print("Element at index 1:", numbers[1])

person = {

'name': 'Alice',

'age': 28,

'city': 'New York'

}

print("\nOriginal dictionary:", person)

person['email'] = 'alice@example.com'

print("After adding email:", person)

del person['age']

print("After removing age:", person)

person['city'] = 'San Francisco'

print("After modifying city:", person)

print("Value of 'name' key:", person['name'])

print("Value of 'email' key:", person.get('email', 'Not Found'))

### G.Exception Handling in Python

Exception handling in Python is a mechanism to handle runtime errors, allowing a program to continue execution or terminate gracefully. It helps in managing and responding to unexpected events (exceptions) that occur during program execution, such as file not found, division by zero, or type errors.

### Key Components

* **try**: The block of code that you want to monitor for exceptions.
* **except**: The block of code that runs if an exception occurs in the try block.
* **finally**: The block of code that runs whether an exception occurs or not, typically used for cleanup actions.

### Example Script

def divide\_numbers(a, b):

try:

result = a / b

except ZeroDivisionError as e:

print(f"Error: Division by zero is not allowed. Details: {e}")

result = None

except TypeError as e:

print(f"Error: Invalid input type. Details: {e}")

result = None

else:

print("Division successful.")

finally:

print("Execution of the try-except block is complete.")

return result

print("Test Case 1: Valid Division")

print(f"Result: {divide\_numbers(10, 2)}")

print("\nTest Case 2: Division by Zero")

print(f"Result: {divide\_numbers(10, 0)}") message

print("\nTest Case 3: Invalid Input Type")

print(f"Result: {divide\_numbers(10, 'a')}")

H.Modules and Packages in Python

A module is a single file containing Python code, which can include functions, classes, variables, and runnable code. It helps in organizing code into manageable sections and promotes code reuse.

Modules are used to break down large programs into smaller, more manageable, and organized files.

Packages:

A package is a collection of related modules bundled together under a directory with a special \_\_init\_\_.py file, which indicates that the directory is a Python package. Packages allow for a hierarchical structuring of the module namespace.

Packages can contain sub-packages, which further contain modules.

Importing and Using a Module

To use a module in your script, you need to import it using the import statement. You can then access the functions, classes, and variables defined in the module.

Example Using the math Module

The math module provides access to mathematical functions like trigonometry, logarithms, and constants.

import math

print("Using math.sqrt() to calculate the square root of 16:") print(math.sqrt(16))

print("\nUsing math.factorial() to calculate the factorial of 5:") print(math.factorial(5))

print("\nUsing math.pi to get the value of pi:") print(math.pi) print("\nUsing math.sin() to calculate the sine of pi/2:") print(math.sin(math.pi / 2))

### I.Reading from and Writing to Files in Python

In Python, you can use the built-in open() function to read from and write to files. Here are the basic steps and examples for both operations.

### Reading from a File

To read the contents of a file, you typically:

1. Open the file using open().
2. Read the content using methods like read(), readline(), or readlines().
3. Close the file using close().

#### Example Script to Read from a File

file\_path = 'example.txt'with open(file\_path, 'r') as file:

### Writing to a File

To write to a file, you typically:

1. Open the file using open() with the appropriate mode ('w' for writing, 'a' for appending).
2. Write the content using methods like write() or writelines().
3. Close the file using close().

#### Example Script to Write a List of Strings to a File

lines = [

"First line of text.\n",

"Second line of text.\n",

"Third line of text.\n"

]

file\_path = 'output.txt'with open(file\_path, 'w') as file:

file.writelines(lines)

print(f"Content written to {file\_path}")