Chapter # 02
Python Programming

Q.1. What is Python, who created it, what inspired its name, and when were its major versions released?

Ans. **Python** is a **programming language** used to give instructions to **computers**. It was **created** by **Guido van Rossum** from the **Netherlands** in the late **1980s** and first **released** in **1991**.

The name **"Python"** was inspired by a **comedy TV show** called *Monty Python's Flying Circus*, which Guido enjoyed.

Major **versions** of Python include:

- **Python 2.0**, released in **2000**
- **Python 3.0**, released in **2008**

Python is known for being **simple** and **easy to learn**.

Q.2. Describe the characteristics of Python that make it a popular programming language worldwide. Why is it considered a good programming language for beginners and professionals?

Ans. Python is a high-level, easy-to-learn programming language. It has simple syntax and clear structure. These characteristics make it a great choice for both beginners and experienced programmers.

Key Features of Python:

1. **Simplicity and Easy to Read:**
   Python is easy to read and write because its syntax (rules for writing code) is simple. The way Python code is written looks like regular English, which makes it easier for beginners to understand.
2. **Versatility:**
   Python can be used in many different fields, such as building websites, analyzing data, making computer games, and even creating artificial intelligence (AI). No matter what type of project we want to work on, we can do it with Python.
3. **Free and Open-Source:**
   Python is free to use, and its code is open for anyone to see and modify. This has made it very popular because anyone can use it without paying for it.
4. **Large Support Community:**
   There are a lot of people who use Python and help each other when learning or solving problems. If we have trouble while coding, it is easy to find solutions and advice from others on the Internet.

Why Python is Good for Beginners:

Python is great for people who are new to programming because:

- **Easy Syntax:** Syntax of a language are the rules for writing code in the language. Python uses simple rules that make it easier to understand compared to other languages.
- **Fast Learning:** A person can learn Python quickly and start making programs right away.
- **Focus on Learning Concepts:** Python focuses on important ideas like variables, loops, and functions, rather than getting confused by difficult language rules.

Why Professionals Like Python:

Even though Python is good for beginners, many professionals also use it because:

- **It's Powerful:** Python can handle big projects like building websites, analyzing data, and creating machine learning models.

- **Support large Number of Tools and Libraries:** Python has many libraries (pre-written code) that help professionals build things faster. For example, if we want to create a website, we can use tools like Django or Flask, or if we want to analyze data, you can use libraries like Pandas and Matplotlib.

Where Python is Used:

Python is used in many different areas, such as:

- **Web Development:** Python helps create websites using frameworks like Django and Flask.
- **Data Science and Analysis:** Python is great for working with data and finding useful information from it.
- **Artificial Intelligence (AI) and Machine Learning:** Python is the main language used to build intelligent systems and make computers learn from data.
- **Automation:** Python is often used to write programs that automatically do tasks, like sending emails or organizing files.
- **Game Development:** Python can also be used to create simple video games.
- **Desktop Applications:** Python is also used to create programs that run on computer, such as apps or tools.

Q.3. What are basic programming concepts, and what are the main steps in writing a computer program?

Ans. Computer programming is the process of designing and writing instructions that a computer can follow to perform specific tasks. These instructions are written in a programming language such as Python. Programming allows us to control the behavior of a computer and solve real-world problems efficiently. It is similar to giving step-by-step directions to a person — the instructions must be clear and correct to get the desired result.

Steps in Writing a Computer Program:

1. **Write Code:**
   This is the first step, where the programmer writes a set of instructions using a programming language like Python.
2. **Compile or Interpret the Code:**
   The written code is then translated into a form that the computer can understand. Python uses an interpreter for this purpose.
3. **Execute the Program:**
   After the code is translated, it is run or executed by the computer to perform the required task.
4. **Display Output:**
   Once the program runs successfully, the output or result is shown, such as displaying text or performing an action.

Q.4. Explain how to set up a Python development environment and how to install Python?

Ans. **Setting Up Python Development Environment:**

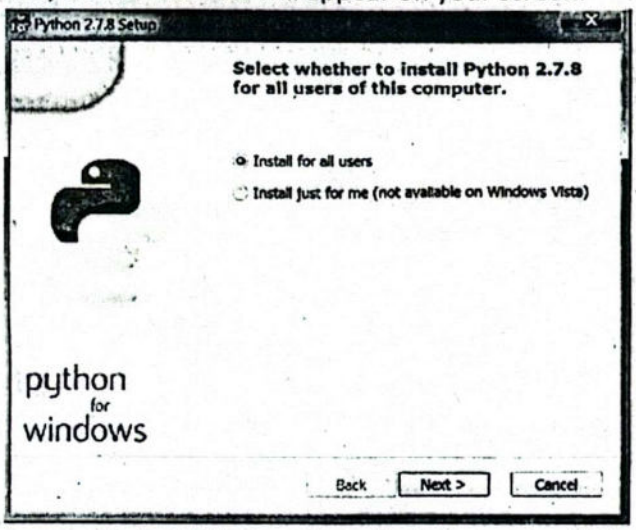
The development environment refers to the setup needed to write, run, and test Python programs effectively. This consists of the following steps:

- **Installing Python:**
  Python can be downloaded and installed from its official website. It is necessary to install Python on the computer to begin programming.
  How to install Python?
  Python 2.x can be downloaded from this website: https://www.python.org/downloads.

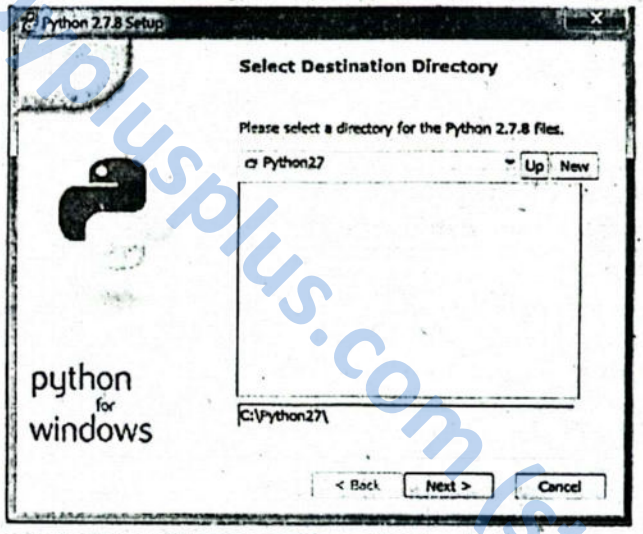
Tidbits:
When installing Python, make sure to check the box that says "Add Python to PATH." This makes it easier to run Python from the command line. We can also use online services to write and run Python program.

The installer is simple and easy to use. Follow these steps to install Python:

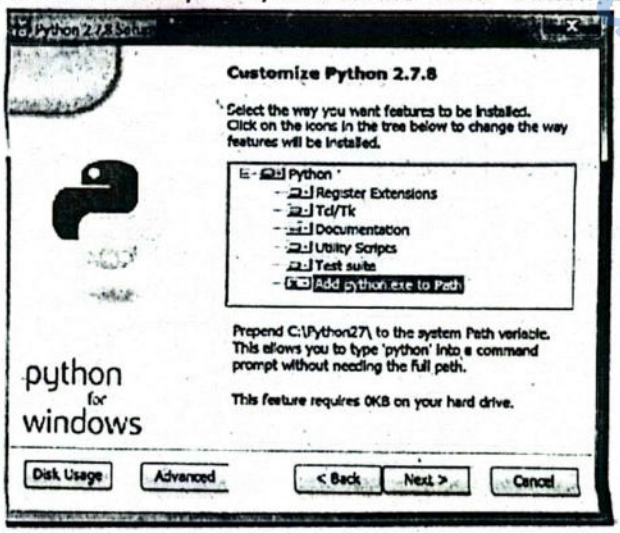
1. After clicking on the installer file, a small window will appear on your screen.



2. You can choose a folder where you want to install Python. If you don't choose any folder, the installer will automatically create one for you. You will see a screen showing the folder name. Click the "**Next**" button to continue.



3. After step 2 is done, a new window will appear to customize Python. You will see an option called **"Add python.exe to Path"** with an **X mark**. Click on this option to **select it**. This will help your computer find Python easily (you will learn more about this later). Now, click on the "**Next**" button to continue.



4. Finally, click on **Finish** to complete the installation:

- **Choosing an IDE (Integrated Development Environment)**
  An IDE is special software that helps programmers write and run their code more easily. It offers helpful features such as auto-completion, syntax highlighting, and error checking. Popular Python IDEs include **IDLE**, **PyCharm**, and **VS Code**.

By setting up a proper development environment, programming becomes easier and more organized. It also helps in identifying and fixing errors in the code efficiently.

Q.5. What are the file formats associated with Python, and explain the difference between the two interactive shells provided with the Python installer?

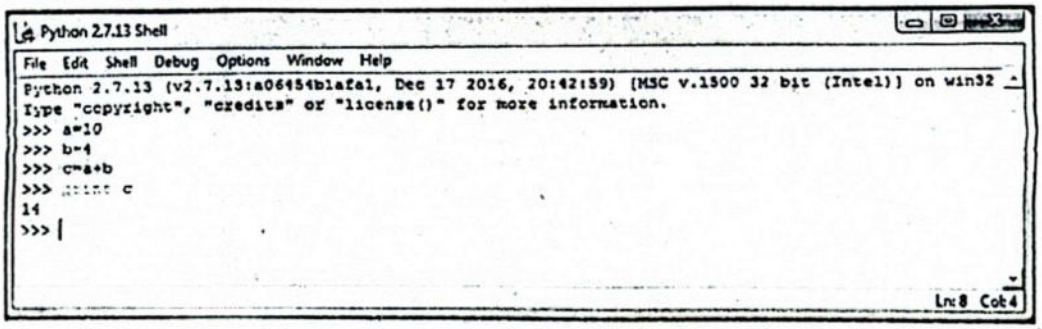
Ans. Python File Formats:

- The standard file format for Python source code is **.py**. This is the extension used for human-readable Python scripts.
- The bytecode file format is **.pyc**. These files are generated by the Python interpreter after compiling the **.py** files and are used to speed up subsequent executions of the program.

Difference Between the Two Interactive Shells:

1. **IDLE (Python GUI):**
   - **IDLE**, which stands **for Integrated Development and Learning Environment**, is a graphical user interface that comes with **Python**. It allows you to write and run Python code easily.
   - It provides a graphical user interface with features like:
     1. Syntax highlighting
     2. Auto-completion
     3. Built-in debugger
   - IDLE is suitable for running simple programs.
   - **How to Open IDLE?**

   On Windows:
   - Click the **Search bar** (next to the Start button)
   - Type **IDLE**
   - Click on **IDLE (Python GUI)** from the search results

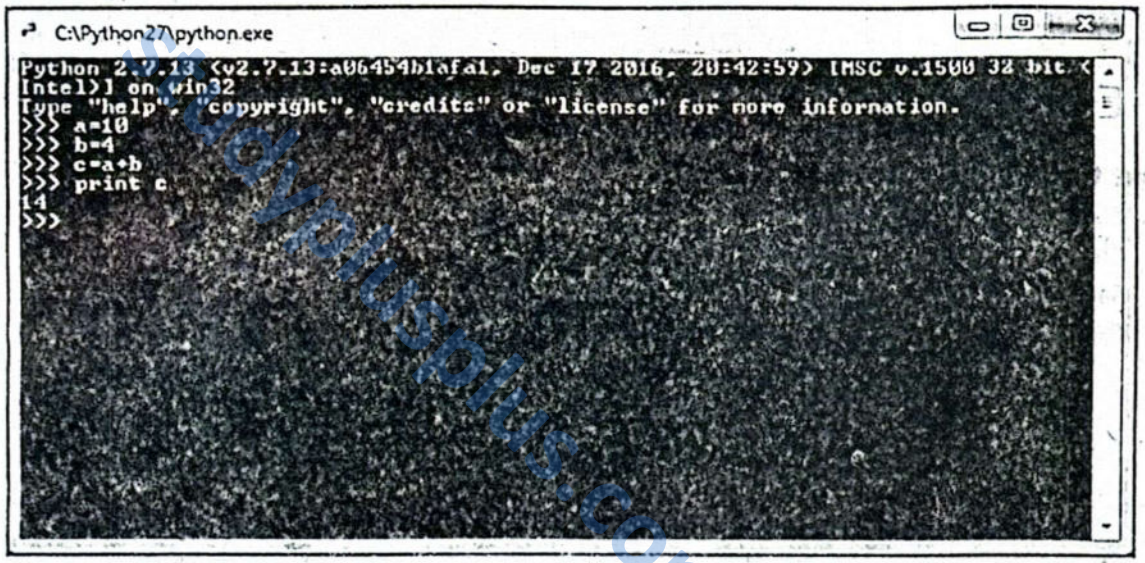


2. **Python Command Line:**
   - The **Python command line** is a text-based tool that comes with Python. You can open it using the **Command Prompt** in Windows.
   - It is more suitable for running **large or complex programs**, especially when your program needs special settings or works with the system.
   - For **big projects or professional work**, programmers usually prefer the command line instead of IDLE because it is more flexible and works better with system tools.
   - **How to open Python Command Line:**

On Windows:

- Click the **Search bar**
- Type **cmd** and press Enter to open **Command Prompt**
- In the Command Prompt window, type python and press Enter

  Or:
- Go to **Start Menu → Python (version) → Python (command line)**



Q.6. Explain the basic syntax of Python programming with the help of an example.

Ans. The syntax of a programming language refers to the set of rules that define how programs are written and understood. Python is known for its simple and readable syntax, which makes it easy to learn and use, especially for beginners.

In Python, statements are written in a straightforward manner without the need for extra symbols like semicolons (;) or curly brackets ({}) used in many other languages.

Example:

```
print("As - Salaam-Alaikum, College Students!")
```

In this example, the **`print()`** function is used to display a message on the screen. The text inside the quotation marks (" ") is the message that will be shown when the program runs.

- **`print`** is a built-in Python function.
- **"As - Salaam-Alaikum, College Students!"** is a **string**, which is a data type in Python used to represent text.
- The function displays the output exactly as written inside the quotes.

This example shows how Python uses clear and clean syntax to perform tasks, such as displaying output.

Q.7. How to Write, Save, and Run a Python Program?

Ans. *To write, save, and run a Python program, we can use* **IDLE**, which is the default editor that comes with Python. It is very simple and easy for students to use.

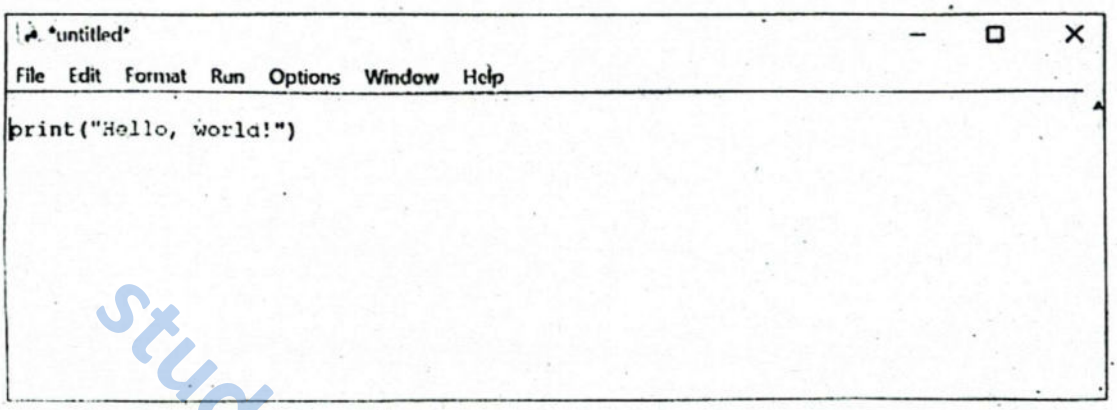
Step 1: Open IDLE

- Click on the **Start Menu**.

- Search for **IDLE (Python 3.x GUI)**.
- Click on it to open the Python shell window.

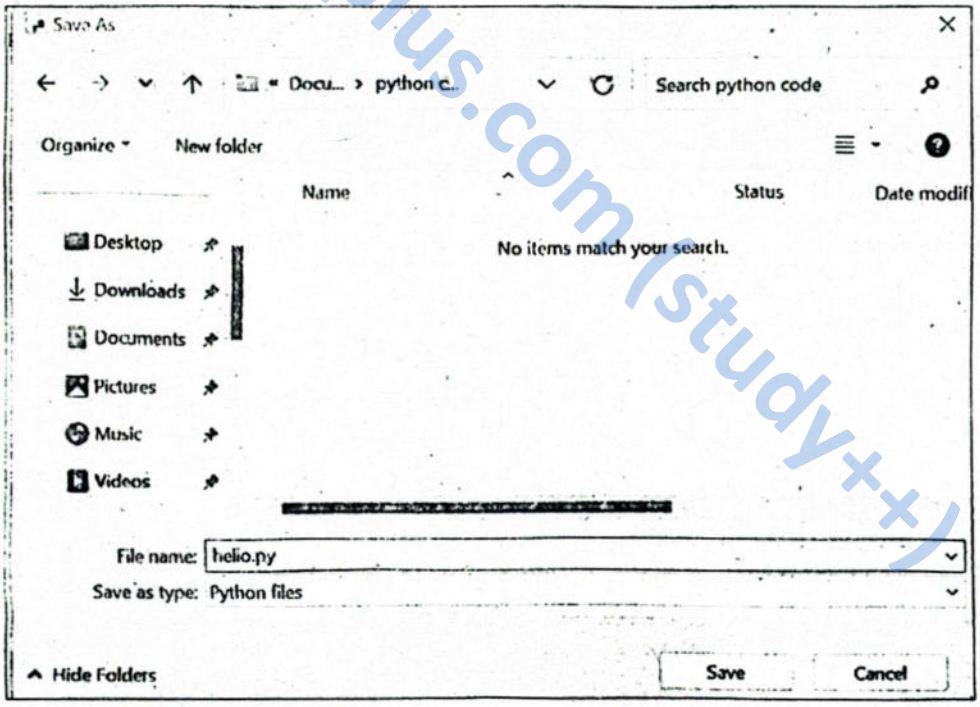
Step 2: Write the Python Program

- In the IDLE window, click on **File → New File**.
- A new editing window will open.
- Type your Python code here.



Step 3: Save the Program

- Click on **File → Save** or press **Ctrl + S**.
- Choose a folder (like Desktop or Documents).
- Save the file with a name and **.py** extension.
  Example: hello.py



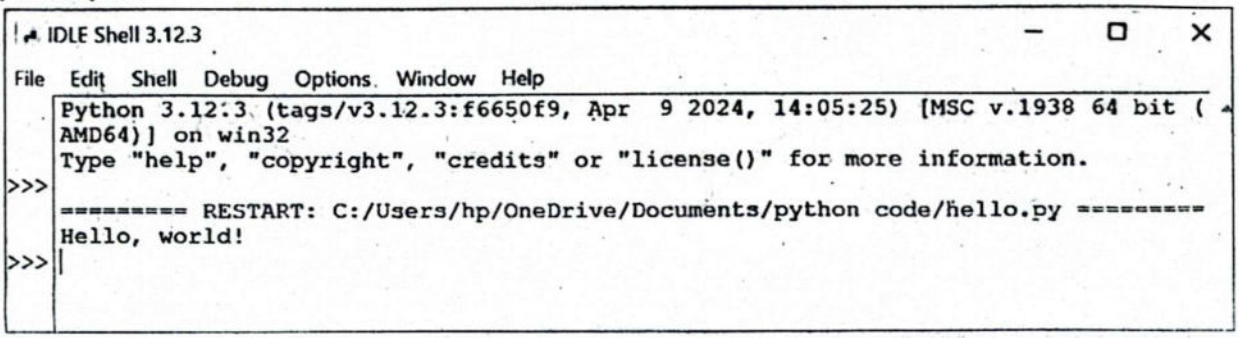
Step 4: Run the Program

- Click on **Run → Run Module** or press **F5** on your keyboard.
- The output will appear in the **Python Shell window.**

Step 5: See the Output

After running the program, you will see the result.

Example Output:



Q.8. What is the purpose of using comments in Python? Explain its types.

Ans. Comments are explanatory statement in a program that are not executed by the Python interpreter. They are used to explain the code and make it easier for programmers to understand the purpose and function of each part of the program.

Purpose of Using Comments in Python:

- To explain what the code is doing in simple words.
- To make the program easier to read and understand.
- To help the programmer or others understand the code later.
- To write notes or instructions inside the code.
- To temporarily disable a line of code during testing or debugging.

Types of Comments in Python:

Python provides two main types of comments:

1. **Single-Line Comment:**
   - Starts with the # symbol.
   - Used to explain a single line or step in the code.
   - Python ignores the part after # while running the program.

Example:

File Edit Format Run Options Window Help

```
# This is a single-line comment
print("K2 is the second-highest mountain in the world")
```

2. **Multi-Line Comment:**
   - Multi-line comments cover more than one line.
   - They can be written using triple quotation marks (''' or """) at the beginning and end of the comment.
   - Useful for writing longer explanations or notes across multiple lines.

Example:

File Edit Format Run Options Window Help

```
'''
This program performs the addition of two numbers.
It stores the result in a variable called 'sum' and prints it.
'''
num1 = 10
num2 = 20
sum = num1 + num2
print("The sum is:", sum)
```

Q.9. Differentiate between single-line and multi-line comments in Python.

Ans. The following are the main differences between single-line and multi-line comments:

Single-line Comment	Multi-line Comment
Definition: Used to comment a single line of code, starting with the # symbol.	**Definition:** Used to comment multiple lines of code, enclosed in triple quotes (''' or """).
Syntax: Begins with #. The rest of the line is ignored by Python.	**Syntax:** Starts and ends with triple quotes (''' or """). Everything inside is ignored by Python.
Usage: Ideal for brief explanations or commenting out one line.	**Usage:** Best for longer explanations or commenting out multiple lines.
Example: # This is a single-line comment	**Example:** ''' This is a multi-line comment. '''

Q.10. Explain the use of single quotes ('), double quotes ("), and triple quotes (''' or """) in Python.

Ans. Use of Single Quotes, Double Quotes, and Triple Quotes in Python:

1. **Single Quotes (')**: Single quotes are used to show a message.
   Example:

```
In [4]: print('Hello!')

        Hello!
```

2. **Double Quotes (")** : Double quotes work the same as single quotes. But they are better if the sentence has a ' (apostrophe).
   Example:

```
In [5]: print("It's a sunny day")

        It's a sunny day
```

3. **Triple Quotes (''' or """)**: Triple quotes are like magic! It is used to write a **long message** on many lines.
   Example:

```
In [6]: print('''Hello!
        How are you?
        Have a great day!''')


        Hello!
        How are you?
        Have a great day!
```

Q.11. What is a variable. Explain its purpose with an example. / Explain the concept of variables in Python.

Ans. Definition of a Variable:

In programming, a variable is the name of a storage location in the computer's memory that holds a value. This value can be used and changed throughout the execution of the program.

A variable works like a container that stores information such as numbers or text, which can be accessed and modified in the program.

Purpose of a Variable:

- A variable is used to **store data temporarily** during the execution of a program.
- The value stored in a variable can be **used, changed, or updated** as needed.
- Variables make programs more **flexible and useful**.
- They allow data to be **changed and shown** in different ways.
- Variables help in **writing reusable** code.

Example:

Consider the following Python code:

```
File  Edit  Format  Run  Options  Window  Help
age = 71
# Storing Quaid-i-Azam's age in the variable 'age'
print("Quaid-i-Azam lived for", age, "years")
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/hp/OneDrive/Documents/python code/var.py ==========
Quaid-i-Azam lived for 71 years
>>>
```

Updating Code:

```
File  Edit  Format  Run  Options  Window  Help
age = 64
# Updating the variable 'age' to store Allama Iqbal's age
print("Allama Iqbal lived for", age, "years")
```

Output:

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/hp/OneDrive/Documents/python code/var.py ==========
Quaid-i-Azam lived for 71 years
>>>
========== RESTART: C:/Users/hp/OneDrive/Documents/python code/var.py ==========
Allama Iqbal lived for 64 years
>>>
```

Explanation:

- In this example, we use the **variable age** to store the ages of two great leaders, Quaid-i-Azam Muhammad Ali Jinnah and Allama Iqbal.
- First, the **value** 71 is **assigned** to the variable **age**, which represents Quaid-i-Azam's age, and it is printed.
- Then, the value of **age** is updated to **64**, which represents Allama Iqbal's age, and the updated value is printed.

This shows how the variable age stores different values at different times, and how the data it holds can be easily updated or used in the program.

Q.12. List and Explain the Rules for Naming Variables in Python. Provide Examples of Valid and Invalid Variable Names.

Ans. Rules for Naming Variables in Python:

When writing a Python program, we must follow some rules for naming variables. These rules help avoid errors and make the code easy to read.

Tidbits:
Always use meaningful names for variables to make your code easier to understand. For example, use age instead of a.

1. A variable name must start with a letter or an underscore (_)

- It can begin with a small letter (a–z), a capital letter (A–Z), or an underscore.

- It **cannot start with a number or symbol.**

  ✔**Correct:** age, _name

  ✗ **Incorrect:** 1age, @value

2. **A variable name can only have letters, numbers, or underscores**
   - After the first letter or underscore, you can use letters, numbers, or underscores in the name.

     ✔**Correct:** roll_no, student1, user_name

     ✗ **Incorrect:** name@1, my-name
3. **Variable names are case-sensitive**
   - This means **capital and small letters are treated as different.**
   - For example, age and Age are two different variables.

     ✔**Correct:** age, Age (both are allowed but they are different variables)
4. **Python keywords cannot be used as variable names**
   - Keywords like if, while, for, etc. are used by Python for special purposes.
   - They **cannot be used** as variable names.

     ✗ **Incorrect:** if, class, while, True
5. **Do not use names of built-in Python functions**
   - Python has many built-in functions like print(), input(), etc.
   - Using these names for variables can **create confusion or errors.**

     ✗ **Incorrect:** print, input, sum
6. **Special characters and spaces are not allowed in variable names**
   - You **cannot use special symbols** like @, #, -, !, %, or **spaces** in variable names.
   - Only **letters, numbers,** and **underscores (_)** are allowed.

     ✔**Correct:** user_name, student1, _score

     ✗ **Incorrect:** user-name, marks@home, total marks (has space)

Q.13. What are the different types of variables in Python? Explain each type with an example.

Ans. The main types of variables in Python are:

> **Tidbits:**
> It's a good practice to use lowercase letters for variable names and underscores to separate words in variable names (e.g., student_name).

1. **Integer (int):**
   - **Definition**: Stores whole numbers without decimal points.
   - **Example:**

     age = 17

   In this case, age is an integer variable storing the value 17.
2. **Floating-point (float):**
   - **Definition**: Stores numbers with decimal points.
   - **Example:**

     price = 19.99

   Here, price is a floating-point variable storing the value 19.99.
3. **String (str):**
   - **Definition**: Stores text or characters. Strings are enclosed in quotes, either single (' ') or double (" ").
   - **Example:**

     name = "Ali"

   In this example, name is a string variable holding the text "Ali".
4. **Boolean (bool):**
   - **Definition**: Stores one of two values, either True or False.
   - **Example:**

     is_student = True

Here, is_student is a Boolean variable that stores the value True.

These are the basic types of variables in Python. Each type is used to store a different kind of data depending on your requirements.

Q.14. Describe how input and output operations are performed in Python. Give examples for each.

Ans. In Python, input and output operations are used to interact with the user. These operations allow the program to receive data from the user and display results on the screen.

1. Input Operation

- **Definition:** Input is used to take data from the user during program execution.
- **Function Used:** `input()` function
- **How it works:**
  - It is used to display a message on the screen asking for input.
  - The user types something and presses Enter.
  - The typed value is stored in a variable.
- **Example:**

```
File  Edit  Format  Run  Options  Window  Help
name = input("Enter your name: ")
```

This line asks the user to enter their name and stores it in the variable **name**.

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/inpuut.py
Enter your name:
```

2. Output Operation

- **Definition:** Output is used to display messages or results to the user.
- **Function Used:** print() function
- **How it works:**
  - It takes values (text, numbers, variables) and shows them on the screen.
- **Example:**

```
File  Edit  Format  Run  Options  Window  Help
name = input("Enter your name: ")

print("As - Salaam - Alaikum, " + name + "!")
```

This line shows a greeting message that includes the user's name entered earlier.

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/inpuut.py
Enter your name: Anum
As - Salaam - Alaikum, Anum!
>>>
```

Q.15. Describe the difference between integer and float data types in Python. Provide an example of each.

Ans. In Python, both **integer** and **float** data types are used to store numbers, but they are used for different kinds of values.

Integer (int)	Float (float)
Definition: An integer is a whole number that does not contain any decimal point.	**Definition:** A float is a number that includes a decimal point to show fractional values.
Usage: Integers are used when only whole numbers are needed, such as age or quantity.	**Usage:** Floats are used when precise measurements are required, like height or weight.
Decimal Point: Integer values do **not** have decimal points.	**Decimal Point:** Float values **do** include decimal points.
Example Code: user_age = **int**(input("Enter your age: ")) print("Your age is:", user_age)	**Example Code:** user_height = **float**(input("Enter your height in meters: ")) print("Your height is", user_height, "meter")
Sample Input: If the user enters 25, it is stored as an integer.	**Sample Input:** If the user enters 1.75, it is stored as a float.
Output Example: The program will display: Your age is: 25	**Output Example:** The program will display: Your height is 1.75 meter

Q.16. What are operators in Python? Explain arithmetic operators with examples.

Ans. Operators are special symbols used to perform operations on variables and values. They are used to carry out computations, comparisons, and logical decisions in a program.

> **Do You Know?**
> A tutorial on Python is available at
> https://docs.python.org/3/tutorial/

Python supports the following types of operators:

- Arithmetic operators
- Comparison operators
- Logical operators
- Assignment operators
- Bitwise operators, etc.

Arithmetic Operators in Python:

Arithmetic operators are used to perform basic mathematical operations on numeric values. The commonly used arithmetic operators are:

Operator	Name	Description	Example (a=10, b=3)	Result
+	Addition	Adds two operands	a + b	13
-	Subtraction	Subtracts second operand from the first	a - b	7
*	Multiplication	Multiplies both operands	a * b	30
/	Division	Divides first operand by second and give answer in floating points	a / b	3.33
//	Floor Division	Returns integer part of the division result	a // b	3
%	Modulus	Returns remainder after division	a % b	1
**	Exponentiation	Raises first operand to the power of second	a ** b	1000

Example Python Program:

File Edit Format Run Options Window Help

```
a = 10
b = 3
print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b)
print("a / b =", a / b)
print("a // b =", a // b)
print("a % b =", a % b)
print("a ** b =", a ** b)
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/arithmetic.py
a + b = 13
a - b = 7
a * b = 30
a / b = 3.3333333333333335
a // b = 3
a % b = 1
a ** b = 1000
>>>
```

Q.17. What are comparison operators in Python? Explain with suitable examples.

Ans. Comparison Operators in Python:

Comparison operators are used to compare two values or expressions. These operators check the relationship between the values and return a **boolean result: True or False.**

They are commonly used in **conditional statements and loops.**

Here are the comparison operators in Python:

Operator	Description	Example (x = 10, y = 5)	Result
>	Greater than	x > y	True
<	Less than	x < y	False
==	Equal to	x == y	False
!=	Not equal to	x != y	True
>=	Greater than or equal to	x >= y	True
<=	Less than or equal to	x <= y	False

Example Python Program:

```
File  Edit  Format  Run  Options  Window  Help
# Define variables
x, y = 10, 5
# Greater than
print(x, ">", y, "=", x > y)
# Less than
print(x, "<", y, "=", x < y)
# Equal to
print(x, "==", y, "=", x == y)
# Not equal to
print(x, "!=", y, "=", x != y)
# Greater than or equal to
print(x, ">=", y, "=", x >= y)
# Less than or equal to
print(x, "<=", y, "=", x <= y)
```

Output:

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/comparison.py
10 > 5 = True
10 < 5 = False
10 == 5 = False
10 != 5 = True
10 >= 5 = True
10 <= 5 = False
>>>
```

Q.18. What are assignment operators in Python? Explain with examples.

Ans. Assignment Operators in Python:

Assignment operators are used to assign values to variables. The most basic assignment operator is the **equal sign** =, which assigns the value on the right to the variable on the left.

Python also provides **compound assignment operators** that combine arithmetic operations with assignment. **These operators help to write shorter and more efficient code.**

Here are common assignment operators:

Operator	Description	Example	Meaning
=	Assign value	a = 10	Assigns 10 to a
+=	Add and assign	a += 5	a = a + 5
-=	Subtract and assign	a -= 5	a = a - 5
*=	Multiply and assign	a *= 5	a = a * 5
/=	Divide and assign (float result)	a /= 5	a = a / 5
//=	Floor divide and assign	a //= 5	a = a // 5
%=	Modulus and assign	a %= 5	a = a % 5
**=	Exponentiate and assign	a **= 2	a = a ** 2

Example Python Program:

This program shows how **assignment and compound assignment operators** work in Python.

- It begins by assigning initial values to two variables, a and b.
- Then, a series of operations are performed on a using operators like **+=, -=, *=, /=, //=, %=, and **=.**
- After each operation, the updated value of a is printed, showing how its value changes step by step.

File Edit Format Run Options Window Help

```
# Define initial values
a = 10
b = 5
# Simple assignment
assignment = a
print("a =", assignment)

# Addition assignment
a += b
print("a after addition = ", a)

# Subtraction assignment
a -= b
print("a after subtraction = ", a)

# Multiplication assignment
a *= b
print("a after multiplication = ", a)

# Division assignment
a /= b
print("a after division = ", a)

# Floor division assignment
a //= b
print("a after floor division = ", a)

# Modulus assignment
a %= b
print("a after modulus = ", a)

# Exponentiation assignment
a **= b
print("a after exponentiation = ", a)
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/assignmnet.py
a = 10
a after addition =  15
a after subtraction =  10
a after multiplication =  50
a after division =  10.0
a after floor division =  2.0
a after modulus =  2.0
a after exponentiation =  32.0
>>>
```

Q.19. What are logical operators in Python? Explain with the help of suitable examples.

Ans. Logical operators are used to combine multiple conditions or expressions. They return a Boolean result (True or False) based on the logic of the expressions being evaluated. **These operators are mostly used in decision-making statements like if, while, and for loops.**

Types of Logical Operators:

Operator	Description	Example (x=True, y=False)	Result
and	Returns True if both conditions are true	x and y	False
or	Returns True if at least one is true	x or y	True
not	Reverses the result (True becomes False)	not x	False

Truth Table for and Operator:

x	y	x and y
True	True	True
True	False	False
False	True	False
False	False	False

Truth Table for or Operator:

x	y	x or y
True	True	True
True	False	True
False	True	True
False	False	False

Example Python Program:

```
File  Edit  Format  Run  Options  Window  Help
# Define Boolean variables
x = True
y = False

# Logical AND
logical_and = x and y
print(x, "and", y, "=", logical_and)

# Logical OR
logical_or = x or y
print(x, "or", y, "=", logical_or)

# Logical NOT
logical_not_x = not x
print("not", x, "=", logical_not_x)

logical_not_y = not y
print("not", y, "=", logical_not_y)
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/and or not.py
True and False = False
True or False = True
not True = False
not False = True
```

Q.20. What is an expression in Python? Explain with examples.

Ans. Expression in Python:

An **expression** is a combination of **variables**, **values**, and **operators** that Python evaluates to produce a **result**. Expressions are used to perform calculations, assign values, and control the flow of programs.

Types of Expressions:

- **Arithmetic expressions:** Involve arithmetic operators (+, -, *, /, etc.)
- **Logical expressions:** Involve logical operators (and, or, not)
- **Relational expressions:** Involve comparison operators (>, <, ==, etc.)

Examples:

```
File  Edit  Format  Run  Options  Window  Help
# 1.    Simple arithmetic expression:
result = 3 + 4
print ("Simple arithmetic expression result:", result)
# 2.    Expression with variables:
a = 5
b = 2
result = a * b
print ("Expression with variables result:", result)
#3.    Expression with parentheses to control order:
result = (3 + 4) * 2
print ("Expression with parentheses to control order result:", result)
# Here, (3 + 4) is evaluated first, giving 7. Then 7 * 2 = 14.
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/expressions.py
Simple arithmetic expression result: 7
Expression with variables result: 10
Expression with parentheses to control order result: 14
>>>
```

Note: Parentheses () are used to change the default order of operations in expressions.

Q.21. What is operator precedence in Python? Explain with examples.

Ans. Operator Precedence in Python:

Operator precedence refers to the order in which operations are performed in a Python expression. Just like in mathematics, Python follows specific rules to decide which operation to perform first in complex expressions. Operators with **higher precedence** are evaluated **before** those with **lower precedence**.

Rules of Operator Precedence (from highest to lowest):

Precedence Level	Operator(s)	Description	Example	Result
1 (Highest)	()	**Parentheses** – operations inside are done first	(3 + 2) * 4	20
2	**	**Exponentiation** (power)	2 ** 3	8
3	*, /, //, %	**Multiplication, Division, Floor Division, Modulus**	4 * 3 = 12, 11 % 3 = 2	
4 (Lowest)	+, -	**Addition and Subtraction**	5 + 2 = 7, 10 - 4 = 6	

Important Note:

If **two operators of the same precedence level** appear in an expression, Python evaluates them **from left to right** (except for exponentiation **, which is right to left).

Example Expression:

Example 1:

File Edit Format Run Options Window Help

```
# Expression: 3 + 2 * 5
# According to operator precedence:
# First: 2 * 5 = 10
# Then: 3 + 10 = 13

result = 3 + 2 * 5
print("3 + 2 * 5 =", result)

# If you want to change the order of evaluation, use parentheses:
# Expression: (3 + 2) * 5
# Parentheses are evaluated first

result = (3 + 2) * 5
print("(3 + 2) * 5 =", result)
```

Output:

File Edit Shell Debug Options Window Help

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/precedence1.py
3 + 2 * 5 = 13
(3 + 2) * 5 = 25
>>>
```

Example 2:

File Edit Format Run Options Window Help

```
# Expression: 10 - 4 + 2
# Subtraction and addition have the same precedence.
# So evaluate left to right:
# 10 - 4 = 6
# 6 + 2 = 8

print("10 - 4 + 2 = ", 10 - 4 + 2)
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/precedence2.py
10 - 4 + 2 =  8
>>>
```

Example 3:

```
File  Edit  Format  Run  Options  Window  Help
# Expression: 2 ** 3 ** 2
# Exponentiation is evaluated right to left:
# 3 ** 2 = 9
# 2 ** 9 = 512

print("2 ** 3 ** 2 = ", 2 ** 3 ** 2)
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/precedence3.py
2 ** 3 ** 2 =  512
>>>
```

Tip: Use parentheses () to clearly define the order of operations and avoid confusion in complex expressions.

Q.22. Compute the following expressions.

1. **10+3*2 ** 2-5/5**
2. **(10+3) * (2 ** (2-1))/5**

Ans.

Expression 1:

10 + 3 * 2 ** 2 - 5 / 5

Step-by-step evaluation:

1. **Exponentiation (**):**
   2 ** 2 = 4
   → Now the expression is: 10 + 3 * 4 - 5 / 5
2. **Multiplication (*) and Division (/)** (left to right):
   3 * 4 = 12
   5 / 5 = 1.0
   → Now the expression is: 10 + 12 - 1.0
3. **Addition and Subtraction** (left to right):
   10 + 12 = 22
   22 - 1.0 = 21.0

Final Answer: 21.0

Expression 2:

(10 + 3) * (2 ** (2 - 1)) / 5

Step-by-step evaluation:

1. **Innermost parentheses (2 - 1)**
   Parentheses () have the highest precedence. So, solve the innermost parentheses first.
   2 - 1 = 1
   → Now the expression becomes:
   (10 + 3) * (2 ** 1) / 5
2. **Left-side parentheses (10 + 3)**
   Now both parentheses are at the same level of precedence, so we solve them **from left to right.**
   10 + 3 = 13
   → Now the expression becomes:
   13 * (2 ** 1) / 5
3. **Next parentheses (2 ** 1)**
   2 ** 1 = 2
   → Now the expression becomes:
   13 * 2 / 5
4. **Multiplication and Division (left to right):**
   - First, we perform multiplication:
     13 * 2 = 26
   - Then, we perform division:
     26 / 5 = 5.2

Final Answer: 5.2

Q.23. Translate the following in Python's syntax:

1. $5+3\times4-72^2+1$
2. $10\times(2+3)-4\text{^}2\ 8:2+5$

Ans.

1. 5 + 3 × 4 - 7 2^2 + 1

Translated to Python syntax:

5 + 3 * 4 - 7 * 2**2 + 1

Explanation:

- × becomes *
- Missing operator between 7 and 2^2 implies multiplication → 7 * 2**2
- ^ is not exponentiation in Python → use ** instead

2. 10 × (2 + 3) - 4^2 8:2 + 5

Translated to Python syntax:

10 * (2 + 3) - 4**2 * 8 / 2 + 5

Explanation:

- × becomes *
- ^ becomes **
- : is assumed to mean division → becomes /
- No operator between 4^2 and 8 implies multiplication → 4**2 * 8

Do You Know?
Using parentheses can help clarify complex expressions and ensure the operations are performed in the desired order.

Q.24. What are control structures in programming? Name the two main types of control structures and briefly explain each with one example in Python.

Ans. Control Structures in Programming:

In programming, control structures are used to control the flow of a program. They allow the program to make decisions or repeat actions depending on certain conditions. This helps the program behave in different ways based on different situations, just like how we make decisions in real life.

There are **two main types of control structures:**

1. Decision Making
2. Looping

1. Decision Making in Python

Decision making allows the program to choose different actions based on conditions. Python provides different conditional statements to perform decision making. These include:

- if statement
- if-else statement
- if-elif-else statement
- Short-hand if-else statement

Each of these is explained below.

1.1 if Statement

The if statement is used to check a condition. If the condition is true, a block of code will run.

Syntax:

```
if condition:
    # code to run if the condition is true
```

Example:

```
File  Edit  Format  Run  Options  Window  Help
temperature = 45
if temperature > 30:
    print("It's a hot day")
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/cond.py
It's a hot day
>>>
```

Explanation:

In this example, the condition temperature > 30 is true, so the message "It's a hot day" will be printed.

1.2 if-else Statement:

The if-else statement runs one block of code if the condition is true, and another block if the condition is false.

Syntax:

```
if condition:
    # code to run if the condition is true
else:
    # code to run if the condition is false
```

Example:

```
File  Edit  Format  Run  Options  Window  Help
temperature = 15
if temperature > 30:
    print("It's a hot day")
else:
    print("It's not a hot day")
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/cond.py
It's not a hot day
>>>
```

Explanation:

In this case, the temperature is not greater than 30, so the program will print "It's not a hot day".

1.3 if-elif-else Statement:

The if-elif-else statement is used when we want to check **multiple conditions**. It allows the program to run different blocks of code depending on which condition is true.

Syntax:

```
if condition1:
    # code to run if condition1 is true
elif condition2:
    # code to run if condition2 is true
else:
    # code to run if none of the above conditions are true
```

Example:

```
File  Edit  Format  Run  Options  Window  Help
weather = "cloudy"
if weather == "sunny":
    print("Wear sunglasses")
elif weather == "rainy":
    print("Take an umbrella")
else:
    print("Enjoy your day!")
```

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/multiple cond.py
Enjoy your day!
>>>
```

Explanation:

In this example, the program checks the value of the variable weather. If it is "sunny", it prints "Wear sunglasses". If it is "rainy", it prints "Take an umbrella". If the weather is neither sunny nor rainy, it prints "Enjoy your day!". This is useful when we have more than two possible conditions to check.

1.4 *Shorthand* if-else Statement:

Python also allows us to write the if-else statement in a shorter form. This is useful when the condition and actions are simple.

Syntax:

```
action_if_true if condition else action_if_false
```

Example:

```
File  Edit  Format  Run  Options  Window  Help
temp = 15
print("It's a hot day") if temp > 30 else print("It's not a hot day")
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/shortand.py
It's not a hot day
>>>
```

Explanation:

This is a shorter version of the if-else statement. It checks the condition and prints one of the two messages based on whether the condition is true or false.

Q.25. Write a program to calculate Body Mass Index (BMI).

Ans. Ask the user for their weight and height, then compute and display their BMI and classification. The Body Mass Index (BMI) is calculated using the formula given below:

BMI = weight(kg) / (height (m))^2

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
# Ask the user for weight and height
weight = float(input("Enter your weight in kilograms (kg): "))
height = float(input("Enter your height in meters (m): "))

# Calculate BMI using the correct formula (height squared)
bmi = weight / (height ** 2)

# Display BMI
print("Your BMI is:", round(bmi, 2))

# Classification based on BMI
if bmi < 18.5:
    print("Classification: Underweight")
elif bmi >= 18.5 and bmi < 24.9:
    print("Classification: Normal weight")
elif bmi >= 25 and bmi < 29.9:
    print("Classification: Overweight")
else:
    print("Classification: Obese")
```

Output:

```
Enter your weight in kilograms (kg): 34
Enter your height in meters (m): 5.6
Your BMI is: 1.08
Classification: Underweight
```

Q.26. Write an if-else statement and a short-hand if-else statement to check if a number is even or odd and print the appropriate message.

Ans. Using if-else Statement:

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
number = 16                    # You can change this number for testing
if number % 2 == 0:
    print("The number is even.")
else:
    print("The number is odd.")
```

Explanation:

This if-else statement checks if the number is divisible by 2 (i.e., if the remainder is 0). If true, it prints "The number is even." Otherwise, it prints "The number is odd."

Using Short-hand if-else Statement:

Python Code:

```
number = 16                    # You can change this number for testing
print("The number is even.") if number % 2 == 0 else print("The number is odd.")
```

Explanation:

This is a more compact version of the if-else statement. It checks if the number is even or odd in a single line.

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/check numnebr.py
The number is even.
>>>
```

Q.27. Write a Python program that takes a number as input and checks whether it is positive, negative, or zero using an if-elif-else statement.

Ans. Python Code:

```
File  Edit  Format  Run  Options  Window  Help
number = int(input("Enter a number: "))
if number > 0:
    print("The number is positive.")
elif number < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

Explanation:

This program takes input from user and checks whether the number is positive, negative, or zero using an if-elif-else statement.

1. If the number is greater than 0, it prints "The number is positive."
2. If the number is less than 0, it prints "The number is negative."
3. If neither of these conditions is true (i.e., the number is 0), it prints "The number is zero."

Output:

```
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/check pos,neg.py
Enter a number: 0
The number is zero.
>>>
```

Q.28. What is a nested if statement in Python? Explain how it works and provide an example.

Ans. A **nested if statement** is when an if statement is placed inside another if statement. This is useful when you need to make decisions based on multiple conditions. In Python, a nested if statement allows you to check one condition first, and then check additional conditions within that block if the first condition is true.

Nested if statements help to make decisions based on multiple levels of conditions, which allows you to create more complex decision-making structures in your program.

Syntax of a Nested if Statement:

The general syntax of a nested if statement is:

```
if condition1:
    if condition2:
        # code to run if both condition1 and condition2 are true
    else:
        # code to run if condition1 is true but condition2 is false
else:
    # code to run if condition1 is false
```

- The **first if** checks the first condition.
- The **second if** (inside the first if) checks the second condition.
- If the second condition is **true**, it runs the code inside the inner if.
- If the second condition is **false**, it runs the code inside the else of the inner if.
- If the **first condition** is **false**, the program skips the inner if-else and runs the code in the **outer else**.

Example:

Here's an example where we check the weather and temperature:

- If the weather is **rainy** and the temperature is below 5°C, you wear a **raincoat**.
- If the weather is **rainy** but the temperature is **5°C or higher**, you take an **umbrella**.
- If the weather is **not rainy**, you just **enjoy the day**.

Here's how to write that using a nested if statement:

Explanation:

1. **First Condition**: The program first checks if the **weather is rainy** using the if weather == "rainy" statement.
2. **Second Condition (Nested)**: If the weather is rainy, the program checks if the **temperature is below 5°C** using the nested if temperature < 5.
   - If both conditions are true, it prints "Wear a raincoat".
   - If the weather is rainy but the temperature is 5°C or higher, it prints "Take an umbrella".
3. **Else**: If the weather is not rainy, it prints "Enjoy your day!" as the action to take.

Python Code:

File Edit Format Run Options Window Help

```
# Example of nested if statement for weather and temperature

weather = input("Enter the weather (rainy/sunny): ")
temperature = int(input("Enter the temperature in °C: "))

if weather == "rainy":
    if temperature < 5:
        print("Wear a raincoat.")
    else:
        print("Take an umbrella.")
else:
    print("Enjoy your day!")
```

Output:

```
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/nested if.py
Enter the weather (rainy/sunny): sunny
Enter the temperature in °C: 23
Enjoy your day!
>>>
```

Q.29. Explain the concept of loops in Python. Describe the two main types of loops (while loop and for loop), providing their syntax and examples. Also, explain how the range() function is used in a for loop.

Ans. Loops are a type of control structure that help repeat a block of code multiple times until a certain condition is met. This is useful for performing repetitive tasks, reducing redundancy in code, and increasing efficiency. There are **two main types of loops** in Python: **while loops** and **for loops**.

1. While Loop:

The **while loop** in Python repeatedly executes a block of code as long as a given condition is true. It checks the condition **before** each round (iteration) of the loop. and if the condition becomes false, the loop terminates.

Syntax of the while loop:

```
while condition:
    # code to run while the condition is true
```

- **condition**: The condition is checked before each iteration(repetition). If it is true, the loop runs the block of code; if false, the loop stops.

Example: Let's consider an example where we want to print numbers starting from 1 and stop when the number reaches 10.

Python Code:

File Edit Format Run Options Window Help

```
number = 1
while number < 10:
    print(number)
    number = number + 1
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/while.py
1
2
3
4
5
6
7
8
9
>>>
```

Explanation:

In this example, the while loop starts with a variable number set to 1. The loop checks if number is less than 10. If the condition is **true**, it prints the current value of number and then increases it by 1 using number += 1. This process continues (repeats) until the value of number becomes 10. At that point, the condition becomes **false**, and the loop stops.

Note: In Python, **indentation** (spaces before the code inside the loop) is very important. It tells Python which lines of code belong to the loop block. Without proper indentation, Python will give an error.

2. The For Loop:

The **for loop** in Python is a control structure that is used to repeat a block of code for each item in a sequence such as a list, tuple, string, or a range of numbers.

A for loop goes through each item in the sequence one by one and runs the same code for each item. It is useful when we know **how many times,** we want to repeat something or when we want to perform an action for every item in a group.

Syntax of the for loop:

```
for variable in sequence:
    # code to run for each element in the sequence
```

- **variable**: The loop variable takes the value of each item in the sequence one at a time.
- **sequence**: This is the collection of items (like a list, tuple, or string) that will go through one by one.

Example 1:

Assume we want to greet each of our friends from a list of names:

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
friends = ["Ahmad", "Ali", "Hassan"]
for friend in friends:
    print("As-Salaam-Alaikum", friend)
```

Output:

```
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/for looop.py
As-Salaam-Alaikum Ahmad
As-Salaam-Alaikum Ali
As-Salaam-Alaikum Hassan
>>>
```

Explanation:

In this example, we have a list named friends that contains three names: "Ahmad", "Ali", and "Hassan". The for loop is used to go through each name in the list one by one.

- The loop variable friend takes the value of each name from the list during each round (iteration) of the loop.
- In the first iteration, friend becomes "Ahmad", and the program prints: **As-Salaam-Alaikum Ahmad**
- In the second iteration, friend becomes "Ali", and the program prints: **As-Salaam-Alaikum Ali**
- In the third iteration, friend becomes "Hassan", and the program prints: **As-Salaam-Alaikum Hassan**

This way, the for loop repeats the same print statement for each friend in the list.

3. The range() Function:

The range() function in Python is used to create a list of numbers. It is mostly used in **for loops** to tell how many times the loop should run. We can give it a **starting number**, a **stopping number**, and a **step size** (how much to increase each time).

It helps repeat something a fixed number of times easily.

Syntax of range():

```
range(stop)
# Generates numbers from 0 to stop - 1

range(start, stop)
# Generates numbers from start to stop - 1

range(start, stop, step)
# Generates numbers from start to stop - 1, incrementing by step
```

- **start**: The starting value (inclusive) means included. Default is 0.
- **stop**: The ending value (exclusive) means not included.
- **step**: The increment (or step size) between each number. Default is 1.

Example:

If you want to print numbers from 0 to 4, you can use the range() function in a for loop as follows:

Python Code:

```
Edit  Format  Run  Options  Window  Help
for i in range(5):
    print(i)
```

Explanation:

The range(5) means:
"Start from 0 and go up to 4" (it stops before 5).
The for loop takes each number from the range and prints it.

Output :

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/somple for.py
0
1
2
3
4
>>>
```

You can also use the range() function to specify a starting point and step size.

For example:

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
for i in range(2, 11, 2):
    print(i)
```

Output :

```
Type "help", "copyright", "credits" or "license()" for more information.
>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/somple for.py
2
4
6
8
10
>>
```

Class Activity

Activity 1: Using a while loop to print even and count the odd numbers from 1 to 20.

Python Code:

```
number = 1
even_count = 0
odd_count = 0

while number <= 20:
    if number % 2 == 0:
        print(f"{number} is even")
        even_count += 1
    else:
        odd_count += 1
    number += 1

print(f"Total even numbers: {even_count}")
print(f"Total odd numbers: {odd_count}")
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/activity 1.py
2 is even
4 is even
6 is even
8 is even
10 is even
12 is even
14 is even
16 is even
18 is even
20 is even
Total even numbers: 10
Total odd numbers: 10
>>>
```

Activity 2: Using a for loop to iterate over a list of temperatures and print "Warm" for temperatures above 20 degrees and "Cold" for temperatures 20 degrees or below.

Python Code:

```
temperatures = [15, 25, 30, 10, 18, 22]
for temp in temperatures:
    if temp > 20:
        print(f"{temp}°C is Warm")
    else:
        print(f"{temp}°C is Cold")
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/activty 2.py
15°C is Cold
25°C is Warm
30°C is Warm
10°C is Cold
18°C is Cold
22°C is Warm
>>>
```

Activity 3: Write a for loop using range() to print the even numbers from 2 to 10.

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
print("Even number from 2 to 10 are:")
# Starts from 2, goes up to 10 (not 11), step is 2 (even numbers)
for i in range(2, 11, 2):
    print(i)
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/activty 3.py
Even number from 2 to 10 are:
2
4
6
8
10
>>>
```

Activity 4: Write a Python program that prints the first 10 multiples of 3 using a for loop and the range() function.

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
print("First 10 multiples of 3 are:")

for i in range(1, 11): # Loop from 1 to 10 (but don't include 11)
    print(i * 3)       # Multiply each number by 3 to get multiples of 3
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/activity 4.py
First 10 multiples of 3 are:
3
6
9
12
15
18
21
24
27
30
>>>
```

Q.30. Define a function in Python and explain how it works. How do you invoke a function? Explain function parameters, return values, default parameters, and keyword arguments with examples. Also, demonstrate the usage of *args and kwargs.

Ans. Defining a Function:

A **function** in Python is a block of code that performs a specific task. It is defined using the **def** keyword, followed by the function name and parentheses (). Inside the parentheses, you can specify **parameters** (input values), which the function will use. The body of the function contains the logic that will be executed when the function is called.

> **Do You Know?**
> You can call a function multiple times with different arguments to reuse the same code for different inputs.

Advantages of Using Functions:

1. **Code Reusability:**
   - You write a function once and reuse it wherever needed.
   - This saves time and avoids repetition.
2. **Easy to Read and Manage:**
   - Functions break the program into smaller parts.
   - This makes the code **clean, neat, and easier to understand.**
3. **Avoids Repetition:**
   - Instead of copying the same lines of code, just **call the function.**
4. **Makes Testing and Debugging Easier:**
   - If there's an error, you can test the function separately.
   - Fixing issues becomes faster and simpler.
5. **Improves Teamwork and Collaboration:**
   - In large programs, different team members can work on **different functions.**
   - This makes **group work more efficient.**
6. **Helps with Logical Thinking:**
   - Writing functions helps students and programmers **think logically** by breaking problems into **smaller tasks.**

Example:

Here is an example of how to define a simple function that prints a greeting message:

Python Code:

```
def greet(name):
    print("As-Salaam-Alaikum", name)
```

In this example:

- **def** is the keyword that starts the function definition.
- **greet** is the function name.
- **name** is the parameter (input) the function takes.

Calling (Invoking) a Function:

To **invoke** or **call** a function, you simply write its name followed by parentheses, and pass the necessary arguments if required.

Example:

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
def greet(name):                          # Defining the greet function
    print("As-Salaam-Alaikum", name)
greet("Bilal")                            # Calling the greet function
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/hp/OneDrive/Documents/python code/func.py ========
As-Salaam-Alaikum Bilal
>>>
```

This calls the greet function and passes "Ali" as the argument, which is used in the function to print the greeting.

Function Parameters and Return Values:

Functions can also **return values**. You can use the return statement to send a result back to the caller.

Example:

Here is an example of a function that adds two numbers and returns their sum:

Python Code:

```
def add (a, b):
    return a + b
# You can call this function with two arguments to get the result:
result = add(3, 5)
# Calling the function and storing the result
print("SUM = ", result)
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/parameter and return.py
SUM =  8
>>>
```

Default Parameters:

A function can have **default values** for its parameters. These default values are used if no argument is passed for that parameter during the function call.

Example with Default Parameter:

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
def greet(name="Student"):
    return "As-Salaam-Alaikum, " + name + "!"
# If you call the function without passing any argument:
print(greet())
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/defult.py
As-Salaam-Alaikum, Student!
>>>
```

 # But if you provide a name:

```
File  Edit  Format  Run  Options  Window  Help
def greet(name="Student"):
    return "As-Salaam-Alaikum, " + name + "!"
# If you call the function without passing any argument:
print(greet("Bilal"))
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/defult.py
As-Salaam-Alaikum, Bilal!
>>>
```

In this example, the parameter name has a default value of "Student". If no argument is provided, it uses this default value.

Class Activity

Activity 1: Define a function that takes a list of numbers and returns the maximum value.

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
def find_max(numbers):
    return max(numbers)
print("Maximum number is: ", find_max([4, 9, 2, 7]) )
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/func activitty.py
Maximum number is:  9
>>>
```

Explanation:

- The function find_max() takes a list of numbers.
- It uses the built-in Python function max() to find and return the largest number in the list.

Q.31. What are Python libraries and modules? Explain with examples how standard libraries such as random, datetime, and statistics are imported and used in Python. Also, explain the benefit of using libraries in Python programming.

Ans. Python Libraries and Modules:

In Python, **libraries and modules** are like toolkits that contain many useful functions and features. These help programmers solve different problems easily, without writing all the code themselves.

- A **module** is a file that contains Python code like functions, variables, or classes.
- A **library** is a collection of one or more modules that are grouped together for a specific purpose.

Python has many **built-in libraries** (also called standard libraries), which are already available in Python. Some common examples include random, datetime, and statistics.

How to Import and Use Libraries in Python:

We use the **import** statement in Python to bring a library into our program so we can use its features.

Example 1: random Library

```
File  Edit  Format  Run  Options  Window  Help
import random
# Generate a random number between 1 and 10
number = random.randint(1, 10)
print("The random number is:", number)
```

Explanation:

- The random library is used to create random numbers.
- The function random.randint(1, 10) generates a random number between 1 and 10.
- This is useful in games, simulations, or lucky draw programs.

Output :

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/random lib.py
The random number is: 4
>>>
```

Example 2: datetime Library

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
import datetime
# Get the current date and time
current_time = datetime.datetime.now()
print("Current date and time:", current_time)
```

Explanation:

- The datetime library is used to work with dates and times.
- The function datetime.datetime.now() gives the current date and time.
- It is useful for setting reminders, logging activities, or managing calendars.

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/datetime lib.py
Current date and time: 2025-05-23 22:35:40.633249
>>>
```

Example 3: statistics Library

```
File  Edit  Format  Run  Options  Window  Help
import statistics
# Calculate the mean of a list of numbers
data = [23, 45, 67, 89, 12, 44, 56]
mean_value = statistics.mean(data)
print("The mean value is:", mean_value)
```

Explanation:

- The statistics library is used for statistical calculations.
- statistics.mean(data) calculates the average (mean) of the numbers in the list.
- It is useful in data analysis, reports, and scientific work.

Output:

```
The mean value is: 48
```

Advantages of Using Libraries in Python:

1. Saves time and effort.
2. Reduces the need to write code from the beginning.
3. Makes the program shorter and easier to read.
4. Gives access to tested and error-free functions.
5. Helps focus only on the main problem instead of coding everything manually.

Tip for good practice:

Always import only the parts of the library you need. This keeps your program clean and faster.
You can explore more Python libraries by visiting the official website: **https://docs.python.org/3/**

Q.32. Write a Python program that imports the random and statistics libraries. Use random to generate a list of 10 random numbers between 1 and 50. Then, use statistics to calculate and print the mean (average) of those numbers.

Ans. Python Code:

```
File  Edit  Format  Run  Options  Window  Help
# Importing the required libraries
import random                          # For generating random numbers
import statistics                      # For calculating the average (mean)

# Create an empty list to store numbers
numbers = []

# Generate 10 random numbers between 1 and 50
for i in range(10):
    num = random.randint(1, 50)
    numbers.append(num)

# Print the list of random numbers
print("Random Numbers:", numbers)

# Find the average (mean) of the numbers
average = statistics.mean(numbers)

# Print the average value
print("Average of Numbers:", average)
```

Explanation:

1. **import random** – This allows us to make random numbers.
2. **import statistics** – This helps us to find the average.
3. Created an **empty list** named numbers.
4. Used a **loop** to make 10 random numbers between **1 and 50**, and add them to the list.
5. **Printed** the list of numbers.
6. Used statistics.mean() to find the **average**.
7. Finally, we **printed the average.**

Output:

```
Random Numbers: [17, 29, 16, 30, 2, 24, 23, 50, 20, 13]
Average of Numbers: 22.4
```

Q.33. What is the Module Search Path in Python? Explain with examples how Python finds modules. Also explain what a package is in Python, and how it helps in organizing code. Write examples to support your answer.

Ans. Module Search Path in Python:

When we use the import statement in Python to include a module, Python searches for that module in a specific order. This path is known as the **module search path.**

Let us understand how Python searches for a module using simple steps and examples.

Tidbits:
Organizing your modules into packages is like organizing books into sections of a library—it makes finding and maintaining your code much easier.

1. Current Directory:

First, Python checks the **current folder** where your main program file is saved. If it finds the required module in the same folder, it imports it directly.

Example:

Suppose we have two files in the same folder:

File: helper.py

```
def greet():
    return "As-Salaam Alaikum from helper.py!"
```

File: main.py

```
import helper
message = helper.greet()
print(message)
```

Output:

As-Salaam Alaikum from helper.py!

Explanation:

Since helper.py is in the same folder as main.py, Python finds and imports it successfully.

2. Standard Library

If Python does not find the module in the current directory, it checks its **built-in standard library,** which *includes many* useful modules.

Example:

main.py

```
import random
number = random.randint(1, 10)
print(f"Random number: {number}")
```

Output:

Random number: 7 (or any number between 1 and 10)

Explanation:

The random module is part of Python's standard library, so Python can find and use it even if it's not in your folder.

6 **Package Structure in Python:**

A **package** in Python is a folder (directory) that contains several related **modules**. It helps in keeping the project organized, especially when the project is large. It is just like arranging books in different sections in a library.

Example:

Suppose we are building an **e-commerce system**. We create a package named ecommerce, and inside it, we add several files:

Folder Structure:

```
ecommerce/
  └── products.py
  └── customers.py
  └── orders.py
```

File: ecommerce/products.py

```
def list_products():
    return ["Laptop", "Mobile", "Tablet"]
```

File: main.py

```
from ecommerce import products

available_products = products.list_products()
print(available_products)
```

Output:

['Laptop', 'Mobile', 'Tablet']

Explanation:

Here, ecommerce is the **package**, and products.py is the **module** inside that package. This makes the code neat and easy to manage.

Q.34. What is a list in Python? Explain how to create, access, modify, and perform different operations on lists using examples. Also describe important list methods: append(), sort(), remove(), and slicing.

Ans. **Built-in Data Structures: Lists**

Python provides built-in data structures to store and manage data. One of the most commonly used data structures is the **list**.

What is a List?

A **list** is a collection of items written inside square brackets [ ]. Each item is separated by a comma. A list can hold **numbers, text (strings)**, or even other lists. Lists are very flexible and allow us to **add, change, or remove** items easily.

1. Creating, Accessing, and Modifying Lists:

Creating a List:

We can create a list by placing items inside square brackets:

File Edit Format Run Options Window Help

```
fruits = ["Mango", "Apple", "Banana"]
print(fruits)
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/lis.py
['Mango', 'Apple', 'Banana']
>>
```

Explanation:

Here, we created a list named fruits which contains three fruit names: "Mango", "Apple", and "Banana". The list is then printed using the print() function. Lists can contain any data type, and the items are stored in the same order they are entered.

Accessing List Items:

Each item in a list has an **index number** starting from 0. We can use this index number to access any item.

```
fruits = ["Mango", "Apple", "Banana"]
print(fruits[1])
```

Output:

```
Apple
```

Explanation:

In Python, indexing starts from 0, so fruits[1] refers to the **second item** in the list, which is "Apple". This is how we access individual elements in a list.

Modifying a List:

We can **change the value** of a list item by using its index and **add** a new item using the append() method.

File Edit Format Run Options Window Help

```
fruits = ["Mango", "Apple", "Banana"]
fruits[0] = "Orange"
print("List before modification:", fruits)
fruits.append("Pineapple")
print("List after modification:", fruits)
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/lis.py
List before modification: ['Orange', 'Apple', 'Banana']
List after modification: ['Orange', 'Apple', 'Banana', 'Pineapple']
>>>
```

Explanation:

- fruits[0] = "Orange" changes the **first item** from "Mango" to "Orange".
- fruits.append("Pineapple") adds "Pineapple" to the **end of the list.**
- The print() function shows the updated list.

Inserting an item at a specific index using insert()

We can also insert an item at any **specific index** in the list using the insert() method.

File Edit Format Run Options Window Help

```
fruits = ["Mango", "Apple", "Banana"]
fruits.insert(1, "Grapes")                    # Insert "Grapes" at index 1
print(fruits)
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/ins.py
['Mango', 'Grapes', 'Apple', 'Banana']
>>>
```

Explanation:

- *fruits.insert*(1, "Grapes") inserts "Grapes" at index 1, pushing the existing items "Apple" and "Banana" to the right.
- The print() function shows the list after the insertion.

2. List Methods and Their Use:

Python provides useful built-in **methods** to work with lists:

- append(item) → Adds an item at the end.
- remove(item) → Removes the first occurrence of the item.

- sort() → Sorts the list in ascending (A-Z) order.
- reverse() → Reverses the order of the list.

Example:

```
File  Edit  Format  Run  Options  Window  Help
students = ["Zain", "Areeba", "Bilal"]
students.append("Fatima")
students.sort()
print(students)
```

Output:

```
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/lis app.py
['Areeba', 'Bilal', 'Fatima', 'Zain']
>>>
```

Explanation:

- append("Fatima") adds a new student to the list.
- sort() arranges the names in **alphabetical order** (A to Z).
- The final sorted list is printed.

3. List Operations: Slicing and Concatenation:

Slicing

Slicing allows us to get a **part of the list** using the format **`list[start:end]`**.

```
File  Edit  Format  Run  Options  Window  Help
numbers = [1, 2, 3, 4, 5]
slice = numbers[1:4]
print(slice)
```

Output:

```
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/lis slice.py
[2, 3, 4]
>>>
```

Explanation:

- numbers[1:4] means we want the items from **index 1 to index 3** (4 is not included).
- So we get the numbers 2, 3, and 4.

Concatenation:

You can **join two lists** using the + operator.

```
numbers = [1, 2, 3, 4, 5]
slice = numbers[1:4]
print("sliced list is:", slice)
extra_numbers = [6, 7]
combined = slice + extra_numbers
print("Slice + extra numbers = ", combined)
```

Output:

```
>>>
======= RESTART: C:/Users/hp/OneDrive/Documents/python code/lis slice.py =======
sliced list is: [2, 3, 4]
Slice + extra numbers =  [2, 3, 4, 6, 7]
>>>
```

Explanation:

Here, slice and extra_numbers are combined using +. This gives one new list with all the elements.

4. Combined Example: Sort and Remove:

We can also use **remove()** to delete a specific item.

```
File  Edit  Format  Run  Options  Window  Help
student_names = ["Ahmed", "Sara", "Ali", "Hina"]
student_names.sort()
student_names.remove("Sara")
print(student_names)
```

Tidbits:
Use list methods like append() and remove () to efficiently manage and modify your lists. For larger projects, organizing data in lists helps keep your code clean and manageable.

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/s, rem.py
['Ahmed', 'Ali', 'Hina']
>>>
```

Explanation:

- sort() arranges the names in A-Z order.
- remove("Sara") removes "Sara" from the list.
- The final list is printed without "Sara".

Use list methods like append() and remove () to efficiently manage and modify your lists. For larger projects, organizing data in lists helps keep your code clean and manageable.

Class Activity

Q.35. Imagine you are maintaining a list of your favorite books: ["To Kill a Mockingbird", "1984", "The Great Gatsby", "Pride and Prejudice"].

Perform the following tasks using Python:

1. **Add a new book "Moby Dick" to the list.**
2. **Replace "1984" with "Brave New World".**
3. **Remove "The Great Gatsby" from the list.**
4. **Merge this list with another list of books: ["War and Peace", "Hamlet"].**
5. **Print the final list of books.**

Write the Python code to execute these tasks and print the final list of books.

Solution:

```
File  Edit  Format  Run  Options  Window  Help
# Step 1: Create the original list of favorite books
books = ["To Kill a Mockingbird", "1984", "The Great Gatsby", "Pride and Prejudice"]

# Step 2: Add a new book "Moby Dick"
books.append("Moby Dick")

# Step 3: Replace "1984" with "Brave New World"
index_1984 = books.index("1984")  # Find the index of "1984"
books[index_1984] = "Brave New World"  # Replace it

# Step 4: Remove "The Great Gatsby" from the list
books.remove("The Great Gatsby")

# Step 5: Merge with another list of books
more_books = ["War and Peace", "Hamlet"]
books = books + more_books  # Merge the two lists

# Step 6: Print the final list of books
print("Final list of favorite books:")
print(books)
```

Output:

```
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/activitly lis.py
Final list of favorite books:
['To Kill a Mockingbird', 'Brave New World', 'Pride and Prejudice', 'Moby Dick', 'War and Peace', 'Hamlet']
>>>
```

Q.36. What are tuples in Python? Explain with examples. Also define indexing and slicing. How are positive and negative indices used in sequences like lists or tuples? Support your answer with suitable examples.

Ans. Tuples in Python:

A **tuple** is a data structure in Python used to store a collection of items. It is similar to a list, but **tuples are immutable**, meaning we **cannot change their values** once they are created.

Example of a Tuple:

File Edit Format Run Options Window Help

```
# Creating a tuple
my_tuple = (1, 2, 3, "Hello", 4.5)

# Accessing elements by index
print("first element = ", my_tuple[0])      # Output: 1
print("fourth element = ", my_tuple[3])     # Output: Hello

# Finding the length of the tuple
print("Length of list = ", len(my_tuple))   # Output: 5
```

Tidbits:
Indexing and slicing are powerful tools for working with sequences in Python. Practice these techniques to become more proficient in manipulating data and accessing specific parts of sequences.

Explanation:

- The tuple my_tuple contains 5 elements of different types (integers, string, float).
- We use index numbers to access elements in a tuple.
- len() is used to find how many elements are in the tuple.

Output:

```
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/tpl.py
first element =  1
fourth element =  Hello
Length of list =  5
>>
```

Indexing and Slicing:

Python provides two important techniques to access and work with elements in sequences like **lists**, **tuples**, and **strings**.

1. Indexing:

Indexing is used to access individual elements in a tuple using their position number. Python uses **zero-based indexing**, so the first element has an index of 0, the second element has an index of 1, and so on.

Example:

```
fruits = ("Apple", "Banana", "Cherry")
print(fruits[0])
```

Output: Apple

```
print(fruits[1])
```

Output: Banana

Explanation:

- fruits[0] gives the first element: **"Apple"**
- fruits[1] gives the second element: **"Banana"**

2. Slicing:

Slicing is used to access a range of elements (a subset of the tuple).

The syntax is:

```
sequence[start : stop : step]
```

- start: the index where slicing starts (included)
- stop: the index where slicing ends (not included)
- step: optional, the jump between elements (default is 1)

Example:

```
File  Edit  Format  Run  Options  Window  Help
my_tuple = (10, 20, 30, 40, 50)

# Slicing a tuple
print("Sliced tuple =", my_tuple[1:4])
```

Output:

```
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/tp slice.py
Sliced tuple = (20, 30, 40)
>>>
```

Explanation:

This slices the tuple from index 1 to index 3 (4 is excluded), which returns a new tuple (20, 30, 40).

3. **Indexing and Slicing with Negative Indices:**

Python also allows **negative indexing**, where:

- -1 is the **last element**
- -2 is the **second last**, and so on

Example:

```
fruits = ("Apple", "Banana", "Cherry", "Date", "Elderberry")

# Indexing
print("First fruit:", fruits[0])    # Positive index
print("Last fruit:", fruits[-1])    # Negative index

# Slicing with positive indices
print("Fruits from index 1 to 3:", fruits[1:4])

# Slicing with negative indices
print("Fruits from index -4 to -1:", fruits[-4:-1])
```

Output:

```
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/neg indextuple.py
First fruit: Apple
Last fruit: Elderberry
Fruits from index 1 to 3: ('Banana', 'Cherry', 'Date')
Fruits from index -4 to -1: ('Banana', 'Cherry', 'Date')
>>>
```

Explanation:

- fruits[0] gives the first fruit: **Apple**
- fruits[-1] gives the last fruit: **Elderberry**
- fruits[-4:-1] returns 2nd to 4th elements (counting from the end)

Conclusion:

- **Tuples** store fixed (unchangeable) data, just like lists but immutable.
- **Indexing** helps us access single items, and **slicing** helps us get parts of sequences.
- Python supports both **positive** and **negative indexing** to make data access easier.

Q.37. Consider the following list, tuple, and string:

 # List:[10, 20,30, 40, 50, 60, 70, 80]
 # Tuple:("Math", "Science", "English", "History", "Geography")
 # String:"Python Programming"

Perform the following operations:

1. **Access and print the third element from each sequence (list, tuple, and string).**
2. **Slice and print elements from index 2 to 5 from the list and the tuple.**
3. **Slice and print characters from index 7 to the end of the string.**
4. **Use negative indexing to print the last two elements from the list and the tuple.**
5. **Use negative slicing to print characters from the second last to the last character of the string.**

Write the Python code to perform these operations and print the results.

Ans. Python Code:

File Edit Format Run Options Window Help

```
# List, Tuple, and String
my_list = [10, 20, 30, 40, 50, 60, 70, 80]
my_tuple = ("Math", "Science", "English", "History", "Geography")
my_string = "Python Programming"

# 1. Access and print the third element from each sequence (list, tuple, and string)
print("Third element from list:", my_list[2])
print("Third element from tuple:", my_tuple[2])
print("Third element from string:", my_string[2])

# 2. Slice and print elements from index 2 to 5 from the list and the tuple
print("Elements from index 2 to 5 in list:", my_list[2:6])
print("Elements from index 2 to 5 in tuple:", my_tuple[2:6])

# 3. Slice and print characters from index 7 to the end of the string
print("Characters from index 7 to the end in string:", my_string[7:])

# 4. Use negative indexing to print the last two elements from the list and the tuple
print("Last two elements from list:", my_list[-2:])
print("Last two elements from tuple:", my_tuple[-2:])

# 5. Use negative slicing to print characters from the second last to the last character of the string
print("Second last to last character in string:", my_string[-2:])
```

Explanation:

- **Accessing the third element**: We use [2] as indexing in Python starts from 0. The third element is at index 2.
- **Slicing from index 2 to 5**: The slice 2:6 will return elements starting from index 2 to index 5 (6 is excluded).
- **Slicing the string from index 7 to the end**: By using my_string[7:], we get all characters starting from index 7 till the end of the string.
- **Negative indexing**: -1 refers to the last element, -2 refers to the second-last element, and so on. Thus, my_list[-2:] gives the last two elements.
- **Negative slicing**: my_string[-2:] gives the last two characters of the string, using negative indexing.

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/tupl activ.py
Third element from list: 30
Third element from tuple: English
Third element from string: t
Elements from index 2 to 5 in list: [30, 40, 50, 60]
Elements from index 2 to 5 in tuple: ('English', 'History', 'Geography')
Characters from index 7 to the end in string: Programming
Last two elements from list: [70, 80]
Last two elements from tuple: ('History', 'Geography')
Second last to last character in string: ng
>>>
```

Q.38. Explain the concept of modular programming in Python. How does it help manage complex programs and reduce code duplication? Also, describe the role of the main function and show how to create and use Python modules.

Ans. Modular programming means breaking a large Python program into smaller parts called **modules**. Each module has some code that does a specific task. This makes the program easier to read, fix, and use again.

Modular programming helps:

- Manage big programs easily
- Avoid repeating the same code
- Reuse code in other programs
- Make the code easier to test and update

Tidbits:
Using the main function with modules helps keep your code organized, making it easier to maintain. Always use the main function to define the starting point of your program, and use modules to separate different parts of your code.

What is Code Duplication?

Code duplication means writing the same code in many places. This is not good because:

- It makes the program longer
- If you want to change the code, you have to change it everywhere
- It can cause mistakes

Do You Know?
Python's standard library is made up of hundreds of modules that you can be used to perform common tasks, like working with dates, generating random numbers, or reading files.

Modular programming fixes this by letting you write the code once and use it many times using a module.

What is the main() Function?

In Python, the main() function shows where the program should start. It is a good practice to write your main code inside a function called main().

We use this code to run the main function only when the file is executed directly:

Python code:

```
if __name__ == "__main__":
    main()
```

This line means:

- If the file is run directly → run the main() function
- If the file is imported into another file → do not run it automatically

Example: Writing a Main File

Create a file and name it **main.py**:

Python Code:

```
# main.py

def main():
    print("This is the main function.")

if __name__ == "__main__":
    main()
```

When you run this file, the output will be:

This is the main function.

How to Create and Use a Module?

A module is just a Python file (.py) that has some code, like functions, which can be used in other files.

Step 1: Create a module file

Name this file **greetings.py**:

Python Code:

```
def say_As_Salaam_Alaikum():
    return "As-Salaam-Alaikum, everyone!"
```

This module has a function you want to use in another file.

Step 2: Create the main program

Name this file **main.py**:

Python Code:

```
# main.py

import greetings                    # import the greetings module
def main():
    # call the function from module
        message = greetings.say_As_Salaam_Alaikum()
        print(message)

    if __name__ == "__main__":
        main()
```

When you run **main.py**, the output will be:

As-Salaam-Alaikum, everyone!

Now your program is organized:

- greetings.py has the reusable function
- main.py is the starting point of the program

Python's Standard Library:

Python comes with many built-in modules that you can use. These modules help you do common tasks without writing extra code.

Examples:

- random – to generate random numbers
- datetime – to work with dates and time
- os – to work with files and folders

These modules save time and make your programs better.

Class Activity

Q39. Create a Python module named calculator.py that includes two functions:

1. **add(a,b) - This function should return the sum of two numbers.**
2. **subtract (a,b) - This function should return the difference between two numbers. Then, write a script named main.py that imports your calculator module and uses these functions to perform the following:**
3. **Print the result of adding 15 and 8.**
4. **Print the result of subtracting 10 from 25.**

Make sure to run your main.py script and verify that the output is correct.

Solution:

1. Create the module file named calculator.py

Python Code:

File Edit Format Run Options Window Help

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

def subtract(a, b):
    return a - b
```

Explanation:

- add() function takes two numbers and returns their sum.
- subtract() function takes two numbers and returns the result after subtracting the second from the first.

2. Create the script file named main.py

Python Code:

```
File  Edit  Format  Run  Options  Window  Help
import calculator                    # This imports the calculator module

def main():
    # Call the add function
    result1 = calculator.add(15, 8)
    print("The sum of 15 and 8 is:", result1)

    # Call the subtract function
    result2 = calculator.subtract(25, 10)
    print("The result of subtracting 10 from 25 is:", result2)

# Call the main function
main()
```

Explanation:

- The main.py script imports the functions from calculator.py.
- It uses add(15, 8) to get the sum and subtract(25, 10) to get the difference.
- The results are printed using print().

Output when you run main.py:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/main.py
The sum of 15 and 8 is: 23
The result of subtracting 10 from 25 is: 15
>>>
```

Q.40. Define Object-Oriented Programming (OOP) and explain the concepts of classes and objects with examples. How do these concepts contribute to code organization and understanding in Python?

Ans. What is Object-Oriented Programming (OOP)?

Object-Oriented Programming (OOP) is a method of writing programs using **objects**. These objects are based on **classes**, which represents real world objects. OOP helps in breaking down a complex program into **smaller and reusable parts**. This makes it easier to write, understand, and manage code.

How OOP Helps in Python Programming:

1. **Easy to manage:** You break the code into small parts (classes/objects).
2. **Reusability:** Same class can be used to create many objects.
3. **Clear structure:** Group related data and behavior together.
4. **Security:** Access modifiers protect sensitive data.
5. **Real-life modeling:** Easy to represent real-world entities like cars, books, students, etc.

Class and Object

Class:

A class is a **blueprint** or **template** for creating objects. It defines the structure and behavior of an object. Inside a class, we write:

- **Attributes (Variables)** — describe the object (e.g., color, size)
- **Methods (Functions inside a class)** — define actions or behavior (e.g., describe a car)

Syntax Example:

```
File  Edit  Format  Run  Options  Window  Help
class ToyCar:
    def __init__(self, color, size, wheels):
        self.color = color          # attribute
        self.size = size            # attribute
        self.wheels = wheels        # attribute

    def describe(self):
        return f"This toy car is {self.color}, {self.size}, and has {self.wheels} wheels."
```

Explanation:

- **class ToyCar:**
  This line defines a **class** named ToyCar. A class is like a plan or design for making toy cars.
- **def init(self, color, size, wheels):**
  This is the **constructor method**. It runs automatically when we create a new toy car.
  - __init__ stands for initialize. It starts and sets values when an object is created.
  - Self refers to the object being created. It is used to access the variables and methods inside the class.
  - Color, size, and wheels are called parameters. They represent inputs given when creating an object e.g creating a new toy car.
- **self.color = color**
  This line saves the input value color into the object. It becomes the car's color.
- **self.size = size**
  Stores the size of the car in the object.
- **self.wheels = wheels**
  Stores the number of wheels the car has.
- **def describe(self):**
  This is a **method** (function inside a class). It gives a description of the toy car using the object's data.
- **return f"This toy car is {self.color}, {self.size}, and has {self.wheels} wheels."**
  This line returns a sentence showing the color, size, and number of wheels of the car.

Object:

An object is an **actual item created from a class**. You can make many objects from one class, and each will have its own values.

Example of creating objects:

Python Code:

```
car1 = ToyCar("red", "small", 4)
car2 = ToyCar("blue", "large", 6)

print(car1.describe())
print(car2.describe())
```

Explanation:

- **car1 = ToyCar("red", "small", 4)**
  This line creates the **first object** named car1 from the ToyCar class.
  - color = "red"
  - size = "small"
  - wheels = 4
    So, this toy car is red, small, and has 4 wheels.
- **car2 = ToyCar("blue", "large", 6)**
  This line creates **another object** named car2 from the same class, but with different values.
  - color = "blue"
  - size = "large"
  - wheels = 6
    So, this toy car is blue, large, and has 6 wheels.
- **print(car1.describe())**
  This calls the describe() method for car1. It returns a sentence describing the car's color, size, and number of wheels.
- **print(car2.describe())**
  This does the same, but for car2.

Complete Code:

```
File  Edit  Format  Run  Options  Window  Help
class ToyCar:
    def __init__(self, color, size, wheels):
        self.color = color        # attribute
        self.size = size          # attribute
        self.wheels = wheels.     # attribute

    def describe(self):
        return f"This toy car is {self.color}, {self.size}, and has {self.wheels} wheels."


car1 = ToyCar("red", "small", 4)
car2 = ToyCar("blue", "large", 6)

print(car1.describe())
print(car2.describe())
```

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/toycar.py
This toy car is red, small, and has 4 wheels.
This toy car is blue, large, and has 6 wheels.
>>>
```

Q.41. Explain the concept of Exception Handling in Python. Discuss try-except blocks with suitable examples.

Ans. Exception Handling in Python:

In Python, **exception handling** is a way to deal with errors that occurs during the execution of a program. Sometimes, an error can occur due to invalid input or a mistake in the program. Instead of stopping the program, Python provides a way to **catch** and **handle** these errors so that the program can continue or show a meaningful message to the user.

Python uses the **try**, **except**, **else**, and **finally** blocks to manage exceptions.

1. Try-Except Block:

The **try** block contains the code that might produce an error. If an error occurs, the **except** block is executed to handle the error.

Example:

```
File  Edit  Format  Run  Options  Window  Help
a = int(input("Enter a number: "))
try:
    result = 10 / a
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Explanation:

- If the user enters 0, it will cause a **ZeroDivisionError**.
- The error is caught in the **except** block, and a message is shown instead of crashing the program.

Output:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/t c.py
Enter a number: 0
You can't divide by zero!
>>>
```

Q.42. Explain file handling in Python. Describe how to open, read, write, and append data to a file with the help of suitable examples. Also explain different file modes used in Python.

Ans. File Handling in Python:

File handling is a way to **store and manage data** in files on the computer. It is used to **read from** or **write to** files. This helps us **save data permanently**, even after the program ends.

Python provides simple functions to handle files using open(), read(), write(), and close().

Importance:

File handling is very important for saving and managing data in Python. We can use different modes like **read, write, append**, and **read+write** depending on the task. The with statement is used to handle files safely and automatically close them after use.

1. Opening and Reading a File:

To read data from a file, we open it in **read mode** ("r"). We often use the with statement to make sure the file closes automatically.

Example:

File Edit Format Run Options Window Help

```
with open("file.txt", "r") as file:
    content = file.read()
    print(content)
```

Explanation:

- The file "file.txt" is opened in read mode.
- Its contents are stored in the variable **content** and printed.
- The **with statement** automatically closes the file after use.

2. Writing to a File:

To write data, we open the file in **write mode** ("w"). This will overwrite any existing content.

Example:

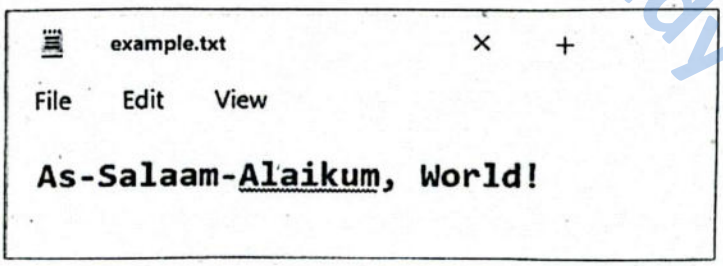
File Edit Format Run Options Window Help

```
with open("example.txt", "w") as file:
    file.write("As-Salaam-Alaikum, World!\n")
```

Explanation:

- The file is opened in write mode.
- The line is written into the file.
- If the file does not exist, it will be created.

Output :



3. Appending to a File:

To add new data without removing old data, we use **append mode** ("a").

Example:

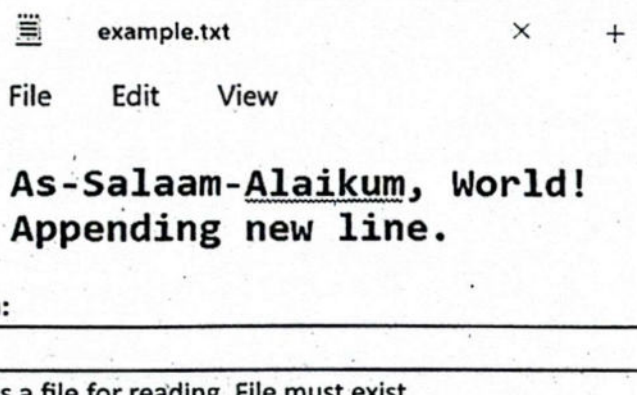
File Edit Format Run Options Window Help

```
with open("example.txt", "a") as file:
    file.write("Appending new line.\n")
```

Explanation:

- The file is opened in append mode.
- New text is added at the end of the file.

Output:



4. File Modes in Python:

Mode	Description
r	Read mode – opens a file for reading. File must exist.
w	Write mode – opens a file for writing. Overwrites the file. Creates new file if it doesn't exist.
a	Append mode – opens a file for adding data at the end. Creates file if it doesn't exist.
r+	Read and Write mode – allows both reading and writing. File must exist.

Q.43. What is testing and debugging in Python? Explain different types of testing and common debugging techniques with the help of examples.

Ans. Testing and Debugging in Python:

In Python programming, **testing and debugging** are very important steps. They help programmers make sure that their programs work properly and do not contain any mistakes (bugs).

Testing and debugging are necessary for writing reliable and correct Python programs.

- **Testing** checks if your program is working as expected.
- **Debugging** helps to find and correct mistakes.

By using both, a programmer can create a program that is safe, efficient, and error-free.

1. What is Testing?

Testing means checking your program by giving it different inputs to see if it gives the correct results. The main aim of testing is to **find errors before the program is used in real life.**

When we test a program, we want to:

- Make sure it gives the correct output.
- Check that all parts of the program work properly.
- Confirm that changes in the program do not break anything.

Types of Testing in Python:

1. **Unit Testing**
   - This tests **one small part** of the program, like a single function or class.
   - Python provides a built-in module called unittest for unit testing.
   - It helps check that each function works correctly on its own.
2. **Integration Testing**
   - This tests how **different parts** of the program work together.
   - For example, checking if two or more functions work properly when combined.
3. **Functional Testing**
   - This type of testing checks if the program behaves as expected from the **user's point of view.**
   - It focuses on **what the program does,** not how it does it.

4. **Regression Testing**
   - This is done **after making changes** to the program.
   - It makes sure that the **new changes do not affect the old working parts** of the program.

Example of Unit Testing in Python:

File Edit Format Run Options Window Help

```
import unittest

def add(a, b):
    return a + b

class TestMathOperations(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2, 3), 5)
        self.assertEqual(add(-1, 1), 0)

if __name__ == '__main__':
    unittest.main()
```

Explanation:

- import unittest:
  This line imports Python's **unittest module**, which provides tools for testing code.
- def add(a, b): return a + b:
  This is a **simple function** named add that takes two numbers and returns their sum.
- class TestMathOperations(unittest.TestCase):
  This is a **test class**. It is used to check if the add() function works correctly. It must inherit from unittest.TestCase.
- def test_add(self):
  This is a **test method**. It contains test cases for the add function.
- self.assertEqual(add(2, 3), 5):
  This checks if add(2, 3) gives **5**. If yes, the test passes.
- self.assertEqual(add(-1, 1), 0):
  This checks if add(-1, 1) gives **0**. If yes, the test passes.
- if __name__ == '__main__': unittest.main():
  This line makes sure the test runs only when this file is run directly.
  unittest.main() starts running all the test cases.

Result:

- If all test cases pass, the program confirms that the add() function is correct.
- If any test fails, Python shows an error message explaining what went wrong.

2. What is Debugging?

Debugging is the process of **finding and fixing errors (called bugs)** in your code. Even experienced programmers make mistakes, so debugging is an important skill.

When a program does not work correctly, debugging helps us:

- **Find the reason** why the program is not working.
- **Fix the error** so the program gives correct results.
- **Understand** how the program is behaving during execution.

Common Debugging Techniques:

1. **Print Statements**
   - Adding print() statements to show the value of variables at different points in the program.
   - It helps understand what is happening in the code step-by-step.

2. **Debugging Tools (like pdb)**
   - Python has a tool called **pdb (Python Debugger)**.
   - It allows the programmer to go through the program line-by-line to find where the error is.
3. **Reading Error Messages**
   - Python shows clear error messages when something goes wrong.
   - By reading the error message, we can understand **what went wrong** and **where** in the code it happened.

Example of Debugging Using Try-Except Block:

File Edit Format Run Options Window Help

```
def divide(a, b):
    return a / b
try:
    result = divide(10, 0)
except ZeroDivisionError as e:
    print("Error occurred:", e)
```

Explanation:

- The divide() function divides two numbers.
- When the user tries to divide 10 by 0, Python gives a ZeroDivisionError.
- The error is caught using the try-except block, and a friendly message is shown instead of crashing the program.

Output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/debugging.py
Error occurred: division by zero
>>>
```


Exercise

Multiple Choice Questions

1. **An action needed during Python installation to run from the command line easily:**
   a) Uncheck "Add Python to PATH" b) Choose a different IDE
   c) Check "Add Python to PATH" d) Install only the IDE
2. **A valid variable name in Python is:**
   a) variable 1 b) 1variable c) variable-name d) variable name
3. **Output of the following piece of code is:**

```
age = 25
print(" Age : ", age)
```

   a) Age: 25 b) 25 c) Age d) age
4. **The operator used for exponentiation in Python is:**
   a) * b) ** c) // d) /
5. **A loop used to iterate over a collection such as lists is:**
   a) while b) for c) do-while d) repeat
6. **A range( )function used to generate a sequence of numbers:**
   a) Generates a list of numbers b) Creates a sequence of numbers
   c) Calculates the sum of numbers d) Prints a range of numbers

7. **A keyword used to define a function in Python:**
   a) define b) function c) def d) func
8. **The Output of the following code is: temperature, humidity, wind_speed = 25,60,15 print("Hot and humid" if temperature > 30 and humidity > 50 else "Warm and breezy" if temperature == 25 and wind_speed > 10 else "Cool and dry" if temperature < 20 and humidity < 30 else "Moderate")**
   a) Hot b) Warm c) Cool d) Nothing
9. **The operation used to combine two lists in Python:**
   a) combine() b) concat() c) + d) merge()

Answers:

Q. No.	Correct Option & Answer
1.	c) Check "Add Python to PATH"
2.	a) variable1
3.	a) Age: 25
4.	b) **
5.	b) for
6.	b) Creates a sequence of numbers
7.	c) def
8.	b) Warm and breezy
9.	c) +

Short Answer Questions

Q.1. Explain the purpose of using comments in Python code?

Ans. Comments in Python are used to make the code easier to understand. They are explanatory statements written in the program, but are not executed by the computer. Instead, they help programmers and readers understand what the code is doing.

Purposes of using comments in Python:

1. **Explain Code Logic:** Comments help in understanding what a specific part of the program is doing.
2. **Improve Readability:** They make the code easier to read and maintain for others and for the programmer in the future.
3. **Disable Code Temporarily for Debugging:** Programmers can use comments to stop or skip certain lines of code during testing and debugging without deleting them. This is done by simply commenting out those specific lines of code.

Q.2. Describe the difference between integer and float data types in Python. Provide an example of each.

Ans. See answer of chapter question # 15

Q.3. Define operator precedence and give an example of an expression where operator precedence affects the result.

Ans. Operator precedence in Python determines the order in which different operators are evaluated in an expression. Operators with higher precedence are evaluated before those with lower precedence.

Example:

result = 2 + 3 * 2 ** 2

Step-by-step:

1. First, exponent is solved: 2 ** 2 = 4
2. Then multiplication: 3 * 4 = 12
3. Then addition: 2 + 12 = 14

So, the final result is **14.**

This shows how the answer changes if we don't use brackets. If we write it like (2 + 3) * 2 ** 2, the answer will be different.

Q.4. How does the short hand if-else statement differ from the regular if-else statement?

Ans.

Feature	Regular if-else Statement	Short-hand if-else Statement
Definition	Used to execute one block of code if a condition is true and another block if false.	A compact way to write if-else in a single line.

Syntax	if condition:
	  statement1
	else:
	  statement2
		statement1 if condition else statement2

Example	if age > 18:
	  print("Adult")
	else:
	  print("Minor")
		print("Adult") if age > 18 else print("Minor")

Q.5. Explain the use of the range() function in a for loop.

Ans. The range() function is used in a for loop to generate a sequence of numbers automatically. It tells the loop how many times to run and what values to use.

Syntax: range(start, stop, step)

- start: (optional) starting number (default is 0)
- stop: number up to which loop runs (not included)
- step: (optional) difference between each number

Example:

```
for i in range(1, 5):
  print(i)
```

Output :

```
1
2
3
4
```

Explanation: The loop runs from 1 to 4 (5 is not included). So, range() helps control the number of iterations.

Q.6. Explain how default parameters work in Python functions.

Ans. In Python, **default parameters** are values assigned to function parameters in case no value is provided by the user during the function call. This makes the function flexible and avoids errors when fewer arguments are passed.

Syntax Example:

```
def greet(name="Student"):
  return "Hello, " + name
# Function Calls:
print(greet())          # Output; Hello, Student
print(greet("Umer"))    # Output: Hello, Umer
```

Explanation:

- If no name is given, it uses "Student" as default.
- If a name is provided, it overrides the default.

Q.7. Explain why modular programming is useful in Python.

Ans. **Modular programming** is a method where a program is divided into smaller, independent units (modules) that perform specific tasks.

It is useful in Python for these reasons:

1. **Code Reusability**: You can reuse code in different programs by placing it in separate modules. For example, a calculator.py module can be used in multiple programs.

2. **Easier Maintenance**: It's easier to fix bugs(errors) because you can work on specific modules without affecting the entire program.
3. **Better Organization**: It keeps your code neat and organized by dividing it into smaller, manageable parts, making it easier to read and understand.

Q.8. Explain the difference between a class and an object in Python.

Ans.

Feature	Class	Object
Definition	A **class** is like a **blueprint** or **template** that defines what an object will look like and what it can do.	An **object** is a **real thing** created from the class. It is an instance of the class.
Purpose	A class tells us **what** an object will have (its properties) and **what** it can do (its functions or methods).	An object is a **real example** of a class. It has its own specific values.
Data	A class does not store data. It is just a plan or design.	An object stores its own data, like the color or size of a car.
Creation	A class is defined **once** to create many objects.	An object is created **using** a class to give it specific values.
Example	class Car: - This defines the general idea of a car.	car1 = Car("Red", "Toyota") - This creates a real car with specific details.

Programming Conceptual Questions (Practical Based)

S.No.	Question	Solution
1.	What is the output of 10 // 3? Explain.	**Output:** 3 **Explanation:** // is the floor division operator, which returns the quotient without the remainder. So, 10 // 3 = 3.

2.	What is the output of the following code?
	x = 5;
	y = 2;
	x = x + y;
	y = y + 1;
	print(x, y)?
		Output: 12 3
		Explanation:
		x = 10 + 2 = 12,
		y = 2 + 1 = 3.

3.	What is the output of the following code? x = 3; y = 4; print(x == y)?	**Output:** False **Explanation:** 3 == 4 is False.
4.	What will be the output of following expression? 10 + 3 * 2 ** 2 - 7 / 2?	**Output:** 18.5 Explanation: 2 ** 2 = 4 then . 3 * 4 = 12, expression becomes 10 + 12 - 7 / 2 then 7 /2 = 3.5, So, 10 + 12 - 3.5 = 22 – 3.5 = **18.5**
5.	What is the output of following expression? x = 10; y = 5; x // y * 3?	**Output:** 6 Explanation: **Floor Division:** x // y = 10 // 5 = 2 **Multiplication:** 2 * 3 = 6

6.	**Find the output of the following code:** x = 10
	if x % 2 == 0:
	print("Even")
	else:
	print("Odd")
		Output: Even
		Explanation: The condition x % 2 == 0 checks if the number is even. Since 10 % 2 == 0, it prints "Even".

7.	**Identify and fix the error in the following code:** for i in range(0, 5) print(i)	**Error:** Missing colon (:) at the end of the for statement and indentation error in the print statement. **Corrected Code:** for i in range(0, 5): print(i)
8.	**Convert the following while loop to a for loop:** count = 0 while count < 4: print(count) count += 1	**Solution:** for count in range(4): print(count)
9.	**What will be the output of the following code:** for i in range(1, 6, 2): print(i)	**Output:** 1 3 5 **Explanation:** The range(1, 6, 2) generates numbers starting from 1, up to 5, with a step size of 2.
10.	**Convert the following for loop to a while loop:** for num in range(2, 6): print(num ** 2)	**Solution:** num = 2 while num < 6: print (num ** 2) num += 1
11.	**What is the output of the following code:** total = 0 for i in range(1, 4): total += i print(total)	**Output:** 6 **Explanation:** The loop adds 1 + 2 + 3 to total, resulting in 6.

12.	**Fix the syntax error in the following code:**
	x = 5
	if x > 3
	print("Greater than 3")
	else
	print("Not greater than 3")
		Error: Missing colons at the end of the if and else statements.
		Corrected Code:
		x = 5
		if x > 3:
		print("Greater than 3")
		else:
		print("Not greater than 3")

13.	**Convert the following for loop to a while loop:**
	for i in range(1, 4):
	if i % 2 == 0:
	print(f"{i} is even")
	else:
	print(f"{i} is odd")
		Solution:
		i = 1
		while i < 4:
		if i % 2 == 0:
		print(f"{i} is even")
		else:
		print(f"{i} is odd")
		i += 1

14.	**What is the output of this code?** def welcome(user): print("Welcome", user) welcome("Ahmed")	**Output**: Welcome Ahmed **Explanation**: The function prints the welcome message using the given argument.
15.	**Identify the error in this function:** def calculate(x, y): return x * y print(calculate(4))	**Errors:** 1. **Missing argument**: The function calculate(x, y) expects **two arguments**, but only one (4) is given in the function call. **2. Indentation Error**: The line return x * y must be indented inside the function.
16.	What is the result of this code? def repeat(word, times): return word * times print(repeat("Hi", 3))	**Output**: HiHiHi **Explanation**: String "Hi" is repeated 3 times.
17.	**What is the output of the following code?** def say_hello(name="Guest"): return "Hello " + name print(say_hello())	**Output**: Hello Guest **Explanation**: Default value "Guest" is used when no argument is passed.
18.	**What happens in this case?** def add(a, b): return a + b print(add(5, "10"))	**Error**: TypeError **Explanation**: Cannot add an integer and a string. Use str(b) or convert before adding.

19.	**Find and fix the error:**
	def show(**info):
	for k v in info.items():
	print(k, v)
		Error: Missing comma between k and v.
		Correct Code:
		for k, v in info.items():

20.	**Predict the output:** def total_marks(*marks): return sum(marks) print(total_marks(30, 25, 40))	**Output**: 95 **Explanation**: All positional arguments are summed.
21.	**What will this code print?** def display(course, year): return f"{course} - {year}" print(display(year=2023, course="Math"))	**Output**: Math – 2023 **Explanation**: Keyword arguments used in different order.
22.	**What is the result of this code?** def power(base=2, exponent=3): return base ** exponent print(power(3))	**Output**: 27 **Explanation**: base = 3, exponent = 3, so $3^3 = 27$.

23.	**Find the output:**
	def profile(**data):
	for item in data:
	print(item, "=", data[item])
	profile(name="Aisha", age=19)
		Output:
		name = Aisha
		age = 19
		Explanation: **data collects the keyword arguments into a dictionary.

24.	**Find the output:** my_list = [10, 20, 30] my_list[1] = 50 print(my_list)	**[10, 50, 30]**

25.	**Identify the error in the code:** my_tuple = (10, 20, 30) my_tuple[0] = 100	**TypeError: 'tuple' is immutable does not support item assignment**
26.	**Find the output of the following slice:** values = (1, 2, 3, 4, 5) print(values[1:4])	**(2, 3, 4)**
27.	**Find the output:** colors = ["red", "blue", "green"] colors.append("yellow") print(colors)	**['red', 'blue', 'green', 'yellow']**
28.	**Find the output of the slice:** nums = [5, 10, 15, 20, 25, 30] print(nums[2:5])	**[15, 20, 25]**
29.	**Find the output:** names = ["Ali", "Sara", "Usman"] x = names.pop() print(x)	**'Usman'**
30.	**Find the output using:** fruits = ["apple", "banana", "cherry"] fruits.insert(1, "grape") print(fruits)	**['apple', 'grape', 'banana', 'cherry']**
31.	**Identify the error in the string code:** my_str = "Hello" my_str[0] = "h"	**TypeError: 'str' object does not support item assignment**
32.	**Find the output:** letters = ['a', 'b', 'c', 'd'] letters.reverse() print(letters)	**['d', 'c', 'b', 'a']** The reverse() method reverses the list in-place.
33.	**Find the output:** marks = [40, 90, 70, 60] marks.sort() print(marks)	**[40, 60, 70, 90]** The sort() method arranges the list in ascending order.
34.	**Find the output:** nums = [1, 2, 3, 2, 4, 2, 5] print(nums.count(2))	**3** count(2) returns the number of times 2 appears in the list.
35.	**Find the output:** cities = ["Lahore", "Karachi", "Peshawar"] print(cities.index("Karachi"))	**1** index() returns the position of the first occurrence of "Karachi".
36.	**Find the output:** languages = ("Python", "C++", "Java") print(languages[-1])	**'Java'** Negative index -1 accesses the **last element** of the tuple.
37.	**Find the output:** word = "Programming" print(word[3:8])	**'gramm'** Slicing from index 3 to 7 (8 is excluded): "g", "r", "a", "m", "m".

38.	**Find the output:** nums = [10, 20, 30, 40, 50, 60] print(nums[1:5:2])	**[20, 40]** Slicing from index 1 to 4, step 2: picks elements at index 1 and 3.

39.	**Identify the error in the following code:**
	class Car:
	def __init__(self, color):
	self.color = color
		Error: Indentation is missing after def __init__(self, color):.
		Fix: Indent lines inside the __init__ method.
		class Car:
		def __init__(self, color):
		self.color = color

40.	**What is the error in the following code?**
	class Toy:
	def describe()
	print("This is a toy")
		Error: Missing: after method name describe (). **Fix:** Add a colon after describe() to define the method properly.
		class Toy:
		def describe():
		print("This is a toy")

41.	**What will be the output of the following code?**
	class Book:
	def __init__(self, title):
	self.title = title
	book1 = Book("Python")
	print(book1.title)
		Output: Python

42.	**What will the following code output?**
	class Animal:
	def make_sound(self):
	print("Animal sound")
	dog = Animal()
	dog.make_sound()
		Output: Animal sound

43.	**Identify the error in the following code:**
	def __init__(self):
	self.name = "Car"
	obj = MyClass()
		Error: No class definition before using the constructor.
		Fix: Define a class (MyClass) before using the constructor __init__.
		class MyClass:
		def __init__(self):
		self.name = "Car"
		
		obj = MyClass()

Long Question Answers

Q.1. Evaluate the following Python expressions:

(a) (18/3+4 ** 2) - (2 * (7-3))/(97,4)

Solution: Error in the expression:

(97, 4) is not a valid operation. It seems like a tuple (97+4), so this would cause an error in Python.

So, we will solve it by considering 97 + 4

Step-by-Step Evaluation:

1. **Parentheses**: Inside the first parentheses, solve 18 / 3 + 4 ** 2:
   - 18 / 3 = 6

- 4 ** 2 = 16
- So, 6 + 16 = 22.

2. **Parentheses**: Inside the second parentheses, solve 7 - 3:
   - 7 - 3 = 4
   - Then, 2 * 4 = 8.
3. Now, we have:
   (22) - (8) / (97 + 4)
4. **Addition**: Inside the last parentheses, solve 97 + 4:
   - 97 + 4 = 101.
5. Now, the expression becomes: 22 - 8 / 101
6. **Division**: Perform the division 8 / 101:
   - 8 / 101 = 0.0792 (approximately).
7. **Subtraction**: Now, subtract 0.0792 from 22:
   - 22 - 0.0792 = 21.9208.

Final result for (a):
21.9208

(b) (25+3*4 ** 2-6)/(2 ** 3+1)-7

Solution: Step-by-Step Evaluation:

1. **Parentheses**: Inside the first parentheses, evaluate 25 + 3 * 4 ** 2 - 6:
   - First, exponentiation: 4 ** 2 = 16
   - Then, multiplication: 3 * 16 = 48
   - Now add and subtract: 25 + 48 - 6 = 73 – 6 = 67
2. **Parentheses**: Inside the second parentheses, evaluate 2 ** 3 + 1:
   - Exponentiation first: 2 ** 3 = 8
   - Then addition: 8 + 1 = 9
3. The expression becomes: 67 / 9 - 7
4. **Division**: 67 / 9 = 7.4444
5. **Subtraction**: 7.4444 - 7 = 0.4444

Final result for (b): 0.4444

(c) (12+6*(5-2)) ** 2/((4 ** 2-7)+10)

Solution: Step-by-Step Evaluation:

1. **Parentheses**: Solve 5 - 2 inside the first set of parentheses:
   - 5 - 2 = 3
   - Then, 6 * 3 = 18
   - Now, 12 + 18 = 30
2. **Parentheses**: Inside the second parentheses, solve 4 ** 2 - 7 + 10:
   - Exponentiation first: 4 ** 2 = 16
   - Then subtraction and addition: 16 - 7 + 10 = 9 + 10 = 19
3. **Exponentiation**: Now, square 30: 30 ** 2 = 900
4. The expression becomes: 900 / 19 = 47.3684

Final result for (c): 47.3684

(d) 45/(2 ** 2+3 * 4)+8 * (7-3)

Solution: Step 1: Solve inside parentheses (highest precedence)

- **Exponentiation first:**
  2 ** 2 = 4
- **Multiplication next:**
  3 * 4 = 12

- **Now solve inner parenthesis:**
  4 + 12 = 16
- **Also solve (7 - 3):**
- 7 - 3 = 4

Now the expression becomes:
45 / 16 + 8 * 4

Step 2: Perform multiplication and division (same precedence, left to right)

- **Left to right:**
  45 / 16 = 2.8125
  8 * 4 = 32

Step 3: Perform final addition

2.8125 + 32 = 34.8125

Final Answer:

Output = 34.8125

Q.2. **Translating the following Mathematical Expressions to Python Syntax:**

(a) $\frac{15\times(3+2^2)}{6-2\times3}$

Ans. **Python Syntax:**

(3 + 2 ** 2)) / (6 - 2 * 3)

(b) $7+2^2$

Python Syntax:

7 + 2 ** 2

Q.3. **Explain the concept of variables in Python.**

Ans. See Answer of chapter Q6

Q.4. **Write a Python program that takes a number as input and checks whether it is positive, negative, or zero using an if-elif-else statement.**

Ans. See Answer of chapter Q27

Q.5. **Write a Python program using a while loop that prints all the odd numbers between 1 and 100 Also, count and print the total number of odd numbers.**

Ans. **Python Code:**

File Edit Format Run Options Window Help

```
# Initialize variables
number = 1
odd_count = 0

# Using a while loop to print odd numbers
while number <= 100:
    if number % 2 != 0:
        print(number, end=",")
        odd_count += 1
    number += 1

# Print the total count of odd numbers
print("Total number of odd numbers:", odd_count)
```

Explanation:

- **Variable Initialization:**
  - number = 1 starts the loop, and odd_count = 0 keeps track of odd numbers found.

- **While Loop:**
  - The loop runs while number <= 20. It checks if the number is odd using if number % 2 != 0:. If true, it prints the number and increments odd_count.
- **Incrementing the Number:**
  - After each iteration, number is incremented by 1 (number += 1).
- **Final Output:**
  - After the loop ends, the program prints the total count of odd numbers found.

Output:

```
>>>
= RESTART: C:/Users/hp/OneDrive/Documents/python code/q6.py
1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99,
Total number of odd numbers: 50
>>>
```

SLO Based Questions

Q.1. What is Python and why is it widely used?

Ans. Python is a high-level programming language known for its simple and easy-to-understand syntax. It is widely used in many fields such as web development, data science, artificial intelligence, and automation. Python's readability and flexibility make it popular among both beginners and professionals.

Q.2. Why is Python considered a suitable language for beginners?

Ans. Python is suitable for beginners because:

- It has a simple syntax similar to English.
- It is easy to read, write, and understand.
- Beginners can focus on learning logic rather than complex rules.
- It has large community support and free resources.

Q.3. Mention any four areas where Python is commonly used.

Ans. Python is commonly used in:

- Web Development (e.g., Django, Flask)
- Data Analysis and Data Science
- Artificial Intelligence and Machine Learning
- Automation and Scripting

Q.4. List four key features of Python.

Ans. Important features of Python include:

- Simple and readable syntax
- Versatile and supports multiple applications
- Open-source and free to use
- Large community for support and learning

Q.5. What is meant by computer programming?

Ans. Computer programming is the process of writing instructions in a programming language to tell a computer what to do. These instructions must be clear and precise so that the computer can perform the desired task correctly

Q.6. What are the basic steps involved in writing and running a program?

Ans. The basic steps are:

- **Write Code** – using a programming language
- **Compile/Interpret** – translate it to machine-understandable form
- **Execute** – run the code
- **Output** – get the result or perform specified action

Q.7. What is a development environment in Python?

Ans. A development environment is a setup where a programmer writes, runs, and debugs Python code. It includes installing Python, choosing an IDE like IDLE or PyCharm, and using libraries or tools to help with programming.

Q.8. From where can Python be downloaded and installed?

Ans. Python can be downloaded for free from the official website: https://www.python.org. It works on Windows, macOS, and Linux.

Q.9. What is the use of "Add Python to PATH" during installation?

Ans. Checking the "Add Python to PATH" option allows you to run Python from the command line (CMD) easily. It saves time and avoids manual configuration of system paths.

Q.10. Can we write and run Python code online? Give one example.

Ans. Yes, Python code can be written and run online using tools like **Replit (https://replit.com)**. These platforms allow programming without installing Python.

Q.11. Explain Python syntax with an example.

Ans. Python uses a clean and simple syntax.

Example:

```
print("As - Salaam-Alaikum, College Students!")
```

In this example, print() is a built-in function that displays the text written inside the quotation marks.

Q.12. What is the use of the print() function in Python?

Ans. The print() function is used to show output on the screen. It displays the message or result written inside it.

For example:

```
print("Hello World")
```

This prints: **Hello World**

Q.13. What is a string in Python?

Ans. A string is a data type in Python used to store text. It is written inside single (' ') or double (" ") quotation marks.

Example: "Hello" is a string.

Q.14. What is a comment in Python? Explain single-line comment with example.

Ans. A comment is an explanatory statement in a program that is not executed. It is used to explain the code.

Single-line comments begin with the # symbol.

Example:

```
# This is a single-line comment
```

Q.15. What is a multi-line comment? Give an example.

Ans. A multi-line comment is used to write notes over more than one line. It starts and ends with triple quotes (''' or """).

Example:

```
'''
This program adds two numbers.
It shows the result on screen.
'''
```

Q.16. Write three purposes of using comments in Python.

Ans.
1. To explain what a part of the code does.
2. To make code easy to understand for others.
3. To disable lines of code without deleting them for debugging purpose.

Q.17. Define a variable with an example.

Ans. A variable is the name of a memory location in a computer that is used to store data in a program. It can hold values that may change during the execution of the program.

Example:

```
age = 17
```

Q.18. What are the rules for naming variables in Python? Give examples.

Ans.
- A variable name must start with a letter or underscore.
- It can contain letters, numbers, or underscores.

- It cannot use spaces, special characters, or Python keywords.
  Valid: student_name, _value
  Invalid: 2name, class, student-name

Q.19. How do you take user input in Python? Give an example.

Ans. The input() function is used to take user input in Python. It displays a message and waits for the user to type a response, which is then stored in a variable.

Example:

```
name = input("Enter your name: ")
```

Q.20. How is output displayed in Python? Give an example.

Ans. Python uses the print() function to show output. It can print text, numbers, or variables.

Example:

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

Q.21. How do you take and print an integer input in Python?

Ans. To take an integer input in Python, use the int(input()) function, which converts the user's input into an integer. Use the print() function to display the result.

Example:

```
age = int(input("Enter your age: "))
print("Age is:", age)
```

Q.22. Write a Python statement to store and print a float value.

Ans. To store a decimal (floating-point) value entered by the user, use the float(input()) function. Then, display it using the print() function.

Example:

```
height = float(input("Enter height: "))
print("Height is", height)
```

Q.23. Why do we use int() and float() functions in Python?

Ans. In Python, the input() function always takes user input as a string. To work with numbers, we need to convert that input into the correct numeric type:

- We use the **int() function** to convert the input into a **whole number** (integer).
- We use the **float() function** to convert the input into a **decimal number** (floating-point).

This is important because mathematical operations like addition, subtraction, or comparison require numeric data types — not strings.

Q.24. Define an expression in Python. Give one example.

Ans. An expression is a combination of **values, variables, and operators** that Python can evaluate to produce a result. It is the smallest unit of code that returns a value.

Example: a + b * 2 is an expression that adds a to the product of b and 2.

Q.25. List four arithmetic operators used in Python. Also mention their symbols.

Ans. Python supports the following arithmetic operators:

- + for Addition
- - for Subtraction
- * for Multiplication
- / for Division

These operators are used to perform basic mathematical calculations on numeric values.

Q.26. What is the purpose of the modulus operator (%) in Python?

Ans. The modulus operator (%) in Python is used to **find the remainder** when one number is divided by another.

Purpose:

1. **Finding Remainders:** It returns the remainder after division.
   - **Example:** 10 % 3 = 1.

2. **Checking Even or Odd:** It helps determine if a number is even or odd.
   - **Example:** 8 % 2 = 0 (Even), 7 % 2 = 1 (Odd).

Q.27. What does the exponentiation operator do in Python? Give one example.

Ans. The exponentiation operator ** is used to **raise one number to the power of another.**
Example: 2 ** 3 = 8 because 2 raised to power 3 means 2 * 2 * 2 = 8.

Q.28. Name any three comparison operators and write their purpose.

Ans. Python provides several comparison operators. Three of them are:

- **== (Equal to):** Checks if two values are equal.
  Example: 5 == 5 returns True.
- **!= (Not equal to):** Checks if two values are not equal.
  Example: 5 != 3 returns True.
- **< (Less than):** Checks if the left value is smaller than the right value.
  Example: 3 < 5 returns True.

These operators return a Boolean value, either True or False, based on the comparison result.

Q.29. What is operator precedence in Python? Explain with an example.

Ans. **Operator precedence** in Python defines the order in which operators are evaluated. Operators with higher precedence are evaluated first. For example, multiplication (*) has higher precedence than addition (+).
Example: In 3 + 2 * 4, multiplication is evaluated first:
2 * 4 = 8
Then 3 + 8 = 11
Thus, the result is 11.

Q.30. How does Python handle operators with the same precedence? Give an example.

Ans. When two operators in an expression have the **same precedence level,** Python evaluates them **from left to right.**
Example: 10 / 2 * 5
→ First 10 / 2 = 5.0, then 5.0 * 5 = 25.0

Q.31. What will print(10 < 3) display? Explain why.

Ans. It will print False because 10 is **not less than** 3.
Python evaluates < and returns Boolean value False.

Q.32. What is the use of comparison operators in Python?

Ans. Comparison operators are used to **compare values** and return True or False.
They are useful in **decision-making** (e.g., in if statements).
Example: x > y returns True if x is greater than y.

Q.33. Differentiate between = and == operators in Python.

Ans.

= (Assignment Operator)	== (Equality Operator)
Purpose: Used to assign a value to a variable.	**Purpose:** Used to compare two values to check if they are equal.
Function: Assigns the value on the right to the variable on the left.	**Function:** Compares the values on both sides and returns True if they are equal.
Example: x = 5 assigns the value 5 to the variable x.	**Example:** x == 5 checks if the value of x is equal to 5 and returns either True or False.

Q.34. What is the purpose of logical and comparison operators in Python?

Ans. **Comparison operators** (like ==, !=, <, >) are used to compare values and return a Boolean (True or False).
Logical operators (and, or, not) are used to combine multiple conditions in decision-making.
Together, they help in writing conditional statements and controlling the flow of a program based on multiple checks.

Q.35. Differentiate between / and // operators in Python.

Ans.

/ Division Operator	// Floor Division Operator
Definition: *The* / operator performs standard division and returns a float result.	**Definition:** The // operator performs floor division and returns only the integer part.
Result Type: It includes the decimal part even if the result is a whole number.	**Result Type:** It removes the decimal part and rounds down to the nearest whole number.
Example: 10 / 3 gives 3.333... as output.	**Example:** 10 // 3 gives 3 as output.

Q.36. Difference between comparison and logical operators in Python.

Ans.

Comparison Operators	Logical Operators
Purpose: Comparison operators are used to compare two values or expressions, such as checking equality (==), greater than (>), or less than (<).	**Purpose:** Logical operators are used to combine multiple conditions or expressions. The most common logical operators are and, or, and not.
Functionality: Comparison operators are used to evaluate the relationship between two values. They return True or False based on the comparison.	**Functionality:** Logical operators are used to combine two or more conditions and return a True or False result based on the combined conditions.
Example: For example, 5 > 3 returns True, as 5 is greater than 3.	**Example:** For example, (5 > 3) and (4 == 4) returns True because both conditions are True.

Q.37. What is a decision-making statement in Python? Explain the if statement with an example.

Ans. A decision-making statement in Python allows the program to make decisions based on certain conditions. The if statement is used to check whether a condition is true or false. If the condition is true, the program runs a block of code.

Example:

```
temperature = 35
if temperature > 30:
    print("It's a hot day")
```

Explanation:

In this example, the program checks if the temperature is greater than 30. If the condition is true, it prints "It's a hot day".

Q.38. What is the difference between the if statement and the if-else statement in Python? Explain with examples.

Ans.

Feature	if Statement	if-else Statement
Definition	Executes a block of code only if the condition is true.	Executes one block of code if the condition is true, and another if the condition is false.
Purpose	Used when an action is needed only if the condition is true.	Used when you need to perform one action if the condition is true, and a different one if false.

Syntax	if condition:
	  # code
		if condition:
		  # code
		else:
		  # code

Example	temperature = 35
	if temperature > 30:
	  print("It's a hot day")
		temperature = 15
		if temperature > 30:
		  print("It's a hot day")
		else:
		  print("It's not a hot day")

Explanation	If the condition temperature > 30 is true, it prints "It's a hot day".	If the condition temperature > 30 is true, it prints "It's a hot day" and if false, it prints "It's not a hot day".

Q.39. Explain the working of a short-hand if-else statement in Python with an example.

Ans. The short-hand if-else statement is a compact version of the if-else statement. It is written in a single line.

Syntax:

```
action_if_true if condition else action_if_false
```

Example:

```
temperature = 15
print("It's a hot day") if temperature > 30 else print("It's not a hot day")
```

Explanation:

This one-line if-else statement checks if temperature is greater than 30. If true, it prints "It's a hot day", otherwise, it prints "It's not a hot day".

Q.40. What is an if-elif-else statement? Explain with an example where multiple conditions are checked.

Ans. The if-elif-else statement allows us to check multiple conditions. If the first condition is false, it checks the next condition, and so on.

Example:

```
weather = "rainy"
if weather == "sunny":
    print("Wear sunglasses")
elif weather == "rainy":
    print("Take an umbrella")
else:
    print("Enjoy your day!")
```

Explanation:

Here, the program first checks if weather is sunny. If not, it checks if it's rainy. If neither condition is true, it prints "Enjoy your day!".

Q.41. Explain the range() function in Python. How can it be used with a for loop? Write an example.

Ans. The range() function generates a sequence of numbers and is commonly used in for loops to specify how many times the loop should run.

Syntax:

```
range(start, stop, step)
```

Example:

```
for i in range(1, 6):
    print(i)
```

Explanation:

This example generates numbers from 1 to 5 using range(1, 6). The loop iterates over each number and prints it.

Q.42. What is a for loop in Python? How is it different from a while loop?

Ans. A for loop in Python is used to repeat a block of code a specific **number of times.** It is mostly used when we **know how many times**, we want to run the loop or when we are going through a sequence like a list, string, or range of numbers.

A while loop, on the other hand, is used when **we don't know exactly how many times** to run the loop. It keeps running as long as a condition is true.

For Loop	While Loop
Used when the number of repetitions is known.	Used when the number of repetitions is not known.
Works with sequences like lists or range().	Works based on a condition.

Example:	Example:

`for i in range(5):`
`  print(i)`
	`i = 0`
	`while i < 5:`
	`  print(i)`
	`i += 1`

Q.43. Explain the concept of f-string formatting in Python. How is it used to print variables in a string?

Ans. In Python, **f-string** (formatted string literal) is used to insert variables or expressions directly into a string using curly brackets {}. It begins with the letter f before the opening quotation mark.

Purpose: It helps make output easier to write and more readable by combining text and variables in one line.

Example:

```
name = "Alice"
age = 25
print(f"Hello, {name}. You are {age} years old.")
```

Explanation:

Here, f"Hello, {name}. You are {age} years old." is an f-string, which replaces {name} and {age} with the values of the variables.

Q.44. What is the purpose of using range() in a for loop? How does it affect the iteration process?

Ans. The range() function in Python is used with a **for loop** to generate a sequence of numbers. It helps control how many times the loop runs by specifying the **start, stop,** and **step** values.

Example:

```
for i in range(5):
    print(i)
```

Explanation:

The range(5) generates numbers from 0 to 4, and the loop iterates over these values, printing them one by one.

Q.45. What are break and continue statements in Python? Explain the difference with examples.

Ans. In Python, break and continue are loop control statements used to change the normal flow of loops.

1. break Statement:

- The break statement **completely stops** the loop when a certain condition is met.
- It is used when we want to exit the loop early.

Example:

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

Output:

```
1
2
```

Explanation: The loop stops when i becomes 3.

2. continue Statement:

- The continue statement **skips the current iteration** and moves to the next one.
- It does **not stop** the loop, only skips certain values.

Example:

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

```
Output:
1
2
4
5
```

Explanation: The number 3 is skipped, but the loop continues.

Q.46. Write a Python program using a nested loop to print a multiplication table for numbers 1 to 3.

Ans.

```
for i in range(1, 4):
    for j in range(1, 11):
        print(f"{i} * {j} = {i * j}")
print()  # Adds a blank line between tables
```

Explanation:

The **outer loop** runs through numbers 1 to 3, and the **inner loop** multiplies each number by 1 to 10. This prints the full multiplication table for each number from 1 to 3.

Q.47. What is the purpose of indentation in Python? How does it affect control structures like loops and conditionals?

Ans. Indentation in Python is used to define the scope or body of control structures such as loops, conditionals, and functions. It indicates which lines of code belong to the same block.

Without proper indentation, Python will raise an IndentationError because it cannot determine the grouping of code.

Example:

```
if 10 > 5:
    print("Condition is true")  # This line is properly indented
```

In the example, the indented line under the if statement shows that it belongs to the if block. Correct indentation is important for the code to run properly in Python.

Q.48. How do you use the continue statement inside a loop? Provide an example where you skip even numbers while printing numbers from 1 to 10.

Ans. The continue statement is used to skip the current iteration of a loop and move to the next one.

Example:

```
for i in range(1, 11):
    if i % 2 == 0:
        continue
    print(i)
```

Explanation:

The loop skips even numbers using continue and prints only the odd numbers between 1 and 10.

Q.49. Write a Python program using a for loop to print the squares of numbers from 1 to 5.

Ans. **Python Code:**

```
for i in range(1, 6):
    print(i * i)
```

Explanation:

This program uses a for loop to iterate over numbers 1 to 5 and prints the square of each number.

Q.50. What is the purpose of using functions in Python?

Ans.

- The main purpose of using functions in Python is to break the program into smaller parts so it becomes easier to understand, write, and test.
- Functions help avoid repetition because the same code can be reused many times by just calling the function.
- They also help organize the code better and improve readability.
- Using functions makes debugging easier, and large programs can be handled more efficiently.

Q.51. Write a function that returns the sum of two numbers.

Ans. **Python Code:**

```
def add(a, b):
    return a + b
print (add(3, 5))
```

Output:

8

This function takes two inputs, a and b, adds them, and gives the result using return.

Q.52. What is a default parameter? Give an example.

Ans. A default parameter is a value assigned to a function argument when no value is passed during the function call.

Example:

```
def greet(name="Student"):
    return "Hello " + name
```

If we call greet() it will return "Hello Student", but if we call greet("Ali"), it will return "Hello Ali". This makes the function flexible and user-friendly.

Q.53. What are two advantages and two disadvantages of using default parameters in Python functions?

Ans. **Advantages:**

1. **Simplifies Function Calls:**
   If a default value is given, the function can be called without passing that argument every time. This reduces effort and avoids repetition.
2. **Increases Flexibility:**
   Default parameters allow the function to work for both basic and advanced use cases by providing optional arguments.

Disadvantages:

1. **Order Sensitivity:**
   All default parameters must come after non-default parameters. If the order is wrong, Python gives a syntax error.
2. **Can Hide Errors:**
   If a user forgets to provide an argument, the default value is used silently. This may lead to logic errors that are hard to detect.

Q.54. What is a keyword argument in Python?

Ans. Keyword arguments are used when we pass values to a function by specifying the name of the parameter. This improves readability and allows arguments to be passed in any order.

Example:

```
def introduce(name, age):
        return f"My name is {name} and I am {age} years old."
introduce(age=20, name="Ali")
```

Here, the use of keywords helps avoid confusion between arguments.

Q.55. What is a dictionary in Python? Write a short example.

Ans. A dictionary is a built-in data structure in Python that stores data in **key-value pairs**. It is used to organize related information.

Example:

```
student = {"name": "Ali", "age": 20}
```

Here, "name" and "age" are keys, and "Ali" and 20 are their values. We can get values using keys like student["name"].

Q.56. Write a function that returns the maximum number from a list.

Ans.

```
def find_max(numbers):
    return max(numbers)
```

This function takes a list as input and uses the max () function to find the largest value.

Example:

`find_max([5, 10, 2])` returns 10.

It helps find the biggest number from any list quickly.

Q.57. What error occurs if you try to add a string and a number directly in Python?

Ans. If we try to add a string and a number without converting, Python gives a **TypeError**.

Example:

```
"Age is " + 20                    # This will give a TypeError
```

To fix this, you can either:

Convert the number to a string using str():

```
"Age is " + str(20)               # Correct output: "Age is 20"
```

Write the number as a string directly:

```
"Age is " + "20"                  # Correct output: "Age is 20"
```

It's important to match the data types before combining them to avoid errors.

Q.58. Write a function to greet a user with a name.

Ans.

```
def greet(name):
    print("As-Salaam-Alaikum", name)
```

This function takes one input name and prints a greeting.

Example: greet("Ali") will print "As-Salaam-Alaikum Ali".

It shows how we can personalize output using parameters.

Q.59. How do keyword arguments make code easy to read?

Ans. Keyword arguments allow you to name each input value. This makes it easy to understand what each value is used for.

Example: introduce(name="Ali", age=20) is clear and avoids confusion about which value belongs to which parameter.

Q.60. Why is indentation important in Python functions?

Ans. Python uses indentation to show the block of code that belongs to a function. Unlike other languages that use braces {}, Python uses spacing. If indentation is missing or wrong, Python gives an error. So, correct indentation is necessary to run the function properly.

Q.61. How does max() work when we give it a list?

Ans. The max() function checks all the values in the list and returns the largest one.

Example: max([4, 9, 2]) returns 9.

It saves time and makes finding the biggest number easy and fast.

Q.62. Why is a dictionary better than a list for storing name and age?

Ans. A dictionary stores data using **keys** like "name" and "age", which makes the data more understandable. In a list, we only use indexes, which can be confusing. Dictionaries provide faster data access and more meaningful structure.

Q.63. What happens if we don't use return in a function?

Ans. If we don't use return, the function runs but does not send any value back. So, if we try to store its result, it gives None. return is important when we want to use the result later in calculations or another part of the program.

However, **whether to use a return statement or not is your choice**, depending on what your function needs to do. If you don't need to return a value (e.g., just printing something or modifying a global variable), you can leave it out. But if you need to use the result of the function later, the return statement *is important.*

Q.64. What are Python libraries and modules? How are they used in a program?

Ans. **Python libraries** are collections of pre-written code that allow you to perform common tasks without writing all the code from scratch. They provide ready-made functions and classes.

Modules are individual files within a library that contain functions, classes, and variables.
To use a module or library in Python, you can **import** it using the import statement.
Example: import random to use the random module.

Q.65. How does Python search for modules when they are imported?

Ans. When you import a module, Python looks for it in the following order:

- **Current directory**: First, Python checks the directory where your script is located.
- **Standard library**: If the module is not found in the current directory, Python looks in the built-in libraries.
- **Third-party libraries**: Lastly, Python checks external libraries installed using package managers like pip.

Q.66. What is the purpose of using a package in Python?

Ans. A **package** is a directory containing related modules, which helps organize large projects.
By grouping related modules into packages, you can easily manage and maintain your code.
Example: A package could be named ecommerce, containing modules like products.py, customers.py, and orders.py.

Q.67. What is the difference between a list and a tuple?

Ans.

Feature	List	Tuple
Mutability	Mutable (can be modified)	Immutable (cannot be modified)
Syntax	Defined using square brackets [ ]	Defined using parentheses ()
Performance	Slower due to mutability	Faster due to immutability
Example	my_list = [10, 20, 30]	my_tuple = (10, 20, 30)

Q.68. How can you access elements from a list, tuple, or string using indexing?

Ans. **Indexing** allows you to access individual elements from a sequence (list, tuple, string).
Python uses **zero-based indexing**, meaning the first element has an index of 0.
Example:

```
my_list = [10, 20, 30]
print(my_list[0]) # Output: 10 (First element)
```

Q.69. What is the difference between positive and negative indexing in Python?

Ans.

Feature	Positive Indexing	Negative Indexing
Start Point	Starts from the beginning of the sequence (0)	Starts from the end of the sequence (-1)
Access Example	my_list[0] for the first element	my_list[-1] for the last element

Q.70. What is slicing in Python and how do you use it?

Ans. **Slicing** is used to access a subset of a sequence, such as a list, tuple, or string.
The syntax for slicing is: sequence[start:stop:step]
start: Index to start the slice (inclusive)
stop: Index to end the slice (exclusive)
step: The step size (default is 1)
Example: my_list[2:5] returns elements starting from index 2 up to, but not including, index 5.

Q.71. What is negative slicing and how is it different from regular slicing?

Ans. **Negative slicing** uses negative indices to access elements from the end of the sequence.
It works in the same way as regular slicing but starts counting from the end instead of the beginning.
Example: my_list[-3:-1] will return the third and second last elements from the list.

Q.72. What is the purpose of the len() function in Python?

Ans. The len() function is used to determine the number of elements in a sequence (list, tuple, string).

```
my_list = [1, 2, 3]
print(len(my_list))                    # Output: 3
```

Q.73. What is the difference between a list and a string in Python in terms of mutability?

Ans.

Feature	List	String
Mutability	Lists are **mutable**, meaning their elements can be changed after creation.	Strings are **immutable**, meaning their characters cannot be changed once created.
Modification	Elements can be added, removed, or modified.	You must create a new string to modify it.
Example	my_list = [1, 2, 3] my_list[0] = 10 → [10, 2, 3] (can change elements)	my_string = "hello" my_str[0] = "H" → Error cannot be change

Q.74. Write a Python program to import the random and statistics libraries. Generate a list of 10 random numbers between 1 and 50 and calculate the mean of those numbers.

Ans.

```
import random
import statistics

# Generate a list of 10 random numbers between 1 and 50
random_numbers = [random.randint(1, 50) for _ in range(10)]

# Calculate the mean of the numbers
mean_value = statistics.mean(random_numbers)

# Print the list and the mean
print("Random numbers:", random_numbers)
print("Mean of numbers:", mean_value)
```

Q.75. What are Python modules and how do you import them?

Ans. A **module** is a file containing Python code (functions, classes, variables) that can be imported into your script.
You can import a module using the import statement.
Example: import math to import the math module.
To import specific elements: from math import pi

Q.76. What is the role of the import statement in Python?

Ans. The import statement is used to bring a module or specific elements of a module into your Python program.
You can import a whole module or specific functions or classes from a module.
Example:
import math (imports the entire math module)
from math import pi (imports only pi from math)

Q.77. What is the difference between using import module and from module import name?

Method	Usage
import module	Imports the entire module and you access its functions via module.function_name
from module import name	Imports a specific function or class directly, allowing you to use name() directly without prefixing with the module name

Q.78. Explain the concept of tuple immutability and provide an example.

Ans. A **tuple** is immutable, meaning once created, its elements cannot be changed, added, or removed.
Example:

```
my_tuple = (1, 2, 3)
```

Trying to change an element, e.g., my_tuple[0] = 10, will result in a TypeError.

Q.79. What is the purpose of negative indexing in Python?

Ans. **Negative indexing** allows you to access elements from the end of a sequence.

Example:

```
my_list = [10, 20, 30]
print(my_list[-1]) # Output: 30 (Last element)
```

Q.80. What is the difference between indexing and slicing in Python?

Feature	Indexing	Slicing
Definition	Accessing a single element from a sequence using its position.	Accessing a range (subset) of elements from a sequence.
Syntax	sequence[index]	sequence[start:stop:step]
Example	fruits = ["apple", "banana"]; fruits[0] → "apple"	fruits[1:3] → ["banana"]

Q.81. What is a Python module?

Ans. A **Python module** is a file containing Python code (functions, classes, or variables) that can be reused in other programs. Modules help organize code into manageable parts and promote code reuse. **For example,** calculator.py is a module where we define functions for arithmetic operations like addition and subtraction.

Q.82. How do you import a module in Python?

Ans. To **import a module** in Python, use the import keyword followed by the module name.

For example:

import calculator

This statement imports the functions and variables from the calculator.py module so they can be used in the script.

Q.83. Write the syntax for defining a function in Python.

Ans. The syntax for defining a function in Python is:

```
def function_name(parameters):
    # function body
    return result
```

For example, to define an add function:

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

Q.84. How do you use functions from an imported module in Python?

Ans. After importing a module, you use its functions by referring to the module name followed by a dot (.) and the function name.

For example:

import calculator

result = calculator.subtract(20, 5)

print(result)

This calls the subtract() function from the calculator module and prints the result.

Q.85. What is the purpose of the __name__ == "__main__" check in Python?

Ans. The if __name__ == "__main__": statement is used to ensure that code inside the main() function only runs when the script is executed directly, not when it is imported as a module. This allows the script to act as both a reusable module and a standalone program.

Q.86. Why is it important to use functions in Python?

Ans. Using functions in Python helps in organizing code into smaller, reusable blocks. It makes the code easier to read, debug, and maintain. **For example,** the add() and subtract() functions in the calculator.py module allow the same logic to be reused in multiple parts of the program without rewriting it.

Q.87. How can you perform addition and subtraction using a custom Python module?

Ans. To perform addition and subtraction, first, create a module with the functions add() and subtract(). Then, import the module and call these functions:

```
import calculator
sum_result = calculator.add(10, 5)
diff_result = calculator.subtract(20, 5)
print(sum_result)
print(diff_result)
```

Q.88. What is the benefit of creating a separate module for functions like add() and subtract()?

Ans. Creating a separate module for functions like add() and subtract() has the benefit of **code reusability.** You can use the same functions in different parts of your program or in other programs, without rewriting the logic. It also makes your code **organized** and easier to maintain.

Q.89. What will happen if you call the function add(15) without the second argument?

Ans. An **error** will occur because the add() function requires two arguments (a and b). If only one argument is passed, Python will raise a **TypeError** indicating that a required positional argument is missing.

Q.90. What is the output of the following code:

```
import calculator
result = calculator.add(15, 8)
result2 = calculator.subtract(25, 10)
print(result)
print(result2)
```

Ans. The output will be:

23
15

Explanation: The add(15, 8) function returns 23 and the subtract(25, 10) function returns 15. Both results are printed.

Q.91. How would you define a function that returns the sum of two numbers?

Ans. To define a function that returns the sum of two numbers, use the following code:

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

This function takes two numbers, adds them, and returns the result.

Q.92. How can we organize our Python code to make it modular?

Ans. We can organize our Python code by creating **modules**. A module is a separate file that contains related functions or classes. **For example,** the calculator.py module contains arithmetic functions like add() and subtract(). This allows us to import and use these functions in different programs, making our code more organized and reusable.

Q.93. Why is it necessary to call the main() function in Python?

Ans. It is necessary to call the main() function in Python to execute the program. The main() function is typically used to organize the flow of the program and ensure that the script runs when executed. Without calling main(), the program might not start as expected.

Q.94. What is Object-Oriented Programming (OOP)? How is it useful in Python?

Ans. OOP is a method of programming where we write code using **objects** and **classes**.

It helps us organize code better by grouping data (attributes) and actions (methods) together.

OOP makes programs easier to **understand, reuse, and maintain**. Python supports OOP features like classes, objects, inheritance, and more.

Q.95. What is a class in Python?

Ans. A class is like a **blueprint or template** used to create objects.

It defines the **attributes** (data) and **methods** (functions) that the objects will have.

Example: class Car: creates a Car class, and we can make car objects from it.

Q.96. What is an object in Python? How is it related to a class?

Ans. An object is a **real thing created from a class**. It represents single instance of a class.
Many objects can be made from one class, and each object has its **own values**.
Example:

```
car1 = Car("Red")

car2 = Car("Blue")
```

Q.97. What is the role of the __init__() method in a class?

Ans. The __init__() method is called a **constructor**.
It runs **automatically** when a new object is created.
It is used to set the starting values of the object (attributes).

Q.98. What is the use of the self keyword in Python classes?

Ans. The keyword **self** refers to the **current object**.
It is used to access the object's **own variables and methods**.
It must be the **first parameter** in every method of a class.

Q.99. What are access modifiers in Python? Name them.

Ans. Access modifiers control the **visibility** of class variables and methods.
Python has three types:

Access Modifier	Syntax Example	Description
Public	self.name	Can be accessed from **anywhere** (inside or outside the class). *(default)*
Protected	self._name	Can be accessed **inside the class** and its **subclasses** (not from outside by convention).
Private	self.__name	Cannot be accessed directly from outside the class. Only accessible **inside the class**.

Q.100. How does OOP help in organizing and understanding code better?

Ans. OOP allows us to divide the code into **small parts** (classes and objects).
It helps us **reuse** code using inheritance.
It makes code **easy to understand, debug, and extend** for bigger programs.

Q.101. What is exception handling in Python? Why is it important?

Ans. **Exception handling** is the process of catching and handling errors during the execution of a program. It is done using try, except, else, and finally blocks in Python.
It is important because:

- It allows the program to handle errors gracefully without crashing.
- It helps in finding errors and provides meaningful error messages to the user.
- It ensures that the program can continue running even after encountering an error.

Q.102. What is the purpose of the try-except block in Python?

Ans. The try-except block is used to catch and handle exceptions (errors) that may occur during the execution of a program.

- The **try** block contains code that might cause an error.
- The **except** block catches the error and allows the program to continue without crashing.

Q.103. Explain how file handling works in Python.

Ans. **File handling** in Python allows you to read from and write to files. You can open a file using the open() function, read its content, or write data to it. It is important to close the file after use, or use the with statement for automatic closure.

- **Reading**: file.read() or file.readlines()
- **Writing**: file.write()
- **Modes**: "r" (read), "w" (write), "a" (append)

Q.104. What is the purpose of the with statement in file handling?

Ans. The with statement is used to handle file opening and closing automatically. It ensures that the file is properly closed after it is read or written, even if an error occurs during the operation. This prevents resource leaks and makes the code cleaner.

Q.105. How do you append data to an existing file in Python?

Ans. To append data to an existing file, open the file in **append mode** ("a"). This will allow you to add data to the end of the file without overwriting its existing content.

Example:

```
with open("example.txt", "a") as file:
    file.write("Appending new line.\n")
```

Q.106. What are the different file modes in Python, and what do they do?

Ans.

- **r**: Opens the file for reading. The file must exist.
- **w**: Opens the file for writing. If the file exists, it will be overwritten; if it doesn't exist, a new file is created.
- **a**: Opens the file for appending. Data is added to the end of the file without overwriting.
- **r+**: Opens the file for both reading and writing. The file must exist.

Q.107. What is unit testing in Python?

Ans. **Unit testing** is the practice of testing individual parts (functions or methods) of a program to ensure they work as expected. It helps find and fix bugs early. Python provides the unittest module for writing and running test cases.

Q.108. How do you create a unit test for a function in Python?

Ans. To create a unit test, you define a class that inherits from unittest.TestCase and then define test methods inside it.

Example:

```
import unittest

def add(a, b):
    return a + b

class TestMathOperations(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2, 3), 5)

if __name__ == '__main__':
    unittest.main()
```

This test checks if the add() function returns the correct sum.

Q.109. How do you handle errors using try-except blocks while testing code in Python?

Ans. We use **try-except** blocks to find errors that occur during execution. If an error is raised in the try block, the code in the except block is executed, allowing you to handle the error.

Example:

```
try:
    result = divide(10, 0)
except ZeroDivisionError as e:
    print("Error occurred:", e)
```

Q.110. What is debugging in Python, and how do you perform it?

Ans. **Debugging** is the process of identifying and fixing errors (bugs) in your code.

You can debug by:

- Using **print statements** to track the values of variables.
- Using **debugging tools** like pdb (Python Debugger) to step through the code and inspect variables.
- Analyzing **error messages** to understand the cause of the problem.

Q.111. What are some common techniques used for debugging Python code?

Ans. Common techniques for debugging include:

- **Print Statements**: Inserting print() statements to check variable values at different points in the program.
- **Using a Debugger**: Using Python's built-in debugger (pdb) to step through the code and inspect variables.
- **Error Messages**: Reading error messages carefully to understand the cause of the issue and where it occurred.

Important MCQs

Introduction to Python Programming

1. Which of the following is NOT a basic step in writing a program?
A) Write Code B) Compile/Interpret
C) Execute D) Paint GUI

2. What is Python mainly used for?
A) Hardware testing
B) Web development, data science, and automation
C) Drawing designs
D) Making mobile covers

3. What makes Python easy to learn?
A) Complex syntax
B) Low-level code
C) Simple and readable syntax
D) Long code lines

4. Python is an example of which type of language?
A) Assembly language B) High-level language
C) Binary language D) Markup language

Python Environment and Syntax

5. Which statement is used to display text on the screen in Python?
A) echo B) write
C) print D) show

6. What symbol is used for single-line comments in Python?
A) // B) #
C) ** D) ;

7. What is indentation in Python used for?
A) To decorate code
B) To align code in the center
C) To define blocks of code
D) To comment lines

8. What happens if indentation is not correct in Python?
A) Code runs faster
B) No effect
C) Error will occur
D) Only comments will work

Variables, Data Types, and I/O

9. Which is a valid variable name in Python?
A) 1value B) value_1
C) value-1 D) value 1

10. What data type is used for whole numbers in Python?
A) float B) int
C) str D) char

11. What will input("Enter your name: ") do?
A) Display output B) Accept user input
C) Print the name D) Check name length

12. Which function is used to convert a string to an integer?
A) str() B) int()
C) float() D) input()

13. Which data type represents True or False?
A) string B) boolean
C) integer D) float

Operators and Expressions

14. What is the output of 10 % 3?
A) 3 B) 1
C) 0 D) 10

15. What does the ** operator do in Python?
A) Modulus
B) Division
C) Exponentiation
D) Floor division

16. What is the order of precedence for the expression 3 + 2 * 5?
A) 25 B) 13
C) 15 D) 30

17. Which is a comparison operator?
A) and B) +
C) == D) =

18. What does a += 5 mean?
A) a = a * 5 B) a = a - 5
C) a = a + 5 D) a = 5

Control Structures

19. What is the output of the code if temperature = 35?

```
if temperature > 30:
    print("It's hot")
```

A) It's cold B) Nothing
C) It's hot D) Error

20. Which statement checks multiple conditions in Python?

A) if B) for
C) elif D) while

21. Which loop runs until a condition becomes false?

A) for B) loop
C) if D) while

22. What does the range(5) function return?

A) Numbers from 1 to 5 B) Numbers from 0 to 4
C) 5 only D) Nothing

23. Which keyword is used to skip the rest of the loop and move to the next iteration?

A) stop B) skip
C) continue D) next

Functions and Modules

24. Which keyword is used to define a function?

A) function B) define
C) def D) func

25. What is a default parameter?

A) A parameter with no value
B) A parameter that must be passed
C) A parameter with a pre-set value
D) A special keyword

26. What is the purpose of *args in Python?

A) Return a list
B) Handle many arguments
C) Create variables
D) Multiply strings

27. What does this code return: def add(a, b): return a + b if add(2, 3) is called?

A) 23 B) 5
C) a + b D) Error

28. What is the use of import in Python?

A) Export values B) Call a function
C) Use built-in libraries D) Close files

Data Structures (Lists and Tuples)

29. How do you add an item to the end of a list?

A) push() B) add()
C) append() D) extend()

30. What is the correct way to access the second item in a list?

A) list(1) B) list[1]
C) list[2] D) list{1}

31. Which method is used to remove an item from a list?

A) delete() B) pop()
C) remove() D) cut()

32. Which of the following is an immutable data structure?

A) List B) Set
C) Dictionary D) Tuple

Object-Oriented Programming (OOP)

33. What is a class in Python?

A) A set of if statements
B) A loop
C) A blueprint to create objects
D) A function

34. Which function initializes a new object?

A) create B) start
C) init D) new

35. What does self refer to in a class?

A) Current module B) The object itself
C) A function D) A class variable

36. Which of the following defines inheritance?

A) Reusing variables
B) Accessing files
C) One class taking properties of another
D) Writing same function again

Exception and File Handling

37. What is exception handling used for?

A) Making code faster B) Fixing bugs
C) Managing errors D) Saving files

38. Which block checks for errors?

A) try B) check
C) error D) run

39. What keyword is used to handle an error?

A) handle B) fix
C) except D) warn

40. What does finally do?

A) Stops the program
B) Runs code only if there's an error
C) Runs no matter what
D) Returns result

41. Which mode is used to read a file?

A) "w" B) "r"
C) "a" D) "x"

42. Which function writes data into a file?

A) read() B) write()
C) print() D) input()

43. What does the "a" mode do in file handling?

A) Appends data B) Removes data
C) Replaces content D) Deletes file

Testing and Debugging

44. What is unit testing?

A) Testing full software
B) Testing one part of code
C) Testing the interface
D) Debugging

45. Which module is used for testing in Python?

A) testit B) unittest
C) pytest D) checkit

46. What does debugging mean?

A) Adding bugs B) Running programs
C) Fixing problems D) Installi

47. What is a common debugging tool?

A) error() B) check
C) print() D) stop()

48. What happens if you divide a number by zero in Python?

A) 0 B) Infinity
C) Error D) 1

49. Why do we use with open(...) for files?

A) To write large data
B) To open web pages
C) To automatically close files
D) To print without errors

50. What happens if a file is not found and not handled properly?

A) It creates a new one
B) Nothing
C) File Not Found Error occurs
D) File renames itself

51. An action needed during Python installation to run from the command line easily:

A) Uncheck "Add Python to PATH"
B) Choose a different IDE
C) Check "Add Python to PATH"
D) Install only the IDE

52. A valid variable name in Python is:

A) variable1 B) 1variable
C) variable-name D) variable name

53. Output of following piece of code is:

```
age = 25
print(" Age : ", age)
```

A) Age: 25 B) 25
C) Age D) age

54. The operator used for exponentiation in Python is:

A) * B) **
C) // D) /

55. A loop used to iterate over a collection such as lists is:

A) while B) for
C) do-while D) repeat

56. A function used to generate a sequence of numbers:

A) Generates a list of numbers
B) Creates a sequence of numbers
C) Calculates the sum of numbers
D) Prints a range of numbers

57. A keyword used to define a function in Python:

A) define B) function
C) def D) func

58. The Output of the following code is:

```
temperature, humidity, wind_speed = 25, 60, 15
print("Hot and humid" if temperature > 30 and
humidity > 50 else
"Warm and breezy" if temperature == 25 and
wind_speed > 10 else
"Cool and dry" if temperature < 20 and humidity <
30 else
"Moderate")
```

A) Hot B) Warm
C) Cool D) Nothing

59. The operation used to combine two lists in Python:

A) combine() B) concat()
C) + D) merge()

ANSWER KEY

1.	D	2.	B	3.	C	4.	B	5.	C	6.	B	7.	C	8.	C	9.	B	10.	B
11.	B	12.	B	13.	B	14.	B	15.	C	16.	B	17.	C	18.	C	19.	C	20.	C
21.	D	22.	B	23.	C	24.	C	25.	C	26.	B	27.	B	28.	C	29.	C	30.	B
31.	C	32.	D	33.	C	34.	C	35.	B	36.	C	37.	C	38.	A	39.	C	40.	C
41.	B	42.	B	43.	A	44.	B	45.	B	46.	C	47.	C	48.	C	49.	C	50.	C
51.	C	52.	A	53.	A	54.	B	55.	B	56.	B	57.	C	58.	B	59.	C		