Module-6: Python Fundamentals

Theory /exercise

* 1.INTRODUCTION TO PYTHON:-

🡪Que-1: Introduction to Python and its Features (simple, high-level, interpreted language)

Ans.

What is Python?

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

🡪 Key Features:

* Simple Syntax: Easy to read and write (similar to English).
* High-Level Language: You don’t need to manage memory manually.
* Interpreted: Code runs line-by-line, which makes debugging easier.
* Dynamically Typed: No need to declare variable types.
* Portable: Run on any platform (Windows, Mac, Linux).
* Extensive Libraries: Built-in and external modules for everything from math to web apps.
* Object-Oriented & Procedural: Supports both programming paradigms.
* Free and Open Source: Available to everyone with an active community.

-------------------------------------------------------------------------------------

🡪Que-2: History and evolution of Python.

Ans.

* **Created** by: Guido van Rossum
* Year: Late 1980s (released in 1991)
* Named after: "Monty Python’s Flying Circus" (not the snake!)
* Major Versions:
  + Python 2.x: Legacy version (no longer supported after 2020)
  + Python 3.x: Modern version with improvements and better Unicode support

🡪 Timeline:

* 1991 – Python 0.9.0 released
* 2000 – Python 2.0 introduced (features like garbage collection)
* 2008 – Python 3.0 launched (not backward compatible with 2.x)
* 2020 – Python 2 officially retired
* Now – Latest versions: Python 3.10, 3.11, 3.12 (actively developed)

-------------------------------------------------------------------------------------

🡪Que-3: Advantages of using Python over other programming languages.

Ans.

|  |  |
| --- | --- |
| **Feature** | **Benefit** |
| Easy to Learn | Ideal for beginners |
| Cross-platform | Runs on Windows, Linux, Mac |
| Huge Libraries | NumPy, Pandas, Matplotlib, etc. |
| Community Support | Help is easy to find |
| Versatile | Used in web dev, AI, ML, automation, data science, etc. |
| Rapid Development | Write less code, get more done |

**Compared to other languages:**

* **vs C/C++**: No need to manage memory manually
* **vs Java**: Less boilerplate, simpler syntax
* **vs JavaScript**: More powerful for data and automation

------------------------------------------------------------------------------------

🡪Que-4:Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).

Ans.

🡪Option 1: Anaconda (Recommended for Data Science)

* Includes Python, Jupyter Notebook, Spyder, and many libraries
* Download from: https://www.anaconda.com

🡪Option 2: VS Code

* Lightweight editor with Python support via extensions
* Download: https://code.visualstudio.com

🡪 Option 3: PyCharm

* Full-featured IDE from JetBrains
* Best for professional Python development
* Download: <https://www.jetbrains.com/pycharm>

🡪Direct Python Installation (if not using Anaconda)

* Download from: <https://www.python.org>
* Add to PATH during installation

-------------------------------------------------------------------------------------

🡪Que-5:Writing and executing your first Python program.

Ans.

**Using Python Console:**

>>> print("Hello, Python!")

Hello, Python!

**🡪 Using a .py file:**

# filename: hello.py

print("Welcome to Python Programming!")

Run it via terminal:

python hello.py

🡪 **Using Jupyter Notebook:**

print("Hello from Jupyter!")

Just click **Run** in the notebook

-------------------------------------------------------------------------------------

* 2.Programming Style-

🡪Que-1:Understanding Python’s PEP 8 guidelines.

Ans.

**What is PEP 8?**

PEP 8 stands for **Python Enhancement Proposal 8**, which is the **official style guide** for writing clean and readable Python code.

It covers rules and conventions for:

* Code layout
* Naming styles
* Imports
* Indentation
* Comments
* Spacing
* Function and class definitions

**🔹 Purpose of PEP 8:**

* Make Python code more **readable and consistent**.
* Help teams collaborate effectively.
* Improve code **maintainability** in the long term.

-------------------------------------------------------------------------------------

🡪Que-2:Indentation, comments, and naming conventions in Python.

**🡪1. Indentation in Python**

* Python **uses indentation instead of braces {}** to define code blocks.
* Default is **4 spaces per indentation level** (tabs are discouraged).
* Incorrect indentation leads to **IndentationError**.

**Example:**

if age > 18:

print("Adult") # Indented 4 spaces

else:

print("Minor")

🡪**2. Comments**

* Used to **describe code** or **temporarily disable** lines.
* Ignored by the Python interpreter.

**Single-line comment:**

# This is a comment

**Multi-line comment:**

"""

This is a

multi-line comment

"""

Note: Python doesn’t have a true multi-line comment syntax; triple quotes are often used for that purpose but are technically docstrings.

**🡪3. Naming Conventions (from PEP 8)**

|  |  |  |
| --- | --- | --- |
| **Type** | **Convention** | **Example** |
| Variable | lowercase\_with\_underscores | user\_name, total\_amount |
| Constant | UPPERCASE | PI = 3.14 |
| Function name | lowercase\_with\_underscores | get\_data() |
| Class name | CapitalizedWords | StudentData |
| Private variable/function | \_single\_leading\_underscore | \_secret\_value |
| Python internal methods | **double\_underscores** | \_\_init\_\_, \_\_str\_\_ |

------------------------------------------------------------------------------------

🡪Que-3:Writing readable and maintainable code.

Ans.

**Writing Readable and Maintainable Code**

Writing readable and maintainable Python code means following best practices that make your code:

* **Clear** for others to understand.
* **Modular** and broken into reusable pieces.
* **Easy to debug, update, or scale.**

**🡪 Best Practices:**

1. **Follow PEP 8**: Consistent style helps readability.
2. **Use meaningful variable/function names**: Names should describe the purpose.

def calculate\_total\_price(items):

...

1. **Break long code into functions**: Avoid repeating the same logic.
2. **Write comments and docstrings**: Explain why, not just what.

def add(a, b):

"""Return the sum of two numbers."""

return a + b

1. **Avoid deep nesting**: It makes code hard to read.
2. **Use whitespace properly**: Around operators and after commas.

x = 10 + 5

fruits = ['apple', 'banana']

1. **Keep lines short (max 79 characters)**: Easier to read in all editors.
2. **Use version control (Git)**: Helps in maintaining and tracking changes.

-------------------------------------------------------------------------------------

* Que-3. SQL CONSTRAINTS:-

🡪Que-1:Understanding data types: integers, floats,strings, lists, tuples, dictionaries,sets.

Ans.

Python is **dynamically typed**, which means you don't need to declare a variable's type. It determines the type at runtime.

**🡪a. Integers (int)**

Whole numbers without a decimal point.

x = 10

**🡪 b. Floats (float)**

Numbers with decimal points.

y = 10.5

**🡪c. Strings (str)**

A sequence of characters enclosed in quotes.

name = "Yogesh"

🡪**d. Lists (list)**

Ordered, mutable collection. Can contain mixed data types.

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

fruits[0] = "grape" # lists are mutable

🡪**e. Tuples (tuple)**

Ordered, **immutable** collection.

dimensions = (1920, 1080)

# dimensions[0] = 1280 X Error: tuples can't be changed

🡪**f. Dictionaries (dict)**

Unordered key-value pairs.

student = {"name": "Yogesh", "age": 21}

print(student["name"])

**🡪 g. Sets (set)**

Unordered collection of unique elements.

colors = {"red", "green", "blue"}

colors.add("red") # no effect — duplicates are ignored

🡪Que-2:Python variables and memory allocation.

Ans.

**Variables:**

Used to store data values. Python automatically handles the data type.

x = 100 # int

name = "Ram" # str

**🡪 Memory Allocation:**

* Python uses **reference-based** memory management.
* When you write x = 10, Python stores 10 in memory and x points to that location.
* Python uses **automatic garbage collection** to free unused memory.

**🡪 Check memory address:**

x = 10

print(id(x)) # prints memory address of the object

Immutable objects (like int, str, tuple) are stored in memory once and reused

-------------------------------------------------------------------------------------

🡪Que-3:Python operators: arithmetic, comparison, logical, bitwise.

Ans.

**🡪a. Arithmetic Operators**

Used for basic math operations.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Result** |
| + | 5 + 2 | 7 |
| - | 5 - 2 | 3 |
| \* | 5 \* 2 | 10 |
| / | 5 / 2 | 2.5 |
| // | 5 // 2 | 2 |
| % | 5 % 2 | 1 |
| \*\* | 5 \*\* 2 | 25 |

🡪**b. Comparison Operators**

Used to compare values.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Result** |
| == | 5 == 5 | True |
| != | 5 != 3 | True |
| > | 5 > 3 | True |
| < | 5 < 3 | False |
| >= | 5 >= 5 | True |
| <= | 5 <= 3 | False |

🡪**c. Logical Operators**

Used with boolean values.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Result** |
| and | True and False | False |
| or | True or False | True |
| not | not True | False |

**🡪d. Bitwise Operators**

Operate on binary numbers.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Result** | **Description** |
| & | 5 & 3 | 1 | AND |
| ` | ` | `5 | 3` |
| ^ | 5 ^ 3 | 6 | XOR |
| ~ | ~5 | -6 | NOT |
| << | 5 << 1 | 10 | Left shift |
| >> | 5 >> 1 | 2 | Right shift |

**🡪Example Program Using All Concepts:**

name = "Yogesh"

age = 20

marks = [85, 90, 95]

student = {"name": name, "age": age, "passed": True}

print("Name:", student["name"])

print("Average marks:", sum(marks) / len(marks))

print("Is age >= 18?", age >= 18)

print("Bitwise AND of 5 & 3:", 5 & 3)

------------------------------------------------------------------------------------

* Que-4. Conditional Statement:-

Que-1:- Introduction to conditional statements: if, else, elif.

Ans.

Conditional statements are used to make decisions in a program.  
They allow the program to execute different code blocks depending on whether a condition is **True** or **False**.

**a) if Statement**

* The simplest form of decision-making.
* Executes a block of code only if the condition is **True**.
* If the condition is **False**, the block is skipped.

**Syntax:**

if condition:

statement(s)

**Example:**

age = 20

if age >= 18:

print("You are eligible to vote.")

**Output:**

You are eligible to vote.

**b) else Statement**

* Used with if to run code when the condition is **False**.
* Ensures that one block is executed in any case.

**Syntax:**

if condition:

statement(s)

else:

statement(s)

**Example:**

age = 16

if age >= 18:

print("You are eligible to vote.")

else:

print("You are not eligible to vote.")

**Output:**

You are not eligible to vote.

**c) elif (else if) Statement**

* Used when multiple conditions need to be checked.
* Executes the first elif (or if) condition that evaluates to **True** and skips the rest.

**Syntax:**

if condition1:

statement(s)

elif condition2:

statement(s)

elif condition3:

statement(s)

else:

statement(s)

**Example:**

marks = 75

if marks >= 90:

print("Grade: A+")

elif marks >= 75:

print("Grade: A")

elif marks >= 60:

print("Grade: B")

else:

print("Grade: C")

**Output:**

Grade: A

Que-2 **Nested if-else Conditions**

* An if (or else) statement **inside** another if or else block.
* Used when decisions depend on **multiple levels** of conditions.

**Syntax:**

if condition1:

if condition2:

statement(s)

else:

statement(s)

else:

statement(s)

**Example:**

age = 20

citizen = True

if age >= 18:

if citizen:

print("You can vote in the election.")

else:

print("You must be a citizen to vote.")

else:

print("You are too young to vote.")

**Output:**

You can vote in the election.

**Key Points to Remember**

* Indentation is **mandatory** in Python to define code blocks.
* elif is **optional** — you can use just if and else.
* Conditions use **comparison operators** (==, !=, <, >, <=, >=) and **logical operators** (and, or, not).
* Nested if-else should be used only when necessary; too much nesting can make code hard to read.

-------------------------------------------------------------------------------------

* Que-5.Looping (for,while):-

Que-1: Introduction to for and while loops.

Que-2:How loops work in Python.

Que-3:Using loops with collections (lists, tuples, etc.).