**Python Fundamentals**

q.1 Introduction to Python and its Features (simple, high-level, interpreted language).

Python is a powerful, easy-to-learn, and widely used programming language known for its simplicity and readability. It was created by **Guido van Rossum** and first released in **1991**. Python is used in various domains, including web development, data science, artificial intelligence, automation, and more

**(1 )Simple and Easy to Learn**

Python has a straightforward syntax that resembles English, making it beginner-friendly

(2) **High-Level Language**

* Python allows programmers to write code without worrying about low-level system details like memory management.

(3) **Interpreted Language**

* Python does not need compilation like C or Java. Instead, the code is executed line-by-line by the Python interpreter.
* This makes debugging easier

Q.2 History and evolution of Python.

**Origins of Python**

* **Creator**: Guido van Rossum
* **Year**: Late 1980s, officially released in 1991
* **Inspiration**: Derived from ABC (a teaching language) with features from C, Modula-3, and Unix shell scripting.
* **Goal**: Create a language that is easy to read, write, and maintain, while supporting both object-oriented and procedural programming.

**2. Python 1.x (1991-2000)**

* **Python 1.0 (1991)**: The first official release.
* **Key Features**:
  + Exception handling
  + Core data types (lists, dictionaries, strings)
  + Modules and functions

**3. Python 2.x (2000-2010)**

* **Python 2.0 (2000)**: Introduced major enhancements.
* **Key Features**:
  + List comprehensions
  + Garbage collection (cycle detection)
  + Unicode support
* **End of Life**: Python 2 was officially discontinued on January 1, 2020.

**4. Python 3.x (2008-Present)**

* **Python 3.0 (2008)**: A major overhaul with backward-incompatible changes.
* **Key Features**:
  + Print statement changed to print() function
  + Improved Unicode support
  + Better integer division (/ returns float, // returns int)
  + Enhanced standard library and performance improvements

**5. Recent Versions and Advancements**

* **Python 3.6+ (2016-Present)**
  + f-strings for better string formatting
  + Type hints for improved code clarity
  + Performance improvements
* **Python 3.9-3.12**:
  + Pattern matching (Python 3.10)
  + Structural pattern matching (like switch-case)
  + Faster execution and memory optimization

**6. Python's Popularity and Future**

* **Widespread Use**: Data science, AI, web development, automation, cybersecurity.
* **Community Support**: One of the largest and most active developer communities.
* **Future Trends**:
  + More AI/ML optimizations
  + Better performance with JIT (Just-In-Time) compilation
  + Expanding its role in cloud computing and automation

Q.3 Advantages of using Python over other programming languages

1. Easy to Learn and Use

2. Extensive Libraries and Frameworks

 Large standard library for tasks like data manipulation, web development, and machine learning.

 Popular frameworks:

* **Web development**: Django, Flask
* **Data Science & AI**: TensorFlow, PyTorch, Pandas, NumPy
* **Automation**: Selenium, PyAutoGUI

**3. Cross-Platform Compatibility**

* Runs on Windows, macOS, Linux, and even mobile platforms.
* Write once, run anywhere (WORA) without modifications

**4. Strong Community Support**

* Large global community for troubleshooting and learning.
* Thousands of open-source projects and extensive documentation.

**5. Versatility and Multi-Purpose**

* Used in **Web Development, Data Science, AI, Cybersecurity, Automation, IoT, Game Development**, etc.
* Unlike JavaScript (web-focused) or R (data-focused), Python is more general-purpose

**6. High Productivity**

* Faster development cycle due to fewer lines of code.
* Rapid prototyping and easy debugging.

**7. Dynamic Typing and Flexibility**

* No need to declare variable types explicitly.
* Allows quick and easy modifications during development.

**8. Integration and Interoperability**

* Can be integrated with C, C++, Java, and .NET.
* Supports APIs for database management and cloud services.

**9. Strong Support for Automation**

* Simple scripting capabilities make it perfect for automating repetitive tasks.
* Used in DevOps, network administration, and testing.

**10. Growing Demand and Career Opportunities**

* High demand in industries like AI, machine learning, and web development.
* Python developers are among the highest-paid programmers.

Q.4 Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).

. Step 1: Install Python

Download from the official site: https://www.python.org/downloads/

Install it and check the installation using:

python --version

Step 2: Choose an IDE (Integrated Development Environment)

Anaconda – Best for data science and machine learning.

PyCharm – Powerful IDE with smart features (good for professionals).

VS Code – Lightweight and flexible (good for beginners).

Q.5 Writing and executing your first Python program.

. Step 1: Open Python (IDLE or any IDE)

Step 2: Write a simple program

print("Hello, World!")

Step 3: Run the Program

If using an IDE, click Run.

If using a terminal, save the file as hello.py and run:

python hello.py

This will output:

Hello, World!

LAB :

Q.1 :-- • Write a Python program that prints "Hello, World!".

print("Hello, World!")

Q.2 • Set up Python on your local machine and write a program to display your name.

**Install Python** by running the installer and checking **“Add Python to PATH”** during setup.

**Verify Installation** by opening the terminal/command prompt and typing:

python –version

Python Program to Display Name

**Step 1: Create a Python File**

* Open a text editor or IDE (VS Code, or Notepad).
* Save the file as **name\_display.py**.

print("My name is rakib shaikh ")

**Step 3: Run the Program**

* **Command Prompt (Windows) or Terminal (Mac/Linux):**  
  Navigate to the file location and run:

**OUTPUT :**

My name is rakib shikh

Programming Style:---

Q.1 Understanding Python’s PEP 8 guidelines.

Key PEP 8 Guidelines:

**1. Indentation**

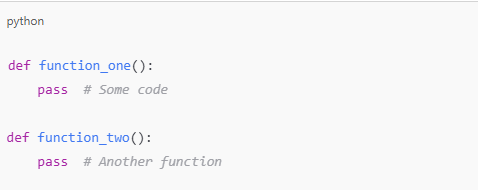
* Use **4 spaces per indentation level** (no tabs)

**2. Maximum Line Length**

* Keep lines **under 79 characters** (72 for docstrings).

**3. Blank Lines**

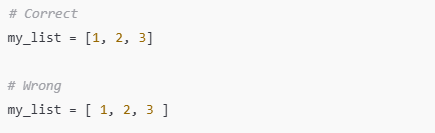
* Use **two blank lines** between top-level functions and classes.
* Use **one blank line** inside a function to separate logic



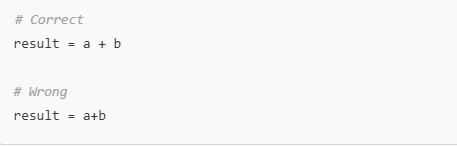
**4. Imports**

* Place **imports at the top** of the file.
* Use **one import per line** and follow this order:
  1. Standard library imports
  2. Third-party imports
  3. Local application imports

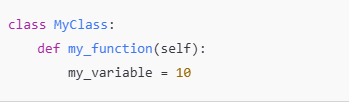
**5. Whitespace Usage**

* **Avoid extra spaces inside parentheses, brackets, or braces.**
  + 

Use a single space around operators.

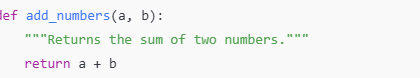


**6. Naming Conventions**

* **Variables & functions:** Use **snake\_case** (my\_variable, calculate\_sum())
* **Classes:** Use **PascalCase** (MyClass)
* **Constants:** Use **UPPER\_CASE** (MAX\_LIMIT = 100)
  + 

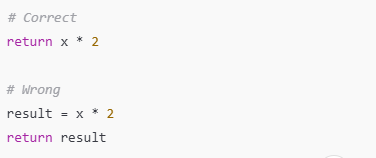
**7. Comments & Docstrings**

* **Use meaningful comments**, not redundant ones.
* **Docstrings (""" """)** for modules, classes, and functions.



**8. Avoid Unnecessary Variables**

* **Use direct returns instead of unnecessary temporary variables.**



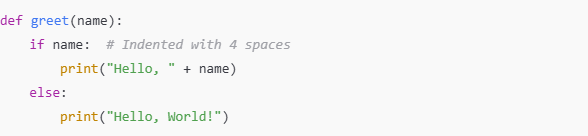
Q.2 Indentation, comments, and naming conventions in Python.

Indentation is crucial in Python as it defines **code blocks** (unlike other languages that use {} or begin...end).

**✅ Rules for Indentation:**

* Use **4 spaces per indentation level** (not tabs).
* Indentation is **mandatory** in Python (unlike C/C++/Java).
* Maintain **consistent** indentation throughout the code.

**Example of Proper Indentation:**

****

Q.3 Writing readable and maintainable code

* Follow PEP 8 – Use consistent style and formatting.
* Use meaningful variable & function names – Make names descriptive.
* Yes:def calculate\_salary(hours, rate):
* no:def cal(hr, r):
* Break long code into functions – Avoid long blocks of code.
* Write comments where necessary – But don’t over-comment obvious things.
* Use list comprehensions for readability – Instead of long loop

**Core Python Concepts :**

Q.1 Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.

**1) Integers (int)**

* Whole numbers (positive, negative, or zero).
* Example:

x = 10 # Integer

**2) Floats (float)**

* Decimal numbers.
* Example:

pi = 3.14

**3) Strings (str)**

* Sequence of characters enclosed in single or double quotes.
* Example:

name = "Alice"

**4) Lists (list)**

* Ordered, **mutable** (changeable), and can store different data types.
* Example:

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

**5) Tuples (tuple)**

* Ordered, **immutable** (cannot be changed after creation).
* Example:

coordinates = (10, 20)

**6) Dictionaries (dict)**

* Stores data in key-value pairs.
* Example

student = {"name": "Alice", "age": 20, "grade": "A"}

**7) Sets (set)**

* **Unordered, unique elements** (no duplicates).
* Example:

numbers = {1, 2, 3, 4, 5, 2}

Q.2 Python variables and memory allocation.

**Python Variables**

* A variable is a **named reference** to a value stored in memory.
* Python does **dynamic typing** (you don’t need to declare the type).
* Example:

x = 10 # Integer

name = "Bob" # String

pi = 3.14 # Float

**2) Memory Allocation**

* Python automatically manages memory using an interpreter and garbage collector.
* **Reference Counting:** Python tracks how many times a variable is used.
* Example:

a = 10

b = a # Both a and b refer to the same memory location

Q.3 Python operators: arithmetic, comparison, logical, bitwise.

**1) Arithmetic Operators**

| **Operator** | **Description** | **Example (a = 10, b = 3)** | **Result** |
| --- | --- | --- | --- |
| + | Addition | a + b | 13 |
| - | Subtraction | a - b | 7 |
| \* | Multiplication | a \* b | 30 |
| / | Division | a / b | 3.333 |
| // | Floor Division | a // b | 3 (removes decimals) |
| % | Modulus (Remainder) | a % b | 1 |
| \*\* | Exponentiation | a \*\* b | 1000 (10³) |

**2) Comparison (Relational) Operators**

| **Operator** | **Description** | **Example (a = 10, b = 3)** | **Result** |
| --- | --- | --- | --- |
| == | Equal to | a == b | False |
| != | Not equal to | a != b | True |
| > | Greater than | a > b | True |
| < | Less than | a < b | False |
| >= | Greater than or equal to | a >= b | True |
| <= | Less than or equal to | a <= b | False |

**3) Logical Operators**

| **Operator** | **Description** | **Example (a = True, b = False)** | **Result** |
| --- | --- | --- | --- |
| and | Returns True if both are True | a and b | False |
| or | Returns True if at least one is True | a or b | True |
| not | Reverses the logical state | not a | False |

**4) Bitwise Operators**

Used for binary-level operations.

| **Operator** | **Description** | **Example (a = 5 (0101), b = 3 (0011))** | **Result** |
| --- | --- | --- | --- |
| & | AND | a & b | 1 (0001) |
| ` | ` | OR | `a |
| ^ | XOR | a ^ b | 6 (0110) |
| ~ | NOT (inverts bits) | ~a | -6 (2’s complement) |
| << | Left shift | a << 1 | 10 (1010) |
| >> | Right shift | a >> 1 | 2 (0010) |

4. Conditional Statements

Theory:

Q.1 Introduction to conditional statements: if, else, elif.

Conditional Statements in Python

Conditional statements allow a program to make decisions based on conditions. Python provides the following conditional statements:

if statement – Executes a block of code if the condition is True.

age = 18

if age >= 18:

print("You are eligible to vote.")

if-else statement – Executes one block if True, another if False.

age = 16

if age >= 18:

print("You are eligible to vote.")

else:

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

if-elif-else statement – Checks multiple conditions sequentially

marks = 85

if marks >= 90:

print("Grade: A+")

elif marks >= 80:

print("Grade: A")

elif marks >= 70:

print("Grade: B")

else:

print("Grade: C")

2) Nested if-else conditions.

When an if statement is inside another if, it’s called nested if-else.

num = 10

if num > 0:

print("Positive number")

if num % 2 == 0:

print("Even number")

else:

print("Odd number")

else:

print("Negative number or Zero")

5. Looping (For, While)

Q.1 Introduction to for and while loops.

1) for Loop

The for loop is used to iterate over a sequence (list, tuple, dictionary, set, string).

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

for fruit in fruits:

print(fruit)

2) while Loop

The while loop runs as long as the condition is True.

count = 1

while count <= 5:

print(count)

count += 1 # Increment count

Q.2 How loops work in Python.

Loops in Python execute a block of code multiple times until a condition is met. They help in automation, iteration over data structures, and reducing repetitive code

**6. Generators and Iterators**

1)Understanding how generators work in Python.

Ans. Generators in Python are a special type of iterable that allow you to iterate over data lazily rather than storing everything in memory at once. They are useful when dealing with large data sets or when you need to generate values dynamically.

2)Difference between yield and return.

Ans.

|  |  |
| --- | --- |
| Yield | return |
| Creates a generator function. | Creates a regular function. |
| Pauses function execution and remembers the state. | Ends function execution immediately. |
| Returns a generator object. | Returns a single value. |
| Used in generators for lazy iteration. | Used in normal functions to return values. |
| Can return multiple values one at a time. | Returns a single value or a tuple. |

3)Understanding iterators and creating custom iterators.

Ans. An iterator is an object that allows sequential traversal of elements in a collection (like lists, tuples, dictionaries, or custom objects) without needing to store all elements in memory.

Iterator: An object that implements two methods:

\_\_iter\_\_() → Returns the iterator object itself.

\_\_next\_\_() → Returns the next value from the sequence, raising StopIteration when finished.

7. Functions and Methods Theory

1)Defining and calling functions in Python,

Ans. Defining: A function is defined using the def keyword, followed by the function name and parentheses ().

Calling: Once defined, a function is called using its name followed by parentheses. If the function takes parameters, pass values inside the parentheses.

2)Function arguments (positional, keyword, default).

1. Positional Arguments:

These arguments must be passed in the correct order as defined in the function.

The function assigns values based on position.

2. Keyword Arguments:

You specify argument names explicitly when calling the function.

Order does not matter since arguments are passed by name.

3. Default Arguments:

Assigns a default value to a parameter if no value is provided.

If an argument is given, it overrides the default value.

3)Scope of variables in Python.

In Python, the scope of a variable refers to the region of the program where the variable is accessible. Understanding variable scope is crucial for managing variable lifetimes and avoiding naming conflicts. Python has several types of scopes, which are:

1. Local Scope:

Variables defined within a function are in the local scope of that function. They can only be accessed from within that function.

2. Enclosing Scope:

This scope refers to variables in the local scope of enclosing functions. It is relevant in nested functions, where an inner function can access variables from its enclosing function.

3. Global Scope:

Variables defined at the top level of a script or module are in the global scope. They can be accessed from any function within the same module.

4)Built-in methods forstrings, lists, etc.

Python provides a rich set of built-in methods for various data types, including strings, lists, dictionaries, sets, and more.

str.lower(): Converts all characters in the string to lowercase.

str.upper(): Converts all characters in the string to uppercase.

str.title(): Converts the first character of each word to uppercase.

str.strip(): Removes leading and trailing whitespace.

**8. Control Statements (Break, Continue, Pass)**

1)Understanding the role of break, continue, and pass in Python loops.

**Break:** The break statement is used to exit a loop prematurely. When break is encountered, the loop is terminated immediately, and the program continues with the next statement following the loop.

**Continue:** The continue statement is used to skip the current iteration of a loop and move to the next iteration. When continue is encountered, the remaining code inside the loop for that iteration is skipped, and the loop proceeds with the next iteration.

**Pass:** The pass statement is a null operation; it is a placeholder that does nothing when executed. It is often used in situations where syntactically some code is required but you do not want to execute any code. This can be useful for creating minimal classes, functions, or loops that you plan to implement later.

**9. String Manipulation**

1)Understanding how to access and manipulate strings.

In Python, strings are a sequence of characters and are one of the most commonly used data types. You can access and manipulate strings in various ways.

**Indexing**: You can access individual characters in a string using indexing. Python uses zero-based indexing, meaning the first character is at index 0.

**Slicing**: You can extract a substring from a string using slicing. The syntax is s[start:end], where start is the index of the first character and end is the index of the character just after the last character you want to include.

Negative Indexing: You can use negative indices to access characters from the end of the string. -1 refers to the last character, -2 to the second last, and so on.

2) Basic operations: concatenation, repetition,string methods(upper(), lower(), etc.).

Concatenation: Concatenation is the operation of joining two or more strings together. In Python, you can concatenate strings using the + operator.

**Repetition**: Repetition allows you to repeat a string multiple times using the \* operator.

**String Methods:** Python provides a variety of built-in string methods that allow you to manipulate and query strings. Here are some commonly used string methods:

**a. upper()**

Converts all characters in the string to uppercase.

**b. lower()**

Converts all characters in the string to lowercase.

3)String slicing.

String slicing in Python allows you to extract a portion of a string by specifying a range of indices. This is a powerful feature that enables you to manipulate and access substrings easily.

Syntax: substring = string[start:end:step]

**10. Advanced Python (map(), reduce(), filter(), Closures and Decorators)**

Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing state and mutable data. Python supports functional programming features, allowing you to write code in a functional style.

2) Using map(), reduce(), and filter() functions for processing data.

In Python, the map(), filter(), and reduce() functions are powerful tools for processing data in a functional programming style. They allow you to apply functions to iterables (like lists) in a concise and readable manner.

**map():** The map() function applies a given function to all items in an iterable (like a list) and returns a map object (which can be converted to a list). It is useful for transforming data.

Syntax: map(function, iterable)

**filter():** The filter() function filters elements from an iterable based on a function that returns True or False. It returns a filter object, which can also be converted to a list.

Syntax: filter(function, iterable)

**reduce():** The reduce() function, which is part of the functools module, reduces an iterable to a single value by applying a binary function cumulatively. It takes two arguments: a function and an iterable.

Syntax: from functools import reduce

reduce(function, iterable)

3)Introduction to closures and decorators.

Closures and decorators are two important concepts in Python that leverage the power of functions as first-class citizens. They allow for more flexible and reusable code.