Name :- ghoghari mohit

Introduction to python

Q1. Introduction to python and its features

(simple, high-level,interpreted language).

ANS

**Introduction to Python and Its Features**

Python is a **simple, high-level, interpreted** programming language known for its **readability and ease of use**. It was created by **Guido van Rossum** and first released in **1991**. Python is widely used in web development, data science, artificial intelligence, automation, and more.

**Key Features of Python:**

1. **Simple & Easy to Learn** – Clean and readable syntax.
2. **High-Level Language** – No need to manage memory manually.
3. **Interpreted** – Executes code line by line, making debugging easier.
4. **Dynamically Typed** – No need to declare variable types.
5. **Object-Oriented** – Supports classes and objects for modular code.
6. **Cross-Platform** – Runs on Windows, Mac, and Linux

Python’s simplicity and power make it an ideal choice for beginners and professionals alike!

**Q2. History and evolution of Python.**

**ANS.**

**1. Origins (Late 1980s - 1991)**

 **Conception**: Python's development began in December 1989 as a successor to the ABC programming language. Guido van Rossum aimed to address ABC's shortcomings while retaining its strengths.

### 2. Early Growth (1991-2000)

 **Python 1.0 (1994)**: Introduced features like functional programming tools (map, filter, reduce) and the module system, allowing for better organization of code.

**3. Maturing Language (2000-2010)**

 **Python 2.0 (2000)**: Introduced list comprehensions, garbage collection, and support for Unicode. This version became widely adopted in various industries.

**4. Current Developments (2010s-present)**

**Python 3.x Series**: Ongoing updates have introduced many enhancements, including:

 **Python 3.3 (2012)**: The "yield from" expression and a new I/O library.

 **Python 3.6 (2016)**: F-strings for easier string formatting.

**Q3. Advantages of using Python over other programming languages**

ANS

Python offers several advantages that make it a popular choice among developers across various domains. Here are some key benefits of using Python over other programming languages:

**1. Ease of Learning and Use**

 **Simple Syntax**: Python’s syntax is clear and readable, making it easier for beginners to learn and understand

2. Versatility

* **Extensive Libraries and Frameworks**: Python has a wide range of libraries and frameworks (e.g., Django, Flask, NumPy, Pandas, TensorFlow), which make it highly adaptable for different fields such as web development, data science, AI, machine learning, automation, and more.

**3.Readable Cod**

* It also promotes a clean coding style, making collaboration between developers easier.

4.**Integration Capabilities**:

.Python can easily integrate with other languages (like C, C++, and Java) and technologies, making it suitable for building complex systems.

5.**Rapid Development**:

. Python's simplicity and the availability of frameworks (like Django and Flask for web development) allow for faster development cycles.

1. **Support for Object-Oriented and Functional Programming** 

Python supports multiple programming paradigms, allowing developers to choose the best approach for their projects.

**Q4. Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).**

**ANS  
Installing Python & Setting Up the Development Environment**

**1. Installing Python:**

* Download Python from the official site: [python.org](https://www.python.org/)
* Install it and ensure **"Add Python to PATH"** is checked.

**2. Setting Up a Development Environment:**

**Anaconda (For Data Science & ML)**

* Download from [anaconda.com](https://www.anaconda.com/)
* Comes with **Jupyter Notebook, Spyder, and Conda** package manager.

**PyCharm (For Professional Development)**

* Download from [jetbrains.com/pycharm](https://www.jetbrains.com/pycharm/)
* Best for large-scale Python projects.
* Install **Python extension** for better support.
* After setup, you can start coding by opening the terminal or IDE and running:
* print("Hello, Python!")
* Now you're ready to code in Python!

**Q4. Writing and executing your first Python program.**

**ANS**

**1. Using Python Interpreter (Quick Test)**

Open the terminal or command prompt and type:

python

print("Hello, Python!")

Press **Enter**, and it will print:

Hello, Python!

**2. Writing a Python Script (.py File)**

1. Open a text editor (Notepad, VS Code, PyCharm, etc.).
2. Write the following code and save it as **first\_program.py**:
3. print("Hello, World!")
4. Run the script in the terminal or command prompt:
5. python first\_program.py
6. Output:
7. Hello, World!

Congratulations! You've written your first Python program!

* **Programming Style**

**Q1.** **Understanding Python’s PEP 8 guidelines**

**ANS**

**Python’s PEP 8 Guidelines (Code Style Best Practices)**

PEP 8 ensures **clean, readable, and consistent** Python code.

✅ **Key Rules:**

1. **Indentation** – Use **4 spaces**, no tabs.
2. **Max Line Length** – **79 characters** (72 for docstrings).
3. **Blank Lines** – **2 lines** between top-level functions/classes.
4. **Imports** – One module per line, properly ordered.
5. **Whitespace** – No extra spaces inside parentheses or before commas.
6. **Naming Conventions** –
   * Variables & functions: **snake\_case** (my\_function)
   * Classes: **PascalCase** (MyClass)
   * Constants: **UPPER\_CASE** (PI = 3.14)
7. **Comments** – Keep them **short and meaningful**.

Following PEP 8 makes your code **more readable and professional**!

**Q2. Indentation, comments, and naming conventions in Python.**

**ANS**

**Indentation, Comments, and Naming Conventions in Python**

.**Indentation** (for code blocks)

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

* Required in loops, functions, classes, etc.
* def greet():
* print("Hello, Python!") # Indented correctly

. **Comments** (for code clarity)

* **Single-line comment:** Use #
* # This is a single-line comment
* print("Hello, World!")
* **Multi-line comment:** Use triple quotes """ """ or ''' '''
* """
* This is a multi-line comment.
* It explains the code in detail.
* """

. **Naming Conventions** (for readability)

* **Variables & Functions:** snake\_case → my\_variable, calculate\_sum()
* **Classes:** PascalCase → MyClass
* **Constants:** UPPER\_CASE → PI = 3.14

Following these practices improves **code readability and maintainability**! 🚀

**Q3. Writing readable and maintainable code.**

**ANS**

**Tips for Readable & Maintainable Python Code**

**Follow PEP 8** – Proper indentation, spacing, and naming.  
**Use Meaningful Names** – Clear variable & function names.  
**Keep Functions Short** – Each function does one task.  
**Write Comments & Docstrings** – Explain complex logic.  
**Use List Comprehensions** – For cleaner loops.  
**Avoid Hardcoding** – Use constants instead.  
**Modular Code** – Use functions & classes for reusability.

Clean code is **easy to read, debug, and maintain!** 🚀

* **Core Python Concepts**

**Q1. Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets**

**ANS**

**Python Data Types (Quick Guide)**

**int** – Whole numbers → x = 10  
**float** – Decimals → y = 3.14  
**str** – Text → name = "Python"  
**list** – Ordered, mutable → fruits = ["apple", "banana"]  
**tuple** – Ordered, immutable → coords = (10, 20)  
**dict** – Key-value pairs → student = {"name": "John"}  
**set** – Unique, unordered → unique\_nums = {1, 2, 3}

Each type serves a **specific purpose**!

**Q2. Python variables and memory allocation.**

**AnsPython Variables & Memory (Easy Explanation)**

**What is a Variable?**  
A variable is like a **container** that holds data.

x = 10 # x stores the number 10

**How Does Python Store Data?**

* **Stack** → Keeps track of variable names.
* **Heap** → Stores actual values (like numbers, lists, etc.).
* **Garbage Collector** → Removes unused data automatically.

**Example:**

a = [1, 2, 3] # A list is created in memory

b = a # `b` also points to the same list

del a # The list still exists because `b` is using it

Python **handles memory automatically**, so you don’t have to worry! 🚀

**Q3. Python operators: arithmetic, comparison, logical, bitwise.**

**ANS**

* **Conditional Statements**

**Q1. Introduction to conditionalstatements: if, else, elif**

**ANS=**

**Conditional Statements in Python**

Conditional statements allow **decision-making** in Python.

**1. if Statement** (Executes if condition is true)

x = 10

if x > 5:

print("x is greater than 5")

**2. if-else Statement** (Executes one of two blocks)

x = 3

i

**Conditional statements help control the flow of a program!**

**Q2. Nested if-else conditions.**

**ANS**

**Nested if-else in Python**

A **nested if-else** means an if inside another if.

**Example:**

x = 10

if x > 5:

print("x is greater than 5")

if x > 8:

print("x is also greater than 8")

else:

print("x is between 5 and 8")

else:

print("x is 5 or less")

.**How it Works?**

* If the **first if** is true, the inner condition is checked.
* If the first if is false, the outer else runs.

Used for **complex decision-making!**

* **Looping**

**Q1. Introduction to for and while loops.**

**ANS**

**Loops in Python (for & while)**

Loops are used to **repeat tasks** in Python.

**1. for Loop** (Iterates over a sequence)

for i in range(5):

print(i) # Output: 0, 1, 2, 3, 4

**2. while Loop** (Repeats while condition is true)

x = 0

while x < 5:

print(x)

x += 1 # Increases x to avoid infinite loop

. **for → Best for fixed loops**  
. **while → Best for loops with conditions**

Loops help automate **repetitive tasks!** 🚀

**Q2. How loops work in Python.**

**ANS**

**How Loops Work in Python**

**1. for Loop**

for i in range(3):

print(i)

**2. while Loop**

x = 0

while x < 3:

print(x)

x += 1

**Q3. Using loops with collections (lists, tuples, etc.).**

**ANS**

**Using Loops with Collections in Python**

**1. for Loop with Lists**

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

for fruit in fruits:

print(fruit)

**2. for Loop with Tuples**

numbers = (1, 2, 3)

for num in numbers:

print(num)

**3. for Loop with Dictionaries**

student = {"name": "John", "age": 25}

for key, value in student.items():

print(key, value)

**4. for Loop with Sets**

unique\_numbers = {10, 20, 30}

for num in unique\_numbers:

print(num)

Loops make working with collections easy!

* **Functions and Method**

**Q1. Defining and calling functions in Python.**

**ANS**

**. Defining a Function**

def greet(name):

return f"Hello, {name}!"

**calling function**

message = greet("Alice")

print(message)

**Q2. Function arguments (positional, keyword, default).**

**ANS**

### 1. Positional Arguments

def multiply(a, b):

return a \* b

result = multiply(3, 4) # 3 is assigned to a, 4 to b

print(result) # Output: 12

### 2. Keyword Arguments

### def introduce(name, age):

### return f"My name is {name} and I am {age} years old."

### result = introduce(age=30, name="Alice") # Order doesn't matter

### print(result) # Output: My name is Alice and I am 30 years old.

### 3. Default Arguments

### def greet(name="User"):

### return f"Hello, {name}!"

### print(greet()) # Output: Hello, User!

### print(greet("Alice")) # Output: Hello, Alice!

**Q3. Scope of variables in Python.**

**ANS**

**Scope of Variables in Python**

### 1. Local Scope

### c def my\_function():

### x = 10 # Local variable

### print(x)

### my\_function() # Output: 10

### # print(x) # This would raise a NameError since x is not accessible here.

**2. Global Scope**

x = 20

def my\_function():

print(x) # Can access global variable

my\_function()

**3. Enclosing (Nonlocal) Scope**

def outer():

y = 30

def inner():

y += 5

print(y)

inner()

outer()

**4. Built-in Scope** (Python’s predefined names)

print(len("Hello")) # `len` is built-in

Understanding scope helps avoid **variable conflicts and errors!**

**Q4. Built-in methods for strings, lists, etc.**

**ANS**

**Buil-in methods for string,liste,etc**

**1.string methods**

Txt=”hello word”

Print(text.upper())

Print(txt.replace(“world”,”java”))

**2.liste methods**

Fruits=[“apple”,”banana”]

Fruits.append(“mango”)

Fruits.remove(“apple”)

Print(fruits)

**3.tuple methods**

Number =(3,3,4,4)

Print(number.count(4))

Print(numbers.index(3))

**4.Dictionary methods**

Studet={“name”:”Rahul”,”age”:18

Print(student.keys())

Print(student.values())

**5.set methods**

Unique\_numbers={1,2,3}

Unique\_numbers.add(4)

Unique\_numbers.discard(2)

Print(unique\_numbers)

**.control statements**

**Q1.** **Understanding the role of break, continue, and pass in Python loops.**

**ANS**

**In Python, the break, continue, and pass statements control the flow of loops in different ways. Here’s a breakdown of each**

** Use break to exit a loop entirely.**

** Use continue to skip the current iteration and move to the next one.**

**. Use pass when you need a placeholder for future code or when a statement is syntactically required.**

for i in range(5):

if i == 2:

break # Stops the loop when i is 2

print(i)

# Output: 0, 1

for i in range(5):

if i == 2:

continue # Skips printing 2

print(i)

# Output: 0, 1, 3, 4

for i in range(5):

if i == 2:

pass # Does nothing, loop continues

print(i)

# Output: 0, 1, 2, 3, 4

**Q2. Practical Example: 1) Write a Python program to skip 'banana' in a list using the continue statement. List1 = ['apple', 'banana', 'mango']**

**ANS**

# List of fruits

list1 = ['apple', 'banana', 'mango']

# Loop through the list

for fruit in list1:

if fruit == 'banana':

continue # Skip 'banana'

print(fruit)

**output:**

apple

mango

**Q3.** **Practical Example: 2) Write a Python program to stop the loop once 'banana' is found using the break statement.**

**ANS**

# List of fruits

list1 = ['apple', 'banana', 'mango']

# Loop through the list

for fruit in list1:

if fruit == 'banana':

break # Stop the loop when 'banana' is found

print(fruit)

**output**:

apple

**.string manipulation**

**Q1.** **Understanding how to access and manipulate strings.**

**ANS**

text = "Hello"

print(text[0])

print(text[-1])

print(text[1:4])

print(text[:3])

print(text[::2])

text = "hello python"

print(text.upper())

print(text.lower())

print(text.replace("python", "world"))

print(text.split())

print(" ".join(["Hello", "world"]))

name=”Rahul”

age=18

print(name+”is”+str(age)+”yers old.”)

print(f“{name}is{age}yers old.”)

print(“hello”in”hello python”)

print(”world”not in”hello python”)

**Q2. Basic operations: concatenation, repetition, string methods (upper(), lower(), etc.).**

**ANS**

str1 = "Hello"

str2 = " World"

result = str1 + str2

print(result)

text = "Python "

print(text \* 3)

text = "Hello World"

print(text.upper())

print(text.lower())

print(text.title())

print(text.strip())

print(text.replace("World", "Python"))

print(text.split())

print(" - ".join(["Hello", "Python"]))

**Q3. String slicing**

**ANS**

text = "Hello World"

print(text[0:5])

print(text[6:])

print(text[-5:])

print(text[::2])

print(text[::-1])