Day 2: History and Internals of Python :

1. **History of Python**

Introduction :

Python is a high-level, interpreted, and general-purpose programming language.

It was created by Guido van Rossum in the late 1980s and was first released in 1991.

Origin of the Name

The name "Python" was inspired by the British comedy series "Monty Python's Flying Circus", not the snake.

Development Timeline

- Late 1980s: Guido van Rossum started working on Python during his time at CWI (Centrum Wiskunde & Informatica) in the Netherlands.
- 1991: The first version, Python 0.9.0, was released. It included features like functions, exception handling, and modules.
- 2000: Python 2.0 was released, introducing list comprehensions and garbage collection using reference counting.
- 2008: Python 3.0 was introduced, which was not backward-compatible but brought major improvements.
- Present: Python continues to evolve, with the latest versions adding features like pattern matching and performance improvements.

Key Organizations Supporting Python :

Python Software Foundation (PSF): A non-profit organization that manages the language's development and distribution.

Open-source contributors: The language thrives because of its strong, collaborative community.

2. Features of Python

1. Easy to Learn and Use

- Python has a simple syntax similar to English, making it easy for beginners to read and understand.

2. Open Source

- Python is free to use and distribute, even for commercial purposes.

3. Interpreted Language

- Python code is executed line by line, which simplifies debugging and makes it platform-independent.

4. High-Level Language

- You don't need to manage memory manually or understand the system architecture deeply.

5. Dynamically Typed

- Variables in Python don't need explicit declarations. Example:

Example code:

```
x = 10  # x is an integer
x = "Hello"  # Now x is a string
```

6. Extensive Standard Library

Python comes with a rich library of modules and functions, covering areas like:

- Web development
- Data science
- Machine learning

- GUI programming

7. Cross-Platform

Python programs can run on different operating systems (Windows, macOS, Linux) without modification.

8. Object-Oriented

- Supports object-oriented programming (OOP), including inheritance, polymorphism, and encapsulation.

9. Embeddable and Extensible

- Python can be embedded within other languages like C or C++ to give scripting capabilities.

10. Scalable

- Python's flexibility makes it suitable for small scripts as well as large-scale projects.

11. Community Support

- Python has a vast and active community, making it easy to find tutorials, documentation, and solutions to problems.

3. Why Python Is Popular Widely used in fields such as:

- Web Development: Frameworks like Django, Flask.
- Data Science: Libraries like NumPy, pandas, and Matplotlib.
- Machine Learning: Libraries like TensorFlow and PyTorch.
- Automation: Simplifies repetitive tasks with minimal code.
- Game Development: Frameworks like Pygame.

Interactive Activities

Quick Quiz:

1. Who created Python, and in which year?
2. Name any two features of Python.

Discussion:

1. Ask students to brainstorm why Python might be popular in different industries.

Coding Challenge:

Write a Python program that uses dynamic typing.

example code

x = 42

print(x)

x = "Python"

print(x)

This structured content provides both theoretical knowledge and interactive learning opportunities to engage students effectively. Let me know if you'd like slides or additional teaching aids!

Memory Management in Python:

1. Introduction

Memory management in Python is a crucial concept that determines how the interpreter allocates, uses, and frees memory during the execution of a program. Python has an efficient built-in memory management system that handles memory allocation and garbage collection automatically.

2. Key Components of Python's Memory Management

Memory Allocation

Python uses different components to manage memory allocation:

Private Heap Space:

All Python objects and data structures are stored in a private heap.

The interpreter manages this heap, and users cannot access it directly.

Memory Manager:

The memory manager oversees the allocation of memory in the heap.

Object-Specific Allocators:

Python has specialized allocators for handling common types of objects like integers, strings, and lists.

Reference Counting

Python uses a reference count mechanism to track the number of references to an object.

When the reference count drops to zero, the memory occupied by the object is deallocated.

Example:

code

```
a = [1, 2, 3]  # Reference count = 1
b = a          # Reference count = 2
del a          # Reference count = 1
del b          # Reference count = 0 (memory is freed)
```

Garbage Collection

Python includes a garbage collector to reclaim unused memory.

It removes objects that are no longer reachable, such as objects in a reference cycle.

Example of a reference cycle:

python

Copy code

```
a = []
b = [a]
a.append(b)  # Reference cycle: a → b → a
del a, b     # Garbage collector handles the cycle
```

Dynamic Memory Management

Python manages memory dynamically at runtime, which allows flexibility but comes with overhead.

Python’s Memory Optimization Techniques

Small Object Pools

Python maintains pools for frequently used objects like integers (-5 to 256) and strings, reducing the overhead of repeatedly allocating and deallocating memory.

Interning

Frequently used strings and numbers are "interned" for reuse:

code

a = "hello"

b = "hello"

print(a is b) # True (both point to the same memory location)

Memory Deallocation

Objects that go out of scope or lose all references are automatically deallocated.

4. Tools to Monitor and Optimize Memory Usage

gc Module:

Allows manual garbage collection and inspection:

python

Copy code

import gc

gc.collect() # Trigger garbage collection

sys.getsizeof():

Returns the size of an object in bytes:

python

Copy code

```
import sys

x = [1, 2, 3]

print(sys.getsizeof(x))  # Output: Memory size of the list
```

Memory Profilers:

Tools like memory_profiler and objgraph help monitor memory usage in real-time.

5. Best Practices for Efficient Memory Management

- Use variables wisely to avoid unnecessary references.
- Use immutable objects like tuples instead of lists where possible.
- Avoid creating reference cycles unnecessarily.
- Utilize libraries like NumPy for memory-efficient computations.
- Explicitly delete large objects if they are no longer needed using del.

6. Limitations

Python's memory management may not be as efficient as lower-level languages like C or C++.

The Global Interpreter Lock (GIL) in CPython can impact multi-threaded memory management.

Interactive Examples

Reference Counting Example:

python

Copy code

```
import sys
a= [1, 2, 3]
print(sys.getrefcount(a))  # Check reference count
b = a
print(sys.getrefcount(a))  # Increased count
del b
print(sys.getrefcount(a))  # Decreased count
```

Garbage Collection Example:

python

Copy code

```
import gc
class Node:
def __init__(self, value):
self.value = value
self.ref = None
a = Node(1)
b = Node(2)
a.ref = b
b.ref = a  # Creates a reference cycle
```

```
del a

del b

gc.collect()  # Reclaims memory from the cycle
```

Setting Up Python Environment & Installation of Python and Code Editors

1. Setting Up the Python Environment

- What You Need
- Python installed on your computer.
- A code editor or Integrated Development Environment (IDE) for writing and running Python code.

2. Installing Python

- Step-by-Step Guide
- Go to the official Python website: https://www.python.org.
- Download the latest stable version of Python for your operating system (Windows, macOS, Linux).
- Run the installer and make sure to check the option "Add Python to PATH" during installation.

Verify installation by opening a terminal or command prompt and typing:

- bash / cmd terminal
- Copy code
- python --version
- or
- bash /cmd terminal
- Copy code
- python3 --version

3. Installing Code Editors

- Popular Code Editors for Python
- VS Code (Visual Studio Code):
- Lightweight and highly customizable.
- Extensions for Python debugging and linting.
- PyCharm:
- Specifically designed for Python development, with advanced features.
- Jupyter Notebook:
- Ideal for interactive coding and data analysis.
- IDLE:
- Comes pre-installed with Python; beginner-friendly.

4. Setting Up VS Code for Python

- Install VS Code from https://code.visualstudio.com.
- Install the Python extension from the Extensions Marketplace.
- Configure the interpreter by pressing Ctrl+Shift+P, typing Python: Select Interpreter, and choosing your Python version.
- Create and save a file with the .py extension to start writing Python code.

First Python Program: "Hello, World!"

1. Introduction to Writing Your First Program

Python is an interpreted language, so you can write and run your code line by line.

2. Writing "Hello, World!" in Python

Open your preferred code editor or Python shell.

Write the following code:

python

Copy code

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

Save the file as hello.py if using a code editor.

Run the program:

From the terminal or command prompt:

bash /cmd terminal

Copy code

python hello.py

In the Python Shell:

Simply type print("Hello, World!") and press Enter.

3. Output

When you run the code, the output will be:

Copy code

Hello, World!

Print Output Using print() Function

1. Understanding the print() Function

The print() function is used to display output on the screen.

Syntax:

python

Copy code

print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

*objects: One or more values to print.

sep: Separator between values (default is a space ' ').

end: String appended after the last value (default is a newline '\n').

2. Examples of Using print()

Basic Printing:

python

Copy code

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

Output:

Copy code

```
Hello, World!
```

Printing Multiple Items:

Copy code

```
print("Python", "is", "fun!")
```

Output:

python

Copy code

```
print("Python", "is", "fun!", sep="-")
```

Output:

Copy code

```
Python-is-fun!
```

Changing the End Character:

python

Copy code

```
print("Python", end=" ")
print("is fun!")
```

Python

Copy code

```
name = "John"
age = 25
```

```
print("Name:", name, "Age:", age)
```

Output:

makefile

Copy code

```
Name: John Age: 25
```

2. Using Escape Characters

Escape characters modify the behavior of strings within print().

Examples:

Newline: \n

python

Copy code

```
print("Hello\nWorld")
```

Output:

Copy code

```
Hello
World
```

Tab: \t

python

Copy code

```
print("Hello\tWorld")
```

Output:

Copy code

```
Hello   World
```

4. Practical Examples

Printing Mathematical Expressions:

python

Copy code

```
print("The sum of 5 and 3 is:", 5 + 3)
```

Output:

python

Copy code

```
The sum of 5 and 3 is: 8
```

Dynamic Output Using Variables:

python

Copy code

```
name = "Alice"
age = 30
print(f"{name} is {age} years old.")
```

Output:

code

```
Alice is 30 years old.
```

Interactive Exercises

1.Write a Python program to print your name and favorite hobby using the print() function.

Modify the separator and end character in the print() function to display:

mathematic

Copy code

Hello-World-End

Use variables and the print() function to display:

Copy code

My name is [your name], and I am learning Python.