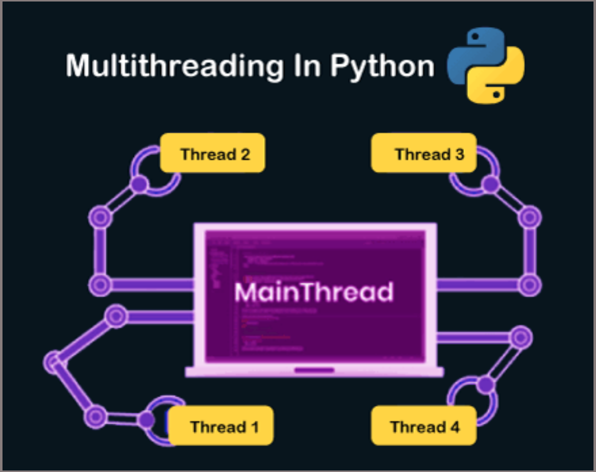
Multithreading in Python

KUMARESAN.R
BHARATH.J.B
PRAGADEESHWARAN.S
VIGNESH.K
SANGEETHARASU.A
YUVARAJ.S
VIGNESHWARAN.G

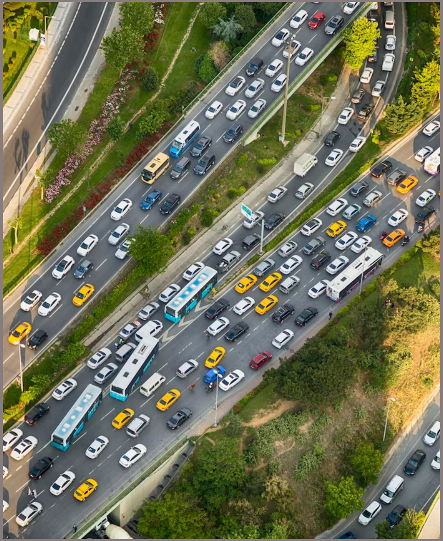


Introduction

Multithreading is a powerful technique for optimizing **Python** programs. In this presentation, we'll explore different **strategies** and **best practices** for unlocking the full **power** of multithreading in Python.

GIL and Multithreading

Global Interpreter Lock (GIL) in Python allows only one thread to execute at a time. This **limits** the **performance** of multithreading in Python. However, we can still use **multithreading** to improve the performance of **I/O-bound tasks** such as network requests or file I/O.

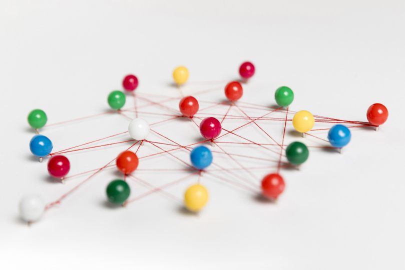


Threading vs Multiprocessing

Threading and **multiprocessing** are two techniques for achieving **concurrency** in Python. **Threading** is suitable for **I/O-bound tasks**, while **multiprocessing** is suitable for **CPU-bound tasks**. However, **multiprocessing** consumes more **system resources** than **threading**.

Thread Synchronization

When multiple threads share **resources**, it's important to ensure that they don't **interfere** with each other. **Thread synchronization** is the process of coordinating the execution of multiple threads to ensure **correctness**. We'll explore different **synchronization techniques** such as **locks**, **semaphores**, and **condition variables**.



Conclusion

Multithreading is a powerful tool for improving the **performance** of Python programs. However, it's important to use the **right technique** for the **right task**. By following **best practices** and using **synchronization techniques**, we can unlock the full **power** of multithreading in Python.

THANK YOU