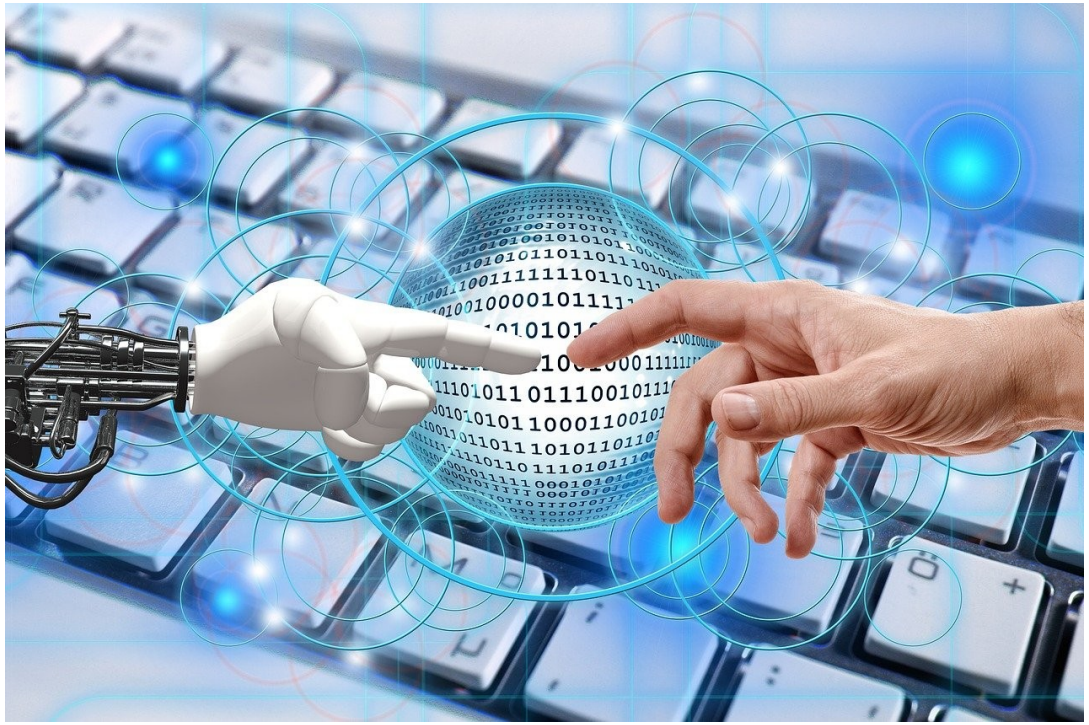
Tutorial 02

Prof. Dr. Teena Hassan
teena.hassan@h-brs.de

Ritwik Sinha
ritwik.sinha@smail.inf.h-brs.de

Department of Computer Science
Hochschule Bonn-Rhein-Sieg
Sankt Augustin

18th April 2024



Content

- *qibullet:* Examples
- Introduction to Multi-threading in Python

qibullet: Examples

Examples: *qibullet*

https://github.com/softbankrobotics-research/qibullet/tree/master/examples

Multi-Threading and Multi-Processing in Python

Introduction

Python offers tools for concurrent execution: multi-threading and multi-processing.

These tools allow us to achieve better performance by leveraging multiple CPU cores or managing concurrent tasks.

Multi-Threading

Multi-threading involves creating multiple threads within a single process.

Threads share the same memory space, making them lightweight and suitable for I/O-bound tasks.

Global Interpreter Lock (GIL) limits true parallelism for CPU-bound tasks, but threads are effective for I/O-bound operations.

Multi-Threading contd.

Example: Running multiple tasks simultaneously, such as downloading files.

Python's threading module enables thread creation and synchronization.

start() launches threads, join() waits for threads to finish, and synchronization primitives like locks prevent data conflicts.

Multi-Processing

Multi-processing involves creating multiple processes, each with its own memory space and Python interpreter.

Processes allow for true parallelism on multi-core systems.

Great for CPU-bound tasks, but communication between processes can be more complex.

Choosing Between Threading and Processing

Use multi-threading for I/O-bound tasks with frequent waiting.

Use multi-processing for CPU-bound tasks that benefit from true parallelism.

Pros and Cons

Multi-Threading:

- Pros: Lightweight, suitable for I/O-bound tasks, efficient for concurrency.
- Cons: GIL limits true parallelism for CPU-bound tasks.

Multi-Processing:

- Pros: Achieves true parallelism, utilizes multi-core systems effectively.
- Cons: Higher memory usage, complex inter-process communication.

Conclusion

Multi-threading and multi-processing are essential for improving performance in concurrent applications.

Choose the right approach based on your task's nature and system architecture.

Python's threading and multiprocessing modules provide powerful tools for managing threads and processes.

Refer to :

- https://docs.python.org/3/library/threading.html
- https://docs.python.org/3/library/multiprocessing.html

Thank you!

For any queries:

- LEA Forum
- email to ritwik.sinha@smail.inf.h-brs.de