


CONCURRENCY IN PYTHON



WHAT IS CONCURRENCY?



Concurrency is a concept where multiple tasks appear to run simultaneously, but not necessarily in parallel. Basically, concurrency means managing access to shared resources, ensuring that data remains consistent, and tasks are executed in an optimized manner. Concurrency also implies faster execution time.

Concurrency in Python can be achieved in various ways like parallelism (using **multiprocessing**), **multithreading** and **asynchronous programming**.

USE CASES OF CONCURRENCY IN REAL WORLD APPLICATIONS

WEB SERVERS AND APPLICATION SERVERS

Used in e-commerce applications and social media sites to handle multiple requests simultaneously.

SCIENTIFIC COMPUTING AND SIMULATIONS

Used to simulate complex phenomena that involve numerous interacting elements.

FINANCIAL TRADING SYSTEMS

High-frequency trading platforms use concurrency to process and analyze vast amounts of market data in real time, execute trades at high speeds, and monitor multiple financial markets simultaneously.

REAL-TIME SYSTEMS

Used in automotive control systems, medical monitoring devices, and industrial automation for critical tasks.

MOBILE APPLICATIONS

Used in apps where they perform network requests, data processing, and UI updates without blocking user interactions.

THREADS AND PROCESSES

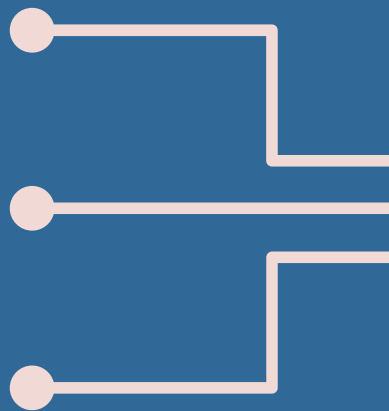


THREADS

A thread is a separate flow of execution, independent of other sub-programs. Multiple threads within the same process can share memory with each other.

PROCESS

A process contains data, files, registers, stacks and code which is executed using threads. Multiple processes can't share memory as they are isolated. A process can have multiple threads.





GLOBAL INTERPRETER LOCK

WHAT IS GIL IN PYTHON?

The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter.

THE PROBLEM WITH GIL

Due to GIL, only one thread can be executed at a time. Other threads are blocked until the locked thread which is running, unlocks the interpreter. This creates problems when multiple tasks are to be executed considering the time complexity.

MULTITHREADING

Multithreading is defined as the ability of a processor to execute multiple threads concurrently. In a simple, single-core CPU, it is achieved using frequent switching between threads. This is termed context switching. In context switching, the state of a thread is saved and the state of another thread is loaded whenever any interrupt (due to I/O or manually set) takes place.





```
import time

def func1():
    time.sleep(2)
    print("Function 1 is running")

def func2():
    time.sleep(3)
    print("Function 2 is running")

start = time.time()
func1()
func2()
time_taken = time.time() - start
print('Time Taken {0}'.format(time_taken))
```



```
Function 1 is running
Function 2 is running
Time Taken 5.004791259765625
```



```
import time
import threading

def func1():
    time.sleep(2)
    print("Function 1 is running")

def func2():
    time.sleep(3)
    print("Function 2 is running")

start = time.time()
x = threading.Thread(target=func1, args=())
y = threading.Thread(target=func2, args=())
x.start()
y.start()
x.join()
y.join()
time_taken = time.time() - start
print('Time Taken {0}'.format(time_taken))
```



```
Function 1 is running
Function 2 is running
Time Taken 3.003720760345459
```


ASYNCHRONOUS PROGRAMMING

Asynchronous programming is a form of concurrency where tasks start and then move on without waiting for the previous task to finish.

It is useful in I/O bound operations (like file reading, making requests) where tasks don't need to wait for other tasks to be completed.

It is achieved using **Asyncio** package and **Aiohttp** package for requests.





SUBROUTINES

A subroutine is a reusable block of code which performs a specific task. It is also called function or method.

COROUTINES

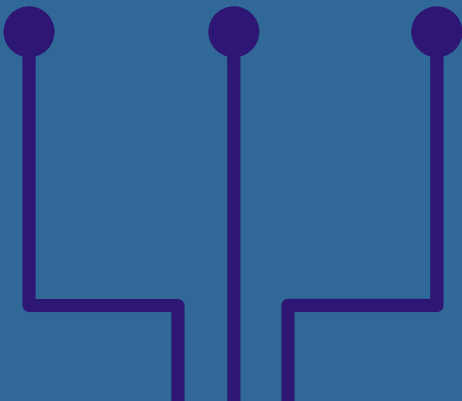
- Used in asynchronous programming
- Give away control when idle in order to enable multiple tasks to run simultaneously.

TASKS

Tasks are a convenient way to manage multiple coroutines running concurrently, allowing us to track their progress and handle errors.

EVENT LOOP

It manages and distributes the execution of multiple tasks. It runs continuously to check and execute tasks or events concurrently.





```
import time

def find_square(n):
    print(f"Calculating the square of {n}\n")
    time.sleep(2)
    number = n*n
    print(f"The square of {n} is {number}\n")

def main():
    for i in range(6):
        find_square(i+1)

start = time.time()
main()
time_taken = time.time() - start
print('Time Taken {0}'.format(time_taken))
```



```
Calculating the square of 1
The square of 1 is 1
Calculating the square of 2
The square of 2 is 4
Calculating the square of 3
The square of 3 is 9
Calculating the square of 4
The square of 4 is 16
Calculating the square of 5
The square of 5 is 25
Calculating the square of 6
The square of 6 is 36
Time Taken 12.009519338607788
```



```
import time
import asyncio

async def find_square(n):
    print(f"Calculating the square of {n}\n")
    await asyncio.sleep(2)
    number = n*n
    print(f"The square of {n} is {number}\n")

async def main():
    tasks = [find_square(i+1) for i in range(6)]
    await asyncio.gather(*tasks)

start = time.time()
asyncio.run(main())
time_taken = time.time() - start
print('Time Taken {0}'.format(time_taken))
```



Calculating the square of 1

Calculating the square of 2

Calculating the square of 3

Calculating the square of 4

Calculating the square of 5

Calculating the square of 6

The square of 1 is 1

The square of 2 is 4

The square of 3 is 9

The square of 4 is 16

The square of 5 is 25

The square of 6 is 36

Time Taken 2.0033071041107178

MULTITHREADING VS. ASYNCIO

MULTITHREADING

- It uses multiple threads
- In threading, we can't determine when to run which code in which thread
- Operating system decides when to run which thread
- So, multithreading sometimes gives random results

ASYNCIO

- It uses single thread
- Asyncio achieves concurrency by cooperative multitasking
- We decide which part of the code can be awaited, which then switches the control to run other parts of the code



CONCURRENCY IN GOLANG VS. PYTHON VS. C++

UNDERSTANDING DIFFERENCES BETWEEN
CONCURRENCY PARADIGMS IN PYTHON,
GOLANG AND C++

- Go has in-built concurrency and no GIL like in Python.
- Go uses Goroutines, Waitgroups and channels to achieve concurrency.
- Go routines are managed by the Go runtime while Python uses OS threads.
- Due to GIL in Python, effectiveness of threading is limited in Python in CPU bound tasks, while Goroutines are effective in both CPU bound as well as I/O bound tasks.
- Goroutines are lightweight and have lower overhead. So large number of goroutines can be created.
- Python being OS threads, have higher overhead costs.

- Also, efficient communication between goroutines is possible using channels.
- Using mutexes, we can achieve partial synchronization in concurrent programs in Go and C++ to prevent data races and make sure that no two goroutines can access a variable at a given time.
- C++ also does not have GIL.
- In C++, the `std::atomic` library provides support for atomic operations. This library is primarily used for ensuring safe access and modification of shared variables in multithreaded programs without the need for explicit locking mechanisms such as mutexes.

REFERENCES

[Concurrency, Multithreading and Asynchronous Programming](#)

[Exploring multithreading, concurrency and parallel execution](#)

[Multithreading – GeeksForGeeks](#)

[Asyncio in Python](#)

[Mutex in Go](#)

[Concurrency in Go](#)

[Concurrency in Python](#)

[Asyncio package in python](#)

[Using aiohttp to download files asynchronously](#)



THANK YOU

