30/oct/2024

Asynchronous Programming

https://youtu.be/Qb9s3UiMSTA

→ multiple tasks can be done simultaniously without the waiting.

→ asyncio

has multiple things in this

① Threads ② Processes

Tasks that wait a lot → Network Requests
Reading files
:
etc

asyncio - does multiple task at a time without using much CPU power.

① Threads → for tasks that needs to wait but also shares the data.
→ Can be seen as parallel within same app making them useful for IO Bound but less CPU intensive

② For CPU Heavy Task → Use Processes
maximizing CPU usage

Key Concepts [decorators]

① The event loop

- each task waiting to be executed
- event loop = core that manages and distributes these tasks

[Python 3-11 >]

② coroutines

① coroutine function

② coroutine object
↳ calling the function

Pass the coroutine object to asyncio.run

```
import asyncio
async def main():
    ...
asyncio.run(main())
```

Steps: ~~define~~ ① import the module
② define some asynchronous function
③ Call the function and pass that asyncio.run

Simply calling the async function without using asyncio.run = RuntimeError coroutine "function_name" was never awaited

here by calling this, we are generating a coroutine ~~function~~ object

→ that coroutine object needs to be awaited inorder for it to actually execute

await → keyword to await a coroutine
can only be used inside
asynchronous function or
inside coroutine.

(Q) How to identify a coroutine function?:-

async def <function_name>:
    Print ("fetching data ---")
    await asyncio.sleep(delay)
    Print ("data fetched")
    return {"data": "Some data"}

before function definition "async" will be written

→ For a coroutine to be executed → it needs to be awaited.

→ When a asynchronous function is called it returns a coroutine that needs to be awaited to actually start executed

③ Tasks

⇒ Way to schedule a coroutine to run as soon as possible and to run multiple coroutines simultaniously

task1 = asyncio.create_task(function_call())

link for youtube current time } See example in youtube video :-

https://youtu.be/Qb9s3UiMSTA?t=787

await asyncio.gather(fn_call1(), fn_call2(), fn.....)

automatically gathers all the coroutine function results in a List

<Link> :-

https://youtu.be/Qb9s3UiMSTA?t=876

Not that great at error handling, if one coroutine has an error then the code doesn't stop

④ Task Group

has inbuit error handling, if one fn has an error, then it automatically cancels all other task

```
async with asyncio.TaskGroup() as tg:
    [ loop ]
```

⑨ Future :-
- Not a code that we write.
- Typically utilised by lower level libraries.
- Promise of a future result.

not the code that is generally used just to understand

```
future = loop.create_future()
loop = asyncio.get_running_loop()
result = await future
```

⑤ Synchronization :- allow us to synchronize the execution of various coroutines

Lock :- Block

```
lock = asyncio.Lock()
```

Here have something and we want to lock it off and only be using it from one coroutine at time
So we can create a lock

② Semaphore : → similar to lock
→ allowing multiple coroutine to have access to same object at same time
But we can decide how many we want that to be

asyclo. Semaphore (n)

n = # of users to allowing coroutine accesses

③ Event :- allow to do simple synchronization act as simple boolean operator that block the code until we set the flag to true in asynchronous way.

Simply a boolean operator
True / False
but in asynchronous way.

④ Condition ——→ Complicated