Python


Introduction to
asyncio in python

asyncio in Python

Asyncio is a Python library that is used for concurrent programming. It is not multi-threading or multi-processing. Asyncio is used as a foundation for multiple Python asynchronous frameworks that provide high-performance network and web servers, database connection libraries, distributed task queues, etc.

Difference between Asynchronous and multi-threading programming

Asynchronous programming basically means that only one part of a program will run at a certain time. For example, suppose we have 3 functions defined in our Python program. Consider a situation when fn1() is not doing anything, it is either asleep or just waiting or has returned a value (done its work). So, to save the CPU time, the other function (fn2()) will start executing and then only the third function (fn3()) will execute. This is the concept of Asynchronous programming (one thing is done at one time) **whereas**, in **multi-threading or multi-processing**, all these three functions will run at the same time, they won't wait for the previous function to finish or sleep. Note that only specific functions are made asynchronous, not the whole program this is done with the help of the Asyncio Python library.

Example of Asyncio in Python

In the example below, we'll create a function and make it asynchronous. To achieve this, an **async** keyword is used. The program will wait for 1 second after it the first print statement is executed and then print the next print statement and so on. Note that we'll make it sleep (or wait) with the help of await **asyncio.sleep(1)** keyword, not with **time.sleep().** To run the program, we'll have to use the **run()** function as it is given below.

```python
import asyncio

async def fn():
    print('This is ')
    await asyncio.sleep(1)
    print('asynchronous programming')
    await asyncio.sleep(1)
    print('and not multi-threading')

asyncio.run(fn())
```

Output:

```
This is
asynchronous programming
and not multi-threading
```

Example 2:

In the program below, we're using **await fn2()** after the first print statement. It simply means to wait until the other function is done executing. So, first, it's gonna print "one", then the control shifts to the second function, and "two" and "three" are printed after which the control shifts back to the first function (because fn() has done its work) and then "four" and "five" are printed.

```python
import asyncio

async def fn():

    print('one')
    await asyncio.sleep(1)
    await fn2()
    print('four')
    await asyncio.sleep(1)
    print('five')
    await asyncio.sleep(1)

async def fn2():
    await asyncio.sleep(1)
    print('two')
    await asyncio.sleep(1)
    print('three')
asyncio.run(fn())
```

Output:

```
one
two
three
four
five
```

Example 3:

Now if you want the program to be actually asynchronous, In the actual order of execution we'll need to make tasks in order to accomplish this. This means that the other function will begin to run anytime if there is any free time using **asyncio.create_task(fn2())**

```python
import asyncio
async def fn():
    task=asyncio.create_task(fn2())
    print("one")
    #await asyncio.sleep(1)
    #await fn2()
    print('four')
    await asyncio.sleep(1)
    print('five')
    await asyncio.sleep(1)

async def fn2():
    #await asyncio.sleep(1)
    print("two")
    await asyncio.sleep(1)
    print("three")

asyncio.run(fn())
```

```python
import asyncio
async def fn():

    task=asyncio.create_task(fn2())
    print("one")
    print('four')
    await asyncio.sleep(1)
    print('five')
    await asyncio.sleep(1)

async def fn2():

    print("two")
    await asyncio.sleep(1)
    print("three")

asyncio.run(fn())
```

1

2

4

3

5

Output: