Python Asyncio Event Loop

Example Code: (Python Docs)

```
import asyncio

async def factorial(name, number):
    f = 1
    for i in range(2, number + 1):
        print(f"Task {name}: Compute factorial({i})...")
        await asyncio.sleep(1)
        f *= i
    print(f"Task {name}: factorial({number}) = {f}")

async def main():
    await asyncio.gather(
        factorial("A", 2),
        factorial("B", 3),
        factorial("C", 4),
    )

loop = asyncio.get_event_loop()
task = loop.create_task(main())

try:
    loop.run_until_complete(task)
finally:
    print("Closing Loop")
    loop.close()
```

- Defining a method with the keyword **async** makes it a coroutine.
- The **await** keyword allows another task to start running.
- The **.gather()** method schedules the three factorial function calls concurrently.
- The **.get_event_loop()** method creates the main event loop.
- The **.create_task()** method creates a task from a coroutine.
- The **.run_until_complete()** method will continue to run the event loop until all tasks are resolved.
- **.close()** must be called to close the event loop.

Output:

```
Task C: Compute factorial(2)...
Task B: Compute factorial(2)...
Task A: Compute factorial(2)...
Task C: Compute factorial(3)...
Task B: Compute factorial(3)...
Task A: factorial(2) = 2
Task C: Compute factorial(4)...
Task B: factorial(3) = 6
Task C: factorial(4) = 24
Closing Loop
```

The event loop runs in a single thread and executes all callbacks and tasks in the same thread.

The example output shows that while waiting for a task to complete, another task can be started before waiting for the current task to finish.

Alysha Iannetta | www.aiannetta.com