“Advanced Python Programming for Everybody”

Instructor: Ernest Bonat, Ph.D.
Senior Software Engineer
Senior Data Scientist
ebonat@15itresources.com
Cell: 503.730.4556

Module 2 “Multithreading and Asynchronous Programming”

GitHub: https://github.com/ebonat/intel_module_2

Multithreaded Programming

(https://www.tutorialspoint.com/python/python_multithreading.htm)

Running several threads is similar to running several different programs concurrently.

Benefits:

- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.

- Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.

A thread has a:

1. Beginning (start)
2. Execution sequence
3. Conclusion (end)

How to start a new Thread:

```
from threading import Thread

thead1 = Thread(target=function, args[,kwargs])

thead1.start()
```

Where:

- args is a tuple of arguments
- kwargs is an optional dictionary of keyword arguments

Exercise 1

Write a multithreading program follow the tasks definition in the table below. Follow Instructor's threading_function_call2.py file. Feel free to use your own ideas if you would like to.

Task Number	Sleep Time (seconds)
task_1	3
task_2	15
task_3	8
task_4	20
task_5	10

Multithreading vs Asynchronous Programming

- Multithreading – run on many threads
- Asynchronous – run on a single thread

Python coroutines are all run on a single thread (application main thread), and don't require extra sockets or memory, it would be a lot harder to run out of resources.

Task Planning

1. Sync	Task 1	Task 2	Task 3	Task 4	Task 5
2. Parallel	Task 1				
	Task 2				
	Task 3				
	Task 4				
	Task 5				
3. Async	Task 1	Task 2	Task 3		
			Task 4		
	Task 5				

Table 1. Sync vs. Parallel vs. Async

asyncio/await Python Code

```
# in python 3.4 (it works in 3.5)
@asyncio.coroutine
def py34_coroutine():
    yield from do_stuff()
```

```
# in python 3.5 and above
async def py35_coroutine():
    await do_stuff()
```

Main running asyncio APIs

```
# create even loop object
ioloop = asyncio.get_event_loop()

# set list of task to run
tasks = [ioloop.create_task(Task1), ioloop.create_task(Task2)),

# create the wait talk object
wait_tasks = asyncio.wait(tasks)

# run all the talks until all complete
ioloop.run_until_complete(wait_tasks)
# release from memory the event loop object

ioloop.close()
```

Practical Example

Calculate Descriptive Statistics using Python Asynchronous programming (summary_statistics_asyncio.py)

Number of Observation		
	Mean	
	Median	
		Standard Deviation