Asynchronous Processing in Python

1. Getting started with asynchrony in Python
2. Managing coroutines via tasks
3. Additional task techniques

Getting started with asynchrony in Python

- Overview
- Asynchrony in Python
- Coroutines
- The asyncio module
- Simple example of asynchrony

Overview

What is asynchrony?

- The ability to perform multiple tasks concurrently

Scenarios where asynchrony is important:

- Processing a large dataset in parallel
- Handling multiple network connections simultaneously
- Performing algorithmic processing in the background
- Etc.

Asynchrony in Python

You can schedule concurrent tasks on a single thread

- The event loop manages task execution on a thread

The event loop can optimize I/O

- If a function is waiting on I/O
- The event loop pauses the function and runs another one instead
- When the first function completes I/O, it is resumed

The event loop can also optimize CPU-intensive functions

- The functions must explicitly "yield", so as not to hog the thread

Note: Python also supports genuine multithreading

- See multipleThreadsAndFutures.py

Coroutines

A coroutine is a special kind of generator function

- It can cede control during its processing (e.g. for I/O)
- The event loop then tries to give another coroutine some time
- The event loop can resume the original coroutine when it's ready

The preferred way to define a coroutine in modern Python is to prefix a function with the async keyword

```
1   async def someFunc(someArgs) :
2       # Some long-running code that might yield control
3       #   e.g. code that does slow I/O
4       #   e.g. code that CPU-intensive processing
```

The asyncio module

The asyncio module provides various methods that allow you to schedule and manage asynchrony

- Some of the common methods are listed here

asyncio.sleep(seconds)

- Sleep for a specified delay (in seconds)

asyncio.run(aCoroutine)

- Creates a new event loop, and runs the coroutine

asyncio.create_task()

- Schedule a coroutine to be executed "soon" on the event loop

Simple example of asynchrony

```
1   import asyncio
2   from time import strftime, localtime
3
4   async def displayAfter(msg, delay) :
5       await asyncio.sleep(delay)
6       now = strftime("%H:%M:%S", localtime())
7       print("%s %s" % (now, msg))
8
9   def main():
10      print("*****Start of main*****")
11      asyncio.run(displayAfter("Hei", 3))
12      asyncio.run(displayAfter("Bye", 5))
13      print("*****End of main*****")
14
15  if __name__ == "__main__" :
16      main()
```

- asyncio.sleep() is a coroutine
- The await keyword yields control back to the event loop, which tries to schedule other coroutines in the meantime
- You can only use the await keyword in coroutines, i.e. functions marked as async
- You can’t just 'invoke' coroutines, you must schedule via asyncio

Managing coroutines via tasks

- Overview
- Simple example of creating a task
- Creating and awaiting multiple tasks
- Awaiting multiple tasks to complete

Overview

The asyncio.create_task() function creates a task

- The task is scheduled for execution "soon" on the event loop
- The task is represented by a Task object

The Task class has methods that allow you to manage the running of the task, such as:

- done() - has the task completed yet?
- cancel() - stop the task now
- result() - get the result of the task (it must have finished!)

Simple example of creating a task

```
from time import strftime, localtime
import asyncio

def doDisplay(msg):
    now = strftime("%H:%M:%S", localtime())
    print("%s %s" % (now, msg))

async def displayAfter(msg, delay) :
    doDisplay("START: " + msg)
    await asyncio.sleep(delay)
    doDisplay("END: " + msg)

async def main():
    print("*****Start of main*****")
    task = asyncio.create_task(displayAfter("Hello", 10))

    for i in range(0,5) :
        print("Doing something useful ... ")
        await asyncio.sleep(1)

    print("Finished doing useful work, now I'll wait for task to finish")
    await task
    print("*****End of main*****")

```

Creating and awaiting multiple tasks

You can create multiple tasks

- All the tasks run concurrently
- You can await for each task to complete individually

```
async def main():
    doDisplay("*****Start of main*****")
    task1 = asyncio.create_task(displayAfter("Bonjour"
    task2 = asyncio.create_task(displayAfter("Bore da"
    task3 = asyncio.create_task(displayAfter("Hei hei"

    for i in range(0,5) :
        doDisplay("Doing something useful ... ")
        await asyncio.sleep(1)

    doDisplay("Waiting for task1 to finish")
    await task1

    doDisplay("Waiting for task2 to finish")
    await task2

    doDisplay("Waiting for task3 to finish")
    await task3

    doDisplay("*****End of main*****")
```

Awaiting multiple tasks to complete

The previous example awaited individual tasks to complete

- If you prefer, you can await multiple tasks to complete
- Use asyncio.gather(), which suspends until all tasks are done

```
async def main():
    doDisplay("*****Start of main*****")
    task1 = asyncio.create_task(displayAfter("Bonjour", 10))
    task2 = asyncio.create_task(displayAfter("Bore da", 15))
    task3 = asyncio.create_task(displayAfter("Hei hei", 20))

    for i in range(0,5) :
        doDisplay("Doing something useful ... ")
        await asyncio.sleep(1)

    doDisplay("Waiting multiple tasks to complete")
    await asyncio.gather(task1, task2, task3)

    doDisplay("*****End of main*****")
```

Additional task techniques

- Awaiting the result of a task
- Polling a task to see if it’s done
- Cancelling a task

Awaiting the result of a task (1/2)

A coroutine can return a value

- The calling code would like to retrieve the value when complete

Here's one way for the calling code to do this:

- Create a task, to schedule the coroutine for execution
- Await completion of the task
- The await expression gives the result of the completed coroutine

```
1   async def createStringAfter(msg, delay) :
2       await asyncio.sleep(delay)
3       now = strftime("%H:%M:%S", localtime())
4       return "{0} {1}".format(now, msg)
5
6   async def main():
7       print("*****Start of main*****")
8       task = asyncio.create_task(createStringAfter("Bonjour", 10))
9       result = await task
10      print(result)
11      print("*****End of main*****")
```

Awaiting the result of a task (2/2)

The previous slide created a task, and then awaited its completion separately:

```
async def main():
    print("*****Start of main*****")
    task = asyncio.create_task(createStringAfter("Bonjour", 10))
    result = await task
    print(result)
    print("*****End of main*****")
```

If it's more convenient, you can combine these two statements into a single statement

```
async def main():
    print("*****Start of main*****")
    result = await asyncio.create_task(createStringAfter("Bonjour", 10))
    print(result)
    print("*****End of main*****")
```

Polling a task to see if it’s done

Sometimes you might want to poll a task to see if it’s done

- Call done() on the task, to see if it’s finished
- If it hasn’t finished, do something else for a bit, then check again
- When it really has finished, call result() on the task

```
async def main():
    doDisplay("*****Start of main*****")
    task = asyncio.create_task(createStringAfter("Bonjour", 10))

    while True:
        if task.done():
            result = task.result()
            doDisplay(result)
            break
        else:
            doDisplay("Doing something useful ... ")
            await asyncio.sleep(1)

    doDisplay("*****End of main*****")
```

Cancelling a task

Sometimes you might want to cancel a task mid-flight

- Call cancel() on the task

```
async def main():
    doDisplay("*****Start of main*****")
    task = asyncio.create_task(createStringAfter("Bonjour", 10))

    while True:
        if task.done():
            result = task.result()
            doDisplay(result)
            break
        else:
            cancel = input("Task not complete yet. Do you want to cancel it? ")
            if cancel == "y":
                doDisplay("OK I'll cancel the task and we'll all just move on in life.")
                task.cancel()
                break
            else:
                doDisplay("OK I'll wait another second and do something useful ... ")
                await asyncio.sleep(1)

    doDisplay("*****End of main*****")
```

Any questions?