async and await

The **async** and **await** keywords were added in Python 3.5 to define a **native coroutine** and make them a distinct type when compared with a generator based coroutine. If you'd like an in-depth description of async and await, you will want to check out PEP 492.

In Python 3.4, you would create a coroutine like this:

```
# Python 3.4 coroutine example
import asyncio

@asyncio.coroutine
def my_coro():
    yield from func()
```

This decorator still works in Python 3.5, but the **types** module received an update in the form of a **coroutine** function which will now tell you if what you're interacting with is a native coroutine or not. Starting in Python 3.5, you can use async def to syntactically define a coroutine function. So the function above would end up looking like this:

```
import asyncio

async def my_coro():
    await func()
```

When you define a coroutine in this manner, you cannot use **yield** inside the coroutine function. Instead it must include a **return** or **await** statement that are used for returning values to the caller. Note that the **await** keyword can only be used inside an **async def** function.

The **async** / **await** keywords can be considered an API to be used for asynchronous programming. The asyncio module is just a framework that happens to use **async** / **await** for programming asynchronously. There is

actually a project called **curio** that proves this concept as it is a separate

implementation of an event loop thats uses **async** / **await** underneath the covers.