A Better Coroutine Example

The **aiohttp** package is designed for creating asynchronous HTTP clients and servers. You can install it with pip like this:

```
pip install aiohttp
```

Once that’s installed, let’s update our code to use aiohttp so that we can download the files:

```
import aiohttp
import asyncio
import async_timeout
import os


async def download_coroutine(session, url):
    with async_timeout.timeout(10):
        async with session.get(url) as response:
            filename = os.path.basename(url)
            with open(filename, 'wb') as f_handle:
                while True:
                    chunk = await response.content.read(1024)
                    if not chunk:
                        break
                    f_handle.write(chunk)
            return await response.release()


async def main(loop):
    urls = ["http://www.irs.gov/pub/irs-pdf/f1040.pdf",
        "http://www.irs.gov/pub/irs-pdf/f1040a.pdf",
        "http://www.irs.gov/pub/irs-pdf/f1040ez.pdf",
        "http://www.irs.gov/pub/irs-pdf/f1040es.pdf",
        "http://www.irs.gov/pub/irs-pdf/f1040sb.pdf"]

    async with aiohttp.ClientSession(loop=loop) as session:
        for url in urls:
            print(await download_coroutine(session, url))


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main(loop))
```

You will notice here that we import a couple of new items: **aiohttp** and **async_timeout**. The latter is a actually one of the aiohttp's dependencies and allows us to create a timeout context manager. Let's start at the bottom of the code and work our way up. In the bottom conditional statement, we start our asynchronous event loop and call our main function. In the main function, we create a **ClientSession** object that we pass on to our download coroutine function for each of the urls we want to download. In the download_coroutine, we call our session's **get()** method which gives us a response object. Now we get to the part that is a bit magical. When you use the **content** attribute of the response object, it returns an instance of **aiohttp.StreamReader** which allows us to download the file in chunks of whatever size we'd like. As we read the file, we write it out to local disk. Finally we call the response's **release()** method, which will finish the response processing.

According to aiohttp's documentation, because the response object was created in a context manager, it technically calls release() implicitly. But in Python, explicit is usually better and there is a note in the documentation that we shouldn't rely on the connection just going away, so I believe that it's better to just release it in this case.

There is one part that is still blocking here and that is the portion of the code that actually writes to disk. While we are writing the file, we are still blocking. There is another library called **aiofiles** that we could use to try and make the file writing asynchronous too, but I will leave that update to the reader.