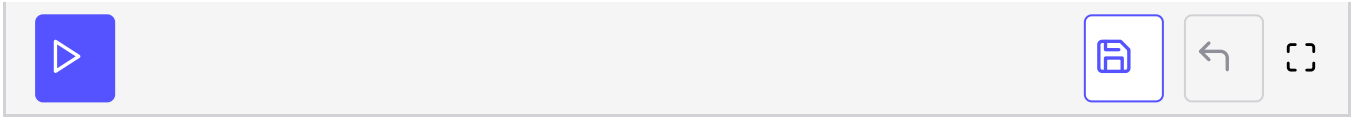
A Bad Coroutine Example

While it is certainly helpful to have a lot of background information into how all this works, sometimes you just want to see some examples so you can get a feel for the syntax and how to put things together. So with that in mind, let's start out with a simple example!

A fairly common task that you will want to complete is downloading a file from some location whether that be an internal resource or a file on the Internet. Usually you will want to download more than one file. So let's create a pair of coroutines that can do that:

```
import asyncio
import os
import urllib.request

async def download_coroutine(url):
    """
    A coroutine to download the specified url
    """
    request = urllib.request.urlopen(url)
    filename = os.path.basename(url)

    with open(filename, 'wb') as file_handle:
        while True:
            chunk = request.read(1024)
            if not chunk:
                break
            file_handle.write(chunk)
    msg = 'Finished downloading {filename}'.format(filename=filename)
    return msg

async def main(urls):
    """
    Creates a group of coroutines and waits for them to finish
    """
    coroutines = [download_coroutine(url) for url in urls]
    completed, pending = await asyncio.wait(coroutines)
    for item in completed:
        print(item.result())


if __name__ == '__main__':
    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"]

    event_loop = asyncio.get_event_loop()
    try:
        event_loop.run_until_complete(main(urls))
    finally:
        event_loop.close()
```



In this code, we import the modules that we need and then create our first coroutine using the **async** syntax. This coroutine is called **download_coroutine** and it uses Python's **urllib** to download whatever URL is passed to it. When it is done, it will return a message that says so.

The other coroutine is our main coroutine. It basically takes a list of one or more URLs and queues them up. We use asyncio's **wait** function to wait for the coroutines to finish. Of course, to actually start the coroutines, they need to be added to the event loop. We do that at the very end where we get an event loop and then call its **run_until_complete** method. You will note that we pass in the **main** coroutine to the event loop. This starts running the main coroutine which queues up the second coroutine and gets it going. This is known as a chained coroutine.

The problem with this example is that it really isn't a coroutine at all. The reason is that the download_coroutine function isn't asynchronous. The problem here is that **urllib** is *not* asynchronous and further, I am not using await or yield from either. A better way to do this would be to use the **aiohttp** package. Let's look at that next!