Scheduling Calls

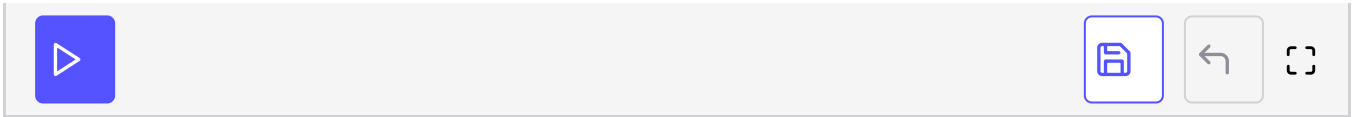
You can also schedule calls to regular functions using the asyncio event loop. The first method we'll look at is **call_soon**. The **call_soon** method will basically call your callback or event handler as soon as it can. It works as a FIFO queue, so if some of the callbacks take a while to run, then the others will be delayed until the previous ones have finished. Let's look at an example:

```
import asyncio
import functools


def event_handler(loop, stop=False):
    print('Event handler called')
    if stop:
        print('stopping the loop')
        loop.stop()


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    try:
        loop.call_soon(functools.partial(event_handler, loop))
        print('starting event loop')
        loop.call_soon(functools.partial(event_handler, loop, stop=True))

        loop.run_forever()
    finally:
        print('closing event loop')
        loop.close()
```



The majority of asyncio's functions do not accept keywords, so we will need the **functools** module if we need to pass keywords to our event handler. Our regular function will print some text out to stdout whenever it is called. If you happen to set its **stop** argument to **True**, it will also stop the event loop.

The first time we call it, we do not stop the loop. The second time we call it, we

do stop the loop. The reason we want to stop the loop is that we've told it to **run_forever**, which will put the event loop into an infinite loop. Once the loop is stopped, we can close it. If you run this code, you should see the following output:

```
starting event loop
Event handler called
Event handler called
stopping the loop
closing event loop
```

There is a related function called **call_soon_threadsafe**. As the name implies, it works the same way as **call_soon**, but it's thread-safe.

If you want to actually delay a call until some time in the future, you can do so using the **call_later** function. In this case, we could change our call_soon signature to the following:

```
loop.call_later(1, event_handler, loop)
```

This will delay calling our event handler for one second, then it will call it and pass the loop in as its first parameter.

If you want to schedule a specific time in the future, then you will need to grab the loop's time rather than the computer's time. You can do so like this:

```
current_time = loop.time()
```

Once you have that, then you can just use the **call_at** function and pass it the time that you want it to call your event handler. So let's say we want to call our event handler five minutes from now. Here's how you might do it:

```
loop.call_at(current_time + 300, event_handler, loop)
```

In this example, we use the current time that we grabbed and append 300 seconds or five minutes to it. By so doing, we delay calling our event handler for five minutes! Pretty neat!