Create a Timing Context Manager

Some programmers like to use context managers to time small pieces of code. So let's create our own timer context manager class!

```
import random
import time

class MyTimer():

    def __init__(self):
        self.start = time.time()

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        end = time.time()
        runtime = end - self.start
        msg = 'The function took {time} seconds to complete'
        print(msg.format(time=runtime))


def long_runner():
    for x in range(5):
        sleep_time = random.choice(range(1,5))
        time.sleep(sleep_time)


if __name__ == '__main__':
    with MyTimer():
        long_runner()
```

In this example, we use the class's **__init__** method to start our timer. The **__enter__** method doesn't need to do anything other then return itself. Lastly, the **__exit__** method has all the juicy bits. Here we grab the end time, calculate the total run time and print it out.

The end of the code actually shows an example of using our context manager where we wrap the function from the previous example in our custom context manager.