Use a decorator

Writing your own timer is a lot of fun too, although it may not be as accurate as just using timeit depending on the use case. Regardless, we're going to write our own custom function timing decorator! Here's the code:

```
import random
import time

def timerfunc(func):
    """
    A timer decorator
    """
    def function_timer(*args, **kwargs):
        """
        A nested function for timing other functions
        """
        start = time.time()
        value = func(*args, **kwargs)
        end = time.time()
        runtime = end - start
        msg = "The runtime for {func} took {time} seconds to complete"
        print(msg.format(func=func.__name__,
                         time=runtime))
        return value
    return function_timer


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

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

For this example, we import the **random** and the **time** modules from Python's standard library. Then we create our decorator function. You will notice that it accepts a function and has another function inside of it. The nested function will grab the time before calling the passed in function. Then it waits for the

function to return and grabs the end time. Now we know how long the

function took to run, so we print it out. Of course, the decorator also needs to return the result of the function call and the function itself, so that's what the last two statements are all about.

The next function is decorated with our timing decorator. You will note that it uses random to "randomly" sleep a few seconds. This is just to demonstrate a long running program. You would actually want to time functions that connect to databases (or run large queries), websites, run threads or do other things that take a while to complete.

Each time you run this code, the result will be slightly different. Give it a try and see for yourself!