A Simple Function Timer

We want to create a simple function that can time how fast a function runs.

We want this function to be generic in the sense that it can be used to time any function (along with it's positional and keyword arguments), as well as specifying the number of the times the function should be timed, and the returns the average of the timings.

We'll call our function **time_it**, and it will need to have the following parameters:

- the function we want to time
- the positional arguments of the function we want to time (if any)
- the keyword-only arguments of the function we want to time (if any)
- the number of times we want to run this function

```
import time

def time_it(fn, *args, rep=5, **kwargs):
    print(args, rep, kwargs)
```

Now we could the function this way:

```
time_it(print, 1, 2, 3, sep='-')
(1, 2, 3) 5 {'sep': '-'}
```

Let's modify our function to actually run the print function with any positional and keyword args (except for rep) passed to it:

```
def time_it(fn, *args, rep=5, **kwargs):
    for i in range(rep):
        fn(*args, **kwargs)

time_it(print, 1, 2, 3, sep='-')
1-2-3
1-2-3
1-2-3
1-2-3
1-2-3
```

As you can see **1, 2, 3** was passed to the **print** function's positional parameters, and the keyword_only arg **sep** was also passed to it.

We can even add more arguments:

```
time_it(print, 1, 2, 3, sep='-', end=' *** ', rep=3)
1-2-3 *** 1-2-3 *** 1-2-3 ***
```

Now all that's really left for us to do is to time the function and return the average time:

```
def time_it(fn, *args, rep=5, **kwargs):
    start = time.perf_counter()
    for i in range(rep):
        fn(*args, **kwargs)
    end = time.perf_counter()
    return (end - start) / rep
```

Let's write a few functions we might want to time:

We'll create three functions that all do the same thing: calculate powers of n**k for k in some range of

integer values

```
def compute_powers_1(n, *, start=1, end):
    # using a for loop
    results = []
    for i in range(start, end):
        results.append(n**i)
    return results

def compute_powers_2(n, *, start=1, end):
    # using a list comprehension
    return [n**i for i in range(start, end)]

def compute_powers_3(n, *, start=1, end):
    # using a generator expression
    return (n**i for i in range(start, end))
```

Let's run these functions and see the results:

```
compute_powers_1(2, end=5)
[2, 4, 8, 16]

compute_powers_2(2, end=5)
[2, 4, 8, 16]

list(compute_powers_3(2, end=5))
[2, 4, 8, 16]
```

Finally let's run these functions through our **time_it** function and see the results:

```
time_it(compute_powers_1, n=2, end=20000, rep=4)
2.5798198230283234

time_it(compute_powers_2, 2, end=20000, rep=4)
2.3151767636341347

time_it(compute_powers_3, 2, end=20000, rep=4)
3.0854032573301993e-06
```

Although the **compute_powers_3** function appears to be **much** faster than the other two, it doesn't quite do the same thing!

We'll cover generators in detail later in this course.