Getting started

What does it mean to benchmark ones code? The main idea behind benchmarking or profiling is to figure out how fast your code executes and where the bottlenecks are. The main reason to do this sort of thing is for optimization. You will run into situations where you need your code to run faster because your business needs have changed. When this happens, you will need to figure out what parts of your code are slowing it down.

This chapter will only cover how to profile your code using a variety of tools. It will not go into actually optimizing your code. Let’s get started!

timeit

Python comes with a module called **timeit**. You can use it to time small code snippets. The timeit module uses platform-specific time functions so that you will get the most accurate timings possible.

The timeit module has a command line interface, but it can also be imported. We will start out by looking at how to use timeit from the command line. Open up a terminal and try the following examples:

```
python -m timeit -s “[ord(x) for x in ‘abcdfghi’]” 100000000 loops, best of
3: 0.0115 usec per loop

python -m timeit -s “[chr(int(x)) for x in ‘123456789’]” 100000000 loops,
best of 3: 0.0119 usec per loop
```

What’s going on here? Well, when you call Python on the command line and pass it the “-m” option, you are telling it to look up a module and use it as the main program. The “-s” tells the timeit module to run setup once. Then it runs the code for n number of loops 3 times and returns the best average of the 3 runs. For these silly examples, you won’t see much difference.

runs. For these silly examples, you won't see much difference.

Your output will likely be slightly different as it is dependent on your computer's specifications.

Let's write a silly function and see if we can time it from the command line:

```python
# simple_func.py
def my_function():
    try:
        1 / 0
    except ZeroDivisionError:
        pass
```

All this function does is cause an error that is promptly ignored. Yes, it's another silly example. To get timeit to run this code on the command line, we will need to import the code into its namespace, so make sure you have changed your current working directory to be in the same folder that this script is in. Then run the following:

```
python -m timeit "import simple_func; simple_func.my_function()"
1000000 loops, best of 3: 1.77 usec per loop
```

Here we import the function and then call it. Note that we separate the import and the function call with semi-colons and that the Python code is in quotes. Now we're ready to learn how to use timeit inside an actual Python script.