Timing your code

Sometimes it's important to know how long your code is taking to run, or at least know if a particular line of code is slowing down your entire project. Python has a built-in timing module to do this.

This module provides a simple way to time small bits of Python code. It has both a Command-Line Interface as well as a callable one. It avoids a number of common traps for measuring execution times.

Let's learn about timeit!

In [1]:

```
import timeit
```

Let's use timeit to time various methods of creating the string '0-1-2-3-.....-99'

We'll pass two arguments: the actual line we want to test encapsulated as a string and the number of times we wish to run it. Here we'll choose 10,000 runs to get some high enough numbers to compare various methods.

In [2]:

```
# For loop
timeit.timeit('"-".join(str(n) for n in range(100))', number=10000)
```

Out[2]:

```
0.21865416520477374
```

In [3]:

```
# List comprehension
timeit.timeit('"-".join([str(n) for n in range(100)])', number=10000)
```

Out[3]:

```
0.19484614421698643
```

In [4]:

```
# Map()
timeit.timeit('"-".join(map(str, range(100)))', number=10000)
```

Out[4]:

```
0.15291817337139246
```

Great! We see a significant time difference by using map()! This is good to know and we should keep this in mind.

Now let's introduce iPython's magic function **%timeit**
NOTE: This method is specific to jupyter notebooks!

iPython's %timeit will perform the same lines of code a certain number of times (loops) and will give you the fastest performance time (best of 3).

Let's repeat the above examinations using iPython magic!

In [5]:

```
%timeit "-".join(str(n) for n in range(100))
```

```
20.4 µs ± 269 ns per loop (mean ± std. dev. of 7 runs, 10000 loops e
ach)
```

In [6]:

```
%timeit "-".join([str(n) for n in range(100)])
```

```
18.1 µs ± 56.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops
each)
```

In [7]:

```
%timeit "-".join(map(str, range(100)))
```

```
14.4 µs ± 64.1 ns per loop (mean ± std. dev. of 7 runs, 100000 loops
each)
```

Great! We arrive at the same conclusion. It's also important to note that iPython will limit the amount of *real time* it will spend on its timeit procedure. For instance if running 100000 loops took 10 minutes, iPython would automatically reduce the number of loops to something more reasonable like 100 or 1000.

Great! You should now feel comfortable timing lines of your code, both in and out of iPython. Check out the documentation for more information: https://docs.python.org/3/library/timeit.html (https://docs.python.org/3/library/timeit.html)