SECTION 14: ADVANCED PYTHON MODULES, 2 hours 23 minutes, 13 sections

10/13 Timing Your Python Code

- **theory**
  - -> we are going to discover multiple solutions for a single task
  - -> we can time the code's performance for this, to find the most efficient approach
  - -> we can track the time elapsed before and after calling the function for this
  - -> we can also use the %%timeit module
  - -> there is a "magic" function for doing this in Jupyter

- **in an ipynb**
  - -> he defines a function
  - -> we are returning the numbers up to that string
  - -> he has defined a second function which does the same thing
  - -> we are mapping the string function to each number

```
[26]: import timeit

n [ ]: timeit.timeit

Signature: timeit.timeit(stmt='pass', setup='pass', timer=<built-in fun
er>, number=1000000, globals=None)
Docstring: Convenience function to create Timer object and call timeit
File:      c:\users\marcial\anaconda3\lib\timeit.py
```

  - -> these are two separate solutions to the same problem
  - -> he imports an entire module -> import timeit, for timing
  - -> then timeit.timeit <- to run the statement code again and again and again
  - -> the code we want to run, the number of times and the setup
    - -> statement and setup are passed in as strings
  - -> import time
  - -> he returns the current time <- time.time()
  - -> then calls the two functions he previously defined
  - -> with timeit, it then runs the code over time and time again
  - -> then running ''''s <- one is run over and over and one is run once
  - -> we have a statement and a setup
  - -> timeit.timeit
  - -> then we run the code
  - -> we are running the code enough times so that we see the a time difference between them
  - -> he repeats this for the second statement
  - -> the second function / statement performs faster
  - **-> this runs the functions a million times and sees how long they take**
  - **-> then we can see in the limit, which of the two functions is more efficient**
  - **-> %%timeit <- this is the first line in the cell**
  - **-> then below this in the cell, the name of the function is called**
  - -> this runs the function for (in this example 100,000 times - and then returns the number of times per microsecond loop)

```
In [30]: setup = '''
         def func_one(n):
             return [str(num) for num in range(n)]
         '''

In [31]: timeit.timeit(stmt,setup,number=100000)

Out[31]: 1.5283261000000001

In [36]: timeit.timeit(stmt2,setup2,number=1000000)

Out[36]: 11.2872208

In [37]: %%timeit
         func_one(100)

         100000 loops, best of 3: 13.7 µs per loop

In [38]: %%timeit
         func_two(100)

         100000 loops, best of 3: 11.2 µs per loop
```