map()

map() is a built-in Python function that takes in two or more arguments: a function and one or more iterables, in the form:

```
map(function, iterable, ...)
```

map() returns an *iterator* - that is, map() returns a special object that yields one result at a time as needed. We will learn more about iterators and generators in a future lecture. For now, since our examples are so small, we will cast map() as a list to see the results immediately.

When we went over list comprehensions we created a small expression to convert Celsius to Fahrenheit. Let's do the same here but use map:

In [1]:
```python
def fahrenheit(celsius):
    return (9/5)*celsius + 32

temps = [0, 22.5, 40, 100]
```

Now let's see map() in action:

In [2]:
```python
F_temps = map(fahrenheit, temps)

#Show
list(F_temps)
```

Out[2]:
```
[32.0, 72.5, 104.0, 212.0]
```

In the example above, map() applies the fahrenheit function to every item in temps. However, we don't have to define our functions beforehand; we can use a lambda expression instead:

In [3]:
```python
list(map(lambda x: (9/5)*x + 32, temps))
```

Out[3]:
```
[32.0, 72.5, 104.0, 212.0]
```

Great! We got the same result! Using map with lambda expressions is much more common since the entire purpose of map() is to save effort on having to create manual for loops.

map() with multiple iterables

map() can accept more than one iterable. The iterables should be the same length - in the event that they are not, map() will stop as soon as the shortest iterable is exhausted.

For instance, if our function is trying to add two values **x** and **y**, we can pass a list of **x** values and another list of **y** values to map(). The function (or lambda) will be fed the 0th index from each list, and then the 1st index, and so on until the n-th index is reached.

Let's see this in action with two and then three lists:

```
In [4]:
a = [1,2,3,4]
b = [5,6,7,8]
c = [9,10,11,12]

list(map(lambda x,y:x+y,a,b))
```

```
Out[4]:
[6, 8, 10, 12]
```

```
In [5]:
# Now all three lists
list(map(lambda x,y,z:x+y+z,a,b,c))
```

```
Out[5]:
[15, 18, 21, 24]
```

We can see in the example above that the parameter **x** gets its values from the list **a**, while **y** gets its values from **b** and **z** from list **c**. Go ahead and play with your own example to make sure you fully understand mapping to more than one iterable.

Great job! You should now have a basic understanding of the map() function.