Higher Order Functions

Below we have a list of Celsius temperatures

temperatures = [12.5, 18.1, 15.6, 17.8, 20.1, 22.6, 18.9]

We want to do the following:

1. Convert all the temperatures into Fahrenheit

To do this you can use the map function
The map function can take a named or lambda function that can convert a floating point Celsius temperature into a Fahrenheit equivalent

For example, you could define a function such as

```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9 / 5) + 32
```

This function could then be used with map to do the conversion.

We could now write:

```
fahrenheit_temps = list(map(celsius_to_fahrenheit, temperatures))
print(fahrenheit_temps)
```

This would give us:

[54.5, 64.58, 60.08, 64.04, 68.18, 72.68, 66.02]

Alternatively we could have used a lambda function – as shown below:

```
fahrenheit_temps = list(map(lambda temp: (temp * 9 / 5) + 32,
temperatures))
print(fahrenheit_temps)
```

The output is again:

[54.5, 64.58, 60.08, 64.04, 68.18, 72.68, 66.02]

2. Filter out all temperatures below 18.0

This can be done using the filter function
Using a lambda or a named function you can that returns a Boolean, you can select which values will be returned from the filter function.

This time we want a lambda (or named function) which will test the temperature and determine if it is the 18.0 threshold.

We can combine the lambda with the filter function as shown below:

```
above_threshold = list(filter(lambda temp: temp >= 18.0, temperatures))
print(above_threshold)
```

The output from this is:

[18.1, 20.1, 22.6, 18.9]

Now try to select all temperatures below 18.0 – what do you need to change?

3. Convert all the temperatures above 14.0 to Fahrenheit

Combine together the `filter` and the `map` functions to convert only those temperatures above 14.0 to Fahrenheit.

You can do this in two steps or in one step using chaining. Using chaining this would be:

```
results = list(
        map(lambda temp: (temp * 9 / 5) + 32,
            filter(lambda temp: temp > 14.0,
                   temperatures)))
print(results)
```

The results of this are:

[64.58, 60.08, 64.04, 68.18, 72.68, 66.02]

4. Further examples

Here are some examples of using `filter` with a simple list of integers:

```
data = [1, 3, 5, 2, 7, 4, 10]
print('data:', data)

# Filter for even numbers using a lambda function
d1 = list(filter(lambda i: i % 2 == 0, data))
print('d1:', d1)

def is_even(i):
    return i % 2 == 0

# Filter for even numbers using a named function
d2 = list(filter(is_even, data))
print('d2:', d2)
```

The output from this is:

```
Data: [1, 3, 5, 2, 7, 4, 10]
d1: [2, 4, 10]
```

```
d2: [2, 4, 10]
```

One difference between the two examples is that it is more obvious what the role is of the test function in the second example as it is explicitly named (i.e. `is_even()`), that is the function is testing the integer to see whether it is even or not. The in-line lambda function does exactly the same, but it is necessary to understand the test function itself to work out what it is doing.

5. Further Map examples

The following example applies a function that adds one to a number, to a list of integers:

```
data = [1, 3, 5, 2, 7, 4, 10]
print('data:', data)

# Apply the lambda function to each element in the list
# using the map function
d1 = list(map(lambda i: i + 1, data))
print('d1', d1)

def add_one(i):
    return i + 1

# Apply the add_one function to each element in the
# list using the map function
d2 = list(map(add_one, data))
print('d2:', d2)
```

The output of the above example is:

```
data: [1, 3, 5, 2, 7, 4, 10]
d1 [2, 4, 6, 3, 8, 5, 11]
d2: [2, 4, 6, 3, 8, 5, 11]
```

Further Reading

- http://book.pythontips.com/en/latest/map_filter.html Summary of map, filter and reduce.
- https://www.w3schools.com/python/ref_func_map.asp The W3C schools `map()` function tutorial.
- https://www.w3schools.com/python/ref_func_filter.asp The W3 schools `filter()` function tutorial.