



Higher Order Functions

Functions as parameters

- Have you ever wanted to pass an entire function as a parameter
- Python has functions as first-class citizens, so you can do this
- You simply pass the functions by name

Higher-Order Functions

- A higher-order function is a function that takes another function as a parameter
- They are “higher-order” because it’s a function of a function
- Examples
 - Map
 - Reduce
 - Filter
- Lambda works great as a parameter to higher-order functions if you can deal with its limitations

Map

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

- Map applies **function** to each element of **iterable** and creates a list of the results
- You can optionally provide more iterables as parameters to map and it will place tuples in the result list
- Map returns an iterator which can be cast to list

Map Example

Example

```
1 nums = [0, 4, 7, 2, 1, 0 , 9 , 3, 5, 6, 8, 0, 3]
2
3 nums = list(map(lambda x : x % 5, nums))
4
5 print(nums)
6 #[0, 4, 2, 2, 1, 0, 4, 3, 0, 1, 3, 0, 3]
7
```

Map Problem

Goal: given a list of three dimensional points in the form of tuples, create a new list consisting of the distances of each point from the origin

Loop Method:

- $\text{distance}(x, y, z) = \sqrt{x^2 + y^2 + z^2}$
- loop through the list and add results to a new list

Map Problem

Solution

```
1 from math import sqrt
2
3 points = [(2, 1, 3), (5, 7, -3), (2, 4, 0), (9, 6, 8)]
4
5 def distance(point) :
6     x, y, z = point
7     return sqrt(x**2 + y**2 + z**2)
8
9 distances = list(map(distance, points))
```

Filter

`filter(function, iterable)`

- The filter runs through each element of **iterable** (any iterable object such as a `List` or another collection)
- It applies **function** to each element of **iterable**
- If **function** returns `True` for that element then the element is put into a `List`
- This list is returned from filter in versions of python under 3
- In python 3, filter returns an iterator which must be cast to type list with `list()`

Filter Example

Example

```
1 nums = [0, 4, 7, 2, 1, 0 , 9 , 3, 5, 6, 8, 0, 3]
2
3 nums = list(filter(lambda x : x != 0, nums))
4
5 print(nums)          #[4, 7, 2, 1, 9, 3, 5, 6, 8, 3]
6
```

Filter Problem

```
NaN = float("nan")
scores = [[NaN, 12, .5, 78, math.pi],
          [2, 13, .5, .7, math.pi / 2],
          [2, NaN, .5, 78, math.pi],
          [2, 14, .5, 39, 1 - math.pi]]
```

Goal: given a list of lists containing answers to an algebra exam, filter out those that did not submit a response for one of the questions, denoted by `NaN`

Filter Problem

Solution

```
1 NaN = float("nan")
2 scores = [[NaN, 12, .5, 78, pi],[2, 13, .5, .7, pi / 2],
3           [2,NaN, .5, 78, pi],[2, 14, .5, 39, 1 - pi]]
4 #solution 1 - intuitive
5 def has_NaN(answers) :
6     for num in answers :
7         if isnan(float(num)) :
8             return False
9     return True
0 valid = list(filter(has_NaN, scores))
1 print(valid)
2 #Solution 2 - sick python solution
3 valid = list(filter(lambda x : NaN not in x, scores))
4 print(valid)
```

Reduce

```
reduce(function, iterable [, initializer] )
```

- Reduce will apply **function** to each element in **iterable** along with the sum so far and create a cumulative sum of the results
- **function** must take two parameters
- If initializer is provided, initializer will stand as the first argument in the sum
- Unfortunately in python 3 reduce() requires an import statement
 - `from functools import reduce`

Reduce Example

Example

```
1 nums = [1, 2, 3, 4, 5, 6, 7, 8]
2
3 nums = list(reduce(lambda x, y : (x, y), nums))
4
5 Print(nums)           # ((((((1, 2), 3), 4), 5), 6), 7), 8)
6
7
```

Reduce Problem

Goal: given a list of numbers I want to find the average of those numbers in a few lines using `reduce()`

For Loop Method:

- sum up every element of the list
- divide the sum by the length of the list

Reduce Problem

Solution

```
1 nums = [92, 27, 63, 43, 88, 8, 38, 91, 47, 74, 18, 16,  
          29, 21, 60, 27, 62, 59, 86, 56]  
2  
3 sum = reduce(lambda x, y : x + y, nums) / len(nums)  
4
```

MapReduce

A framework for processing huge datasets on certain kinds of distributable problems

Map Step:

- master node takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes.
- worker node may chop its work into yet small pieces and redistribute again

MapReduce

Reduce Step:

- master node then takes the answers to all the sub-problems and combines them in a way to get the output

MapReduce

Problem: Given an email how do you tell if it is spam?

- Count occurrences of certain words. If they occur too frequently the email is spam.

MapReduce

map_reduce.py

```
1 email = ['the', 'this', 'annoy', 'the', 'the', 'annoy']
2
3 def inEmail (x):
4     if (x == "the"):
5         return 1;
6     else:
7         return 0;
8
9 map(inEmail, l)                                #[1, 0, 0, 0, 1, 1, 0]
10 reduce((lambda x, xs: x + xs), map(inEmail, email)) #3
```