Map, Filter and Lambda in Python Lab

Objectives

- Apply and combine the skills covered for map and filter functions
- Learn how to write and use lambda functions for transforming data
- Modify given data using map and lambda functions as an alternative to writing for loops
- Filter given data using filter function to only include the data that meets a given criteria

Introduction

In this lab, we'll put our new knowledge about `map` and `filter` to the test. We'll also introduce `lambda` functions as a convenient tool for transforming data on the fly. As a test case, we'll be working with Yelp data again. Let's get started!

Lambda functions

Recall that `map` applies a given function to every element of an iterable. Previously, you've seen `map` used with a variety of built-in Python functions. As you begin to work with more complicated data, you may need to use a custom function that performs a unique task for which there is no built-in Python function. This is exactly what `lambda` functions are used for.

Say you wanted to add 5 to every element in the list of numbers shown below:

In [1]:
```
# List of numbers
numbers = [1, 3, 8, 9, 11, 20]
print(numbers)
sum(numbers)
```

```
[1, 3, 8, 9, 11, 20]
```

Out[1]:
```
52
```

Unfortunately, you can't use the addition operator as this results in a `TypeError`:

In [2]:
```
list(map(lambda x: x + 5,numbers))
```

Out[2]:
```
[6, 8, 13, 14, 16, 25]
```

If there were a built-in Python function to add 5, you might just use that function with `map` to apply it to all of the numbers in the list. But, sadly, no such function exists. The good news is that `lambda` can be used to define a custom function that adds 5! The syntax for defining a lambda function that adds 5 is shown below:

```
lambda x: x + 5
```

As you might have guessed, `x` here is a variable and the `lambda` function simply adds 5 to it. Now that you understand how to write `lambda` functions, use `map` to apply the `lambda` function above to add 5 to every number in the `numbers` list.

In [3]:
```
list(map(lambda x: x + 5, numbers))
```

Out[3]:
```
[6, 8, 13, 14, 16, 25]
```

Note that you don't always have to use `x` as the variable. You can define the variable with any name you want as long as the syntax is correct!

The cool thing is that `lambda` functions are *customizable*. They are not just limited to numeric applications. You'll see how `lambda` functions can be used to transform text data below with the Yelp data set.

Yelp data

Now that you've been introduced to `lambda`, you can practice using it with `map` and `filter` to handle some real-world data. Let's start with the Yelp restaurants data set. The code below uses `lambda` to create a dictionary consisting of 4 keys: `name`, `price`, `is_closed`, and `review_count`. The `map` function is then used to apply the `lambda` function to every restaurant in the data set.

In [4]:
```
from restaurants import yelp_restaurants # in this line we are simply impor
```

In [5]:
```
restaurants = list(map(lambda restaurant: dict(name=restaurant['name'],
                                               price=restaurant['price'],
                                               is_closed=restaurant['is_closed'
                                               review_count=restaurant['review_
                                              ), yelp_restaurants))
```

We now have a list of restaurants from the Yelp Api. Let's take a look at the list.

In [6]:
```
restaurants
```

Out[6]:
```
[{'name': 'Fork & Fig',
  'price': '$$',
  'is_closed': False,
  'review_count': 610},
 {'name': 'Salt And Board',
  'price': '$$',
  'is_closed': False,
  'review_count': 11},
 {'name': 'Frontier Restaurant',
  'price': '$',
  'is_closed': False,
  'review_count': 1373},
 {'name': 'Nexus Brewery',
  'price': '$$',
  'is_closed': False,
  'review_count': 680},
 {'name': "Devon's Pop Smoke",
  'price': '$$',
  'is_closed': False,
  'review_count': 54},
 {'name': 'Cocina Azul',
  'price': '$$',
  'is_closed': True,
  'review_count': 647},
 {'name': 'Philly Steaks',
  'price': '$$',
  'is_closed': False,
  'review_count': 25},
 {'name': 'Stripes Biscuit',
  'price': '$$',
  'is_closed': True,
  'review_count': 20}]
```

Using map

As you can see, it's a little tricky to see the names of all of the restaurants due to amount of data. Let's create a new list `names` to contain only the names of all the restaurants from the list above. Use the `map` and `lambda` functions, along with your understanding of a dictionary's structure to do so.

In [7]:
```
names = None
names = list(map(lambda restaurant: restaurant['name'],restaurants))
names
#['Fork & Fig',
#  'Salt And Board',
#  'Frontier Restaurant',
#  'Nexus Brewery',
#  "Devon's Pop Smoke",
#  'Cocina Azul',
#  'Philly Steaks',
#  'Stripes Biscuit']
```

Out[7]:
```
['Fork & Fig',
 'Salt And Board',
 'Frontier Restaurant',
 'Nexus Brewery',
 "Devon's Pop Smoke",
 'Cocina Azul',
 'Philly Steaks',
 'Stripes Biscuit']
```

This worked well. Now let's get a sense of how many reviews were written for each of these restaurants. Just like above, create a new list `review_counts` to only contain the values of `review_count` for each restaurant.

In [8]:
```
review_counts = None
review_counts = list(map(lambda restaurant: restaurant['review_count'],rest
review_counts # [610, 11, 1373, 680, 54, 647, 25, 20]
```

Out[8]:
```
[610, 11, 1373, 680, 54, 647, 25, 20]
```

Let's say we want to get a sense of total number of reviews in the whole dataset. We can add up the elements in `review_counts` list, and assign the result to a variable named `total_reviews`.

In [9]:
```
total_reviews = None

def summation(X):
    t = 0
    for i in range(0, len(X)):
        t += X[i]
        print(str(i+1)+" review="+str(X[i])+", total="+str(t))
    return t
    pass

total_reviews = summation(review_counts)
print("*** *** *** *** ***")
print("print sum:"+str(total_reviews))

#total_reviews = 0
#total_reviews = sum(review_counts)
total_reviews # 3420
```

```
1 review=610, total=610
2 review=11, total=621
3 review=1373, total=1994
4 review=680, total=2674
5 review=54, total=2728
6 review=647, total=3375
7 review=25, total=3400
8 review=20, total=3420
*** *** *** *** ***
print sum:3420
```

Out[9]:
```
3420
```

It's a little tricky to work with the price in the format of dollars signs i.e. and based on how expensive the restaurant is.

So write a function called `format_restaurants` that changes each restaurant to have the attribute `'price'` point to the number of dollar signs (i.e. 1 for $and 2 for$). We'll get you started with the function, `format_restaurant`.

In [10]:
```
def format_restaurant(restaurant):
    if type(restaurant['price']) == str:
        restaurant['price'] = len(restaurant['price'])
    return restaurant
```

In [11]:
```
format_restaurant(restaurants[0]) # {'name': 'Fork & Fig', 'price': 2, 'is_
```

Out[11]:
```
{'name': 'Fork & Fig', 'price': 2, 'is_closed': False, 'review_count': 61
0}
```

Now write another function called `map_format_restaurants` using `map`, that uses above function and returns a list of restaurants with each of them formatted with price pointing to the respective number.

In [12]:
```
def map_format_restaurants(restaurants):
    return list(map(format_restaurant,restaurants))
    pass
```

In [13]:
```
map_format_restaurants(restaurants)

#[{'name': 'Fork & Fig', 'price': 2, 'is_closed': False, 'review_count': 61
# {'name': 'Salt And Board',
#  'price': 2,
#  'is_closed': False,
#  'review_count': 11},
# {'name': 'Frontier Restaurant',
#  'price': 1,
#  'is_closed': False,
#  'review_count': 1373},
# {'name': 'Nexus Brewery',
#  'price': 2,
#  'is_closed': False,
#  'review_count': 680},
# {'name': "Devon's Pop Smoke",
#  'price': 2,
#  'is_closed': False,
#  'review_count': 54},
# {'name': 'Cocina Azul', 'price': 2, 'is_closed': True, 'review_count': 64
# {'name': 'Philly Steaks', 'price': 2, 'is_closed': False, 'review_count':
# {'name': 'Stripes Biscuit',
#  'price': 2,
#  'is_closed': True,
#  'review_count': 20}]
```

Out[13]:
```
[{'name': 'Fork & Fig', 'price': 2, 'is_closed': False, 'review_count': 6
10},
 {'name': 'Salt And Board',
  'price': 2,
  'is_closed': False,
  'review_count': 11},
 {'name': 'Frontier Restaurant',
  'price': 1,
  'is_closed': False,
  'review_count': 1373},
 {'name': 'Nexus Brewery',
  'price': 2,
  'is_closed': False,
  'review_count': 680},
 {'name': "Devon's Pop Smoke",
  'price': 2,
  'is_closed': False,
  'review_count': 54},
 {'name': 'Cocina Azul', 'price': 2, 'is_closed': True, 'review_count': 6
```

Filter

Now let's use `filter` to search for restaurants based on specific criteria.

Write a function called `open_restaurants` using `filter` and `lambda` that takes in a list of restaurants and only returns those that are open. You can use the distionary key `is_closed` to make a decision in your code.

In [14]:
```
#openness = list(map(lambda restaurant: restaurant['is_closed'],restaurants
#print(openness)
#openlist = list(map(lambda restaurant: dict(name=restaurant['name'],
#                                            is_closed=restaurant['is_close
#                                            restaurants))
#print(openlist)

#def open_restaurants(restaurant):
#    open_restaurants = []
#    for i in restaurant:
#        if (i['is_closed'] == False):
#            open_restaurants.append(i)

#    return open_restaurants
#    pass

#def open_restaurants(restaurant):
#    open_restaurants = []
#    for i in range(0,len(restaurant)):
#        if (restaurant[i]['is_closed'] == False):
#            open_restaurants.append(restaurant[i])

#    return open_restaurants
#    pass

#open_restaurants(restaurants)

#filter(open_restaurants,restaurants)

open_restaurants = list(filter(lambda restaurant: restaurant['is_closed'] =
print(open_restaurants)
```

```
[{'name': 'Fork & Fig', 'price': 2, 'is_closed': False, 'review_count': 6
10}, {'name': 'Salt And Board', 'price': 2, 'is_closed': False, 'review_c
ount': 11}, {'name': 'Frontier Restaurant', 'price': 1, 'is_closed': Fals
e, 'review_count': 1373}, {'name': 'Nexus Brewery', 'price': 2, 'is_close
d': False, 'review_count': 680}, {'name': "Devon's Pop Smoke", 'price':
2, 'is_closed': False, 'review_count': 54}, {'name': 'Philly Steaks', 'pr
ice': 2, 'is_closed': False, 'review_count': 25}]
```

In [15]:
```
def open_restaurants(restaurant):
    open_restaurants = []
    for i in range(0,len(restaurant)):
        if (restaurant[i]['is_closed'] == False):
            open_restaurants.append(restaurant[i])

    return open_restaurants
    pass
open_restaurants(restaurants)

#[{'name': 'Fork & Fig', 'price': 2, 'is_closed': False, 'review_count': 61
# {'name': 'Salt And Board',
#  'price': 2,
#  'is_closed': False,
#  'review_count': 11},
# {'name': 'Frontier Restaurant',
#  'price': 1,
#  'is_closed': False,
#  'review_count': 1373},
# {'name': 'Nexus Brewery',
#  'price': 2,
#  'is_closed': False,
#  'review_count': 680},
# {'name': "Devon's Pop Smoke",
#  'price': 2,
#  'is_closed': False,
#  'review_count': 54},
# {'name': 'Philly Steaks', 'price': 2, 'is_closed': False, 'review_count':
```

Out[15]:
```
[{'name': 'Fork & Fig', 'price': 2, 'is_closed': False, 'review_count': 6
10},
 {'name': 'Salt And Board',
  'price': 2,
  'is_closed': False,
  'review_count': 11},
 {'name': 'Frontier Restaurant',
  'price': 1,
  'is_closed': False,
  'review_count': 1373},
 {'name': 'Nexus Brewery',
  'price': 2,
  'is_closed': False,
  'review_count': 680},
 {'name': "Devon's Pop Smoke",
  'price': 2,
  'is_closed': False,
  'review_count': 54},
 {'name': 'Philly Steaks', 'price': 2, 'is_closed': False, 'review_coun
t': 25}]
```

Let's say we now want to look at restaurants that are comparatively cheaper i.e. $ or 1 as price.

Write a function called `cheap_restaurants` using filter, that returns the restaurants that have a price of 1, or '$'.

In [16]:
```
def cheapest_restaurants(restaurants):
    return list(filter(lambda restaurant: restaurant['price']==1 or str(res
    pass

cheapest_restaurants(restaurants)
```

Out[16]:
```
[{'name': 'Frontier Restaurant',
  'price': 1,
  'is_closed': False,
  'review_count': 1373}]
```

In [17]:
```
cheapest_restaurants(restaurants)

# [{'name': 'Frontier Restaurant',
#   'price': 1,
#   'is_closed': False,
#   'review_count': 1373}]
```

Out[17]:
```
[{'name': 'Frontier Restaurant',
  'price': 1,
  'is_closed': False,
  'review_count': 1373}]
```

So we have only one restaurant in the data that meets the given criteria. Next, we shall write a function that filters out only those restaurants that 100 reviews or more, since we want to make sure there is some solid data points backing the reviews -- we are burgeoning data scientists after all!

In [18]:
```
def sufficiently_reviewed_restaurants(restaurants):
    return list(filter(lambda restaurant: restaurant['review_count']>=100,r
    pass
```

In [19]:
```
sufficiently_reviewed_restaurants(restaurants)

# [{'name': 'Fork & Fig', 'price': 2, 'is_closed': False, 'review_count': 6
# {'name': 'Frontier Restaurant',
#  'price': 1,
#  'is_closed': False,
#  'review_count': 1373},
# {'name': 'Nexus Brewery',
#  'price': 2,
#  'is_closed': False,
#  'review_count': 680},
# {'name': 'Cocina Azul', 'price': 2, 'is_closed': True, 'review_count': 64
```

Out[19]:
```
[{'name': 'Fork & Fig', 'price': 2, 'is_closed': False, 'review_count': 6
10},
 {'name': 'Frontier Restaurant',
  'price': 1,
  'is_closed': False,
  'review_count': 1373},
 {'name': 'Nexus Brewery',
  'price': 2,
  'is_closed': False,
  'review_count': 680},
 {'name': 'Cocina Azul', 'price': 2, 'is_closed': True, 'review_count': 6
47}]
```

Summary

Neat! In this lab, we successfully proved our prowess when it comes to iterating over each element of a list with both `map` and `filter` ! We also learned about `lambda` functions and how to use them. We used `map` to format our data into ways that better help us answer questions and extrapolate insights. We used `filter` to return subsets of our data like our restaurants that were only one $ or our restaurants that had 100 or more reviews.