Looping Over Collections - Lab

Introduction

In this lab, we will be practicing what we know about `for` loops. We will use them to reduce the amount of code we write by hand to iterate through collections. We will use data from the excel file, `cities.xlsx`, that has data on different cities, their populations, and their areas. Finally, we will use this information to plot and compare each city. Let's get started!

Objectives

You will be able to:

- Use a `for` loop to iterate over a collection

Identifying When To Use a For Loop

In the last lesson, we worked with some of our travel data. Additional data has been compiled in the `cities.xlsx` excel spreadsheet. Let's retrieve this data from excel using the Pandas library. Don't worry if Pandas feels unfamiliar, it will be covered in detail later. For now, just follow the provided code and get a feel for what is happening. First, read the information from the excel file as a list of dictionaries, with each dictionary representing a location. Then, assign this list to the variable `cities` .

```
import pandas as pd
file_name = './cities.xlsx'
travel_df = pd.read_excel(file_name)
cities = travel_df.to_dict('records')
```

Next, retrieve the first three city names, stored as the `'City'` attribute of each dictionary, and `'Population'` of each of the cities. Then plot the names as our `x_values` and the populations as our `y_values` using the `matplotlib` library. Again, don't worry about understanding all of the details behind what `matplotlib` is doing. It will be covered in more detail soon.

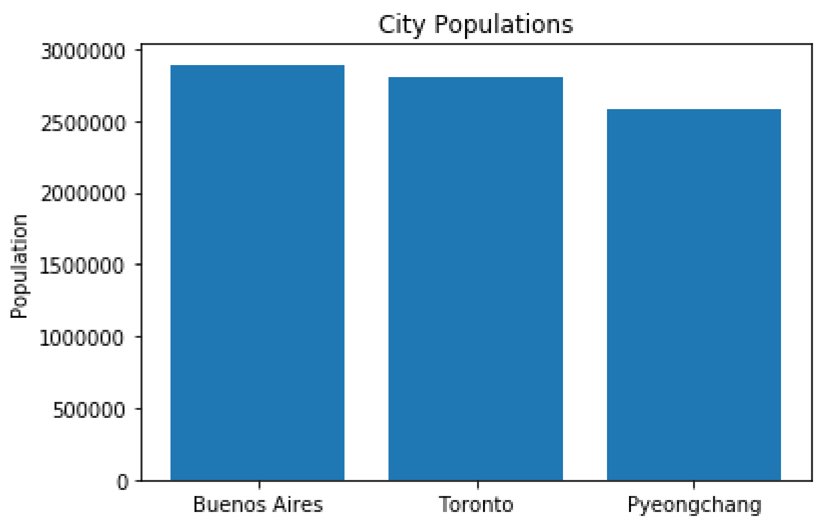
```
import matplotlib.pyplot as plt

%matplotlib inline

x_values = [cities[0]['City'], cities[1]['City'], cities[2]['City']]
y_values = [cities[0]['Population'], cities[1]['Population'], cities[2]['Population'

plt.bar(x_values, y_values)
plt.ylabel('Population')
plt.title('City Populations')

plt.show()
```



Of course, as you may have spotted, there is a good amount of repetition in displaying this data. Just take a look at how we retrieved the data for our `x_values` and `y_values` . And you'll notice that, unless we know the exact number of cities and populations in our excel file, this method of retrieving data might miss some data or try to access values that don't exist.

We can take a close look at this below:

```
x_values = [cities[0]['City'], cities[1]['City'], cities[2]['City']]
y_values = [cities[0]['Population'], cities[1]['Population'], cities[2]['Population'
```

As we can see, if we have any more than 3 lines of data, our `x_values` and `y_values` will be incomplete, and if we had only 2 lines of data, our code would break.

So in this lesson, we will use `for` loop to display information about our travel locations with less repetition and more accuracy.

Instructions

Before we get into creating graphs from our cities data, let's get a bit more comfortable with the data we are working with. Let's see if we can iterate through just one element (i.e. a city **dictionary** object) to get the **area**.

```
buenos_aires = cities[0]
buenos_aires
```

```
{'City': 'Buenos Aires',
 'Country': 'Argentina',
 'Population': 2891000,
 'Area': 4758}
```

```
# here we want to find just the area of buenos_aires
buenos_aires_area = None
# code goes here
for key, value in buenos_aires.items():
    if key == "Area":
        buenos_aires_area = value
buenos_aires_area
```

```
4758
```

Now that we have a bit more familiarity with our dictionaries, we can move to gathering all the information we need to create our traces.

Our `cities` list contains information about the top 12 cities. For our upcoming iteration tasks, it will be useful to have a list of the numbers 0 through 11. Use what we know about `len` and `range` to generate a list of numbers 0 through 11. Assign this to a variable called `city_indices`.

```
city_indices = list(range(0,12))
city_indices # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Now, using the `cities` list, we want to create a list of the names for each city. Loop through each city and append it's name (`'City'`) to the `city_names` list.

```
city_names = []
for city in cities:
    city_names.append(city['City'])
city_names
```

```
['Buenos Aires',
 'Toronto',
 'Pyeongchang',
 'Marakesh',
 'Albuquerque',
 'Los Cabos',
 'Greenville',
 'Archipelago Sea',
 'Walla Walla Valley',
 'Salina Island',
 'Solta',
 'Iguazu Falls']
```

Your task is to assign the variable `names_and_ranks` to a list, with each element equal to the city name and its corresponding rank. For example, the first element would be, `"1. Buenos Aires"` and the second would be `"2. Toronto"`. Luckily for us, the list of cities that we read from our excel file is already in order by most populous to least. So, all we need to do is add numbers 1 through 12 to the beginning of each city name.

Use a `for` loop and the lists `city_indices` and `city_names` to accomplish this. We'll need to perform some nifty string interpolation to format our strings properly. Check out f-string interpolation to see how we can pass values into a string. Remember that list indices start at zero, but we want our `names_and_ranks` list to start at one!

```
names_and_ranks = []
for i in city_indices:
    names_and_ranks.append(f"{i+1}. {city_names[i]}")
names_and_ranks
# write a for loop that adds the properly formatted string to the names_and_ranks li
```

```
['1. Buenos Aires',
 '2. Toronto',
 '3. Pyeongchang',
 '4. Marakesh',
 '5. Albuquerque',
 '6. Los Cabos',
 '7. Greenville',
 '8. Archipelago Sea',
 '9. Walla Walla Valley',
 '10. Salina Island',
 '11. Solta',
 '12. Iguazu Falls']
```

```
names_and_ranks[0] # '1. Buenos Aires'
names_and_ranks[1] # '2. Toronto'
```

```
names_and_ranks[-1] # '12. Iguazu Falls'
```

```
'12. Iguazu Falls'
```

Ok, now use another `for` loop to iterate through our list of `cities` and create a new list called `city_populations` that has the population for each city (`Population`).

```
city_populations = []
for city in cities:
    city_populations.append(city['Population'])
city_populations
# use a for loop to iterate through the list of cities with their corresponding popu
```

```
[2891000,
 2800000,
 2581000,
 928850,
 559277,
 287651,
 84554,
 60000,
 32237,
 4000,
 1700,
 0]
```

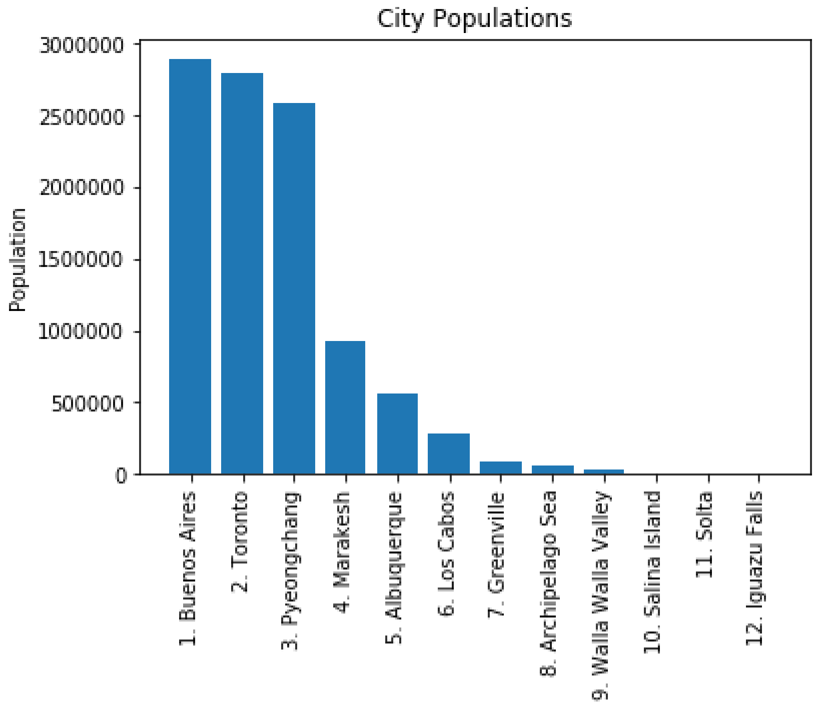
```
city_populations[0] # 2891000
city_populations[1] # 2800000
city_populations[-1] # 0
```

```
0
```

Great! Now we can begin to plot this data. Again, we'll used `matplotlib` to create a bar graph with our cities and their respective population data. To do this, we use the `.bar()` function and pass in our x-axis and y-axis values, add a label and title, and finally we call the `.show()` function to view our new bar graph.

> **Note:** In the example below, we are adding a custom rotation for our x-axis labels so that they do not overlap.

```
plt.bar(names_and_ranks, city_populations)
plt.xticks(rotation='vertical')
plt.ylabel('Population')
plt.title('City Populations')
plt.show()
```



Now we want declare a variable called `city_areas` that points to a list of all of the areas of the cities. Let's use a `for` loop to iterate through our `cities` and have `city_areas` equal to each area of the city.

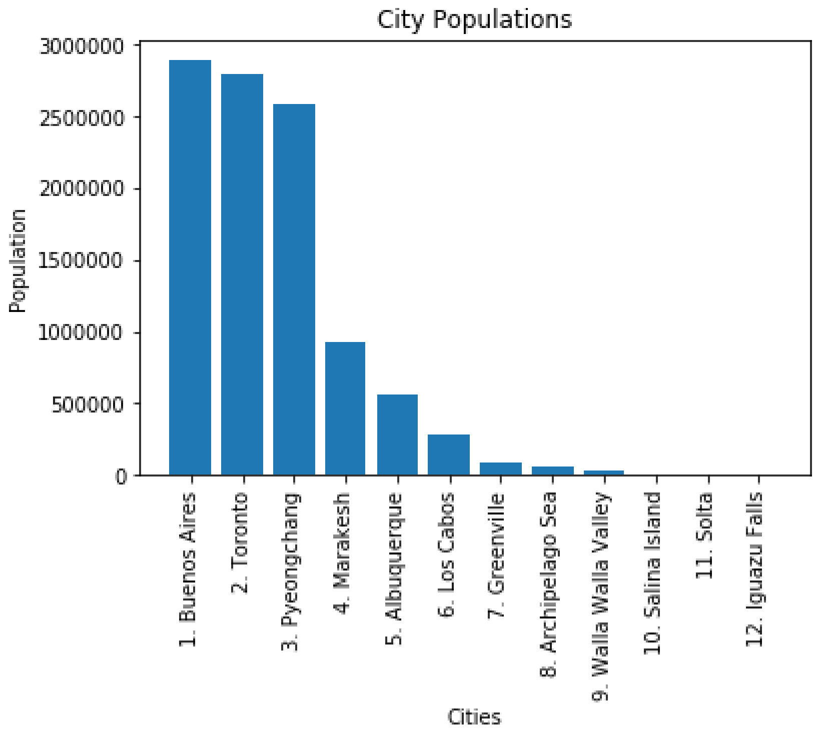
```
city_areas = []
for city in cities:
    city_areas.append(city['Area'])
city_areas
```

```
[4758, 2731, 3194, 200, 491, 3750, 68, 8300, 33, 27, 59, 672]
```

Now that we have the city areas and populations, let's plot them to see how the size of each city compares to its population.

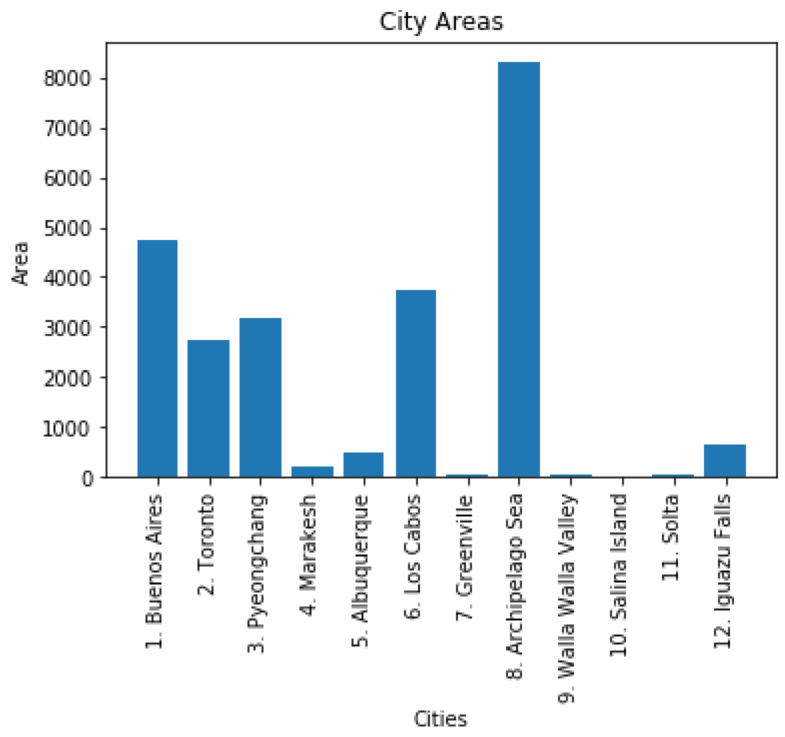
```
plt.bar(names_and_ranks, city_populations)

plt.ylabel('Population')
plt.xlabel('Cities')
plt.title('City Populations')
plt.xticks(rotation='vertical')
plt.show()
```



```
plt.bar(names_and_ranks, city_areas)
plt.ylabel('Area')
plt.xlabel('Cities')
plt.title('City Areas')
plt.xticks(rotation='vertical')

plt.show()
```



Summary

In this section we saw how we can use `for` loops to go through elements of a list and perform the same operation on each. By using `for` loops we were able to reduce the amount of code that we wrote and write more expressive code.

Releases

No releases published

Packages

No packages published

Contributors 6

Languages

● Jupyter Notebook 86.3% ● Python 13.7%

● Jupyter Notebook 86.3% ● Python 13.7%