All About List

List Creation

a = "is" b = "nice" my_list = ["my", "list", a, b]

List of lists

In [ ]:
```
house = [["hallway", hall],
         ["kitchen", kit],
         ["living room", liv],
         ["bedroom", bed],
         ["bathroom", bath]]
```

Subsetting List

In [ ]:
```
x = ["a", "b", "c", "d"]
x[1]
x[-3] # same result!
print(x[1] + x[3])
```

Slicing and Dicing

In [ ]:
```
# my_list[begin:end]
x = ["a", "b", "c", "d"]
x[1:3]
x[:2]
x[2:]
x[:]
```

Subsetting lists of lists

In [ ]:
```
x = [["a", "b", "c"],
     ["d", "e", "f"],
     ["g", "h", "i"]]
#to get 'g'
x[2][0]
#to get ['g','h']
x[2][:2]
```

Replace List Elements

In [ ]:
```
x = ["a", "b", "c", "d"]
x[1] = "r"
x[2:] = ["s", "t"]
```

Extend a List

In [ ]:
```
x = ["a", "b", "c", "d"]
y = x + ["e", "f"]
```

Delete List Elements

In [ ]:
```
x = ["a", "b", "c", "d"]
del(x[1])
```

Inner Working of List

In [ ]:
```
#To prevent changes from original list, use list
areas = [11.25, 18.0, 20.0, 10.75, 9.50]
areas_copy = list(areas)
```

Sorting

Use sorted()

String Methods

In [ ]:
```
#str.upper() to all caps
#str.count() to get the number of times an element appears in a list.
#str.index() to get the index of the first element of a list that matches its input
#str.append(), that adds an element to the list it is called on
#str.remove(), that removes the first element of a list that matches the input
#str.reverse(), that reverses the order of the elements in the list it is called on.
areas = [11.25, 18.0, 20.0, 10.75, 9.50]
print(areas.index(20.0))
print(areas.count(9.50))
areas.append(24.5)
areas.reverse()
```

All about Numpy

In [1]:
```
### Create Numpy Array
import numpy as np
baseball = [180, 215, 210, 210, 188, 176, 209, 200]
np_baseball = np.array(baseball)
```

```
#Boolean Numpy Arrays
high = y > 5
y[high]
```

Subsetting 2D NumPy Arrays

In [ ]:
```
import numpy as np
np_x = np.array(x)
np_x[:,0]
```

In [ ]:
```
### 2D Arithmetic
np_mat = np.array([[1, 2],
                   [3, 4],
                   [5, 6]])
np_mat * 2
np_mat + np.array([10, 10])
np_mat + np_mat
```

In [ ]:
```
# Basic Stat Functions
# np.mean()
# np.median()
# np.std()
# np.corrcoef()
```

In [ ]:
```
# Heights of the goalkeepers: gk_heights
gk_heights = np_heights[np_positions == 'GK']
```

All about Matploblib

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show()
plt.scatter(x,y)
plt.show()
plt.hist()
plt.show()
plt.clf()

#Customization
plt.xlabel()
plt.ylabel()
plt.title()
plt.yticks([0,1,2], ["one","two","three"]) #This replaces 0, 1, 2 in the yaxis with "one", "two", "three"
plt.xticks()
plt.scatter(gdp_cap, life_exp, s = np_pop) #set s argument to np_pop = size is dependent to np_pop
plt.text() #to add words in the plot based from placement
```

Dictionary

In [ ]:
```
my_dict = {
   "key1":"value1",
   "key2":"value2",
}
```

In [ ]:
```
#Add elements to Dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }
europe['iceland'] = 'reykjavik'
#Remove elements
del(europe['australia'])
```

In [ ]:
```
### Dictionariception
europe = { 'spain': { 'capital':'madrid', 'population':46.77 },
           'france': { 'capital':'paris', 'population':66.03 },
           'germany': { 'capital':'berlin', 'population':80.62 },
           'norway': { 'capital':'oslo', 'population':5.084 } }
# Print out the capital of France
print(europe['france']['capital'])
```

Dictionary to DataFrame using Pandas

In [7]:
```
names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']
dr =  [True, False, False, False, True, True, True]
cpc = [809, 731, 588, 18, 200, 70, 45]
import pandas as pd
my_dict = {'country':names, 'drives_right':dr, 'cars_per_cap' : cpc}
cars = pd.DataFrame(my_dict)
```

In [ ]:
```
#Change row labels
row_labels = ['US', 'AUS', 'JPN', 'IN', 'RU', 'MOR', 'EG']
cars.index = row_labels
```

CSV to DataFrame

In [ ]:
```
import pandas as pd
cars = pd.read_csv('cars.csv')
#to specify which column in the CSV file should be used as a row label
cars = pd.read_csv('cars.csv', index_col = 0)
```

Square Brackets

In [ ]:
```
cars['cars_per_cap']
cars[['cars_per_cap']]
#The single bracket version gives a Pandas Series, the double bracket version gives a Pandas DataFrame.
cars[0:5] #selects the first five rows from the cars
cars.loc[['RU', 'AUS']]
cars.iloc[[4, 1]]
cars.loc['MOR', 'drives_right'] #drives_right value of Morocco
# Print sub-DataFrame
cars.loc[['RU','MOR'],['country','drives_right']]
```

Comparison Operators

In [ ]:
```
#Equality
2 == (1 + 1)
"intermediate" != "python"
True != False
#Greater and less than
3 < 4
3 <= 4
"alpha" <= "beta"
#compare arrays
my_house = np.array([18.0, 20.0, 10.75, 9.50])
your_house = np.array([14.0, 24.0, 14.25, 9.0])
print(my_house < your_house)
print(my_house>=18)
```

In [ ]:
```
#Boolean Operators
#and, or, not
print(my_kitchen >10 and my_kitchen <18)
x = 8
y = 9
not(not(x < 3) and not(y > 14 or y > 10))

#Boolean Operators with Numpy
#use np.logical_and/or/not
print(np.logical_or(my_house > 18.5 , my_house < 10))
```

If,elif,else

In [ ]:
```
area = 10.0
if(area < 9) :
    print("small")
elif(area < 12) :
    print("medium")
else :
    print("large")
```

Filtering Pandas DataFrame

In [ ]:
```
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Extract drives_right column as Series: dr
dr = cars['drives_right']
# Use dr to subset cars: sel
sel = cars[dr]
#applying operators
cpc = cars['cars_per_cap']
between = np.logical_and(cpc>100, cpc < 500)
medium = cars[between]
```

In [ ]:
```
## While Loop with conditionals
offset = -6
while offset != 0 :
    print("correcting...")
    if offset > 0 :
      offset = offset -1
    else :
      offset = offset + 1
    print(offset)
```

In [ ]:
```
##For Loop
#with indexes and values
for index, a in enumerate(areas) :
    print("room " +str(index) +": " + str(a))

#Loop over list of lists
house = [["hallway", 11.25],
         ["kitchen", 18.0],
         ["living room", 20.0],
         ["bedroom", 10.75],
         ["bathroom", 9.50]]
for x in house :
    print( "the " + str(x[0]) + " is " + str(x[1]) + " sqm" )

#Loop over dictionary ## you need the items() method to loop over a dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin',
          'norway':'oslo', 'italy':'rome', 'poland':'warsaw', 'austria':'vienna' }
for x, y in europe.items() :
    print ("the capital of " + x + " is " + y)

#Loop over NumPyArray
#1D array
for x in my_array
```

```
#2D array
for x in np.nditer(my_array) :
```

In [ ]:

```
#Loop over DataFrame
#Iterating over a Pandas DataFrame is typically done with the iterrows()
for lab, row in cars.iterrows() :
    print (lab)
    print (row)
#The row data that's generated by iterrows() on every run is a Pandas Series
for lab, row in cars.iterrows() :
    print(lab + ": " + str(row['cars_per_cap']))
#Add Column
for x,row in cars.iterrows() :
    cars.loc[x, 'COUNTRY'] = str.upper(row['country'])
#If you want to add a column to a DataFrame by calling a function on another column, use apply()
cars['COUNTRY'] = cars['country'].apply(str.upper)
```

In [ ]:

```
#Random Float
#seed() sets the random seed, so that your results are reproducible between simulations
   #As an argument, it takes an integer of your choosing. If you call the function, no output will be generated.
#rand() if you don't specify any arguments, it generates a random float between zero and one
#randint() to generate integers randomly
import numpy as np
np.random.randint(4, 8) #generates the integer 4, 5, 6 or 7 randomly. 8 is not included
```

In [ ]:

```
max() #If you pass max() two arguments, the biggest one gets returned
```

In [ ]:

```

```

In [ ]:

```

```