24787 Recitation1 : Installation, basics of Python, Numpy and Matplotlib

Installation

We will be using Jupyter notebooks for this course. Before diving into numpy, vectorization and python basics please make sure that you have python3 installed in your computer. For python installation we would prefer the anaconda distribution.

visit the webpage: https://www.anaconda.com/distribution and follow the instructions. Make sure you install python3 since python 2 will no longer be supported starting Jan 2020

Python basics

Writing a comment

In [1]:
```
# This is how you comment things out. Type '#' followed by your text
```

In [2]:
```
# Here is another example of a comment
```

In [3]:
```
# Now lets use comments
```

In [4]:
```
print(1 + 2) # Addition
print(1 - 2) # Subtraction
print(1 * 2) # Multiplication
print(1 / 2) # Division
print(10 % 3) # return the remainder
print(10 // 3) # return the quotient
print(2 ** 3) # number raised to the power
```

```
3
-1
2
0.5
1
3
8
```

Type and type conversion

In [5]:
```
print(type(9))
print(type(9.0))
print(type(1+3.33))
```

```
<class 'int'>
<class 'float'>
<class 'float'>
```

In [6]:
```
decimal = 12.456435
print(type(decimal))
decimal = int(decimal)
print(decimal)
print(type(decimal))
```

```
<class 'float'>
12
<class 'int'>
```

Variables

- Varialbes are assigned with the variable name on the left, and the value on the right of the equals sign.
- Variables must be assigned before they can be used.
- Can be reassigned any time
- Variable name must start with a letter or underscore, cannot start with a number. Case sensitive.
- Wrong examples: 2cats, hey@you

In [7]:
```
num_1 = 10
num_2 = 20
print(num_1 + num_2)
print(num_1 * num_2)
num_2 = 30
print(num_2)
Num_2 = 40
print(num_2, Num_2)
```

```
30
200
30
30 40
```

Data Types

- bool : True or False
- int: an integer
- str: a sequence of Unicode characters, like "friday"
- list: an ordered sequence of values of other data types
- dict: a collection of key & values

In [8]:
```
print(1 != 2) # not equal
print(3 != 3)
print(4 == 4) # equal to
```

```
True
False
True
```

- "==" operator compares the values and checks for value equality.

In [9]:
```
day = 'Friday' # string
is_it_friday = True # bool
```

In [10]:
```
'Friday' == day
```

Out[10]:
```
True
```

Length

In [11]:
```
print(len(day)) # There are 6 characters in the word 'Friday'
```

```
6
```

In [12]:
```
print(len('day'))
```

```
3
```

Length of a sentence

In [13]:
```
print(len('Ah, Welcome class!')) # space, symbols, numbers are counted as charac
```

```
18
```

Strings

- can be declared with either single or double quotes

In [14]:
```
txt = "I need more and more holidays"
# count method
x = txt.count("holidays") # returns a number equal to the number of occurrences
y = txt.count("more")
print(x)
print(y)
# String concatenation
str_one = "city"
str_two = "PIT"
str_all = str_one + " : " + str_two
print(str_all)
print(len(str_all))
print(str_all[2])

# Operations on string
str_all_sorted = sorted(str_all)
print(str_all_sorted)
# Here is an example of sorting in the reverse direction
str_all_sorted_reverse = sorted(str_all, reverse = True)
print(str_all_sorted_reverse)

# Formatting
course_number = 24787
formatted = f"AI and ML for Engineers: {course_number} ! " # Python 3.6+
formatted2 = "AI and ML for Engineers: {} !".format(course_number) # Python 2 ->
```

```
print(formatted)
print(formatted2)
# Another way of printing the same sentence
print("AI and ML for Engineers:",course_number, '!')
```

```
1
2
city : PIT
10
t
[' ', ' ', ':', 'I', 'P', 'T', 'c', 'i', 't', 'y']
['y', 't', 'i', 'c', 'T', 'P', 'I', ':', ' ', ' ']
AI and ML for Engineers: 24787 !
AI and ML for Engineers: 24787 !
AI and ML for Engineers: 24787 !
```

Lists

In [15]:

```
# Lists - can store anything
list_of_things = [] # Initializing an empty list
list_of_things = ['AI and ML for Engineers']
list_of_things.append(24787)
list_of_things.append('CIT @ CMU')
list_of_things.append(True)

# Accessing objects
print(type(list_of_things[3]))

# Print the list
print(list_of_things)

# Create another list
another_list = ['More objects', ['List inside a list!']]

# Merge the lists
third_lists = list_of_things + another_list
print(third_lists)

# Extend list_of_things
print('\nLength before adding another_list',len(list_of_things),'\n')
list_of_things.extend(another_list)
print('Length after adding another_list',len(list_of_things),'\n')

list_of_things+=another_list # list_of_things = list_of_things + another_list
print(list_of_things)
```

```
<class 'bool'>
['AI and ML for Engineers', 24787, 'CIT @ CMU', True]
['AI and ML for Engineers', 24787, 'CIT @ CMU', True, 'More objects', ['List ins
ide a list!']]

Length before adding another_list 4

Length after adding another_list 6

['AI and ML for Engineers', 24787, 'CIT @ CMU', True, 'More objects', ['List ins
ide a list!'], 'More objects', ['List inside a list!']]
```

List slicing and join

some_list[start : end : step]: like range but separated by colons

In [16]:
```
colors = ['red','blue','green','yellow','purple','indigo']
print(len(colors))
# Python indices start at 0
print(colors[0])
print(colors[:5])
print(colors[0:5])
print(colors[2:-1])
print(colors[2:])
print(colors[::2])
print(colors[::-1])
print(colors[5][::-1])

# Join (a string method)
words = ["Python","Is","Great"]
words =".".join(words)
print(words)
```

```
6
red
['red', 'blue', 'green', 'yellow', 'purple']
['red', 'blue', 'green', 'yellow', 'purple']
['green', 'yellow', 'purple']
['green', 'yellow', 'purple', 'indigo']
['red', 'green', 'purple']
['indigo', 'purple', 'yellow', 'green', 'blue', 'red']
ogidni
Python.Is.Great
```

Boolean and condition

Exercise:

- age < 18 : cannot vote
- age => 18 : can vote

In [17]:
```
age = 21
if age < 18:
    print("Sorry, you cannot vote!")
else:
    print("You can vote!")
```

```
You can vote!
```

if elif else

In [18]:
```
color = 'yellow'
if color == "purple":
    print("great choice")
elif color == "red":
    print("I like how you think")
elif color == "pink":
    print("not bad")
else:
    print('Thats new!')
```

```
Thats new!
```

Looping

General Form :

for item in iterable_object:

```
    do something with item
```

- item is a new variable that you can call it whatever
- item references the current position of our iterator within the iterable

Ranges

- A range is just a slice of number line.
- Ex: range (1,8) will give you integers from 1 to 7

In [19]:

```
numbers = range(10)
print(list(numbers)) # It starts with zero and excludes the number 10
numbers = range(1,10)
print(list(numbers)) # It starts with one and excludes the number 10
odd = range(1,10,2) # start, end, step
print(list(odd)) #It starts with 1 and takes a step of 2
even = range(0,10,2) # start, end, step
print(list(even)) #It starts with 0 and takes a step of 2
```

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

Exercise: Use a for loop for add up every odd number from 10 to 20 (inclusive)

In [20]:

```
add = 0
for n in range(10, 21):  #remember range is exclusive, so we have to go up to 21
    if n % 2 != 0: # % means mod
        add = add + n
        print(n)
print('Sum of the numbers: ',add)
```

```
11
13
15
17
19
Sum of the numbers:  75
```

While Loop and using "len"

In [21]:

```
numbers = [1,2,3,4]
i=0
while i < len(numbers):
```

```
    print(numbers[i])
    i+=1
```

```
1
2
3
4
```

List Comprehension:

- A shorthand syntex that allows us to generate new list

Looping vs List Comprehension

In [101…

```
numbers = [1,2,3,4,5]
doubled_numbers = []

for num in numbers:
    doubled_numbers.append(num * 2)

print(doubled_numbers)
```

```
[2, 4, 6, 8, 10]
```

In [102…

```
numbers = [1,2,3,4,5]

doubled_numbers = [num * 2 for num in numbers]

print(doubled_numbers)
```

```
[2, 4, 6, 8, 10]
```

In [103…

```
numbers = [4,5,6,7,8,9,10]
evens = [num for num in numbers if num % 2 == 0]
odds = [num for num in numbers if num % 2 != 0]
print(evens, odds)
```

```
[4, 6, 8, 10] [5, 7, 9]
```

Dictionaries

- A data structure that contains key-value pairs
- Keys and values are separated by colons
- Keys are almost always numnbers or strings
- Note: dictionary doesnt have a specific order, unlike a list

- update: Update keys and values in a dict with another set of key value pairs.

In [104…

```
first = dict(a=1,b=2,c=3,d=4)
second = {}
second.update(first)
print(second)
# Overwrite an existing ket
```

```
second["a"]= "AMAZING"
print(second)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
{'a': 'AMAZING', 'b': 2, 'c': 3, 'd': 4}
```

In [105…

```
cmu = {
    'founder': "Andrew Carnegie",
    'private school?': True,
    'acceptance rate':0.2,
     4:"number of other campuses"
}
print(cmu)
print(cmu["founder"])
# print(cmu["mascot"])
```

```
{'founder': 'Andrew Carnegie', 'private school?': True, 'acceptance rate': 0.2,
4: 'number of other campuses'}
Andrew Carnegie
```

Accessing all values in a dictionary

- Access value: Use .values()

In [106…

```
for value in cmu.values():
    print(value)
```

```
Andrew Carnegie
True
0.2
number of other campuses
```

- Access key: Use .keys()

In [28]:

```
for key in cmu.keys():
    print(key)
```

```
founder
private school?
acceptance rate
4
```

- Access both: .items()

In [29]:

```
for key,value in cmu.items():
    print(key, ":", value)
```

```
founder : Andrew Carnegie
private school? : True
acceptance rate : 0.2
4 : number of other campuses
```

- Test if a key exists: Use "in"

In [30]:
```
print("founder" in cmu)
print("good sleep" in cmu)
```

```
True
False
```

Dictionary Methods

- pop: Takes a single arguement corresponding to a ket and removes that key-value pair from the dict. Returns the value corresponding to the key that was removed.

In [31]:
```
d = dict(a=1,b=2,c=3)
d.pop("a") # 1
# d.pop() # TypeError: pop expected at least 1 arguement
```

Out[31]:
```
1
```

Introductory Example: Implementing with Lists

To know the difference between an array and a list please Google ;)

This blog also provides a good explanation : https://medium.com/backticks-tildes/list-vs-array-python-data-type-40ac4f294551

In [32]:
```
# Inputs
# Now that we know what a list is lets use it to do a simple calculation
theta_1 = [[2, 1, 3],
     [1, 2, 3],
     [3, 1, 2]]

x = [7, 8, 5]
theta_0 = [1, 1, 1]

# Output
y = [0, 0, 0]

# Computation of y = theta_1*x + theta_0
for i in range(len(theta_1)):
    for j in range(len(x)):
        y[i] = y[i]+theta_1[i][j] * x[j]
    y[i] = y[i] + (theta_0[i])

print(y)
```

```
[38, 39, 40]
```

- What is NumPy?

If you have installed Anaconda correctly, Numpy should come with it. Just in case, if numpy is not installed please follow instructions at https://scipy.org/install.html

NumPy-based Vectorized Implementation

a.dot(b)

a*b

np.matmul(a,b)

a@b

In [33]:
```
import numpy as np
```

In [112…
```
theta_1 = np.array([[2, 1, 3],
     [1, 2, 3],
     [3, 1, 2]])
theta_1.shape
```

Out[112… (3, 3)

In [113…
```
x = np.array([7, 8, 5])
x.shape
```

Out[113… (3,)

In [36]:
```
theta_0 = np.ones(3)
theta_0
```

Out[36]: array([1., 1., 1.])

In [37]:
```
theta_1.dot(x)
```

Out[37]: array([37, 38, 39])

In [38]:
```
np.matmul(theta_1,x)
```

Out[38]: array([37, 38, 39])

In [39]:
```
np.matmul(theta_1,x) + theta_0
```

Out[39]: array([38., 39., 40.])

In [114…
```
theta_1@x
```

Out[114… array([37, 38, 39])

In [115…
```
theta_1*x
```

Out[115…
```
array([[14,  8, 15],
       [ 7, 16, 15],
```

```
       [21,  8, 10]])
```

Multi-Dimensional Arrays

In [41]:
```
T = np.random.random((4,2,4)) # Creates an array of random numbers between 0 and
T
```

Out[41]:
```
array([[[0.68750665, 0.49398625, 0.47858434, 0.35071712],
        [0.23869637, 0.78157449, 0.19127851, 0.19399739]],

       [[0.68619695, 0.97168959, 0.43965528, 0.00547982],
        [0.88144044, 0.75155198, 0.22382978, 0.47952113]],

       [[0.39730076, 0.65537218, 0.69151552, 0.34921768],
        [0.87773041, 0.71257969, 0.84105234, 0.28591784]],

       [[0.26621071, 0.27579241, 0.11845169, 0.45760663],
        [0.10183073, 0.83111632, 0.91528513, 0.41461627]]])
```

Random seed

In [42]:
```
T = np.random.random((2,2)) # Creates an array of random numbers between 0 and 1
print(T)

np.random.seed(24878)
T = np.random.random((2,2)) # Now, all the random numbers are fixed as per the s
print(T)
```

```
[[0.47448561 0.52024984]
 [0.27408603 0.12845574]]
[[0.76643613 0.23417987]
 [0.68789336 0.3656141 ]]
```

In [43]:
```
np.random.seed(24878)
T = np.random.random((2,2))
print(T)
```

```
[[0.76643613 0.23417987]
 [0.68789336 0.3656141 ]]
```

In [44]:
```
W = T[1]
W.shape, W.dtype
```

Out[44]:
```
((2,), dtype('float64'))
```

In [45]:
```
X = np.full((4,3), 24787, dtype=np.int32) # prints out the same in all dimension
X.shape, X.dtype,X
```

Out[45]:
```
((4, 3),
 dtype('int32'),
 array([[24787, 24787, 24787],
        [24787, 24787, 24787],
        [24787, 24787, 24787],
        [24787, 24787, 24787]], dtype=int32))
```

Basic Element-wise Operations

In [46]:
```
y = np.arange(5)
print(y, np.array(range(5)))
```

```
[0 1 2 3 4] [0 1 2 3 4]
```

In [47]:
```
y == np.array(range(5))
```

Out[47]:
```
array([ True,  True,  True,  True,  True])
```

In [48]:
```
y + 1
```

Out[48]:
```
array([1, 2, 3, 4, 5])
```

In [49]:
```
y * 10
```

Out[49]:
```
array([ 0, 10, 20, 30, 40])
```

In [50]:
```
(y + 1)**2
```

Out[50]:
```
array([ 1,  4,  9, 16, 25])
```

In [51]:
```
y + y
```

Out[51]:
```
array([0, 2, 4, 6, 8])
```

In [52]:
```
y / (y + 1)
```

Out[52]:
```
array([0.        , 0.5       , 0.66666667, 0.75      , 0.8       ])
```

In [53]:
```
np.sqrt(y**2)
```

Out[53]:
```
array([0., 1., 2., 3., 4.])
```

In [54]:
```
np.max(y), np.min(y), np.sum(y)
```

Out[54]:
```
(4, 0, 10)
```

Indexing NumPy Arrays

In [55]:
```
np.save('AIML.npy', np.arange(16).reshape(4,4))
```

In [56]:
```
T = np.load('AIML.npy')
T,  T.shape
```

Out[56]:
```
(array([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15]]),
 (4, 4))
```

In [57]:
```
T[0]
```

Out[57]:
```
array([0, 1, 2, 3])
```

In [58]:
```
T[1,0]  # more efficient version of T[1][0]# a similar approach can be used for
```

Out[58]:
```
4
```

In [59]:
```
T[0,0:3] # selecting the specific elements for operations
```

Out[59]:
```
array([0, 1, 2])
```

In [60]:
```
(T % 10) == 0
```

Out[60]:
```
array([[ True, False, False, False],
       [False, False, False, False],
       [False, False,  True, False],
       [False, False, False, False]])
```

In [61]:
```
T[(T % 10) == 0]
```

Out[61]:
```
array([ 0, 10])
```

In [62]:
```
np.minimum(T, 5)
```

Out[62]:
```
array([[0, 1, 2, 3],
       [4, 5, 5, 5],
       [5, 5, 5, 5],
       [5, 5, 5, 5]])
```

In [63]:
```
np.max(T)# please try np.maximum and by following the upper example
```

Out[63]:
```
15
```

Computation Along Axes

In [64]:
```
A = np.random.random((2,2))
A
```

Out[64]:
```
array([[0.43484507, 0.15795619],
       [0.77549348, 0.40143293]])
```

In [65]:
```
A.sum()# all
```

Out[65]:
```
1.7697276711397136
```

In [66]:
```
np.sum(A)
```

Out[66]:
```
1.7697276711397136
```

In [67]:
```
A.sum(axis=0) # column wise # col1 # col2 #col3
```

Out[67]:
```
array([1.21033856, 0.55938911])
```

In [68]:
```
np.sum(A, axis = 0)
```

Out[68]:
```
array([1.21033856, 0.55938911])
```

In [69]:
```
A.sum(axis=1) # row 1
```

Out[69]:
```
array([0.59280126, 1.17692641])
```

In [70]:
```
np.sum(A, axis = 1)
```

Out[70]:
```
array([0.59280126, 1.17692641])
```

In [71]:
```
A.max(axis=0), A.argmax(axis=0)# column# the argmax returns the index where max
```

Out[71]:
```
(array([0.77549348, 0.40143293]), array([1, 1]))
```

In [72]:
```
np.max(A, axis = 0), np.argmax(A, axis = 0)
```

Out[72]:
```
(array([0.77549348, 0.40143293]), array([1, 1]))
```

Combining Arrays: Stacking & Concatenation

In [73]:
```
a = np.full(3, 2)
b = np.full(3, 4)
c = np.full(3, 6)
print(a, b, c)
```

```
[2 2 2] [4 4 4] [6 6 6]
```

Stacking creates a new axis. It joins the supplied arrays, which must have the same shape, along that axis. Indexing a single element from that axis returns the appropriate input array.

In [74]:
```
B = np.stack([a, b, c], axis=1)
B, B.shape
```

Out[74]:
```
(array([[2, 4, 6],
        [2, 4, 6],
        [2, 4, 6]]),
 (3, 3))
```

In [75]:
```
B = np.hstack((a, b, c))
B, B.shape
```

Out[75]:
```
(array([2, 2, 2, 4, 4, 4, 6, 6, 6]), (9,))
```

In [76]:
```
B = np.vstack((a, b, c))
B, B.shape
```

Out[76]:
```
(array([[2, 2, 2],
        [4, 4, 4],
        [6, 6, 6]]),
 (3, 3))
```

In [77]:
```
B[:,1]
```

Out[77]:
```
array([2, 4, 6])
```

Concatenating arrays joins them along an *existing* aixs.

In [78]:
```
C = np.concatenate([B+10, B+50], axis=1 )
C
```

Out[78]:
```
array([[12, 12, 12, 52, 52, 52],
       [14, 14, 14, 54, 54, 54],
       [16, 16, 16, 56, 56, 56]])
```

Transposing and Reshaping

In [79]:
```
C, C.shape
```

Out[79]:
```
(array([[12, 12, 12, 52, 52, 52],
        [14, 14, 14, 54, 54, 54],
        [16, 16, 16, 56, 56, 56]]),
 (3, 6))
```

In [80]:
```
CT = C.T
CT, CT.shape
```

Out[80]:
```
(array([[12, 14, 16],
        [12, 14, 16],
        [12, 14, 16],
        [52, 54, 56],
        [52, 54, 56],
```

```
       [52, 54, 56]]),
 (6, 3))
```

In [81]:
```
CR = C.reshape(9, 2)
CR, CR.shape # it goes row wise
```

Out[81]:
```
(array([[12, 12],
        [12, 52],
        [52, 52],
        [14, 14],
        [14, 54],
        [54, 54],
        [16, 16],
        [16, 56],
        [56, 56]]),
 (9, 2))
```

Saving and Loading

Saving and loading a single NumPy array

In [82]:
```
# Save single array
x = np.random.random((5,))
print(x)

np.save('tmp.npy', x)
```

```
[0.12992504 0.45724977 0.71825714 0.89288735 0.10466857]
```

In [83]:
```
# Load the array
y = np.load('tmp.npy')

print(y)
```

```
[0.12992504 0.45724977 0.71825714 0.89288735 0.10466857]
```

Loading a csv file

In [84]:
```
# load csv file
import csv
with open('imp.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        print(row)
# This is a good starting point
# you may want to use this when coding up your homework
```

```
['-0.08079', '10']
['-0.13378', '10']
['-0.00862', '10']
['-0.04602', '10']
['-0.01998', '10']
['-0.0212', '10']
['-0.00895', '10']
['-0.02897', '10']
['-0.02008', '10']
['0.00202', '10']
```

['-0.01943', '10']
['-0.01903', '10']
['-0.00645', '10']
['-0.00235', '10']
['0.05695', '0']
['0.10163', '0']
['0.13792', '0']
['-0.16741', '110']
['0.09675', '110']
['0.12613', '110']
['0.19621', '110']
['0.11198', '110']
['0.12281', '110']
['0.12198', '110']
['0.11859', '110']
['-0.08413', '110']
['0.07572', '110']
['0.0881', '110']
['0.09133', '110']
['0.17411', '110']
['0.22738', '110']
['0.16986', '110']
['0.19638', '110']
['0.08839', '110']
['0.17157', '110']
['0.16655', '110']
['0.18993', '110']
['0.06115', '110']
['0.12599', '110']
['0.19396', '110']
['0.14168', '110']
['0.0774', '110']
['0.16222', '110']
['0.14302', '110']
['0.20389', '110']
['0.14527', '110']
['0.08859', '110']
['0.13977', '110']
['0.17785', '110']
['0.02958', '110']
['-0.00088', '110']
['0.10586', '110']
['0.09249', '110']
['0.04947', '110']
['0.15746', '110']
['0.08521', '110']
['-0.01344', '110']
['0.18974', '110']
['0.19554', '110']
['0.18589', '110']
['0.20867', '110']
['0.02813', '110']
['0.02775', '110']
['-0.07904', '110']
['0.14697', '110']
['0.05406', '110']
['0.00073', '110']
['-0.00633', '110']
['0.0951', '110']
['-0.05711', '110']
['0.07355', '112']
['0.05525', '112']
['0.0574', '123']

Functions

Need to add parenthesis after the function name:

```
def myfunction():
```

return:

- Exits the function
- Outputs whatever value is places after the return keyword

In [85]:
```
from random import random
def flip_coin():
    # generates random number 0-1
    r = random()
    if r > 0.5:
        result = "Heads"
    else:
        result = "Tails"

    return result

flip_coin()
```

Out[85]:
```
'Heads'
```

Accepting input parameters

In [86]:
```
def double(d):

    return 2*d

double(10)
```

Out[86]:
```
20
```

Default Parameters

- Allows you to be more defensive
- Avoids errors with incorrect parameters

In [87]:
```
def add(a=10, b=20):
    return a+b
add()
```

Out[87]:
```
30
```

Keyworded Argument

*args: A special operator we can pass to functions. Gathers remaining arguments as a tuple.

This is just a parameter. You can call it whatever you want after the "*"!

Instead of doing:

```
def sum_all_nums(num1, num2, num3, num4):
    return num1+num2+num3+num4
```

In [88]:
```
def sum_all_nums(*args):
    total = 0
    for num in args:
        total += num
    return total

print(sum_all_nums(4,6,9,5,8))
print(sum_all_nums(4,6))
```

```
32
10
```

Lambdas: like a function that has no name (anonymous function)

In [89]:
```
def square(num): return num*num
square2 = lambda num: num*num # no return
print(square2(7))
```

```
49
```

References

You can find a more complete introduction at
https://docs.scipy.org/doc/numpy/user/quickstart.html

Graph and plotting using Matplotlib

Matplotlib is a library used for graphing purpose. You can find a more complete introduction at
https://matplotlib.org/tutorials/introductory/pyplot.html

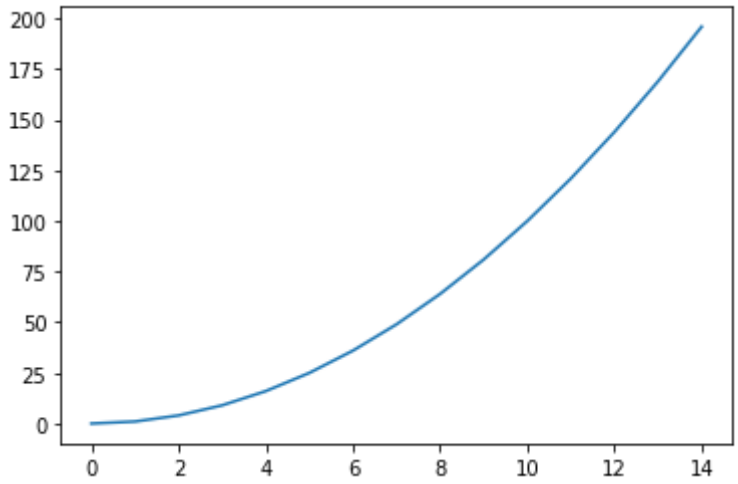
In [90]:
```
import matplotlib.pyplot as plt
```

Continuous plot (Generally used for continuous variables)

Here is an example of a simple plot

In [91]:
```
x1 = np.arange(15)
y1 = x1**2
plt.plot(x1, y1)
```

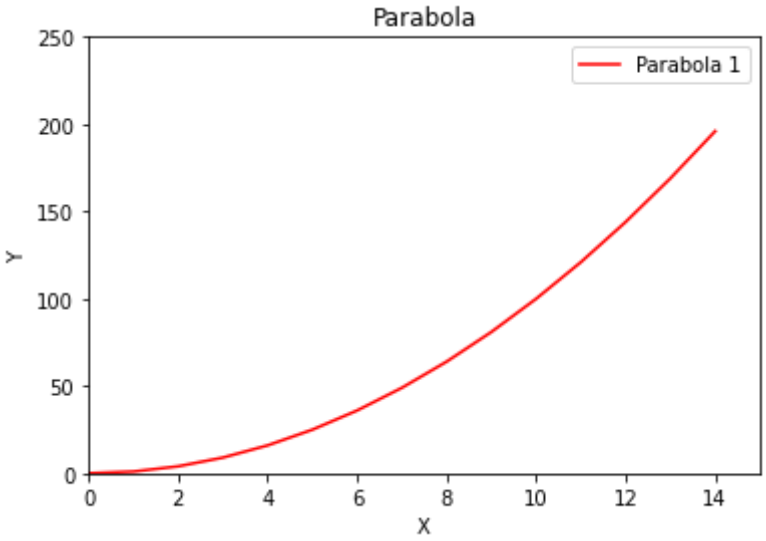
Out[91]:
```
[<matplotlib.lines.Line2D at 0x7faf59591d68>]
```



Adding title, labels, axis limits, etc.

In [92]:

```
x1 = np.arange(15)
y1 = x1**2
plt.plot(x1, y1, label = 'Parabola 1', c = 'r') # 'c' decides the color
plt.axis([0, 15, 0, 250]) # sets the axes limit [xmin, xmax, ymin, ymax]
plt.title('Parabola') # Adds the title to the graph
plt.xlabel('X') # Adds the x axis label
plt.ylabel('Y') # Adds the y axis label
plt.legend() # Adds the legend of all the series
plt.show()
```

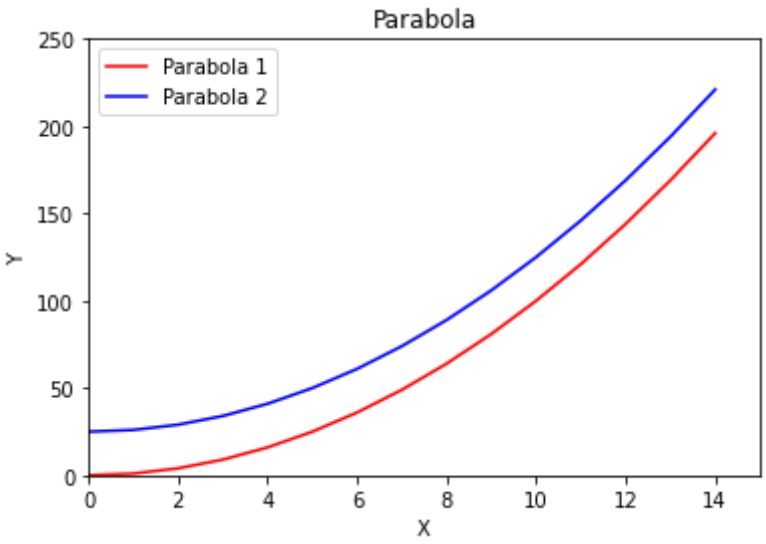


Plotting two different sets of datapoints on the same graph

In [93]:

```
x2 = np.arange(15)
y2 = x2**2 + 25
plt.plot(x1, y1, label = 'Parabola 1', c = 'r')
plt.plot(x2, y2, label = 'Parabola 2', c = 'b')
plt.axis([0, 15, 0, 250])
plt.title('Parabola')
plt.xlabel('X')
plt.ylabel('Y')
```

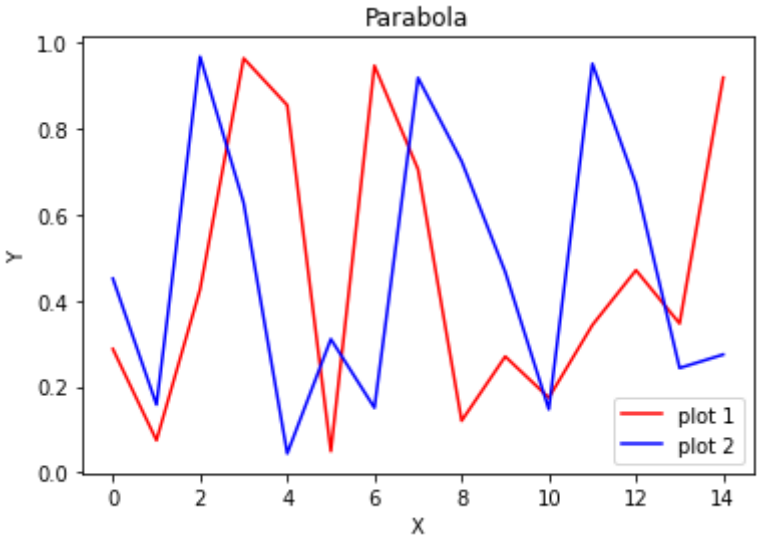
```
plt.legend()
plt.show()
```



plt.plot command takes x values as indices if not passed explicitely

In [94]:

```
plt.plot(np.random.random(15), label = 'plot 1', c = 'r')
plt.plot(np.random.random(15), label = 'plot 2', c = 'b')
plt.title('Parabola')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```



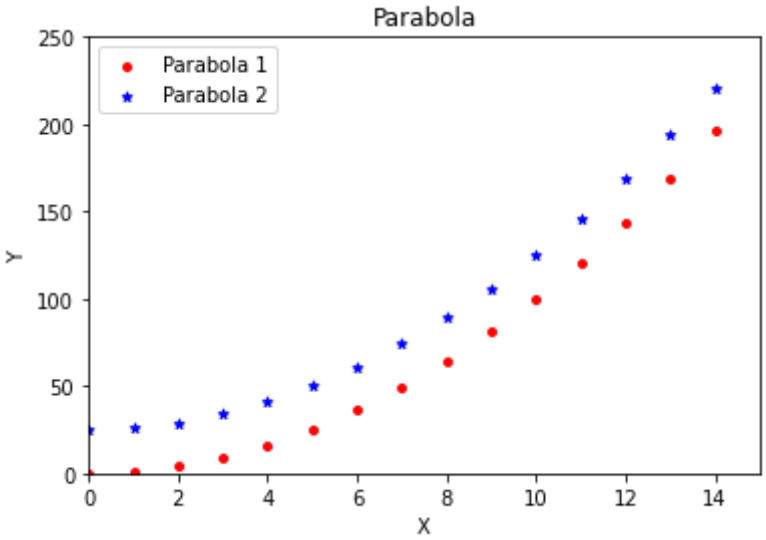
Scatter plot (Generally used for discrete datapoints)

Building on the previous method, making a scatter plot

In [95]:

```
x1 = np.arange(15)
y1 = x1**2

x2 = np.arange(15)
```

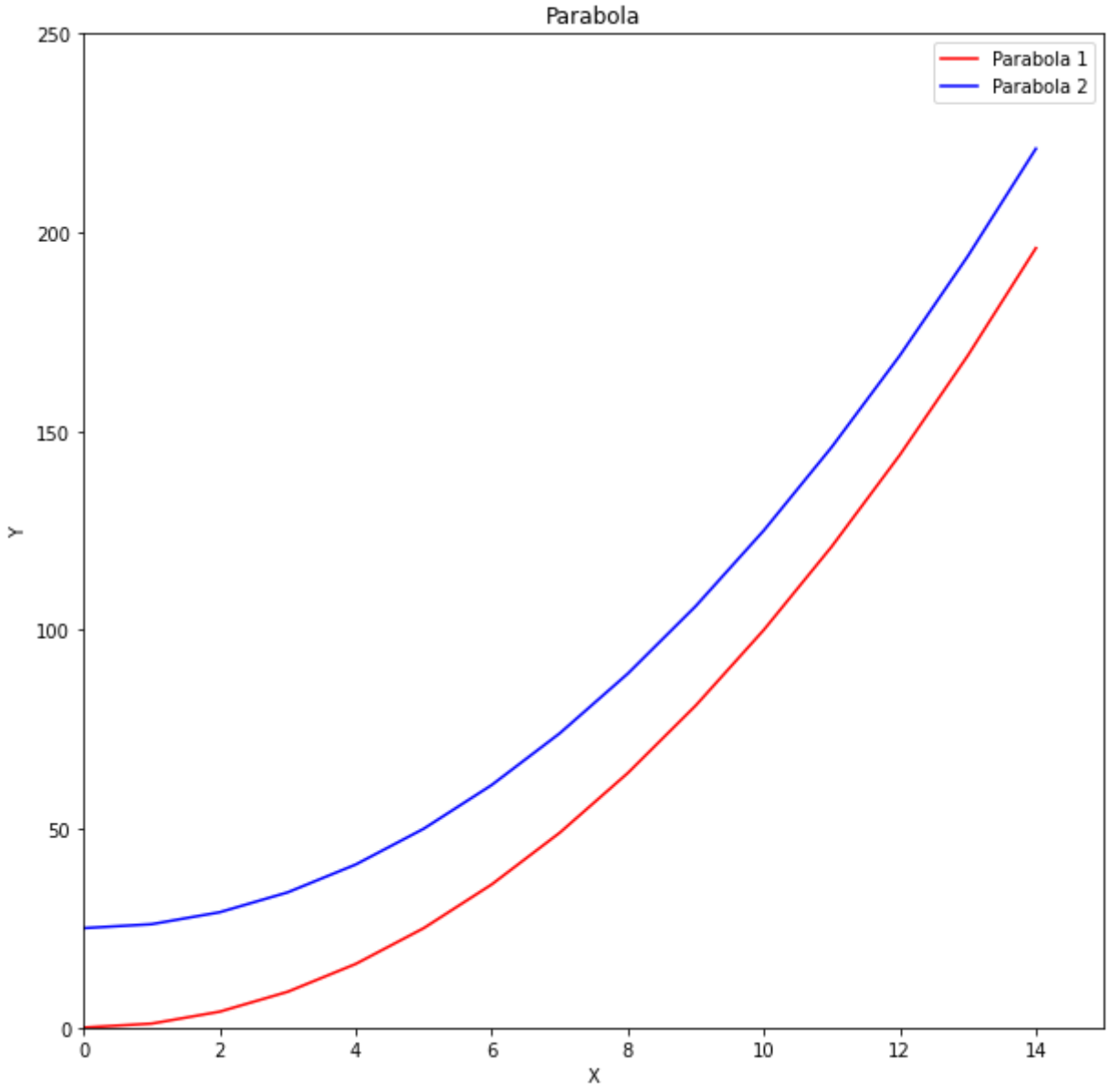
```
y2 = x2**2 + 25

plt.scatter(x1, y1, label = 'Parabola 1', c = 'r', s = 15 ) # 's' decides the si
plt.scatter(x2, y2, label = 'Parabola 2', c = 'b', s = 25, marker = '*' ) # mark
plt.axis([0, 15, 0, 250]) # sets the axes limit [xmin, xmax, ymin, ymax]
plt.title('Parabola') # Adds the title to the graph
plt.xlabel('X') # Adds the x axis label
plt.ylabel('Y') # Adds the y axis label
plt.legend() # Adds the legend of all the series
plt.show()
```



Lets change the figure size!

In [96]:

```
plt.figure(figsize=(10,10)) # Sets the size of the plot
plt.plot(x1, y1, label = 'Parabola 1', c = 'r')
plt.plot(x2, y2, label = 'Parabola 2', c = 'b')
plt.axis([0, 15, 0, 250])
plt.title('Parabola')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.savefig('graph.png') # Saves the graph as a .png file
plt.show()
```


Parabola
Parabola 1
Parabola 2
250
200
150
100
50
0
Y
0
2
4
6
8
10
12
14
X

In [ ]: