PYTHON

PYTHON

- An object oriented interpreter-based programming language
- Some advantages:
  - Free
  - Powerful
  - Widely used (Google, NASA, Yahoo, Electronic Arts, some UNIX scripts etc.)
- Named after a British comedy "Monty Python's Flying Circus"
- Official website (Python the programming language, not the Monty Python comedy troop): http://www.python.org

FOR..

- If you want to learn python systematically, see youtube or other course.
- This ppt show only basic instructions for hw.

these sites may help you!

- https://wikidocs.net/book/1
- YOUTUBE(ENG, …) : type ‘python tutorials’ or anything about python.
- …

KEY WORDS IN PYTHON[1]

and
as
assert
break
class
continue
def

del
elif
else
except
exec
finally
for

from
global
if
import
in
is
lambda

not
or
pass
print
raise
return
try

while
with
yield

[1] From “*Starting out with Python*” by Tony Gaddis

WHAT’S DIFFERENCE.

C Lang.

```
#include <cstdio>
#include <math.h>
int main(int argc, int* argv){
   int a, b, c;
   char a[5] = {‘h’,’e’,’l’,’l’,’o’}     // array
   for(int i=1 ; i<=5 ; i++){
      printf(“%c”,a[i]);
   }
}
```

Python Lang.

```
import math
a=‘h’     no data type required
l = [‘h’,’e’,’l’,’l’,’o’]       #(‘h’,’e’,’l’,’l’,’o’)
           list                          tuple

for index in ragne(5):
🟥print( l )
 not use { }, use tap.
enum = list(enumerate(l))
print(enum)
```

BASIC

```
# Strings
data = 'hello world'
print(data[0])
print(len(data))
print(data)
```

```
h
11
hello world
```

```
# Numbers
value = 123.1
print(value)
value = 10
print(value)
```

```
123.1
10
```

```
# Boolean
a = True
b = False
print(a, b)
```

```
True, False
```

```
a, b, c = 1, 2, 3
print(a, b, c)
```

```
1, 2, 3
```

ESCAPE CODES

Escape sequence	Description
\a	Alarm. Causes the program to beep.
\n	Newline. Moves the cursor to beginning of the next line.
\t	Tab. Moves the cursor forward one tab stop.
\'	Single quote. Prints a single quote.
\"	Double quote. Prints a double quote.
\\	Backslash. Prints one backslash.

```
print ("\a*Beep!*")

print ("hi\nthere")

print ('it\'s')

print ("he\\y \"you\"")
```

```
*Beep!*
hi
there
it's
he\y "you"
```

THE IF STATEMENT

- Syntax:

```
if <condition>:
    <statements>
x = 5
if x > 4:
    print(“x is greater than 4”)
print(“This is not in the scope of the if”)
```

THE IF STATEMENT

- The colon is required for the if
- Note that all statement indented one level in from the if are with in it scope:

```
x = 5
if x > 4:
        print(“x is greater than 4”)
        print(“This is also in the scope of the if”)
```

THE IF/ELSE STATEMENT

```
if <condition>:
      <statements>
else:
      <statements>
```

- Note the colon following the else
- This works exactly the way you would expect

THE FOR LOOP

- This is similar to what you're used to from C or Java, but not the same
- Syntax:

```
    for variableName in groupOfValues:
        <statements>
```

- variableName gives a name to each value, so you can refer to it in the statements.
- groupOfValues can be a range of integers, specified with the range function.

RANGE

- The range function specifies a range of integers:

`range(start, stop)` - the integers between start (inclusive)
and stop (exclusive)

- It can also accept a third value specifying the change between values.

`range(start, stop, step)` - the integers between start (inclusive)
and stop (exclusive) by step

```
for x in range(1, 6):
    print(x, "squared is", x * x)
```

THE WHILE LOOP

- Executes a group of statements as long as a condition is True.
- Good for indefinite loops (repeat an unknown number of times)
- Syntax:

```
while <condition>:
    <statements>
```

- Example:

```
number = 1
while number < 200:
    print (number)
    number = number * 2
```

BASIC

```
# If-Then-Else Conditional

value = 99

if value == 99:

    print('That is fast')

elif value > 200:

    print('That is too fast')

else:

    print('That is safe')
```

```
That is
fast
```

```
# For-Loop

for i in range(10):

    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
# While-Loop

i = 0

while i < 10:

    print(i)

    i += 1
```

```
0
1
2
3
4
5
6
7
8
9
```

BASIC – DATA STRUCTURE

# Tuple

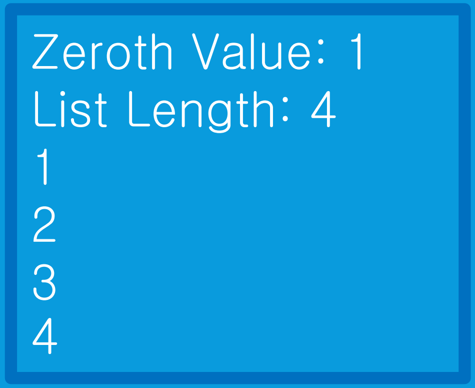
Tuples are read-only collections of items.

```
a = (1, 2, 3)
print(a)
```

# List

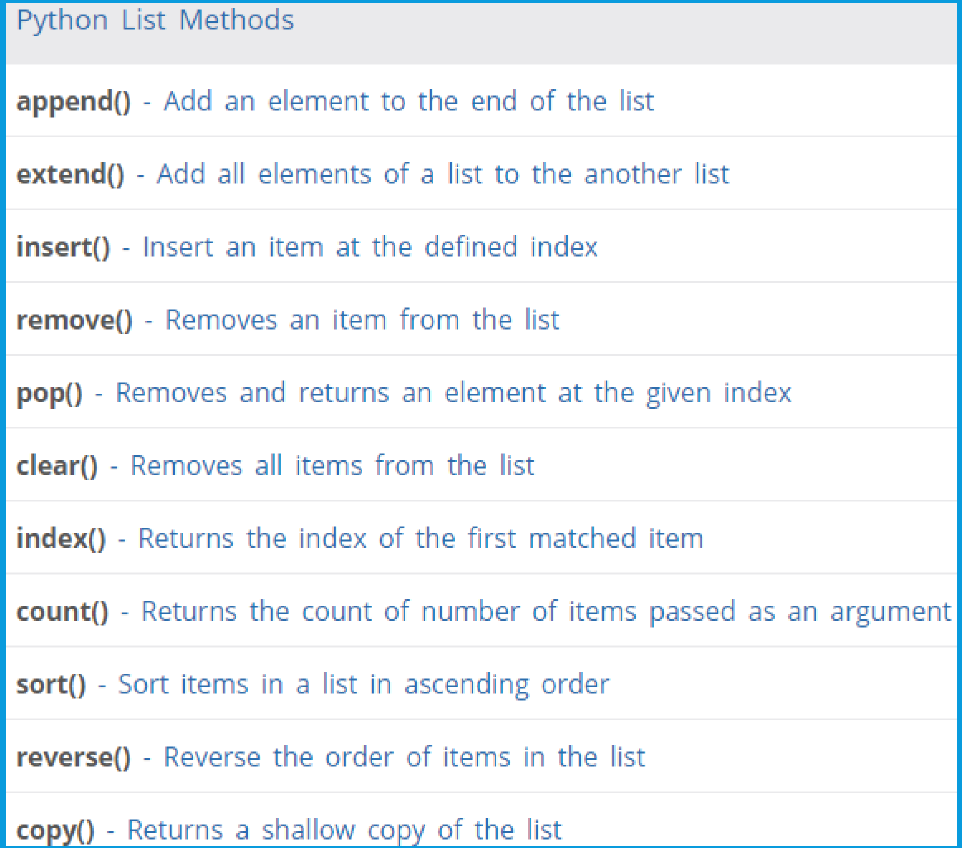
Lists use the square bracket notation and can be index using array notation.

```
mylist = [1, 2, 3]
print("Zeroth Value: %d" % mylist[0])
mylist.append(4)
print("List Length: %d" % len(mylist))
for value in mylist:
    print(value)
```



```
Zeroth Value: 1
List Length: 4
1
2
3
4
```

LIST DATA TYPE



Python List Methods

append() - Add an element to the end of the list

extend() - Add all elements of a list to the another list

insert() - Insert an item at the defined index

remove() - Removes an item from the list

pop() - Removes and returns an element at the given index

clear() - Removes all items from the list

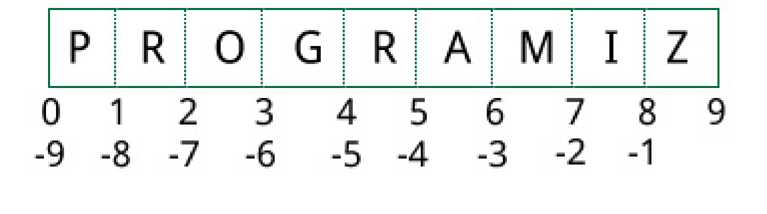
index() - Returns the index of the first matched item

count() - Returns the count of number of items passed as an argument

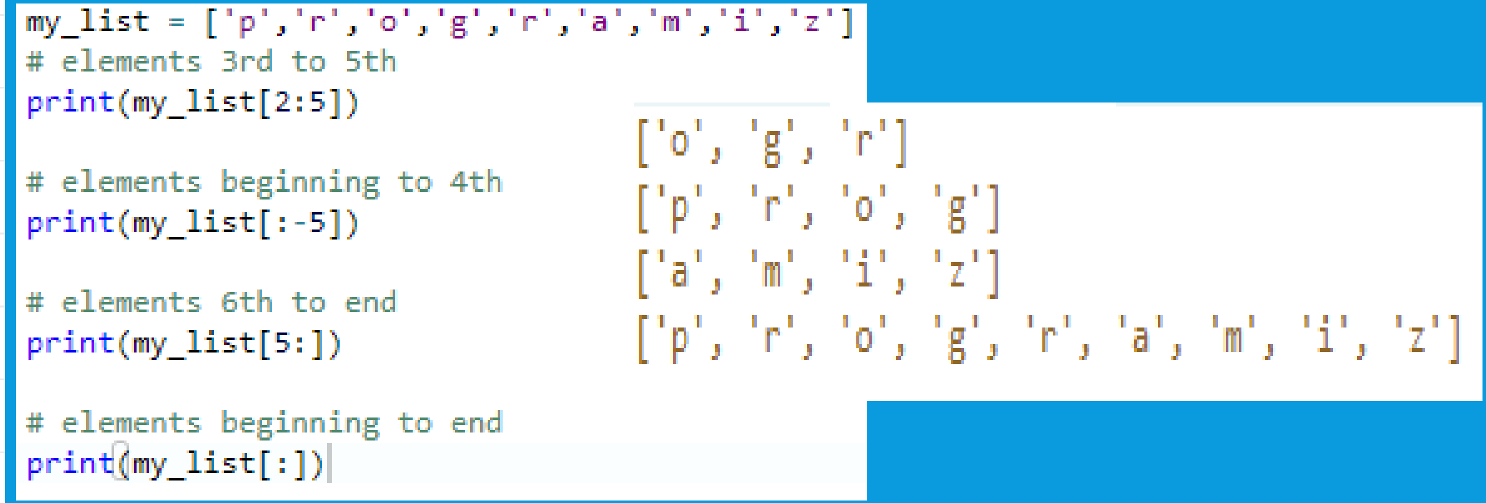
sort() - Sort items in a list in ascending order

reverse() - Reverse the order of items in the list

copy() - Returns a shallow copy of the list



P	R	O	G	R	A	M	I	Z	
0	1	2	3	4	5	6	7	8	9
-9	-8	-7	-6	-5	-4	-3	-2	-1	



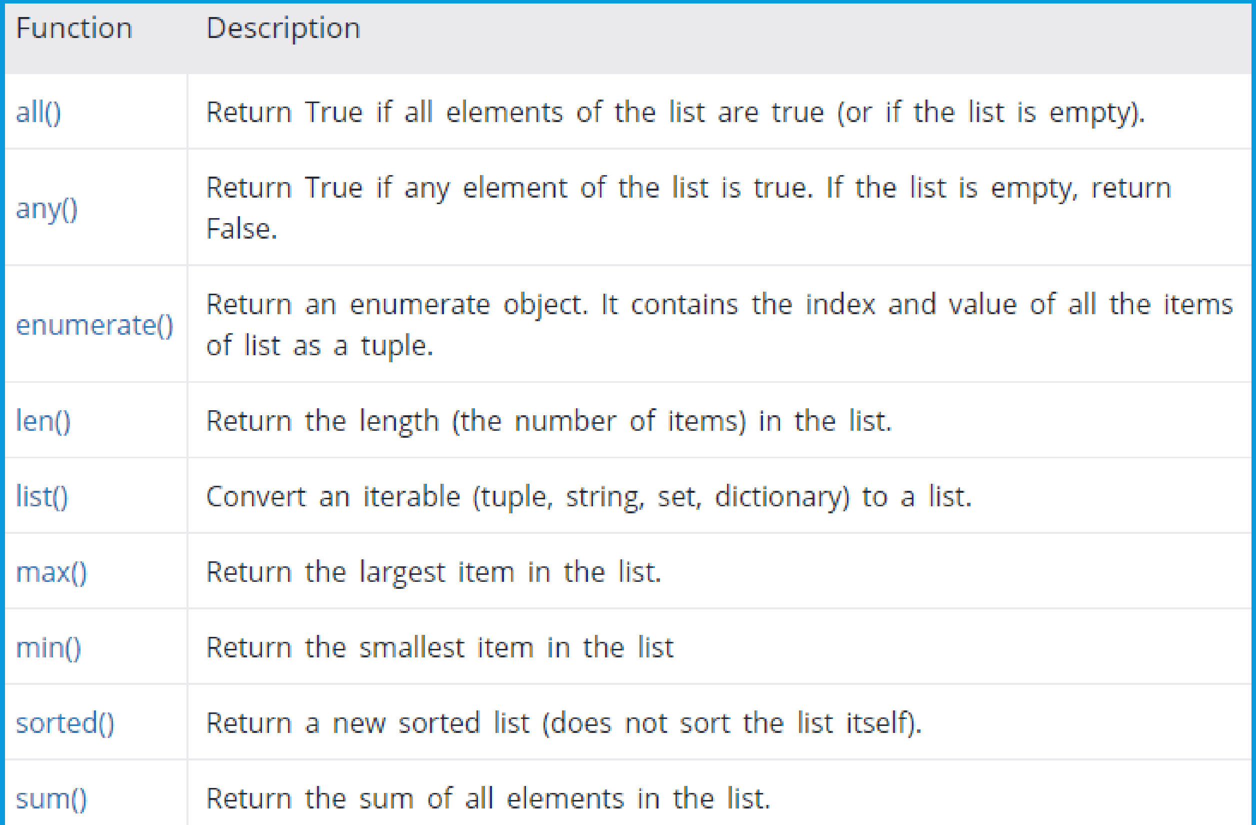
```
my_list = ['p','r','o','g','r','a','m','i','z']
# elements 3rd to 5th
print(my_list[2:5])

# elements beginning to 4th
print(my_list[:-5])

# elements 6th to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

```
['o', 'g', 'r']
['p', 'r', 'o', 'g']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

ref. https://www.programiz.com/python-programming/list

LIST DATA TYPE

Built-in Functions with List



Function	Description
all()	Return True if all elements of the list are true (or if the list is empty).
any()	Return True if any element of the list is true. If the list is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of list as a tuple.
len()	Return the length (the number of items) in the list.
list()	Convert an iterable (tuple, string, set, dictionary) to a list.
max()	Return the largest item in the list.
min()	Return the smallest item in the list
sorted()	Return a new sorted list (does not sort the list itself).
sum()	Return the sum of all elements in the list.

ref. https://www.programiz.com/python-programming/list

LISTS ARE MUTABLE - SOME USEFUL METHODS

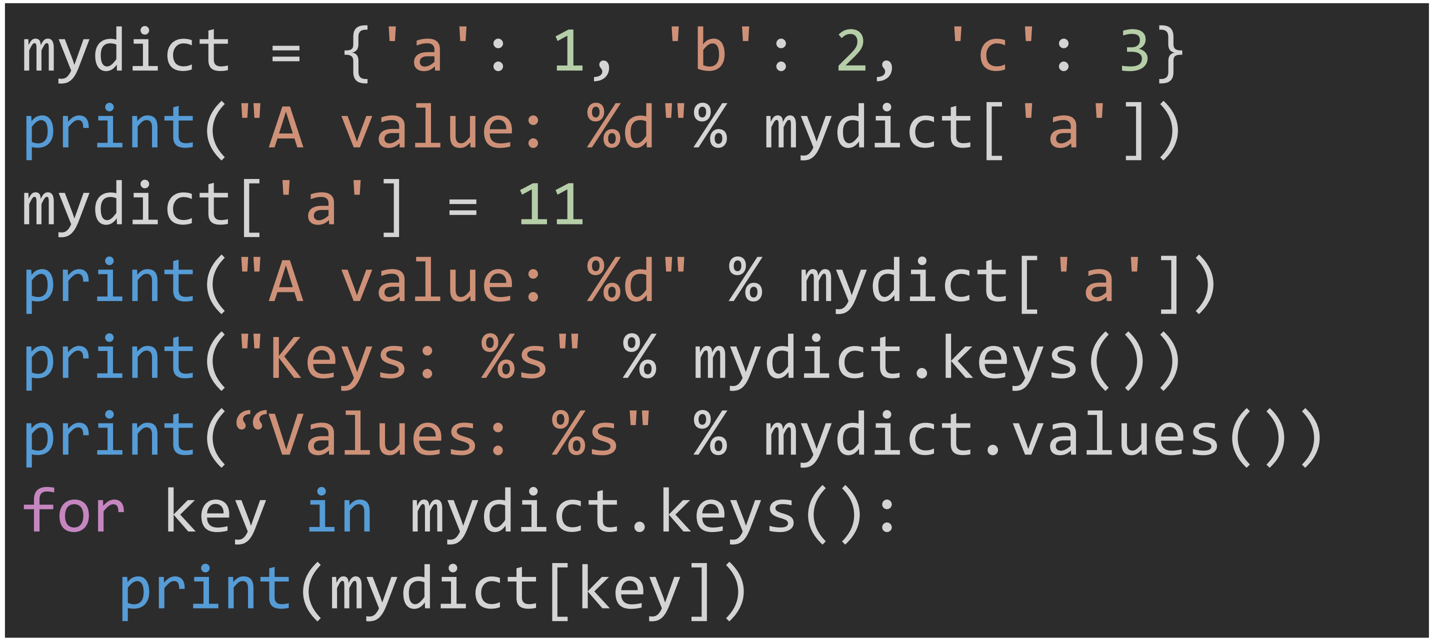
```
language = ['Python', 'Java', 'C++']
language.append("French")
print(language)
language.remove('Java')
print(language)
language.extend(["C#"])
print(language)
del language[0]
print(language)
language.reverse()
print(language)
language.insert(0,"Java")
print(language)
```

```
['Python', 'Java', 'C++', 'French']
['Python', 'C++', 'French']
['Python', 'C++', 'French', 'C#']
['C++', 'French', 'C#']
['C#', 'French', 'C++']
['Java', 'C#', 'French', 'C++']
```

DICTIONARY DATA TYPE

#Dictionary

Dictionaries are mappings of names to values, like key-value pairs. Note the use of the curly bracket and colon notations when defining the dictionary.



```
mydict = {'a': 1, 'b': 2, 'c': 3}
print("A value: %d"% mydict['a'])
mydict['a'] = 11
print("A value: %d" % mydict['a'])
print("Keys: %s" % mydict.keys())
print(“Values: %s" % mydict.values())
for key in mydict.keys():
   print(mydict[key])
```

```
A value: 1
A value: 11
Keys: dict_keys(['a', 'b', 'c'])
Values: dict_values([11, 2, 3])
11
2
3
```

Keys can be any immutable value numbers, strings, tuples, frozenset, not list, dictionary, set, ...

SETS

- Sets are similar to dictionaries in Python, except that they consist of only keys with no associated values.
- Essentially, they are a collection of data with no duplicates.
- They are very useful when it comes to removing duplicate data from data collections.

WRITING FUNCTIONS

- Define a function:

```
def <function name>(<parameter list>)
```

- The function body is indented one level:

```
def computeSquare(x):
     return x * x
```

BASIC – DATA STRUCTURE

#Functions

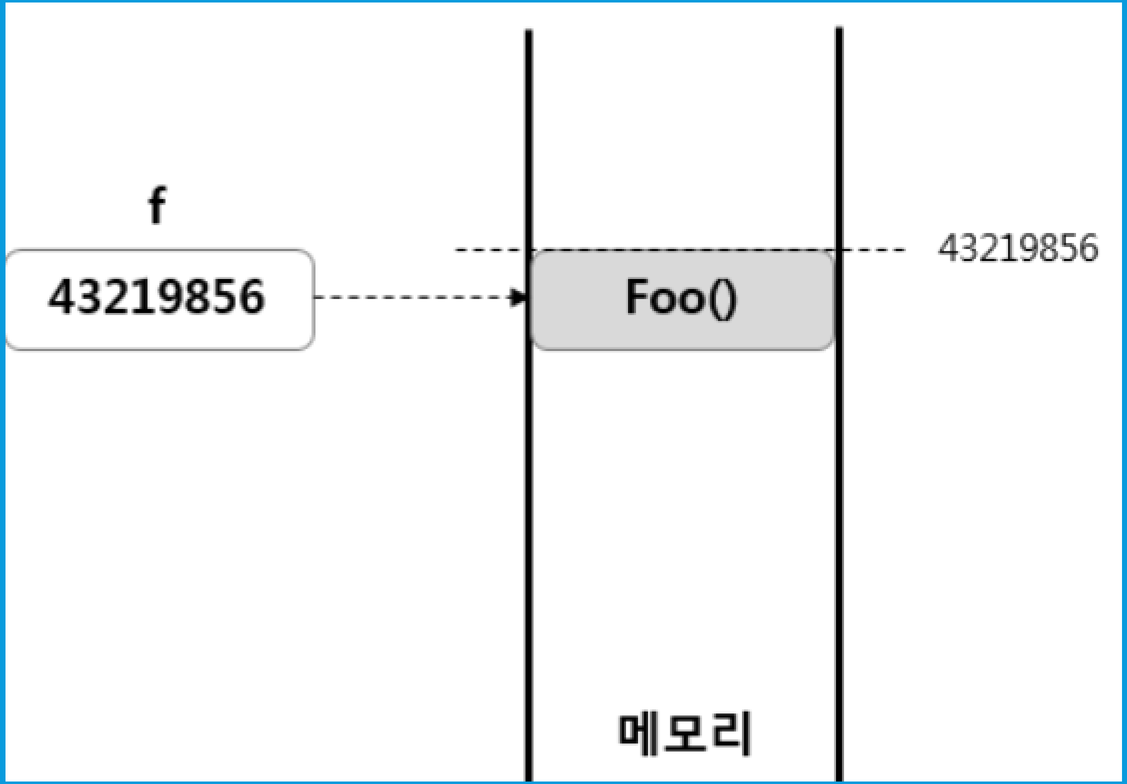
The example below defines a new function to calculate the sum of two values and calls the function with two arguments.

```
4
{'last_letter': 'k'}
```

```
# Sum function
def mysum(x, y):
   return x + y
# Test sum function
result = mysum(1, 3)
print(result)

#Classify text function
def gender_features(word):
      return {'last_letter': word[-1]}
print(gender_features('Shrek'))
```

CLASS

```
class Foo:
        def func1():
                print("function 1")
        def func2(self):
                print(id(self))
                print("function 2")
>>>f = Foo()
>>>id(f)
43219856
```



BASIC – NUMPY CRASH COURSE

#Create Array

NumPy provides the foundation data structures and operations for SciPy. These are arrays(ndarrays) that are efficient to define and manipulate.

```
# define an array
import numpy
mylist = [1, 2, 3]
myarray = numpy.array(mylist)
print(myarray)
print(myarray.shape)
```

```
[1 2 3]
(3,)
```

BASIC – NUMPY CRASH COURSE

```
import numpy as np
narray=np.random.randn(7,5)
print(narray)
print(narray.shape)
```

```
[[-1.47681546  0.36759776 -1.81672664 -0.10873707  0.38206153]
 [-0.70324524 -0.39033639  0.49244305 -0.82453668 -0.39475457]
 [ 0.43889658 -0.86885583  0.15669599  0.87395767  1.01840542]
 [-0.78801537 -0.41610522 -0.78012021  1.63591824  1.28867333]
 [ 0.36339421  0.30189617  0.18593078 -0.11172339  0.07997436]
 [-1.56539099  0.05538636 -1.0509418   0.65868237 -1.03506228]
 [ 0.37501026  0.36386668  1.10073997  0.04242472  0.21530997]]
(7, 5)
```

BASIC – NUMPY CRASH COURSE

```
import numpy as np
narray=np.arange(15)
print("create 1-dim matrix")
print(narray, "\n")

narray=narray.reshape((3,5))
print("matrix 3X5 reshape")
print(narray, "\n")

transpose_narray=narray.T
print("matrix transpose")
print(transpose_narray, "\n")

dot_array=np.dot(narray,transpose_narray)
print("matrix dot product")
print(dot_array)
```

```
create 1-dim matrix
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]

matrix 3X5 reshape
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]

matrix transpose
[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]

matrix dot product
[[ 30  80 130]
 [ 80 255 430]
 [130 430 730]]
```

BASIC – ACCESS DATA

Array notation and ranges can be used to efficiently access data in a NumPy array.

```
[[1 2 3]
 [3 4 5]]
(2, 3)
First row: [1 2 3]
Last row: [3 4 5]
Specific row and col: 3
Whole col: [3 5]
whole row: [3 4 5]
```

```
# access values
import numpy
mylist = [[1, 2, 3], [3, 4, 5]]
myarray = numpy.array(mylist)
print(myarray)
print(myarray.shape)
print("First row: %s"%myarray[0])
print("Last row: %s"%myarray[-1])
print("Specific row and col: %s"%myarray[0,2])
print("whole col: %s"%myarray[:,2])
print("whole row: %s"%myarray[1,:])
```

BASIC – ARITHMETIC

NumPy arrays can be used directly in arithmetic.

```
# arithmetic
import numpy
myarray1 = numpy.array([2, 2, 2])
myarray2 = numpy.array([3, 3, 3])
print("Addtion: %s"%(myarray1+myarray2))
print("Multiplication: %s"%(myarray1*myarray2))
```

```
Addition: [5 5 5]
Multiplication: [6 6 6]
```

FILE I/O

1.

```
f = open("C:/Python/newfile.txt", 'r’)
data = f.read()
print(data)
f.close()
```

read all of words

2.

```
with open(“newfile.txt", “r") as f:
    data=f.read()
    print(data)
```

tap

‘f’ is valid in this sectioin.

1.

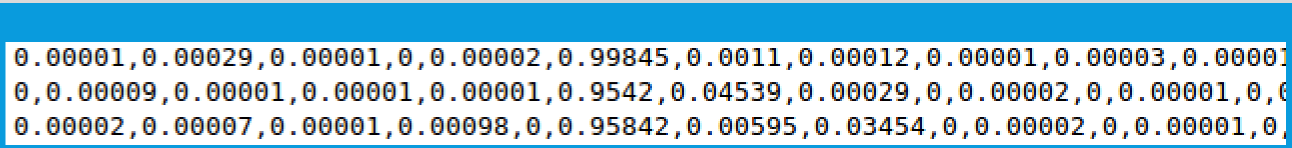
```
f = open("foo.txt", 'w’)
f.write("Life is too short, you need python")
f.close()
```

2.

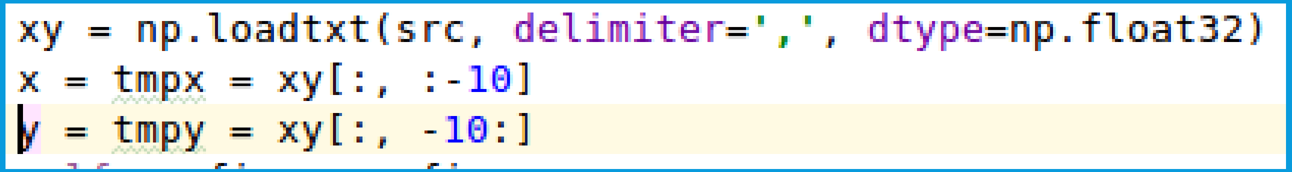
```
with open("foo.txt", "w") as f:
    f.write("Life is too short, you need python")
```

tap

FILE I/O -> .CSV , USING 'NUMPY' PACKAGE



```
0.00001,0.00029,0.00001,0,0.00002,0.99845,0.0011,0.00012,0.00001,0.00003,0.00001
0,0.00009,0.00001,0.00001,0.00001,0.9542,0.04539,0.00029,0,0.00002,0,0.00001,0,
0.00002,0.00007,0.00001,0.00098,0,0.95842,0.00595,0.03454,0,0.00002,0,0.00001,0,
```



```
xy = np.loadtxt(src, delimiter=',', dtype=np.float32)
x = tmpx = xy[:, :-10]
y = tmpy = xy[:, -10:]
```

package 'numpy' have good function to control .csv file.

can set delimiter so can load each by each.

RESULT

x = [all of rows and beginning to, before last 10 values]

y = [all of rows and 11th to end of columns]

FILE I/O -> .CSV , USING ‘PANDAS’ PACKAGE

```
# Load CSV using Pandas

from pandas import read_csv
```

from : package
import : function name

```
filename = 'pima-indians-diabetes.data.csv'

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass',
'pedi', 'age', 'class']

data = read_csv(filename, names=names)

print(data.shape)
```

RESULT

```
(768, 9)
```

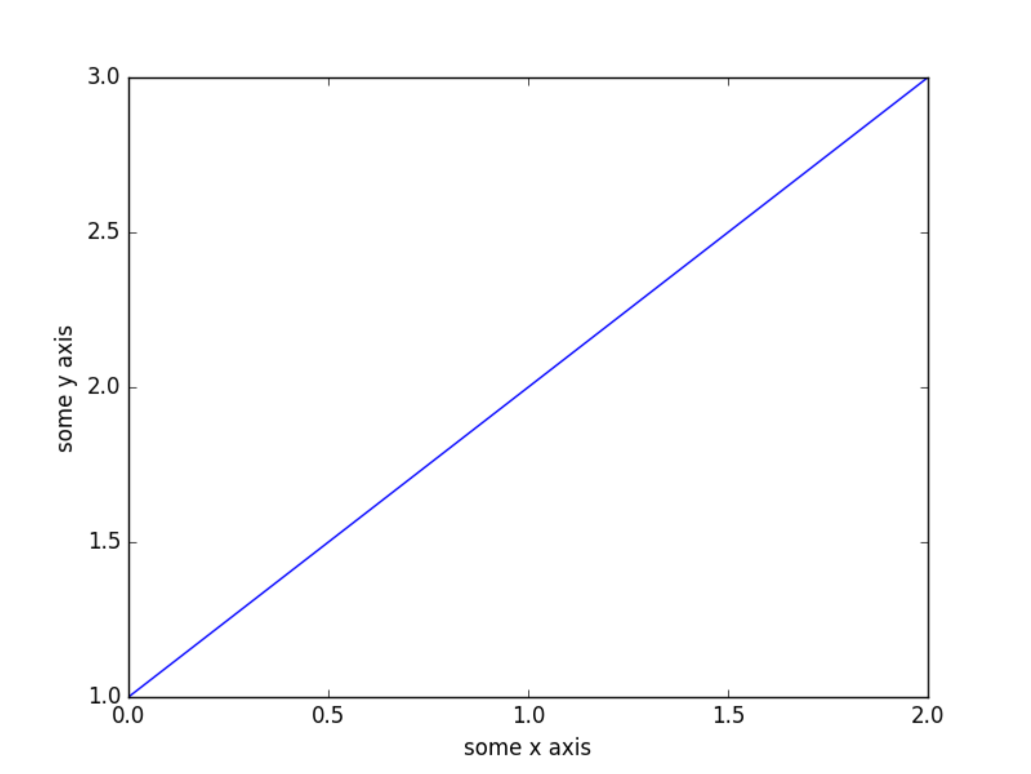
GRAPHICS

	description
distplot	histogram
barplot	estimate of central tendency for a numeric variable
violinplot	similar to boxplot, also shows the probability density of the data
jointplot	Scatterplot
regplot	Regression plot
pairplot	Pairplot
boxplot	boxplot
swarmplot	categorical scatterplot
factorplot	General categorical plot

PACKAGE ‘MATPLOT’

Install package ‘matplot’ = > ‘pip install matplotlib’

#Line Plot

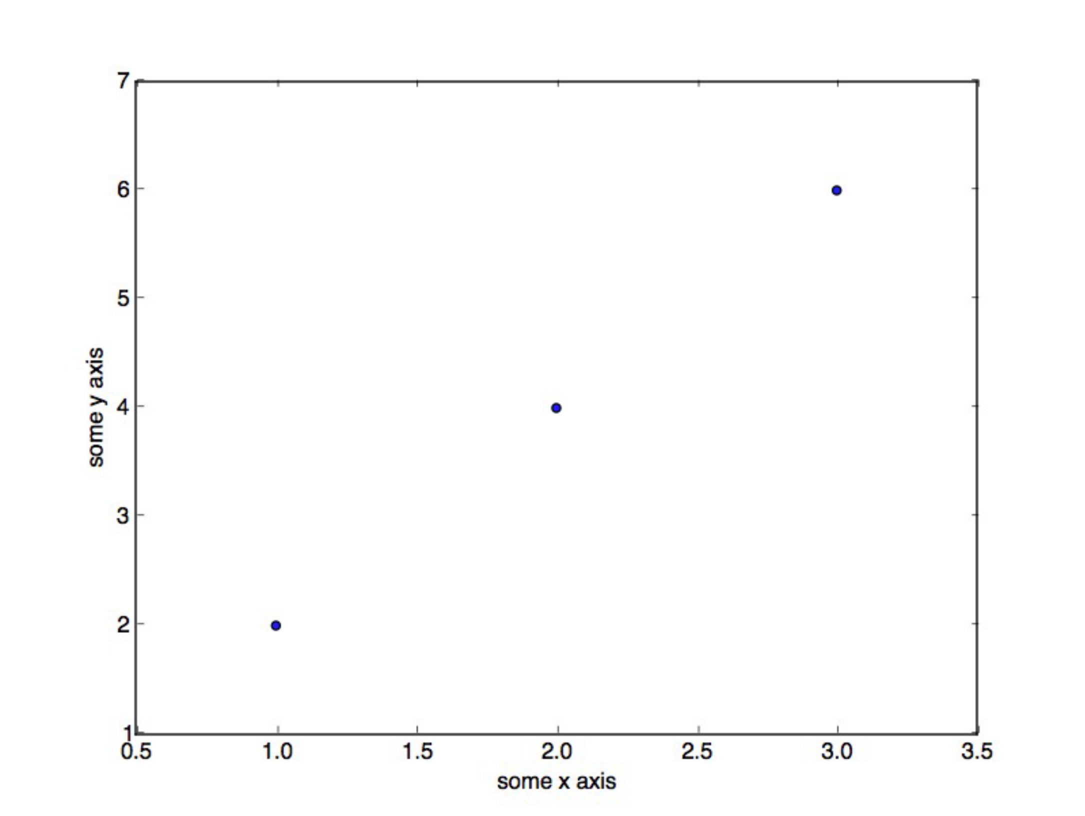


```
# basic line plot
import matplotlib.pyplot as plt
import numpy
myarray = numpy.array([1, 2, 3])
plt.plot(myarray)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```

Ref. https://matplotlib.org/users/pyplot_tutorial.html

PACKAGE ‘MATPLOT’

#Scatter Plot

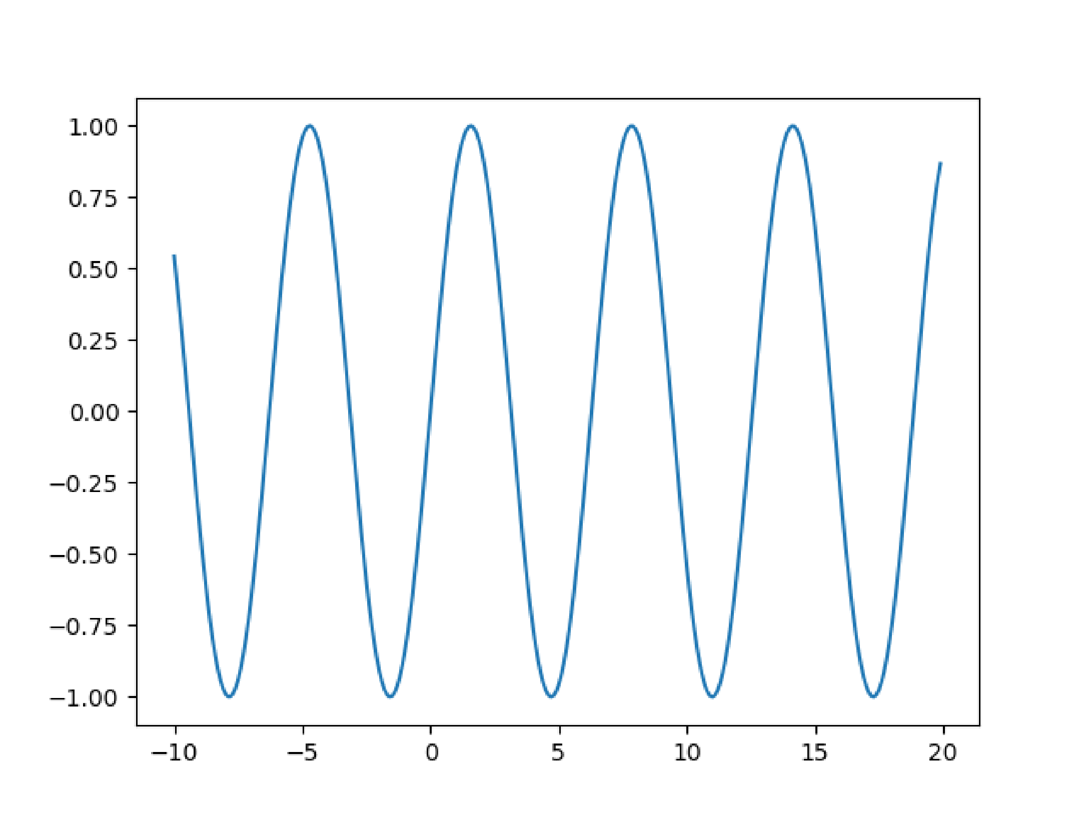


```
# basic scatter plot
import matplotlib.pyplot as plt
import numpy
x = numpy.array([1, 2, 3])
y = numpy.array([2, 4, 6])
plt.scatter(x,y)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```

Ref. https://matplotlib.org/users/pyplot_tutorial.html

PACKAGE 'MATPLOT'

#sin Plot



```
# sin plot
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-10, 20, 0.1)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```