# Python Fundamentals for Machine Learning

In [ ]:
```
# Author : Dr.Thyagaraju G S , Context Innovations Lab
# Date : 15/7/2018
```

# My First Program

In [2]:
```
# Program to find simple interest
p = int(input("\n Enter the principal Amount:"))
t = int(input("\n Enter the time period:"))
r = float(input("\n Enter the rate of interest:"))
si = p*t*r/100
print("\n Simple Interest:",si)
```

```
Enter the principal Amount:1000

Enter the time period:3

Enter the rate of interest:1.14

Simple Interest: 34.199999999999996
```

# 1.0 Output using Print

In [10]:
```
print('This sentence is output to the screen')
a=5
print("The value of a is:",a)
print(1,2,3,4)
x = 5 ; y = 10
print('The value of x is {} and y is {}'.format(x,y))
```

```
print('I love {0} and {1}'.format('bread','butter'))
print('I love {1} and {0}'.format('bread','butter'))
```

```
This sentence is output to the screen
The value of a is: 5
1 2 3 4
The value of x is 5 and y is 10
I love bread and butter
I love butter and bread
```

In [12]:
```
print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning',\
                                        name = 'John'))
```

```
Hello John, Goodmorning
```

In [13]:
```
x = 12.3456789
print('The value of x is %3.2f' %x)
print('The value of x is %3.4f' %x)
```

```
The value of x is 12.35
The value of x is 12.3457
```

In [17]:
```
for x in range(1, 11):
     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

```
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

In [20]:
```
table = {'Raju': 9480123526, 'Ravi': 9480123527, 'Rahul': 9480123527}
for name, phone in table.items():
     print('{0:10} ==> {1:10d}'.format(name, phone))
```

```
Raju        ==> 9480123526
Ravi        ==> 9480123527
Rahul       ==> 9480123527
```

In [19]:
```
import math
print('The value of PI is approximately %5.3f.' % math.pi)
```

```
The value of PI is approximately 3.142.
```

## 1.1 Input using input

In [22]:
```
x = input('Enter a string: ')
print("The enetered string is :",x)
y = int(input('Enter a integer: '))
print("The enetered integer is :",y)
z = float(input('Enter a string: '))
print("The enetered string is :",z)
```

```
Enter a string: abcde
The enetered string is : abcde
Enter a integer: 12345
The enetered integer is : 12345
Enter a string: 256.78
The enetered string is : 256.78
```

## 1.3 Mutiline Statements

In [24]:
```
# Example of implicit line continuation
x = ('1' + '2' +
'3' + '4')
# Example of explicit line continuation
y = '1' + '2' + \
'11' + '12'
weekdays = ['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday']
```

```
weekend = {'Saturday',
'Sunday'}
print ('x has a value of', x)
print ('y has a value of', y)
print(weekdays)
print (weekend)
```

```
x has a value of 1234
y has a value of 121112
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
{'Saturday', 'Sunday'}
```

In [25]:
```
import os; x = 'Hello'; print(x)
```

```
Hello
```

# 2.0 Conditional Execution

#Example code for a simple ‘if’ statement

In [26]:
```
var = -1
if var < 0:
    print(var)
    print("the value of var is negative")
# If there is only a signle cluse then it may go on the same line as th
e
#header statement
if ( var == -1 ) :
    print("the value of var is negative")
```

```
-1
the value of var is negative
the value of var is negative
```

#Example code for ‘if else’ statement

In [27]:
```
var = 1
if var < 0:
    print("the value of var is negative")
```

```
    print(var)
else:
    print("the value of var is positive")
    print(var)
```

```
the value of var is positive
1
```

# Example for nested if else

In [30]:
```
score = 95
if score >= 99:
    print('A')
elif score >=75:
    print('B')
elif score >= 60:
    print('C')
elif score >= 35:
    print('D')
else:
    print('F')
```

```
B
```

# 3.0 Iterations

# Usage of For Loop

In [34]:
```
# First Example
print("First Example")
for item in [1,2,3,4,5]:
    print('item :', item)
# Second Example
print("Second Example")
letters = ['A', 'B', 'C']
for letter in letters:
    print(' First loop letter :', letter)
# Third Example - Iterating by sequency index
print("Third Example")
```

```
for index in range(len(letters)):
    print('First loop letter :', letters[index])
# Fourth Example - Using else statement
print("Fourth Example")
```

```
First Example
item : 1
item : 2
item : 3
item : 4
item : 5
Second Example
 First loop letter : A
 First loop letter : B
 First loop letter : C
Third Example
First loop letter : A
First loop letter : B
First loop letter : C
Fourth Example
```

#While loop: The while statement repeats a set of code until the condition is true.

In [35]:
```
#Example code for while loop statement
count = 0
while (count <3):
    print('The count is:', count)
    count = count + 1
```

```
The count is: 0
The count is: 1
The count is: 2
```

In [36]:
```
#Example code for a ‘while with a else’ statement
count = 0
while count <3:
    print(count, " is less than 5")
    count = count + 1
else:
    print(count, " is not less than 5")
```

```
0  is less than 5
1  is less than 5
2  is less than 5
3  is not less than 5
```

## 4.1 LISTS

Python’s lists are the most flexible data type. It can be created by writing a list of commaseparated values between square brackets. Note that that the items in the list need not be of the same data type.

In [62]:
```
# Example code for accessing lists
# Create lists
list_1 = ['Statistics', 'Programming', 2016, 2017, 2018];
list_2 = ['a', 'b', 1, 2, 3, 4, 5, 6, 7 ];
# Accessing values in lists
print("list_1[0]: ", list_1[0])
print("list2_[1:5]: ", list_2[1:5])
```

```
list_1[0]:  Statistics
list2_[1:5]:  ['b', 1, 2, 3]
```

In [63]:
```
#Example code for adding new values to lists
print("list_1 values: ", list_1)
# Adding new value to list
list_1.append(2019)
print("list_1 values post append: ", list_1)
```

```
list_1 values:  ['Statistics', 'Programming', 2016, 2017, 2018]
list_1 values post append:  ['Statistics', 'Programming', 2016, 2017, 2
018, 2019]
```

In [64]:
```
#Example code for updating existing values of lists
print("Values of list_1: ", list_1)
# Updating existing value of list
print("Index 2 value : ", list_1[2])
```

```
list_1[2] = 2015;
print("Index 2's new value : ", list_1[2])
```

```
Values of list_1:  ['Statistics', 'Programming', 2016, 2017, 2018, 201
9]
Index 2 value :  2016
Index 2's new value :  2015
```

In [65]:

```
#Example code for deleting a list element
print("list_1 values: ", list_1)
# Deleting list element
del list_1[5];
print("After deleting value at index 2 : ", list_1)
```

```
list_1 values:  ['Statistics', 'Programming', 2015, 2017, 2018, 2019]
After deleting value at index 2 :  ['Statistics', 'Programming', 2015,
2017, 2018]
```

# Example code for basic operations on lists

In [66]:

```
import math
import string
import operator
#Example code for basic operations on lists

print("Length: ", len(list_1))

print("Concatenation: ", [1,2,3] + [4, 5, 6])

print("Repetition :", ['Hello'] * 4)

print("Membership :", 3 in [1,2,3])

print("Iteration :")
for x in [1,2,3]: print(x)

# Negative sign will count from the right
```

```python
print("slicing :", list_1[-2])

# If you dont specify the end explicitly, all elements from the specifi
ed
#start index will be printed
print("slicing range: ", list_1[1:])

print("Max of list: ", max([1,2,3,4,5]))

print("Min of list: ", min([1,2,3,4,5]))

print("Count number of 1 in list: ", [1,1,2,3,4,5,].count(1))

list_1.extend(list_2)

print("Extended :", list_1)

print("Index for Programming:",list_1.index('Programming'))
print (list_1)
print("pop last item in list: ", list_1.pop())
print("pop the item with index 2: ", list_1.pop(2))
list_1.remove('b')
print("removed b from list: ", list_1)
list_1.reverse()
print("Reverse: ", list_1)
list_1 = ['a','c','b']
list_1.sort()
print("Sort ascending: ", list_1)
list_1.sort(reverse = True)
print("Sort descending: ", list_1)
```

```
Length:  5
Concatenation:  [1, 2, 3, 4, 5, 6]
Repetition : ['Hello', 'Hello', 'Hello', 'Hello']
Membership : True
Iteration :
1
2
3
slicing : 2017
```

```
slicing range:  ['Programming', 2015, 2017, 2018]
Max of list:  5
Min of list:  1
Count number of 1 in list:  2
Extended : ['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1,
2, 3, 4, 5, 6, 7]
Index for Programming: 1
['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4,
5, 6, 7]
pop last item in list:  7
pop the item with index 2:  2015
removed b from list:  ['Statistics', 'Programming', 2017, 2018, 'a', 1,
2, 3, 4, 5, 6]
Reverse:  [6, 5, 4, 3, 2, 1, 'a', 2018, 2017, 'Programming', 'Statistic
s']
Sort ascending:  ['a', 'b', 'c']
Sort descending:  ['c', 'b', 'a']
```

## 4.2 Tuples

A Python tuple is a sequences or series of immutable Python objects very much similar to the lists. However there exist some essential differences between lists and tuples, which are the following.

1. Unlike list, the objects of tuples cannot be changed.
2. Tuples are defined by using parentheses, but lists are defined by square brackets

In [68]:
```
# Example code for creating tuple
# Creating a tuple
Tuple = ()
print("Empty Tuple: ", Tuple)
Tuple = (1,)
print("Tuple with single item: ", Tuple)
Tuple = ('a','b','c','d',1,2,3)
print("Sample Tuple :", Tuple)
```

```
Empty Tuple:  ()
Tuple with single item:  (1,)
Sample Tuple : ('a', 'b', 'c', 'd', 1, 2, 3)
```

In [69]:

```
#Example code for accessing tuple
# Accessing items in tuple
Tuple = ('a', 'b', 'c', 'd', 1, 2, 3)
print("3rd item of Tuple:", Tuple[2])
print("First 3 items of Tuple", Tuple[0:2])
```

```
3rd item of Tuple: c
First 3 items of Tuple ('a', 'b')
```

In [70]:

```
#Example code for deleting tuple
# Deleting tuple
print("Sample Tuple: ", Tuple)
del Tuple
print(Tuple) # Will throw an error message as the tuple does not exist
```

```
Sample Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
<ipython-input-70-ab07a54b7d35> in <module>()
      3 print("Sample Tuple: ", Tuple)
      4 del Tuple
----> 5 print(Tuple) # Will throw an error message as the tuple does no
t exist

NameError: name 'Tuple' is not defined
```

In [72]:

```
# Example code for basic operations on tupe (not exhaustive)
# Basic Tuple operations
Tuple = ('a','b','c','d',1,2,3)
print("Length of Tuple: ", len(Tuple))
Tuple_Concat = Tuple + (7,8,9)
print("Concatinated Tuple: ", Tuple_Concat)
```

```python
print("Repetition: ", (1,'a',2, 'b') * 3)
print("Membership check: ", 3 in (1,2,3))
# Iteration
for x in (1, 2, 3): print(x)
print("Negative sign will retrieve item from right: ", Tuple_Concat[-2
])
print("Sliced Tuple [2:] ", Tuple_Concat[2:])
# Find max
print("Max of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
max((1,2,3,4,5,6,7,8,9,10)))
print("Min of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
min((1,2,3,4,5,6,7,8,9,10)))
print("List [1,2,3,4] converted to tuple: ", type(tuple([1,2,3,4])))
```

```
Length of Tuple:  7
Concatinated Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3, 7, 8, 9)
Repetition:  (1, 'a', 2, 'b', 1, 'a', 2, 'b', 1, 'a', 2, 'b')
Membership check:  True
1
2
3
Negative sign will retrieve item from right:  8
Sliced Tuple [2:]  ('c', 'd', 1, 2, 3, 7, 8, 9)
Max of the Tuple (1,2,3,4,5,6,7,8,9,10):  10
Min of the Tuple (1,2,3,4,5,6,7,8,9,10):  1
List [1,2,3,4] converted to tuple:  <class 'tuple'>
```

## 4.3 Dictionary

The Python dictionary will have a key and value pair for each item that is part of it. The key and value should be enclosed in curly braces. Each key and value is separated using a colon (:), and further each item is separated by commas (,). Note that the keys are unique within a specific dictionary and must be immutable data types such as strings, numbers, or tuples, whereas values can take duplicate data of any type.

In [73]:
```python
# Example code for creating dictionary
# Creating dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
```

In [74]:
```python
# Example code for accessing dictionary
# Accessing items in dictionary
print("Value of key Name, from sample dictionary:", dict['Name'])
```

```
Value of key Name, from sample dictionary: Jivin
```

In [77]:
```python
#Example for deleting dictionary
# Deleting a dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict)
del (dict['Name']) # Delete specific item
print("Sample dictionary post deletion of item Name:", dict)
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
dict.clear() # Clear all the contents of dictionary
print("dict post dict.clear():", dict)
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
del (dict) # Delete the dictionary
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
Sample dictionary post deletion of item Name: {'Age': 6, 'Class': 'Firs
t'}
dict post dict.clear(): {}
```

In [78]:
```python
#Example code for updating dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict)
dict['Age'] = 6.5
print("Dictionary post age value update: ", dict)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
Dictionary post age value update:  {'Name': 'Jivin', 'Age': 6.5, 'Clas
s': 'First'}
```

In [86]:

```
#!/usr/bin/python
#Example code for basic operations on dictionary
# Basic operations
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Length of dict: ", len(dict))
dict1 = {'Name': 'Jivin', 'Age': 6};
dict2 = {'Name': 'Pratham', 'Age': 7};
dict3 = {'Name': 'Pranuth', 'Age': 7};
dict4 = {'Name': 'Jivin', 'Age': 6};

# String representation of dictionary
dict = {'Name': 'Jivin', 'Age': 6}
print("Equivalent String: ", str (dict))

# Copy the dict
dict1 = dict.copy()
print(dict1)
# Create new dictionary with keys from tuple and values to set value
seq = ('name', 'age', 'sex')
dict = dict.fromkeys(seq)
print("New Dictionary: ", str(dict))
dict = dict.fromkeys(seq, 10)
print("New Dictionary: ", str(dict))
# Retrieve value for a given key
dict = {'Name': 'Jivin', 'Age': 6};
print("Value for Age: ", dict.get('Age'))
# Since the key Education does not exist, the second argument will be
#returned
print("Value for Education: ", dict.get('Education', "First Grade"))
# Return items of dictionary
print("dict items: ", dict.items())
# Return items of keys
print("dict keys: ", dict.keys())
# return values of dict
print("Value of dict: ", dict.values())

# if key does not exists, then the arguments will be added to dict and
#returned
print("Value for Age : ", dict.setdefault('Age', None))
```

```
print("Value for Sex: ", dict.setdefault('Sex', None))

# Concatenate dicts
dict = {'Name': 'Jivin', 'Age': 6}
dict2 = {'Sex': 'male' }
dict.update(dict2)
print("dict.update(dict2) = ", dict)
```

```
Length of dict:  3
Equivalent String:  {'Name': 'Jivin', 'Age': 6}
{'Name': 'Jivin', 'Age': 6}
New Dictionary:  {'name': None, 'age': None, 'sex': None}
New Dictionary:  {'name': 10, 'age': 10, 'sex': 10}
Value for Age:  6
Value for Education:  First Grade
dict items:  dict_items([('Name', 'Jivin'), ('Age', 6)])
dict keys:  dict_keys(['Name', 'Age'])
Value of dict:  dict_values(['Jivin', 6])
Value for Age :  6
Value for Sex:  None
dict.update(dict2) =  {'Name': 'Jivin', 'Age': 6, 'Sex': 'male'}
```

# 5.0 User-Defined Functions

A user-defined function is a block of related code statements that are organized to achieve a single related action. The key objective of the user-defined functions concept is to encourage modularity and enable reusability of code.

Syntax for creating functions without argument: def functoin_name(): 1st block line 2nd block line ...

In [87]:
```
# Example code for creating functions without argument

# Simple function
def someFunction():
    print("Hello World")
```

```
# Call the function
someFunction()
```

```
Hello World
```

Syntax for Creating Functions with Argument def functoin_name(parameters): 1st block line 2nd block line ... return [expression]

In [88]:
```
#Example code for creating functions with arguments

# Simple function to add two numbers
def sum_two_numbers(x, y):
        return x + y

# after this line x will hold the value 3
print(sum_two_numbers(1,2))
```

```
3
```

Scope of Variables The availability of a variable or identifier within the program during and after the execution is determined by the scope of a variable. There are two fundamental variable scopes in Python. 1. Global variables 2. Local variables

In [89]:
```
#Example code for defining variable scopes
# Global variable
x = 10
# Simple function to add two numbers
def sum_two_numbers(y):
    return x + y
# Call the function and print result
print(sum_two_numbers(10))
```

```
20
```

In [92]:
```
#Variable Length Arguments
# Example code for passing argumens as *args
# Simple function to loop through arguments and print them

def sample_function(*args):
```

```
    for a in args:
        print(a)

# Call the function
sample_function(1,2,3)
```

```
1
2
3
```

In [93]:

```
#Example code for passing argumens as **kwargs
# Simple function to loop through arguments and print them

def sample_function(**kwargs):
    for a in kwargs:
        print(a, kwargs[a])
# Call the function
sample_function(name='John', age=27)
```

```
name John
age 27
```

Module A module is a logically organized multiple independent but related set of codes or functions or classes. The key principle behind module creating is it's easier to understand, use, and has efficient maintainability. You can import a module and the Python interpreter will search for the module in interest in the following sequences.

1. Currently active directly, that is, the directory from which the Python your program is being called.
2. If the module isn't found in currently active directory, Python then searches each directory in the path variable PYTHONPATH. If this fails then it searches in the default package installation path.

#Example code for importing modules # Import all functions from a module import module_name from modname import* # Import specific function from module from module_name import function_name

In [96]:

```
import os
content = dir(os)
```

```python
print(content)
```

```
['DirEntry', 'F_OK', 'MutableMapping', 'O_APPEND', 'O_BINARY', 'O_CREA
T', 'O_EXCL', 'O_NOINHERIT', 'O_RANDOM', 'O_RDONLY', 'O_RDWR', 'O_SEQUE
NTIAL', 'O_SHORT_LIVED', 'O_TEMPORARY', 'O_TEXT', 'O_TRUNC', 'O_WRONL
Y', 'P_DETACH', 'P_NOWAIT', 'P_NOWAITO', 'P_OVERLAY', 'P_WAIT', 'PathLi
ke', 'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'TMP_MAX', 'W_OK', 'X_
OK', '_Environ', '__all__', '__builtins__', '__cached__', '__doc__', '_
_file__', '__loader__', '__name__', '__package__', '__spec__', '_execvp
e', '_exists', '_exit', '_fspath', '_get_exports_list', '_putenv', '_un
setenv', '_wrap_close', 'abc', 'abort', 'access', 'altsep', 'chdir', 'c
hmod', 'close', 'closerange', 'cpu_count', 'curdir', 'defpath', 'device
_encoding', 'devnull', 'dup', 'dup2', 'environ', 'errno', 'error', 'exe
cl', 'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp', 'execv
pe', 'extsep', 'fdopen', 'fsdecode', 'fsencode', 'fspath', 'fstat', 'fs
ync', 'ftruncate', 'get_exec_path', 'get_handle_inheritable', 'get_inhe
ritable', 'get_terminal_size', 'getcwd', 'getcwdb', 'getenv', 'getlogi
n', 'getpid', 'getppid', 'isatty', 'kill', 'linesep', 'link', 'listdi
r', 'lseek', 'lstat', 'makedirs', 'mkdir', 'name', 'open', 'pardir', 'p
ath', 'pathsep', 'pipe', 'popen', 'putenv', 'read', 'readlink', 'remov
e', 'removedirs', 'rename', 'renames', 'replace', 'rmdir', 'scandir',
'sep', 'set_handle_inheritable', 'set_inheritable', 'spawnl', 'spawnl
e', 'spawnv', 'spawnve', 'st', 'startfile', 'stat', 'stat_float_times',
'stat_result', 'statvfs_result', 'strerror', 'supports_bytes_environ',
'supports_dir_fd', 'supports_effective_ids', 'supports_fd', 'supports_f
ollow_symlinks', 'symlink', 'sys', 'system', 'terminal_size', 'times',
'times_result', 'truncate', 'umask', 'uname_result', 'unlink', 'urando
m', 'utime', 'waitpid', 'walk', 'write']
```

# 6.0 Machine Learning Python Packages

Machine Learning is a collection of algorithms and techniques used to create computational systems that learn from data in order to make predictions and inferences. AI Process Loop : [ • Observe – identify patterns using the data • Plan – find all possible solutions • Optimize – find optimal solution from the list of possible solutions • Action – execute the optimal solution • Learn and Adapt – is the result giving expected result, if no adapt ] ML Process Loop : [There are six major phases : • Business understanding • Data understanding • Data preparation • Modeling • Evaluation • Deployment ] There is a rich number of open source libraries available to facilitate practical machine learning. These are mainly known as scientific Python libraries and are generally put to use when performing elementary machine learning tasks. At a high level we can divide these libraries into data analysis and core machine learning libraries based on their usage/purpose.

Data analysis packages: These are the sets of packages that provide us the mathematic and scientific functionalities that are essential to perform data preprocessing and transformation. Core Machine learning packages: These are the set of packages that provide us with all the necessary machine learning algorithms and functionalities that can be applied on a given dataset to extract the patterns.

# 6.1: Data Analysis Packages

There are four key packages that are most widely used for data analysis. • NumPy • SciPy • Matplotlib • Pandas

# 6.1.1 : NumPy

NumPy is the core library for scientific computing in Python. It provides a highperformance multidimensional array object, and tools for working with these array

In [1]:
```
#Example code for initializing NumPy array
import numpy as np
# Create a rank 1 array
a = np.array([0, 1, 2])
print(type(a))
# this will print the dimension of the array
print(a.shape)
print(a[0])
print(a[1])
print(a[2])
```

```
# Change an element of the array
a[0] = 5
print(a)
```

```
<class 'numpy.ndarray'>
(3,)
0
1
2
[5 1 2]
```

In [2]:
```
# Create a rank 2 array
b = np.array([[0,1,2],[3,4,5]])
print(b.shape)
print(b)
print(b[0, 0], b[0, 1], b[1, 0])
```

```
(2, 3)
[[0 1 2]
 [3 4 5]]
0 1 3
```

In [3]:
```
# Create a 3x3 array of all zeros
a = np.zeros((3,3))
print(a)
```

```
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
```

In [4]:
```
# Create a 2x2 array of all ones
b = np.ones((2,2))
print(b)
```

```
[[ 1.  1.]
 [ 1.  1.]]
```

In [5]:
```
# Create a 3x3 constant array
c = np.full((3,3), 7)
```

```
print(c)
```

```
[[7 7 7]
 [7 7 7]
 [7 7 7]]
```

In [6]:
```
# Create a 3x3 array filled with random values
d = np.random.random((3,3))
print(d)
```

```
[[ 0.06608011  0.82570932  0.27457936]
 [ 0.14677725  0.22876304  0.00868927]
 [ 0.00649678  0.74876722  0.4325642 ]]
```

In [7]:
```
# Create a 3x3 identity matrix
e = np.eye(3)
print(e)
```

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

In [8]:
```
# convert list to array
f = np.array([2, 3, 1, 0])
print(f)
```

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

In [9]:
```
# arange() will create arrays with regularly incrementing values
g = np.arange(20)
print(g)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

In [10]:
```
# note mix of tuple and lists
h = np.array([[0, 1,2.0],[0,0,0],(1+1j,3.,2.)])
print(h)
```

```
[[ 0.+0.j  1.+0.j  2.+0.j]
```

```
 [ 0.+0.j  0.+0.j  0.+0.j]
 [ 1.+1.j  3.+0.j  2.+0.j]]
```

In [11]:
```
# create an array of range with float data type
i = np.arange(1, 8, dtype=np.float)
print(i)
```

```
[ 1.  2.  3.  4.  5.  6.  7.]
```

In [12]:
```
# linspace() will create arrays with a specified number of items which
 are
# spaced equally between the specified beginning and end values
j = np.linspace(2., 4., 5)
print(j)
```

```
[ 2.   2.5  3.   3.5  4. ]
```

In [13]:
```
# indices() will create a set of arrays stacked as a one-higher
# dimensioned array, one per dimension with each representing variation
# in that dimension
k = np.indices((2,2))
print(k)
```

```
[[[0 0]
  [1 1]]

 [[0 1]
  [0 1]]]
```

In [14]:
```
#NumPy datatypes
# Let numpy choose the datatype
x = np.array([0, 1])
y = np.array([2.0, 3.0])
# Force a particular datatype
z = np.array([5, 6], dtype=np.int64)
print(x.dtype, y.dtype, z.dtype)
```

```
int32 float64 int64
```

In [15]:
```
#Field access
x = np.zeros((3,3), dtype=[('a', np.int32), ('b', np.float64, (3,3))])
print("x['a'].shape: ",x['a'].shape)
print("x['a'].dtype: ", x['a'].dtype)
print("x['b'].shape: ", x['b'].shape)
print("x['b'].dtype: ", x['b'].dtype)
```

```
x['a'].shape:  (3, 3)
x['a'].dtype:  int32
x['b'].shape:  (3, 3, 3, 3)
x['b'].dtype:  float64
```

In [18]:
```
# Basic slicing : The basic slice syntax is i: j: k,
# where i is the starting index,j is the stopping index,
# and k is the step and k is not equal to 0.
x = np.array([5, 6, 7, 8, 9])
print(x[1:7:2])
print(x[-2:5])
print(x[-1:1:-1])
```

```
[6 8]
[8 9]
[9 8 7]
```

In [21]:
```
#Boolean array indexing
a=np.array([[1,2], [3, 4], [5, 6]])
# Find the elements of a that are bigger than 2
print (a > 2)
# to get the actual value
print (a[a > 2])
```

```
[[False False]
 [ True  True]
 [ True  True]]
[3 4 5 6]
```

In [25]:
```
import numpy as np
x=np.array([[1,2],[3,4],[5,6]])
y=np.array([[7,8],[9,10],[11,12]])
```

```
# Elementwise sum; both produce the array
print(x+y)
print(np.add(x, y))
# Elementwise difference; both produce the array
print(x-y)
print(np.subtract(x, y))
```

```
[[ 8 10]
 [12 14]
 [16 18]]
[[ 8 10]
 [12 14]
 [16 18]]
[[-6 -6]
 [-6 -6]
 [-6 -6]]
[[-6 -6]
 [-6 -6]
 [-6 -6]]
```

In [26]:

```
# Elementwise product; both produce the array
print(x*y)
print(np.multiply(x, y))
```

```
[[ 7 16]
 [27 40]
 [55 72]]
[[ 7 16]
 [27 40]
 [55 72]]
```

In [27]:

```
print(x/y)
print(np.divide(x, y))
```

```
[[ 0.14285714  0.25      ]
 [ 0.33333333  0.4       ]
 [ 0.45454545  0.5       ]]
[[ 0.14285714  0.25      ]
```

```
 [ 0.33333333  0.4        ]
 [ 0.45454545  0.5        ]]
```

In [29]:
```
print(np.sqrt(x))
```

```
[[ 1.          1.41421356]
 [ 1.73205081  2.        ]
 [ 2.23606798  2.44948974]]
```

In [31]:
```
x=np.array([[1,2],[3,4]])
y=np.array([[5,6],[7,8]])
a=np.array([9,10])
b=np.array([11, 12])
# Inner product of vectors; both produce 219
print(a.dot(b))
print(np.dot(a, b))
```

```
219
219
```

In [33]:
```
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(a))
print(np.dot(x, a))
```

```
[29 67]
[29 67]
```

In [34]:
```
# Matrix / matrix product; both produce the rank 2 array
print(x.dot(y))
print(np.dot(x, y))
```

```
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
```

In [35]:
```
# Sum function
x=np.array([[1,2],[3,4]])
```

```
# Compute sum of all elements
print (np.sum(x))
# Compute sum of each column
print (np.sum(x, axis=0))
# Compute sum of each row
print (np.sum(x, axis=1))
```

```
10
[4 6]
[3 7]
```

In [36]:

```
#Transpose function
x=np.array([[1,2], [3,4]])
print(x)
print(x.T)
```

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

In [37]:

```
# Note that taking the transpose of a rank 1 array does nothing:
v=np.array([1,2,3])
print(v)
print(v.T)
```

```
[1 2 3]
[1 2 3]
```

Broadcasting : Broadcasting enables arithmetic operations to be performed between different shaped arrays

In [39]:

```
# Broadcasting
# create a matrix
a = np.array([[1,2,3], [4,5,6], [7,8,9]])
# create a vector
v = np.array([1, 0, 1])
# create an empty matrix with the same shape as a
b = np.empty_like(a)
# Add the vector v to each row of the matrix x with an explicit loop
for i in range(3):
```

```
    b[i, :] = a[i, :] + v
print(b)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]]
```

In [40]:

```
# Broadcasting for large matrix
# Stack 3 copies of v on top of each other
vv = np.tile(v, (3, 1))
print(vv)
# Add a and vv elementwise
b = a + vv
print(b)
```

```
[[1 0 1]
 [1 0 1]
 [1 0 1]]
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]]
```

In [41]:

```
# Broadcasting using NumPy
a = np.array([[1,2,3], [4,5,6], [7,8,9]])
v = np.array([1, 0, 1])
# Add v to each row of a using broadcasting
b = a + v
print(b)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]]
```

In [42]:

```
# Compute outer product of vectors
# v has shape (3,)
v = np.array([1,2,3])
# w has shape (2,)
w = np.array([4,5])
# To compute an outer product, we first reshape v to be a column
```

```
# vector of shape (3, 1); we can then broadcast it against w to yield
# an output of shape (3, 2), which is the outer product of v and w:
print(np.reshape(v, (3, 1)) * w)
```

```
[[ 4  5]
 [ 8 10]
 [12 15]]
```

In [43]:

```
# Add a vector to each row of a matrix
x = np.array([[1,2,3],[4,5,6]])
# x has shape (2, 3) and v has shape (3,) so they broadcast to (2, 3)
print(x + v)
```

```
[[2 4 6]
 [5 7 9]]
```

In [44]:

```
# Add a vector to each column of a matrix
# x has shape (2, 3) and w has shape (2,).
# If we transpose x then it has shape (3, 2) and can be broadcast
# against w to yield a result of shape (3, 2); transposing this result
# yields the final result of shape (2, 3) which is the matrix x with
# the vector w added to each column
print((x.T + w).T)
```

```
[[ 5  6  7]
 [ 9 10 11]]
```

In [46]:

```
# Another solution is to reshape w to be a row vector of shape (2, 1);
# we can then broadcast it directly against x to produce the same
# output.
print(x + np.reshape(w,(2,1)))
```

```
[[ 5  6  7]
 [ 9 10 11]]
```

In [45]:

```
# Multiply a matrix by a constant:
# x has shape (2, 3). Numpy treats scalars as arrays of shape ();
# these can be broadcast together to shape (2, 3)
print(x * 2)
```

```
[[ 2  4  6]
 [ 8 10 12]]
```

# 6.1.2 : PANDAS

In [48]:
```
import pandas as pd
```

Pandas are an open source Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. Pandas are well suited for tabular data with heterogeneously typed columns, as in an SQL table or Excel spreadsheet

Data Structures:Pandas introduces two new data structures to Python – Series and DataFrame, both of which are built on top of NumPy (this means it’s fast).

1.Series : This is a one-dimensional object similar to column in a spreadsheet or SQL table. By default each item will be assigned an index label from 0 to N.

In [49]:
```
#Creating a pandas series
# creating a series by passing a list of values, and a custom index lab
el.
#Note that the labeled index reference for each row and it can have
#duplicate values
s = pd.Series([1,2,3,np.nan,5,6], index=['A','B','C','D','E','F'])
print(s)
```

```
A    1.0
B    2.0
C    3.0
D    NaN
E    5.0
F    6.0
dtype: float64
```

2.DataFrame : It is a two-dimensional object similar to a spreadsheet or an SQL table. This is the most commonly used pandas object

In [50]:
```
#Creating a pandas dataframe
data = {'Gender': ['F', 'M', 'M'],'Emp_ID': ['E01', 'E02',
'E03'], 'Age': [25, 27, 25]}
# We want the order the columns, so lets specify in columns parameter
df = pd.DataFrame(data, columns=['Emp_ID','Gender', 'Age'])
df
```

Out[50]:

| | Emp_ID | Gender | Age |
|---|---|---|---|
| **0** | E01 | F | 25 |
| **1** | E02 | M | 27 |
| **2** | E03 | M | 25 |

# Reading and Writing Data

In [59]:
```
#Reading / writing data from csv, text, Excel

# Reading frome csv
df=pd.read_csv('C:/Users/Dr.Thyagaraju/Desktop/Data/PF/CSV/CHS2.csv')
print("READ CSV:\n",df)
#Writing to csv
df.to_csv('C:/Users/Dr.Thyagaraju/Desktop/Data/PF/CSV/CHS0.csv', index=
False)
print("WRITE CSV:\n",df)

# Reading from text Files
df=pd.read_csv('C:/Users/Dr.Thyagaraju/Desktop/Data/PF/TEXT/ex.txt', se
p='\t') # from text file
print("READ TXT:\n",df)
#Writing to text Files
df.to_csv('C:/Users/Dr.Thyagaraju/Desktop/Data/PF/TEXT/ex0.txt', sep='
\t', index=False)
print("WRITE TXT:\n",df)
```

```
#Reading from Excel File
df=pd.read_excel('C:/Users/Dr.Thyagaraju/Desktop/Data/PF/EXCEL/Heart_Pa
tient.xlsx','Sheet1') # from Excel
print("READ EXCEL:\n",df)

#Writing to Excel File
df.to_excel('C:/Users/Dr.Thyagaraju/Desktop/Data/PF/EXCEL/Heart_Patient
0.xlsx',sheet_name='Sheet1', index = False)
print("WRITE EXCEL:\n",df)

# reading from multiple sheets of same Excel into different dataframes
xlsx = pd.ExcelFile('C:/Users/Dr.Thyagaraju/Desktop/Data/PF/EXCEL/Heart
_Patient.xlsx')
sheet1_df = pd.read_excel(xlsx, 'Sheet1')
print("EXCEL SHEET1:\n",sheet1_df)
sheet2_df = pd.read_excel(xlsx, 'Sheet2')
print("EXCEL SHEET2:\n",sheet2_df)

# writing
# index = False parameter will not write the index values, default is T
rue
```

```
READ CSV:
     Gender  Height (in)
0     Male           72
1     Male           72
2   Female           63
3   Female           62
4   Female           62
5     Male           73
6   Female           64
7   Female           63
8   Female           67
9     Male           71
10    Male           72
11  Female           63
12    Male           71
13  Female           67
14  Female           62
```

```
15  Female          63
16    Male          66
17  Female          60
18  Female          68
19  Female          65
20  Female          64
WRITE CSV:
     Gender  Height (in)
0     Male          72
1     Male          72
2   Female          63
3   Female          62
4   Female          62
5     Male          73
6   Female          64
7   Female          63
8   Female          67
9     Male          71
10    Male          72
11  Female          63
12    Male          71
13  Female          67
14  Female          62
15  Female          63
16    Male          66
17  Female          60
18  Female          68
19  Female          65
20  Female          64
READ TXT:
 Empty DataFrame
Columns: [ 0.00632  18.00   2.310  0  0.5380   ]
Index: []
WRITE TXT:
 Empty DataFrame
Columns: [ 0.00632  18.00   2.310  0  0.5380   ]
Index: []
READ EXCEL:
     ATS     SSM   HBP   FH  ADM     O
```

```
0   Yes  Abnorm  High  Yes  Yes  0.94
1   Yes  Abnorm  High  Yes   No  0.93
2   Yes  Abnorm  High   No  Yes  0.92
3   Yes  Abnorm  High   No   No  0.91
4   Yes  Abnorm  Norm  Yes   No  0.84
5   Yes  Abnorm  Norm  Yes   No  0.82
6   Yes  Abnorm  Norm   No  Yes  0.80
7   Yes  Abnorm  Norm   No   No  0.78
8   Yes    Norm  High  Yes  Yes  0.91
9   Yes    Norm  High  Yes   No  0.91
10  Yes    Norm  High   No  Yes  0.90
11  Yes    Norm  High   No   No  0.89
12  Yes    Norm  Norm  Yes  Yes  0.80
13  Yes    Norm  Norm  Yes   No  0.78
14  Yes    Norm  Norm   No  Yes  0.76
15   No  Abnorm  High  Yes  Yes  0.79
16   No  Abnorm  High  Yes   No  0.77
WRITE EXCEL:
     ATS     SSM   HBP   FH  ADM     O
0   Yes  Abnorm  High  Yes  Yes  0.94
1   Yes  Abnorm  High  Yes   No  0.93
2   Yes  Abnorm  High   No  Yes  0.92
3   Yes  Abnorm  High   No   No  0.91
4   Yes  Abnorm  Norm  Yes   No  0.84
5   Yes  Abnorm  Norm  Yes   No  0.82
6   Yes  Abnorm  Norm   No  Yes  0.80
7   Yes  Abnorm  Norm   No   No  0.78
8   Yes    Norm  High  Yes  Yes  0.91
9   Yes    Norm  High  Yes   No  0.91
10  Yes    Norm  High   No  Yes  0.90
11  Yes    Norm  High   No   No  0.89
12  Yes    Norm  Norm  Yes  Yes  0.80
13  Yes    Norm  Norm  Yes   No  0.78
14  Yes    Norm  Norm   No  Yes  0.76
15   No  Abnorm  High  Yes  Yes  0.79
16   No  Abnorm  High  Yes   No  0.77
EXCEL SHEET1:
     ATS     SSM   HBP   FH  ADM     O
0   Yes  Abnorm  High  Yes  Yes  0.94
```

```
1   Yes  Abnorm  High  Yes   No  0.93
2   Yes  Abnorm  High   No  Yes  0.92
3   Yes  Abnorm  High   No   No  0.91
4   Yes  Abnorm  Norm  Yes   No  0.84
5   Yes  Abnorm  Norm  Yes   No  0.82
6   Yes  Abnorm  Norm   No  Yes  0.80
7   Yes  Abnorm  Norm   No   No  0.78
8   Yes    Norm  High  Yes  Yes  0.91
9   Yes    Norm  High  Yes   No  0.91
10  Yes    Norm  High   No  Yes  0.90
11  Yes    Norm  High   No   No  0.89
12  Yes    Norm  Norm  Yes  Yes  0.80
13  Yes    Norm  Norm  Yes   No  0.78
14  Yes    Norm  Norm   No  Yes  0.76
15   No  Abnorm  High  Yes  Yes  0.79
16   No  Abnorm  High  Yes   No  0.77
EXCEL SHEET2:
     ATS     SSM   HBP
0   Yes  Abnorm  High
1   Yes  Abnorm  High
2   Yes  Abnorm  High
3   Yes  Abnorm  High
4   Yes  Abnorm  Norm
5   Yes  Abnorm  Norm
6   Yes  Abnorm  Norm
7   Yes  Abnorm  Norm
8   Yes    Norm  High
9   Yes    Norm  High
10  Yes    Norm  High
11  Yes    Norm  High
12  Yes    Norm  Norm
13  Yes    Norm  Norm
14  Yes    Norm  Norm
15   No  Abnorm  High
16   No  Abnorm  High
```

## Loading Data From URL

In [64]:
```
# Load CSV from URL using NumPy
from numpy import loadtxt
from urllib.request import urlopen
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima
-indians-diabetes.data.csv'
raw_data = pd.read_csv(urlopen(url))
print(raw_data.shape)
print(raw_data.head())
```

```
(767, 9)
   6  148  72  35    0  33.6  0.627  50  1
0  1   85  66  29    0  26.6  0.351  31  0
1  8  183  64   0    0  23.3  0.672  32  1
2  1   89  66  23   94  28.1  0.167  21  0
3  0  137  40  35  168  43.1  2.288  33  1
4  5  116  74   0    0  25.6  0.201  30  0
```

# Loading Data From Library

In [85]:
```
from sklearn.datasets import load_iris
import numpy as np
iris=load_iris()
#iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
#X.head()
print("Shape:\n",X.shape)
print("Head:\n",X.head(5))
print("Tail:\n",X.tail(5))
```

```
Shape:
 (150, 4)
Head:
    Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
```

```
4          5.0          3.6          1.4          0.2
Tail:
     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
145           6.7          3.0           5.2          2.3
146           6.3          2.5           5.0          1.9
147           6.5          3.0           5.2          2.0
148           6.2          3.4           5.4          2.3
149           5.9          3.0           5.1          1.8
```

# Basic Statistics Summary

Pandas has some built-in functions to help us to get better understanding of data using basic statistical summary methods describe()- will returns the quick stats such as count, mean, std (standard deviation), min, first quartile, median, third quartile, max on each column of the dataframe

In [86]:

```
df = pd.DataFrame(iris.data)
df.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'
]
df.describe()
```

Out[86]:

| | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width |
|---|---|---|---|---|
| **count** | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| **mean** | 5.843333 | 3.054000 | 3.758667 | 1.198667 |
| **std** | 0.828066 | 0.433594 | 1.764420 | 0.763161 |
| **min** | 4.300000 | 2.000000 | 1.000000 | 0.100000 |
| **25%** | 5.100000 | 2.800000 | 1.600000 | 0.300000 |
| **50%** | 5.800000 | 3.000000 | 4.350000 | 1.300000 |
| **75%** | 6.400000 | 3.300000 | 5.100000 | 1.800000 |

| | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width |
|---|---|---|---|---|
| **max** | 7.900000 | 4.400000 | 6.900000 | 2.500000 |

cov() - Covariance indicates how two variables are related. A positive covariance means the variables are positively related, while a negative covariance means the variables are inversely related. Drawback of covariance is that it does not tell you the degree of positive or negative relation

In [87]:
```
df.cov()
```

Out[87]:

| | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width |
|---|---|---|---|---|
| **Sepal_Length** | 0.685694 | -0.039268 | 1.273682 | 0.516904 |
| **Sepal_Width** | -0.039268 | 0.188004 | -0.321713 | -0.117981 |
| **Petal_Length** | 1.273682 | -0.321713 | 3.113179 | 1.296387 |
| **Petal_Width** | 0.516904 | -0.117981 | 1.296387 | 0.582414 |

corr() - Correlation is another way to determine how two variables are related. In addition to telling you whether variables are positively or inversely related, correlation also tells you the degree to which the variables tend to move together. When you say that two items correlate, you are saying that the change in one item effects a change in another item. You will always talk about correlation as a range between -1 and 1. In the below example code, petal length is 87% positively related to sepal length that means a change in petal length results in a positive 87% change to sepal lenth and vice versa.

In [88]:
```
df.corr()
```

Out[88]:

| | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width |
|---|---|---|---|---|
| **Sepal_Length** | 1.000000 | -0.109369 | 0.871754 | 0.817954 |

| | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width |
|---|---|---|---|---|
| **Sepal_Width** | -0.109369 | 1.000000 | -0.420516 | -0.356544 |
| **Petal_Length** | 0.871754 | -0.420516 | 1.000000 | 0.962757 |
| **Petal_Width** | 0.817954 | -0.356544 | 0.962757 | 1.000000 |

# Grouping

Grouping involves one or more of the following steps: • Splitting the data into groups based on some criteria, • Applying a function to each group independently, • Combining the results into a data structure

In [90]:

```
#Grouping operation
df = pd.DataFrame({'Name' : ['jack', 'jane', 'jack', 'jane', 'jack', 'j
ane',
'jack', 'jane'],'State' : ['SFO', 'SFO', 'NYK', 'CA', 'NYK', 'NYK','SF
O', 'CA'],
'Grade':['A','A','B','A','C','B','C','A'],
'Age' : np.random.uniform(24, 50, size=8),
'Salary' : np.random.uniform(3000, 5000, size=8),})
# Note that the columns are ordered automatically in their alphabetic o
rder
print(df)
# for custom order please use below code
# df = pd.DataFrame(data, columns = ['Name', 'State', 'Age','Salary'])
# Find max age and salary by Name / State
# with groupby, we can use all aggregate functions such as min, max, me
an,
#count, cumsum
df.groupby(['Name','State']).max()
```

```
         Age Grade  Name       Salary State
0  47.760931     A  jack  3858.576000   SFO
1  26.938656     A  jane  4223.853733   SFO
2  30.178551     B  jack  4847.967235   NYK
```

```
3  28.017794     A  jane  3584.773734   CA
4  32.545801     C  jack  4599.097730  NYK
5  31.020030     B  jane  3080.166423  NYK
6  47.966642     C  jack  3535.443511  SFO
7  48.614870     A  jane  4391.099980   CA
```

Out[90]:

| | | Age | Grade | Salary |
|---|---|---|---|---|
| **Name** | **State** | | | |
| **jack** | **NYK** | 32.545801 | C | 4847.967235 |
| | **SFO** | 47.966642 | C | 3858.576000 |
| **jane** | **CA** | 48.614870 | A | 4391.099980 |
| | **NYK** | 31.020030 | B | 3080.166423 |
| | **SFO** | 26.938656 | A | 4223.853733 |

Pivot Tables Pandas provides a function 'pivot_table' to create MS-Excel spreadsheet style pivot tables. It can take following arguments: • data: DataFrame object, • values: column to aggregate, • index: row labels, • columns: column labels, • aggfunc: aggregation function to be used on values, default is NumPy.mean

In [91]:

```
# by state and name find mean age for each grade
pd.pivot_table(df, values='Age', index=['State', 'Name'], columns=['Gra
de'])
```

Out[91]:

| | Grade | A | B | C |
|---|---|---|---|---|
| **State** | **Name** | | | |
| **CA** | **jane** | 38.316332 | NaN | NaN |
| **NYK** | **jack** | NaN | 30.178551 | 32.545801 |
| | **jane** | NaN | 31.020030 | NaN |

| | Grade | A | B | C |
|---|---|---|---|---|
| State | Name | | | |
| SFO | jack | 47.760931 | NaN | 47.966642 |
| | jane | 26.938656 | NaN | NaN |

# 7.0 Matplotlib

In [97]:

```
import matplotlib.pyplot as plt
```

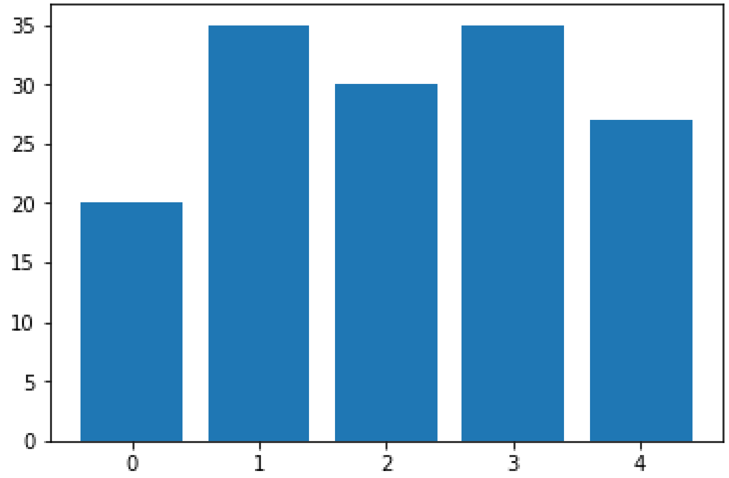
Matplotlib is a numerical mathematics extension NumPy and a great package to view or present data in a pictorial or graphical format. It enables analysts and decision makers to see analytics presented visually, so they can grasp difficult concepts or identify new patterns. There are two broad ways of using pyplot.

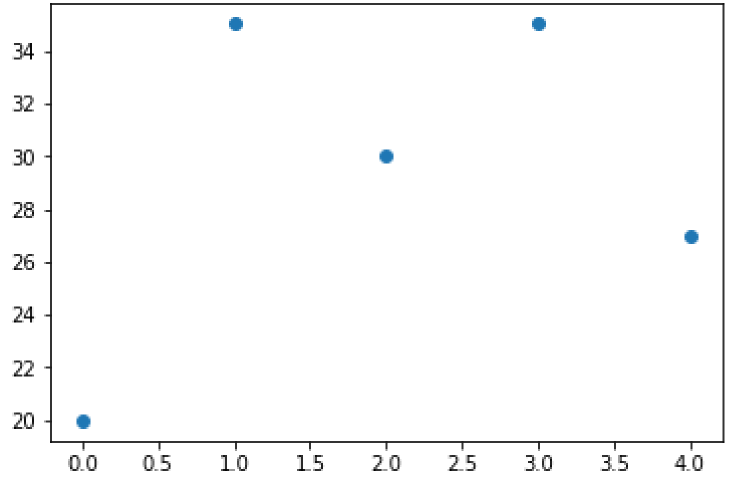
1. Using Global Functions The most common and easy approach is by using global functions to build and display a global figure using matplotlib as a global state machine. Let's look at some of the most commonly used charts. Then see Listing 2-31. • plt.bar – creates a bar chart • plt.scatter – makes a scatter plot • plt.boxplot – makes a box and whisker plot • plt.hist – makes a histogram • plt.plot – creates a line plot

In [98]:

```
# Creating plot on variables
# simple bar and scatter plot

x = np.arange(5) # assume there are 5 students
y = (20, 35, 30, 35, 27) # their test scores
plt.bar(x,y) # Bar plot

# need to close the figure using show() or close(), if not closed
# any follow plot commands will use same figure.

plt.show() # Try commenting this an run
```

```
plt.scatter(x,y) # scatter plot
plt.show()
```



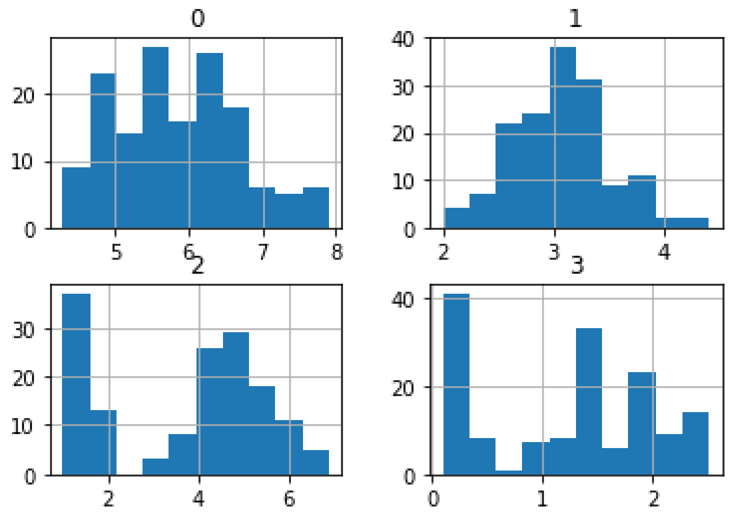




In [104]:

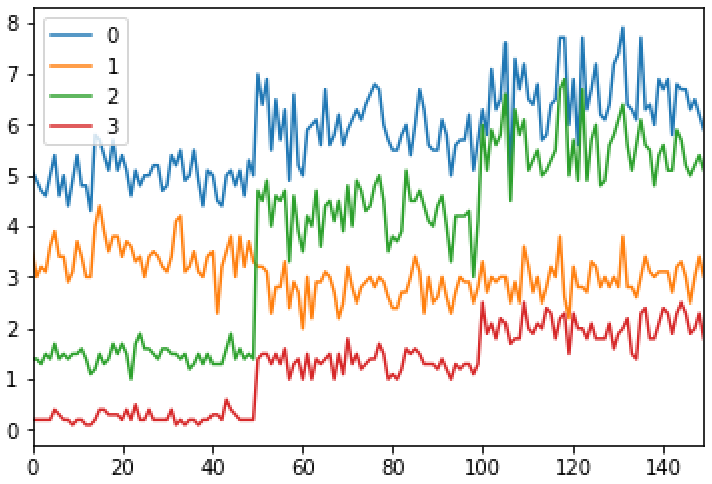
```
#Creating plot on dataframe df = pd.read_csv('Data/iris.csv')
# Read sample data
df = pd.DataFrame(iris.data)
#df.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Widt
h']
```

```
print("Histogram:\n")
df.hist()# Histogram
plt.show()
print("Line Graph:\n")
df.plot() # Line Graph
plt.show()
print("Box Plot:\n")
df.boxplot() # Box plot
plt.show()
```

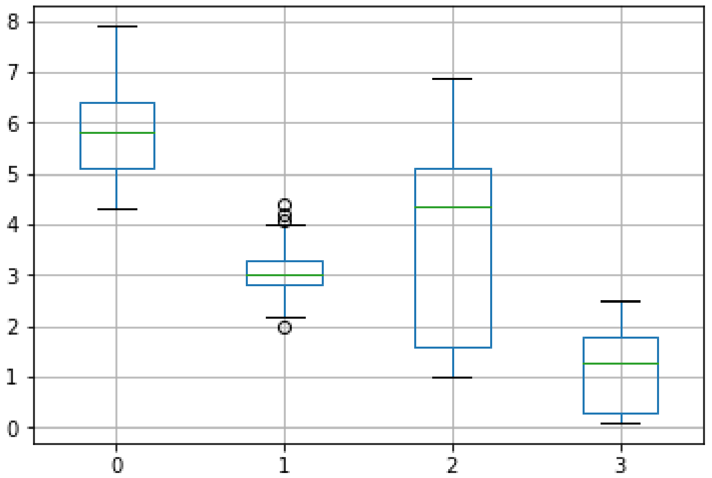
Histogram:




Line Graph:



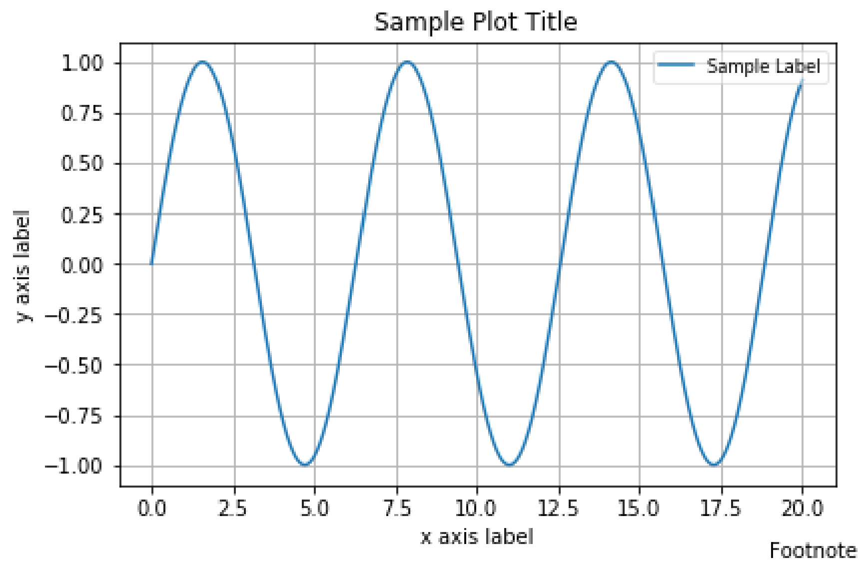

```
Box Plot:
```



# Customizing Labels

In [107]:
```
#Customize labels
```

```
# generate sample data
x = np.linspace(0, 20, 1000) #100 evenly-spaced values from 0 to 50
y = np.sin(x)

# customize axis labels
plt.plot(x, y, label = 'Sample Label')
plt.title('Sample Plot Title') # chart title
plt.xlabel('x axis label') # x axis title
plt.ylabel('y axis label') # y axis title
plt.grid(True) # show gridlines

# add footnote
plt.figtext(0.995, 0.01, 'Footnote', ha='right', va='bottom')

# add legend, location pick the best automatically
plt.legend(loc='best', framealpha=0.5, prop={'size':'small'})

# tight_layout() can take keyword arguments of pad, w_pad and h_pad.
# these control the extra padding around the figure border and between
#subplots. The pads are specified in fraction of fontsize.
plt.tight_layout(pad=1)

# Saving chart to a file
#plt.savefig('filename.png')
#plt.close()
# Close the current window to allow new plot creation on
#separate window / axis, alternatively we can use show()
plt.show()
```




## 8.0 Machine Learning Libraries

In [130]:
```
# Python version
import sys
print('Python: {}'.format(sys.version))
# scipy
import scipy
print('scipy: {}'.format(scipy.__version__))
# numpy
import numpy
print('numpy: {}'.format(numpy.__version__))
# matplotlib
import matplotlib
print('matplotlib: {}'.format(matplotlib.__version__))
# pandas
import pandas
print('pandas: {}'.format(pandas.__version__))
# scikit-learn
```

```
import sklearn
print('sklearn: {}'.format(sklearn.__version__))
import seaborn
print('seaborn: {}'.format(seaborn.__version__))
import pgmpy
print('pgmpy: {}'.format(pgmpy.__name__))
import urllib
print('urlib: {}'.format(urllib.__name__))
import csv
print('csv: {}'.format(csv.__version__))
```

```
Python: 3.6.3 |Anaconda custom (32-bit)| (default, Oct 15 2017, 07:29:1
6) [MSC v.1900 32 bit (Intel)]
scipy: 0.19.1
numpy: 1.13.3
matplotlib: 2.1.0
pandas: 0.20.3
sklearn: 0.19.1
seaborn: 0.8.0
pgmpy: pgmpy
urlib: urllib
csv: 1.0
```

## 8.1 Scipy

SciPy, pronounced as Sigh Pi, is a scientific python open source, distributed under the BSD licensed library to perform Mathematical, Scientific and Engineering Computations. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays and provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install and are free of charge. NumPy and SciPy are easy to use, but powerful enough to depend on by some of the world's leading scientists and engineers.

In [131]:
```
import numpy as np
print(np.linspace(1., 4., 6))
```

```
[ 1.   1.6  2.2  2.8  3.4  4. ]
```

In [135]:

```
#K-Means Implementation in SciPy
from scipy.cluster.vq import kmeans,vq,whiten
from numpy import vstack,array
from numpy.random import rand

# data generation with three features
data = vstack((rand(100,3) + array([.5,.5,.5]),rand(100,3)))
#print(data)
# whitening of data for normalizing
data = whiten(data)
#print(data)
# computing K-Means with K = 3 (2 clusters)
centroids,_ = kmeans(data,3)
print("Centroids:\n",centroids)
# assign each sample to a cluster
clx,_ = vq(data,centroids)
print("Cluster:\n",clx)
```

```
Centroids:
 [[ 0.88885498  1.30111066  0.94727687]
 [ 2.70805616  2.87573485  2.839588  ]
 [ 2.03499214  1.43162913  2.00227741]]
Cluster:
 [1 2 2 1 2 2 2 1 1 1 1 1 2 1 1 2 1 2 1 1 2 1 1 1 1 1 1 2 2 1 2 2 1 1 1
2 1
 2 1 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1
1 1
 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 0 0 0 0 2 0 2 2
2 2
 0 0 0 0 0 0 0 2 0 0 0 0 2 0 2 2 2 2 2 0 0 0 0 0 0 2 2 2 2 0 0 0 0 0 2
2 0
 0 2 2 2 2 2 0 0 2 0 0 2 0 0 2 0 0 2 0 2 2 0 0 2 0 2 2 0 0 0 0 2 2 2 2
0 2
 0 2 0 0 2 2 1 2 0 0 2 0 0 0 0]
```

In [138]:

```
#Fast Fourier Transform
```

```python
#Importing the fft and inverse fft functions from fftpackage
from scipy.fftpack import fft

#create an array with random n numbers
x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])

#Applying the fft function
y = fft(x)
print("FFT :\n",y)

from scipy.fftpack import ifft
yinv = ifft(y)
print("FFT Inverse:\n",yinv)
```

```
FFT :
 [ 4.50000000+0.j          2.08155948-1.65109876j -1.83155948+1.6082204
1j
 -1.83155948-1.60822041j  2.08155948+1.65109876j]
FFT Inverse:
 [ 1.0+0.j  2.0+0.j  1.0+0.j -1.0+0.j  1.5+0.j]
```

In [142]:

```python
#Discrete Cosine Transform
from scipy.fftpack import dct
print ("DCT:\n",dct(np.array([4., 3., 5., 10., 5., 3.])))

#Inverse Discrete Cosine Transform
from scipy.fftpack import idct
print("IDCT:\n",idct(np.array([4., 3., 5., 10., 5., 3.])))
```

```
DCT:
 [ 60.          -3.48476592 -13.85640646  11.3137085    6.          -6.
31319305]
IDCT:
 [ 39.15085889 -20.14213562  -6.45392043   7.13341236   8.14213562
  -3.83035081]
```

## SciPy - Integrate

The general form of quad is scipy.integrate.quad(f, a, b), Where ‘f’ is the name of the function to be integrated. Whereas, ‘a’ and ‘b’ are the lower and upper limits, respectively. Let us see an example of the Gaussian function, integrated over a range of 0 and 1. f(x)=e−x2 ∫f(x)dx

In [143]:
```
# Single Integration
import scipy.integrate
from numpy import exp
f= lambda x:exp(-x**2)
i = scipy.integrate.quad(f, 0, 1)
print(i)
```

```
(0.7468241328124271, 8.291413475940725e-15)
```

Linear Algebra x + 3y + 5z = 10 2x + 5y + z = 8 2x + 3y + 8z = 3

In [145]:
```
#importing the scipy and numpy packages
from scipy import linalg
import numpy as np

#Declaring the numpy arrays
a = np.array([[1, 3, 5], [2, 5, 1], [2, 3, 8]])
b = np.array([10, 8, 3])

#Passing the values to the solve function
x = linalg.solve(a, b)

#printing the result array
print (x)
```

```
[-9.28  5.16  0.76]
```

## Finding a Determinant

In [146]:
```
#importing the scipy and numpy packages
from scipy import linalg
import numpy as np
```

```
#Declaring the numpy array
A = np.array([[1,2],[3,4]])

#Passing the values to the det function
x = linalg.det(A)

#printing the result
print (x)
```

```
-2.0
```

Eigenvalues and Eigenvectors

In [147]:

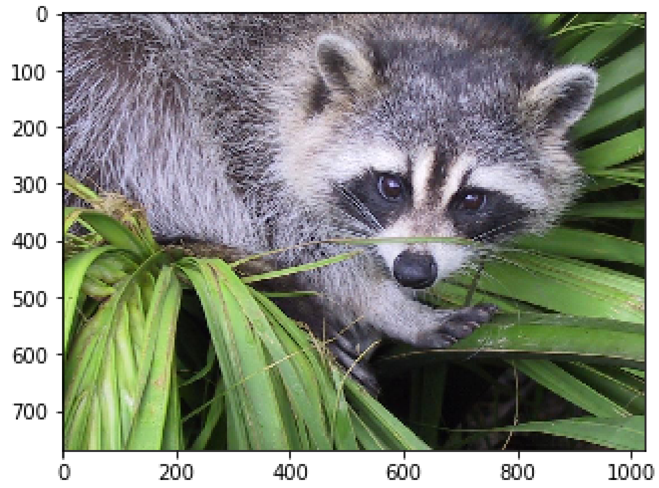
```
#importing the scipy and numpy packages
from scipy import linalg
import numpy as np

#Declaring the numpy array
A = np.array([[1,2],[3,4]])

#Passing the values to the eig function
l, v = linalg.eig(A)

#printing the result for eigen values
print ("Eigen Values :\n",l)

#printing the result for eigen vectors
print ("Eigen Vectors:\n",v)
```

```
Eigen Values :
 [-0.37228132+0.j  5.37228132+0.j]
Eigen Vectors:
 [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

# Image Processing

In [165]:

```
from scipy import misc
f = misc.face()
misc.imsave('face.png', f) # uses the Image module (PIL)

import matplotlib.pyplot as plt
plt.imshow(f)
plt.show()
```




In [166]:

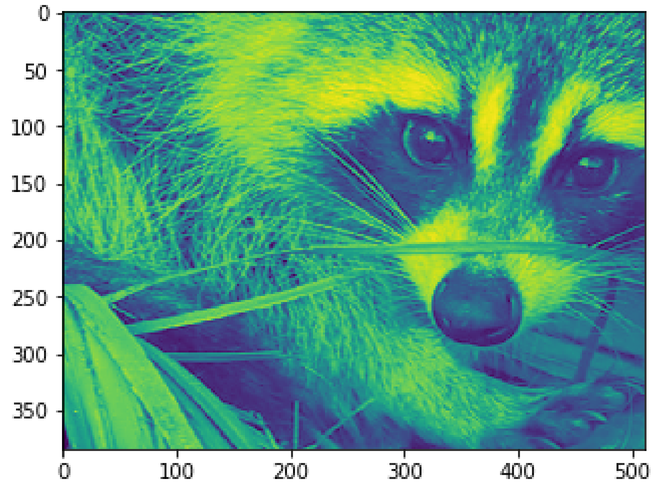
```
# Statistical Information of the image
from scipy import misc
face = misc.face(gray = False)
print(face.mean(), face.max(), face.min())
```

```
110.162743886 255 0
```

In [180]:

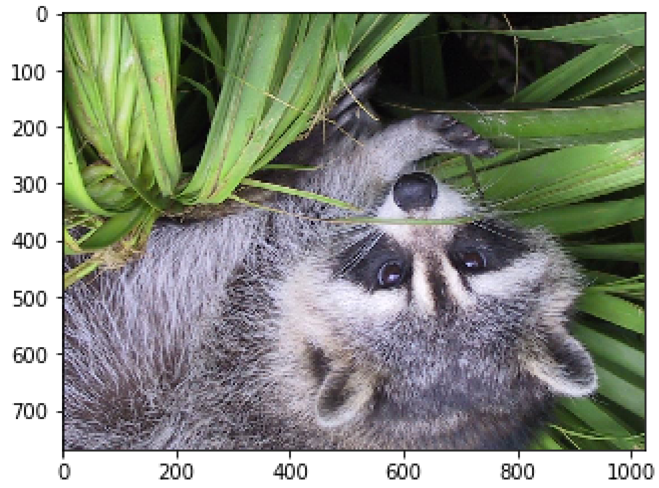
```
# Cropping
from scipy import misc
face = misc.face(gray = True)
lx,ly = face.shape
# Cropping
crop_face = face[lx//4 : -lx//4 , ly//4 : -ly//4]
import matplotlib.pyplot as plt
```

```
plt.imshow(crop_face)
plt.show()
```



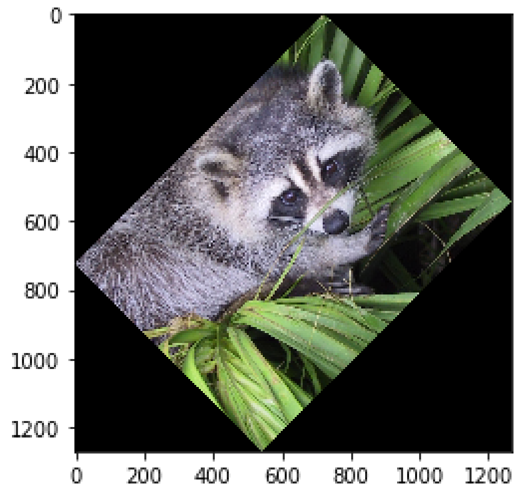
In [168]:

```
# up <-> down flip
from scipy import misc
face = misc.face()
flip_ud_face = np.flipud(face)

import matplotlib.pyplot as plt
plt.imshow(flip_ud_face)
plt.show()
```

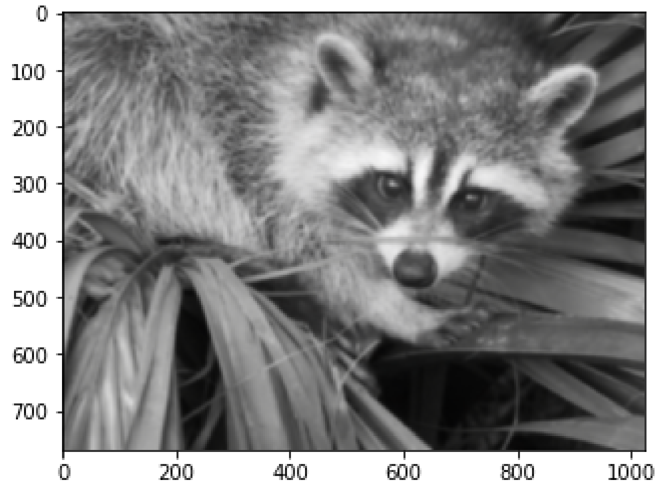



In [169]:

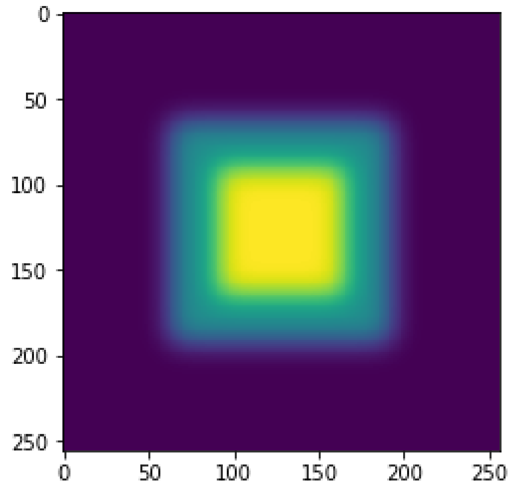
```
# rotation
from scipy import misc,ndimage
face = misc.face()
rotate_face = ndimage.rotate(face, 45)

import matplotlib.pyplot as plt
plt.imshow(rotate_face)
plt.show()
```



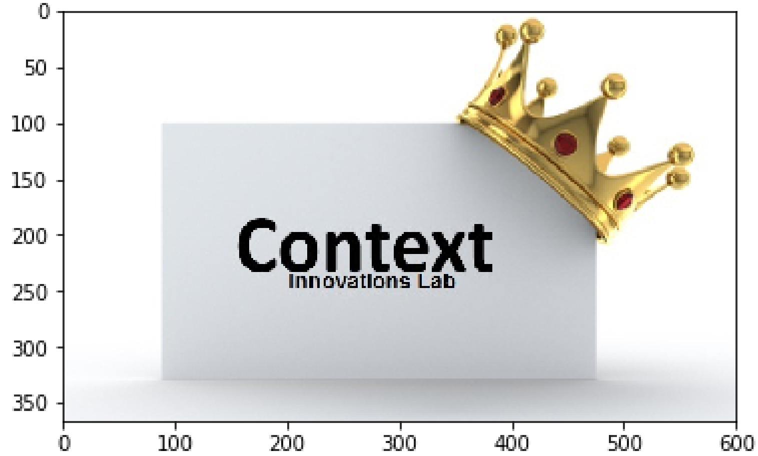
In [170]:
```
# Blurring
from scipy import misc
face = misc.face()
blurred_face = ndimage.gaussian_filter(face, sigma=3)
import matplotlib.pyplot as plt
plt.imshow(blurred_face)
plt.show()
```




In [171]:
```
# Edge Detection
import scipy.ndimage as nd
import numpy as np

im = np.zeros((256, 256))
im[64:-64, 64:-64] = 1
im[90:-90,90:-90] = 2
im = ndimage.gaussian_filter(im, 8)

import matplotlib.pyplot as plt
plt.imshow(im)
plt.show()
```




In [190]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img=mpimg.imread('C:/Users/Dr.Thyagaraju/Desktop/Data/Image/C11.png')
#print(img)
plt.imshow(img)
plt.show()
```



## 8.2 sklearn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes: NumPy: Base n-dimensional array package SciPy: Fundamental library for scientific computing Matplotlib: Comprehensive 2D/3D plotting IPython: Enhanced interactive console Sympy: Symbolic mathematics Pandas: Data structures and analysis

In [191]:
```
import sklearn
```

# Scikit Learn Loading Dataset

In [192]:
```
from sklearn import datasets
```

In [193]:
```
# Data sets available in sklearn
iris= datasets.load_iris()
houseprice = datasets.load_boston()
diabetes = datasets.load_diabetes()
digits = datasets.load_digits()
linerud= datasets.load_linnerud()
wine = datasets.load_wine()
breastcancer = datasets.load_breast_cancer()
```

In [206]:
```
print(digits.data[0])
```

```
[  0.   0.   5.  13.   9.   1.   0.   0.   0.   0.  13.  15.  10.  15.
   5.
   0.   0.   3.  15.   2.   0.  11.   8.   0.   0.   4.  12.   0.   0.
   8.
   8.   0.   0.   5.   8.   0.   0.   9.   8.   0.   0.   4.  11.   0.
   1.
  12.   7.   0.   0.   2.  14.   5.  10.  12.   0.   0.   0.   0.   6.
```

```
13.
  10.    0.    0.    0.]
```

In [207]:
```
print(houseprice.data[0])
```

```
[   6.32000000e-03   1.80000000e+01   2.31000000e+00   0.00000000e+00
    5.38000000e-01   6.57500000e+00   6.52000000e+01   4.09000000e+00
    1.00000000e+00   2.96000000e+02   1.53000000e+01   3.96900000e+02
    4.98000000e+00]
```

In [208]:
```
print(diabetes.data[0])
```

```
[ 0.03807591  0.05068012  0.06169621  0.02187235 -0.0442235  -0.0348207
6
 -0.04340085 -0.00259226  0.01990842 -0.01764613]
```

In [209]:
```
print(linerud.data[0])
```

```
[   5.  162.   60.]
```

In [210]:
```
print(wine.data[0])
```

```
[   1.42300000e+01   1.71000000e+00   2.43000000e+00   1.56000000e+01
    1.27000000e+02   2.80000000e+00   3.06000000e+00   2.80000000e-01
    2.29000000e+00   5.64000000e+00   1.04000000e+00   3.92000000e+00
    1.06500000e+03]
```

In [211]:
```
print(breastcancer.data[0])
```

```
[   1.79900000e+01   1.03800000e+01   1.22800000e+02   1.00100000e+03
    1.18400000e-01   2.77600000e-01   3.00100000e-01   1.47100000e-01
    2.41900000e-01   7.87100000e-02   1.09500000e+00   9.05300000e-01
    8.58900000e+00   1.53400000e+02   6.39900000e-03   4.90400000e-02
    5.37300000e-02   1.58700000e-02   3.00300000e-02   6.19300000e-03
    2.53800000e+01   1.73300000e+01   1.84600000e+02   2.01900000e+03
    1.62200000e-01   6.65600000e-01   7.11900000e-01   2.65400000e-01
    4.60100000e-01   1.18900000e-01]
```

In [195]:
```
# Print shape of data to confirm data is loaded
print("IRIS:\n",iris.data.shape)
print("HOUSEPRICE:\n",houseprice.data.shape)
print("DIABETES:\n",diabetes.data.shape)
print("DIGITS:\n",digits.data.shape)
print("LINERUD:\n",linerud.data.shape)
print("WINE:\n",wine.data.shape)
print("BREASTCANCER:\n",breastcancer.data.shape)
```

```
IRIS:
 (150, 4)
HOUSEPRICE:
 (506, 13)
DIABETES:
 (442, 10)
DIGITS:
 (1797, 64)
LINERUD:
 (20, 3)
WINE:
 (178, 13)
BREASTCANCER:
 (569, 30)
```

In [216]:
```
# see what’s available in iris:
iris.keys()
print("IRIS KEYS:\n",iris.keys())
n_samples, n_features = iris.data.shape
print ("IRIS # SAMPLES:\n",n_samples)
print ("IRIS # FEATURES:\n",n_features)
print ("IRIS FIRST FEW ROWS:\n",iris.data[0:10])
print("IRIS TARGETS NAMES",iris.target_names)
print("IRIS FEATURE NAMES",iris.feature_names)
print("IRIS TARGET",iris.target)
print("IRIS DESCR",iris.DESCR)
iris_X = iris.data
iris_y = iris.target
np.unique(iris_y)
```

```
IRIS KEYS:
```

```
 dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_name
s'])
IRIS # SAMPLES:
 150
IRIS # FEATURES:
 4
IRIS FIRST FEW ROWS:
 [[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]
 [ 5.4  3.9  1.7  0.4]
 [ 4.6  3.4  1.4  0.3]
 [ 5.   3.4  1.5  0.2]
 [ 4.4  2.9  1.4  0.2]
 [ 4.9  3.1  1.5  0.1]]
IRIS TARGETS NAMES ['setosa' 'versicolor' 'virginica']
IRIS FEATURE NAMES ['sepal length (cm)', 'sepal width (cm)', 'petal len
gth (cm)', 'petal width (cm)']
IRIS TARGET [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 2]
IRIS DESCR Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the cla
ss
    :Attribute Information:
```

```
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica
:Summary Statistics:

============== ==== ==== ======= ===== ====================
                Min  Max   Mean    SD   Class Correlation
============== ==== ==== ======= ===== ====================
sepal length:   4.3  7.9   5.84   0.83    0.7826
sepal width:    2.0  4.4   3.05   0.43   -0.4194
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
petal width:    0.1  2.5   1.20  0.76     0.9565  (high!)
============== ==== ==== ======= ===== ====================

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris

The famous Iris database, first used by Sir R.A Fisher

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the fie
ld and
is referenced frequently to this day.  (See Duda & Hart, for example.)
The
data set contains 3 classes of 50 instances each, where each class refe
rs to a
type of iris plant.  One class is linearly separable from the other 2;
the
```

```
latter are NOT linearly separable from each other.

References
----------
   - Fisher,R.A. "The use of multiple measurements in taxonomic problem
s"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contribution
s to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Ana
lysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New Syst
em
     Structure and Classification Rule for Recognition in Partially Exp
osed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Tran
sactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLAS
S II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

Out[216]:

```
array([0, 1, 2])
```

In [218]:

```python
# Split iris data in train and test data
# A random permutation, to split the data randomly
np.random.seed(0)
indices = np.random.permutation(len(iris_X))
iris_X_train = iris_X[indices[:-10]]
iris_y_train = iris_y[indices[:-10]]
iris_X_test = iris_X[indices[-10:]]
iris_y_test = iris_y[indices[-10:]]
# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier()
knn.fit(iris_X_train, iris_y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski'
,
metric_params=None, n_jobs=1, n_neighbors=5, p=2,
weights='uniform')
print("Predicted :\n",knn.predict(iris_X_test))
print("Actual:\n",iris_y_test)
```

```
Predicted :
 [1 2 1 0 0 0 2 1 2 0]
Actual:
 [1 1 1 0 0 0 2 1 2 0]
```

# Linear regression

LinearRegression, in its simplest form, fits a linear model to the data set by adjusting a set of parameters in order to make the sum of the squared residuals of the model as small as possible

In [219]:

```
diabetes = datasets.load_diabetes()
diabetes_X_train = diabetes.data[:-20]
diabetes_X_test = diabetes.data[-20:]
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

In [221]:

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(diabetes_X_train, diabetes_y_train)
print("Regression Coef:\n",regr.coef_)
print("Mean:\n",np.mean((regr.predict(diabetes_X_test)-diabetes_y_test)
**2))
# Explained variance score: 1 is perfect prediction
# and 0 means that there is no linear relationship
# between X and y.
regr.score(diabetes_X_test, diabetes_y_test)
```

```
Regression Coef:
 [  3.03499549e-01  -2.37639315e+02   5.10530605e+02   3.27736980e+02
  -8.14131709e+02   4.92814588e+02   1.02848452e+02   1.84606489e+02
   7.43519617e+02   7.60951722e+01]
Mean:
 2004.56760269
```

Out[221]: 0.58507530226905735

In [222]:

```python
# Sample Decision Tree Classifier
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=N
one,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
            splitter='best')
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        50
          1       1.00      1.00      1.00        50
          2       1.00      1.00      1.00        50
```

```
avg / total       1.00      1.00      1.00       150

[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
```

## 8.3 pgmpy

# Probabilistic Graphical Models using pgmpy

Probabilistic Graphical Model is a way of compactly representing Joint Probability distribution over random variables using the independence conditions of the variables

In [223]:

```
import pgmpy
```

In [234]:

```
# Generate data
import numpy as np
import pandas as pd

raw_data = np.array([0] * 30 + [1] * 70) # Representing heads by 0 and
 tails by 1
data = pd.DataFrame(raw_data, columns=['coin'])
print(data[25:35])
```

```
    coin
25     0
26     0
27     0
28     0
29     0
30     1
31     1
32     1
```

```
33      1
34      1
```

In [235]:

```
# Defining the Bayesian Model
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstima
tor

model = BayesianModel()
model.add_node('coin')

# Fitting the data to the model using Maximum Likelihood Estimator
model.fit(data, estimator=MaximumLikelihoodEstimator)
print(model.get_cpds('coin'))
```

| | |
|---|---|
| coin(0) | 0.3 |
| coin(1) | 0.7 |

In [236]:

```
# Fitting the data to the model using Bayesian Estimator with Dirichlet
 prior with equal pseudo counts.
model.fit(data, estimator=BayesianEstimator, prior_type='dirichlet', ps
eudo_counts={'coin': [50, 50]})
print(model.get_cpds('coin'))
```

```
WARNING:root:Replacing existing CPD for coin
```

| | |
|---|---|
| coin(0) | 0.4 |
| coin(1) | 0.6 |

We can see that we get the results as expected. In the maximum likelihood case we got the probability just based on the data where as in the bayesian case we had a prior of P(H) = 0.5 and P(T) = 0.5 , therefore with 30% heads and 70% tails in the data we got a posterior of P(H) = 0.4

and P(T) = 0.6 . Similarly we can learn in case of more complex model. Let's take an example of the student model and compare the results in case of Maximum Likelihood estimator and Bayesian Estimator.

In [238]:
```
# Generating radom data with each variable have 2 states and equal prob
abilities for each state
import numpy as np
import pandas as pd

raw_data = np.random.randint(low=0, high=2, size=(1000, 5))
data = pd.DataFrame(raw_data, columns=['D', 'I', 'G', 'L', 'S'])
print(data[0:10])
```

```
   D  I  G  L  S
0  1  1  0  0  1
1  1  1  1  1  0
2  1  0  0  1  1
3  1  1  1  0  0
4  1  0  0  1  0
5  0  0  0  1  1
6  1  1  1  0  1
7  0  0  1  1  1
8  1  1  0  0  0
9  1  0  1  0  1
```

In [239]:
```
# Defining the model
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstima
tor

model = BayesianModel([('D', 'G'), ('I', 'G'), ('I', 'S'), ('G', 'L')])

# Learing CPDs using Maximum Likelihood Estimators
model.fit(data, estimator=MaximumLikelihoodEstimator)
for cpd in model.get_cpds():
    print("CPD of {variable}:".format(variable=cpd.variable))
    print(cpd)
```

```
CPD of D:
```

| | |
|---|---|
| D(0) | 0.512 |
| D(1) | 0.488 |

CPD of G:

| D | D(0) | D(0) | D(1) | D(1) |
|---|---|---|---|---|
| I | I(0) | I(1) | I(0) | I(1) |
| G(0) | 0.497942386831275730 | 0.4275092936802974 | 0.4959677419354839 | 0.5541666666666667 |
| G(1) | 0.5020576131687243 | 0.5724907063197026 | 0.5040322580645161 | 0.44583333333333336 |

CPD of I:

| | |
|---|---|
| I(0) | 0.491 |
| I(1) | 0.509 |

CPD of L:

| G | G(0) | G(1) |
|---|---|---|
| L(0) | 0.5040650406504065 | 0.46653543307086615 |
| L(1) | 0.4959349593495935 | 0.5334645669291339 |

CPD of S:

| I | I(0) | I(1) |
|---|---|---|
| S(0) | 0.4623217922606925 | 0.5029469548133595 |
| S(1) | 0.5376782077393075 | 0.49705304518664045 |

As the data was randomly generated with equal probabilities for each state we can see here that all the probability values are close to 0.5 which we expected. Now coming to the Bayesian Estimator:

In [240]:
```
# Learning with Bayesian Estimator using dirichlet prior for each varia
ble.

pseudo_counts = {'D': [300, 700], 'I': [500, 500], 'G': [800, 200], 'L'
: [500, 500], 'S': [400, 600]}
model.fit(data, estimator=BayesianEstimator, prior_type='dirichlet', ps
eudo_counts=pseudo_counts)
for cpd in model.get_cpds():
    print("CPD of {variable}:".format(variable=cpd.variable))
    print(cpd)
```

```
WARNING:root:Replacing existing CPD for D
WARNING:root:Replacing existing CPD for G
WARNING:root:Replacing existing CPD for I
WARNING:root:Replacing existing CPD for S
WARNING:root:Replacing existing CPD for L
```

CPD of D:

| | |
|---|---|
| D(0) | 0.406 |
| D(1) | 0.594 |

CPD of G:

| D | D(0) | D(0) | D(1) |
|---|---|---|---|

D(1) |

| I | I(0) | I(1) | I(0) | I(1) |
|---|---|---|---|---|
| G(0) | 0.7409493161705552 | 0.7210401891252955 | 0.7395833333333334 | 0.7524193548387097 |
| G(1) | 0.2590506838294449 | 0.2789598108747045 | 0.2604166666666667 | 0.24758064516129033 |

CPD of I:

| | |
|---|---|
| I(0) | 0.4955 |
| I(1) | 0.5045 |

CPD of L:

| G | G(0) | G(1) |
|---|---|---|
| L(0) | 0.5013404825737265 | 0.4887267904509284 |
| L(1) | 0.49865951742627346 | 0.5112732095490716 |

CPD of S:

| I | I(0) | I(1) |
|---|---|---|
| S(0) | 0.42052313883299797 | 0.4347249834327369 |
| S(1) | 0.579476861167002 | 0.5652750165672631 |

Since the data was randomly generated with equal probabilities for each state, the data tries to bring the posterior probabilities close to 0.5. But because of the prior we will get the values in between the prior and 0.5.