Python Programming

COE

April 2022

Contents / Agenda

Python Basics,

Logic and Conditional Flow in Python ,

Dictionaries and Sets in Python,

Files and Input/Output ,

Errors and Exceptions ,

Functions and Modules ,

Classes and OOP ,

Python Library: NumPy,Python

Library: SciPy,

Python Library: matplotlib,

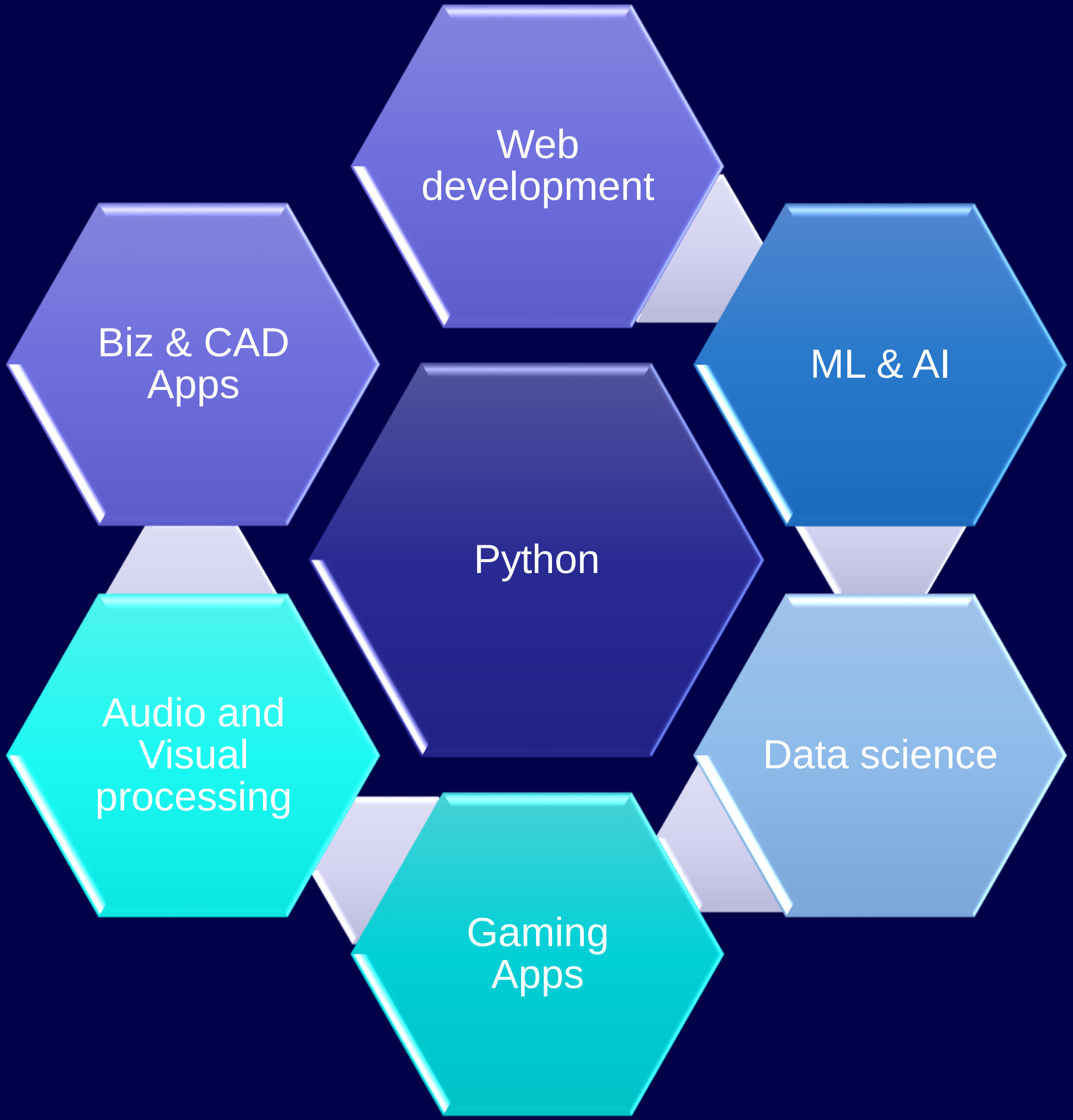
Python Library: Pandas,

Python Library: Seaborne

Origin

Image Credits: Mclek /Shutterstock

Python is ruling the World



Web development

Biz & CAD Apps

ML & AI

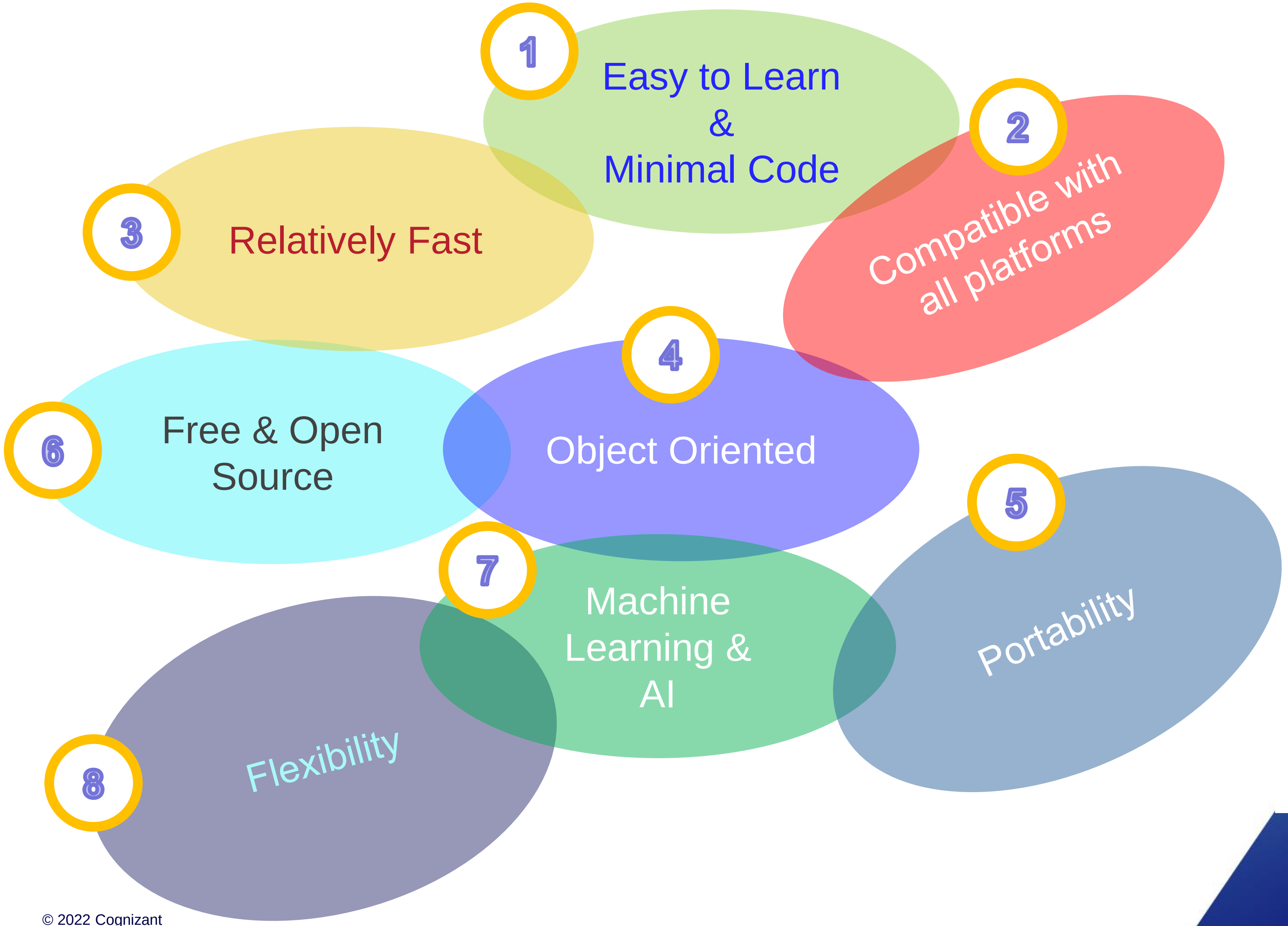
Python

Audio and Visual processing

Data science

Gaming Apps

Why to use Python?



1 Easy to Learn & Minimal Code

2 Compatible with all platforms

3 Relatively Fast

4 Object Oriented

5 Portability

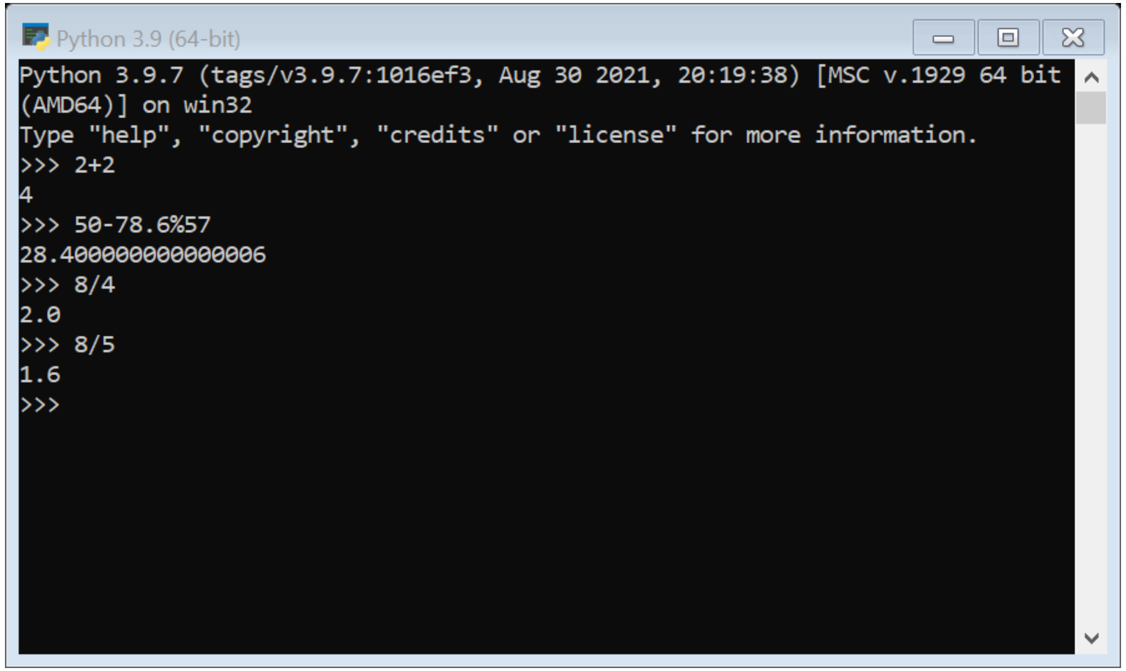
6 Free & Open Source

7 Machine Learning & AI

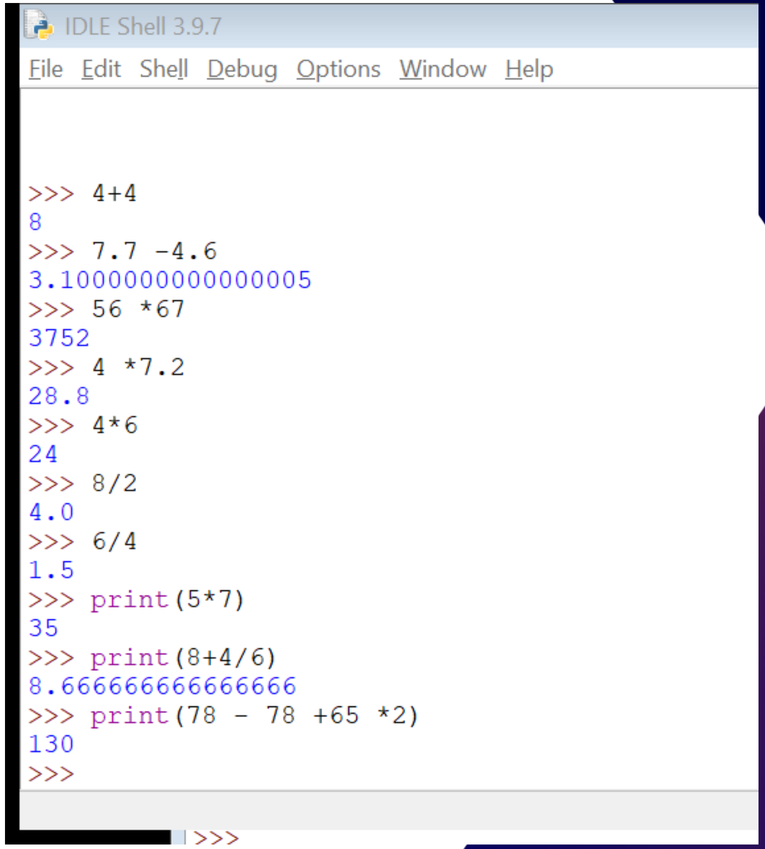
8 Flexibility

Visualization

Python Interpreter As Calculator



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> 50-78.6%57
28.400000000000006
>>> 8/4
2.0
>>> 8/5
1.6
>>>
```

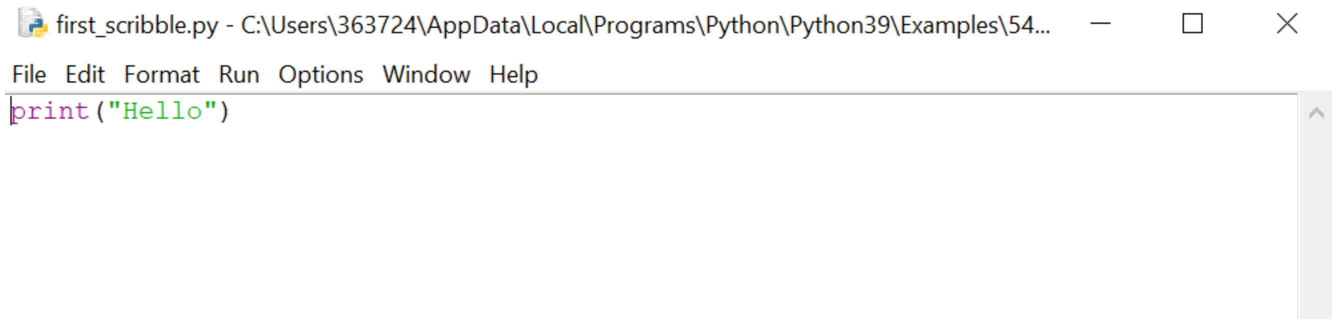


```
>>> 4+4
8
>>> 7.7 -4.6
3.1000000000000005
>>> 56 *67
3752
>>> 4 *7.2
28.8
>>> 4*6
24
>>> 8/2
4.0
>>> 6/4
1.5
>>> print(5*7)
35
>>> print(8+4/6)
8.666666666666666
>>> print(78 - 78 +65 *2)
130
>>>
```

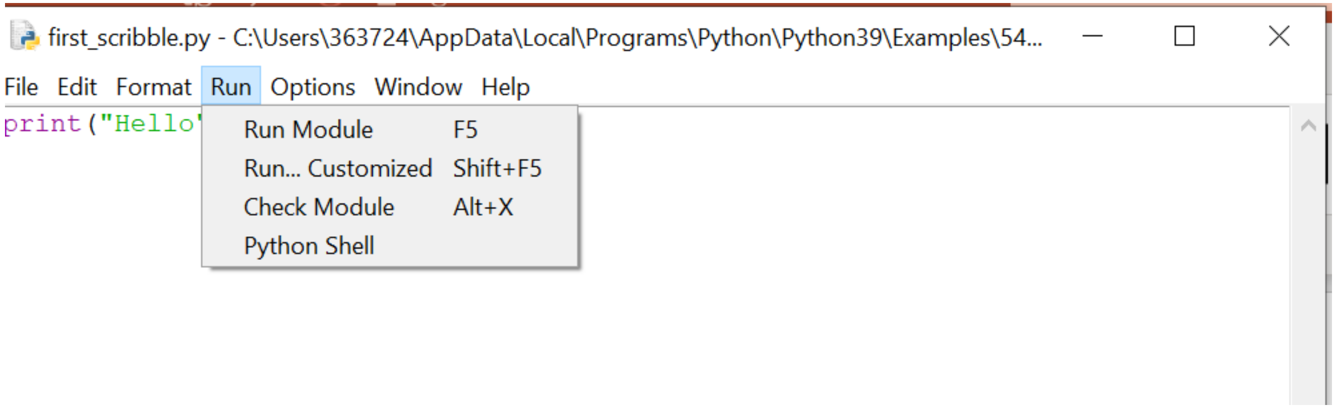
Working with IDLE

1. IDLE is the Python environment we will be using. Look for "IDLE (Python 3.5 32-bit)" entry in the Programs list, under Python 3.5.
2. The IDLE shell window opens up. You can again type in print("hello!") and so forth, and the shell will do the printing. As you can see, it's interactive. Python responds to every line of code you enter.
3. Opening up a new window will create a script file window. Here, print("hello!") does not immediately produce output. That is because this is a script file editing window, which means the commands won't execute until the file is saved and run.
4. You can run the script by going "Run --> Run Module" or simply by hitting F5 (on some systems, Fn + F5).
5. Before running, IDLE prompts you to save the script as a file. Choose a name ending in .py ("hello.py") and save it on Desktop.
6. The script will then run in the IDLE shell window. Since you now have a saved script, you can run it again (and again, and again...).
7. I also have the IDLE shortcut pinned in the START menu (how to do that in next tutorial). I can launch IDLE from there.
8. I can then open up the saved "hello.py" file and run it again, through the "Open..." dialogue.
9. You can also open IDLE directly from your Python script file. Right click the file, then choose "Edit with IDLE".
10. Rather than going through the "Run..." menu, learn to use F5 (on some systems, Fn + F5) to run your script. It's much quicker.

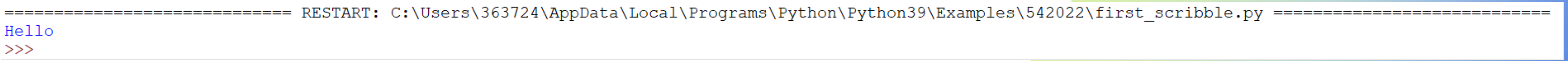
Working with Python Script


first_scribble.py - C:\Users\363724\AppData\Local\Programs\Python\Python39\Examples\54...
File Edit Format Run Options Window Help

```
print("Hello")
```


first_scribble.py - C:\Users\363724\AppData\Local\Programs\Python\Python39\Examples\54...
File Edit Format Run Options Window Help
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell

```
print("Hello'
```



```
============================== RESTART: C:\Users\363724\AppData\Local\Programs\Python\Python39\Examples\542022\first_scribble.py ==============================
Hello
>>>
```

Starting coding with Python

First Steps with Python

Print Strings

Try saving the below in a file and observe the outputs

print "Name", "Marks", "Age"

print "John Doe", 80.67, "27"

print "Bhaskar", 76.908, "27"

print "Mohit", 56.98, "25"

Printing formatted strings

Try saving the below in a file and observe the outputs

print "Name Marks Age"

print ("%s %14.2f %11d" % ("John Doe", 80.67, 27))

print ("%s %12.2f %11d" %("Bhaskar" ,76.901, 27))

print ("%s %3.2f %11d" %("Mohit", 56.98, 25))

Print with escape sequence

Try these in a file and observe the outputs

print 'a’

Print(“'tHermit’ “)

print "i know , they are 'great’”

print "Only way to join" + "two strings"

Variables, Datatypes & operators

Varaibles

- Variable names can begin with _, $, or a letter
- Variable names can be in lower case and uppercase
- Variable names cannot start with a number
- White space characters are not allowed in the naming of a variable
- Syntax:

<variable_name> = < expression >

- Single Assignment

Dept= “ADM”; Role=‘Developer’

- Multiple Assignment

A=B=C=45

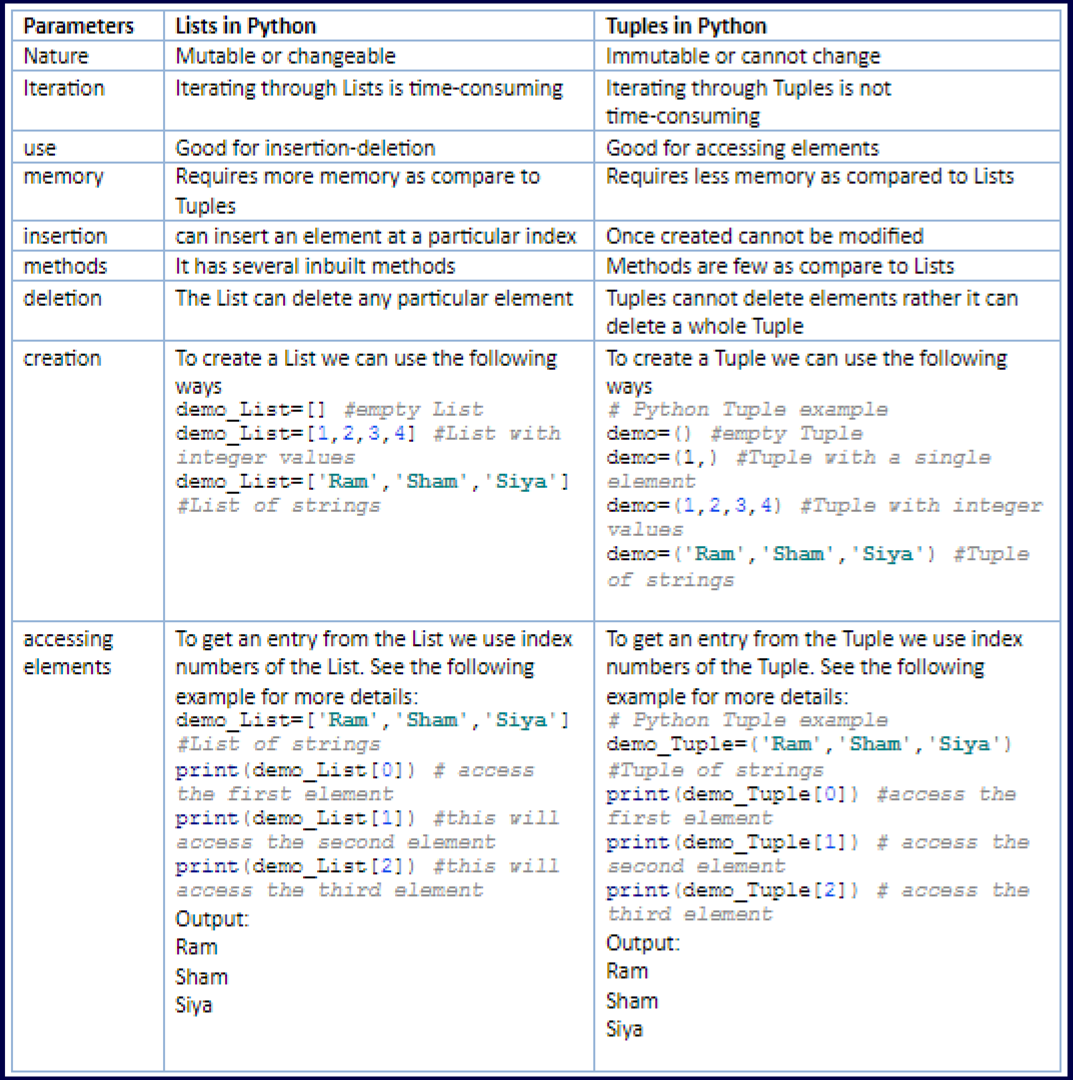
Data Types

- Numbers
  - Simple(Int, longInt, Zeros)
  - Complex
  - Floating point
  - Boolean
- String
- Tuples
- List
- Dictionary

Operators

- Arithmetic operators.
  - ** - exponent
  - * - Multiplication
  - / - Division
  - % - Modulo division
  - + - Addition
  - - - Subtraction
  - BODMAS rule applicable(Bracket, Of, Division, Multiplication, Addition, and Subtraction (BODMAS))
- Comparison operators
- Assignment operators
- Bitwise operators
- Logical operators
- Membership operators
- Identity operators

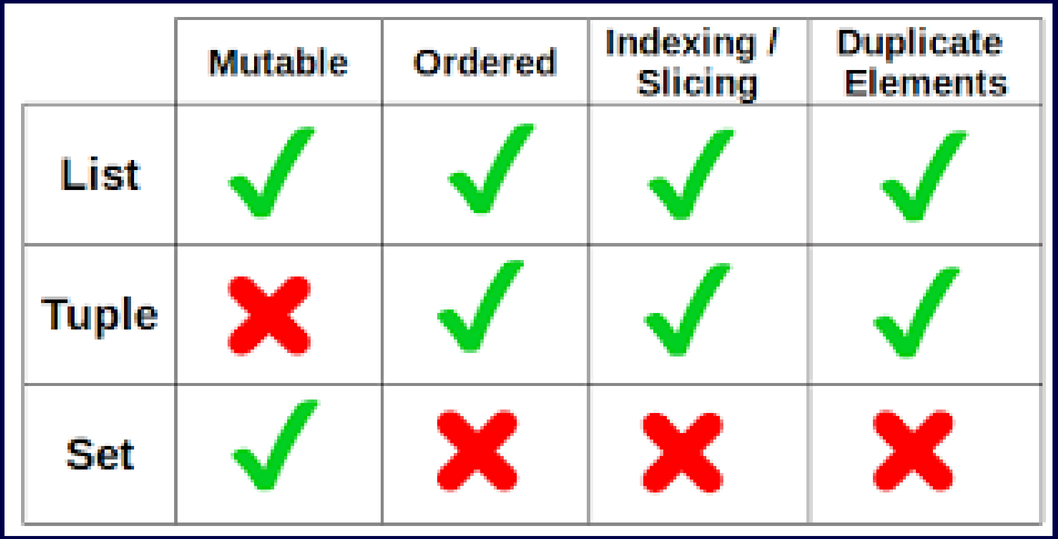
List & Tuple - Comparison



Parameters	Lists in Python	Tuples in Python
Nature	Mutable or changeable	Immutable or cannot change
Iteration	Iterating through Lists is time-consuming	Iterating through Tuples is not time-consuming
use	Good for insertion-deletion	Good for accessing elements
memory	Requires more memory as compare to Tuples	Requires less memory as compared to Lists
insertion	can insert an element at a particular index	Once created cannot be modified
methods	It has several inbuilt methods	Methods are few as compare to Lists
deletion	The List can delete any particular element	Tuples cannot delete elements rather it can delete a whole Tuple

creation	To create a List we can use the following ways
	`demo_List=[] #empty List`
	`demo_List=[1,2,3,4] #List with integer values`
	`demo_List=['Ram','Sham','Siya'] #List of strings`
		To create a Tuple we can use the following ways
		`# Python Tuple example`
		`demo=() #empty Tuple`
		`demo=(1,) #Tuple with a single element`
		`demo=(1,2,3,4) #Tuple with integer values`
		`demo=('Ram','Sham','Siya') #Tuple of strings`

accessing elements	To get an entry from the List we use index numbers of the List. See the following example for more details:
	`demo_List=['Ram','Sham','Siya'] #List of strings`
	`print(demo_List[0]) # access the first element`
	`print(demo_List[1]) #this will access the second element`
	`print(demo_List[2]) #this will access the third element`
	Output:
	Ram
	Sham
	Siya
		To get an entry from the Tuple we use index numbers of the Tuple. See the following example for more details:
		`# Python Tuple example`
		`demo_Tuple=('Ram','Sham','Siya') #Tuple of strings`
		`print(demo_Tuple[0]) #access the first element`
		`print(demo_Tuple[1]) # access the second element`
		`print(demo_Tuple[2]) # access the third element`
		Output:
		Ram
		Sham
		Siya



	Mutable	Ordered	Indexing / Slicing	Duplicate Elements
List	✓	✓	✓	✓
Tuple	✗	✓	✓	✓
Set	✓	✗	✗	✗

Dissimilarity and Similarity of Python List,Tuple,Set,Frozenset,Dictionary

		List	Tuple	Set	Frozen Set	Dictionary
Dissimilarity	**Type name**	Sequence	Sequence	Set	Set	Mapping

	Is Mutable	Yes
		>>> a = [1, 2, 3]
		>>> a[0]=10
		>>> print(a)#a changed
		[10, 2, 3]
			No
			>>> a=(1,2,3)
			>>> a[0]=10#a not changed
			Traceback (most recent call last):
			File "<stdin>", line 1, in <module>
			TypeError: 'tuple' object does not support item assignment
				Yes
				>>> a = set([1, 2, 3])
				>>> b = a
				>>> b |= set([4, 5, 6])
				>>> print (a) #a changed
				set([1, 2, 3, 4, 5, 6])
					No
					>>> a = frozenset([1, 2, 3])
					>>> b = a
					>>> b |= frozenset([4, 5, 6])
					>>> print (a) #a not changed
					frozenset([1, 2, 3])
						Yes
						>>> a= {1: 'apple', 2: 'ball'}
						>>> a[1]='banana'
						>>> print(a) #apple changed into banana
						{1: 'banana', 2: 'ball'}

	Is member Unique	yes/no
		>>> a = list('good')
		>>> print (a) # duplicate 'o'
		['g', 'o', 'o', 'd']
			yes/no
			>>> a = (1, 2, 3,3)
			>>> print (a) # duplicate '3'
			(1, 2, 3, 3)
				Yes
				>>> a = {1, 2, 3,3}
				>>> print (a) # no duplicate
				set([1, 2, 3])
					Yes
					>>> frozenset((1, 2, 3,3))
					>>> print (a) # no duplicate
					set([1, 2, 3])
						Only keys;incase of duplicacy last is preserved
						>>> a= {1: 'apple1', 2: 'ball',1: 'apple2'}
						>>> print(a) # last '1'e.g. 1:apple2 is kept
						{1: 'apple2', 2: 'ball'}

	Insertion order maintained/is ordered	Yes
		>>> a = [1, 2, 3]
		>>> b=a+[6,7]
		>>> print(b) # order is kept
		[1, 2, 3, 6, 7]
			Yes
			>>> a=(1,2,3)
			>>> b = a + (4, 5, 6)
			>>> print(b) # order is kept
			(1, 2, 3, 4, 5, 6)
				No
				>>> a = {'a', 'b', 'c'}
				>>> a.add('d')
				>>> print (a) # order is broken
				set(['a', 'c', 'b', 'd'])
					No
					>>> a = [('a',1,2), ('d',3,4), ('c',5,6), ('e',2,1)]
					>>> y = set(map(frozenset, a))
					>>> print(y) # order is broken
					set([frozenset(['a', 1, 2]), frozenset(['c', 5, 6]), frozenset([3, 4, 'd']), frozenset([1, 2, 'e'])])
						no(before python 3.7 and Cython 3.6)
						Python 2.7
						>>> keywords = ['foo', 'bar', 'bar', 'foo', 'baz', 'foo']
						>>> list(dict.fromkeys(keywords))# order is broken
						['baz', 'foo', 'bar']
						Python 3.9
						>>> keywords = ['foo', 'bar', 'bar', 'foo', 'baz', 'foo']
						>>> list(dict.fromkeys(keywords))# order is kept
						['foo', 'bar', 'baz']

	Construction:each of the types supports construction using their constructor function	>>> a=[1,2,3]
		>>> a=list((1,2,3))
			>>> a=(1,2,3)
			>>> a = tuple([1,2,3])
				>>> a = {1, 2, 3}
				>>> a = set([1, 2, 3])
					>>> a= frozenset((1,2,3))
					>>> a = frozenset({1,2,3})
					>>> a= frozenset({"name": "John", "age": 23, "sex": "male"})
						>>> a= {1: 'apple', 2: 'ball'}
						>>> a = dict({1:'apple', 2:'ball'})
						>>> a= {'name': 'John', wife: [2, 4, 3]}
						>>> a = dict([(1,'apple'), (2,'ball')])

	Supports Mathematical set operation e.g. a.intersection(b)	No >>> hash([1,2,3] Traceback (most recent call last): File "<pyshell#4>", line 1, in <module> hash(a) TypeError: unhashable type: 'list'	No >>> hash((1,2,3)) -378539185	Yes >>> hash({1, 2, 3}) Traceback (most recent call last): File "<pyshell#10>", line 1, in <module> hash({1, 2, 3}) TypeError: unhashable type: 'set'	Yes >>> hash(frozenset((1,2,3))) 409093564	No >>> hash({1: 'apple', 2: 'ball'}) Traceback (most recent call last): File "<pyshell#12>", line 1, in <module> hash({1: 'apple', 2: 'ball'}) TypeError: unhashable type: 'dict'
	Is hashable[try hash(a)]	no	yes	no	yes	no
Similarity	**Is Collection Type**	yes	yes	yes	yes	yes
	Is Iterable	yes	yes	yes	yes	yes
	Is built-in	yes	yes	yes	yes	yes
	Can be created using constructor	yes	yes	yes	yes	yes
	All of they support some method e.g. len	yes	yes	yes	yes	yes

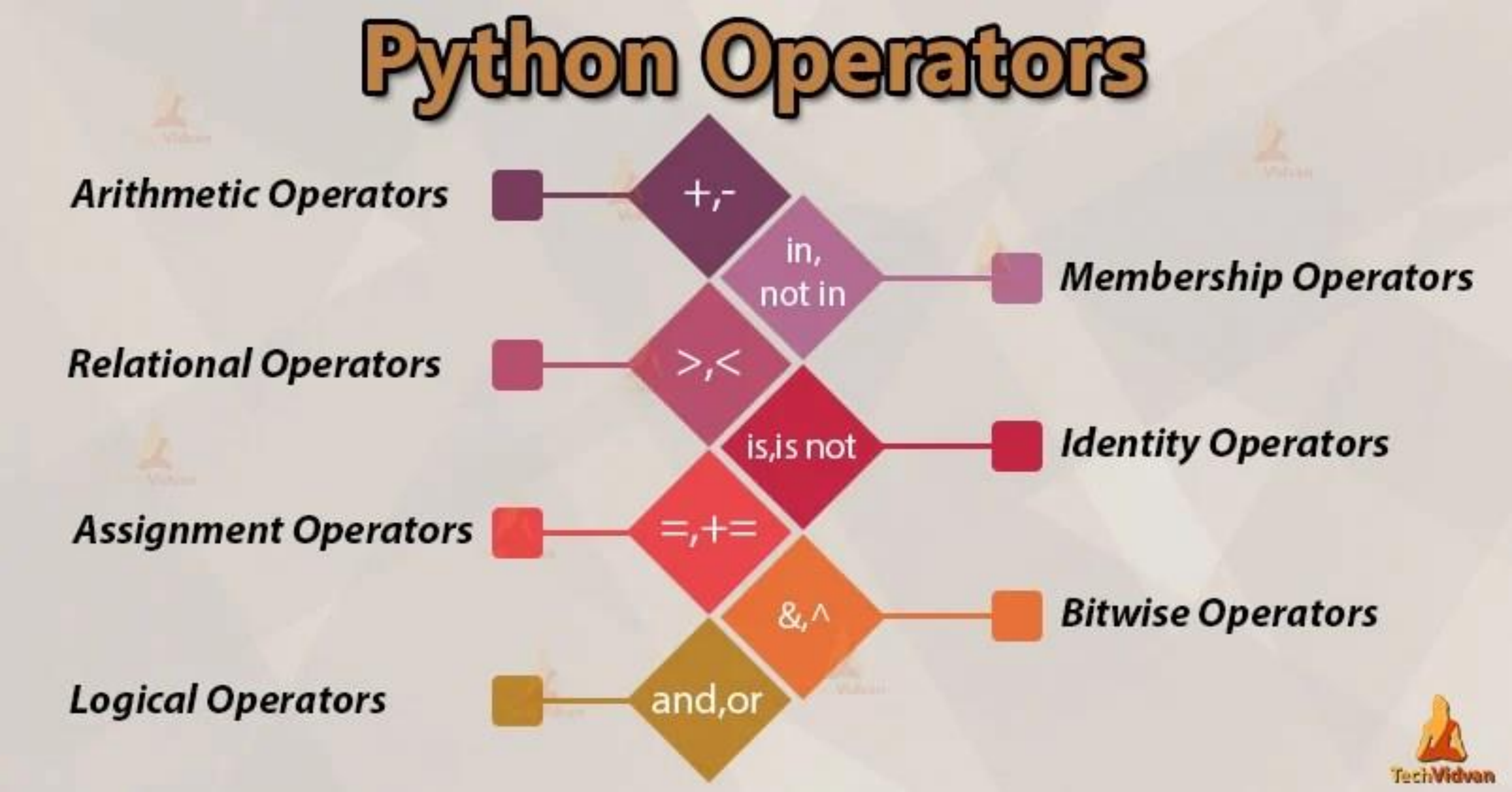
Quick comparison

Parameter	Lists	Sets	Tuples	Dictionary
Syntax	List =[]	Set=set()	Tuple=()	Dict={}
Changing Values	Mutable	Mutable	Immutable(values cannot be changed once assigned)	Mutable
Accepting Duplicates	Can contain duplicate elements	Can't contain duplicate elements	Can contain duplicate elements	Can't contain duplicate keys but can contain duplicate values
Appending	List[2]=100	Set.add(14)	Cannot alter/append	Dict["Key1"]=15
Accessing/ Printing values	Print(list[2])	Print(set)	Print(word[0])	Print(dict["key1"])
Slicing	Can be done	Cannot be done	Can be done	Cannot be done
Usage	1) If you have collection of date that doesn't need random access 2) When you need simple, iterative collection that is frequently modified	1) Membership testing and elimination of duplicate entries 2) When you need uniqueness for the elements	1) Used in combination with dictionaries, where tuple can be a key value 2) When your date doesn't change	1) When you need logical association of data as key/value pairs 2) When you need fast lookup of values 3) Frequently modified data's

Example	List = {2,4,6]	Set = {1,2,3,4}
		Print(set) → {1,2,3,4}
		Set = {1,2,2}
		Print(set) → {1,2,}
			Words =("Books","Pens"}	Dict = {"key1":23,"key2":43}

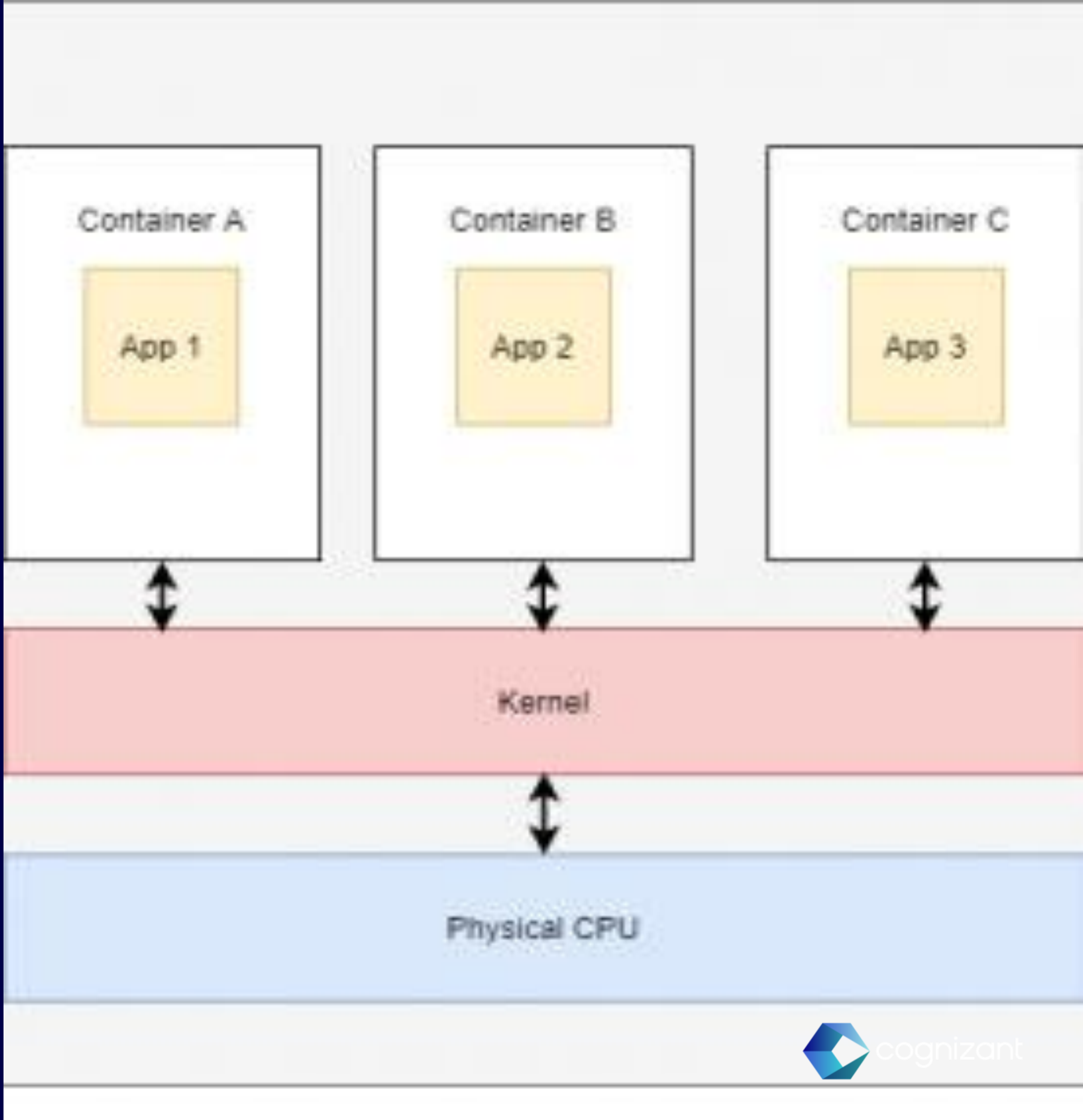
Sorting	Sequence type, sortable	Unordered	Sequence type	Unorder and non sortable since it's a hashmap

Python Operators



Functions & Modules

Minimal code and maximal reusability



Functions

Creating Function

- Without return type:

```
def my_function():
    print("Hello from a function")
```

- With Return type:

```
 def my_function(x):
    return 5 * x

  print(my_function(3))
  print(my_function(5))
  print(my_function(9))
```

Calling a Function

```
def my_function():
    print("Hello from a function")

  my_function()
```

———

- Passing Arbitrary Arguments, *args

```
def my_function(*kids):
    print("The youngest child is " + kids[2])

  my_function("Emil", "Tobias", "Linus")
```

Function is a block of code which only runs when it is called

Arguments

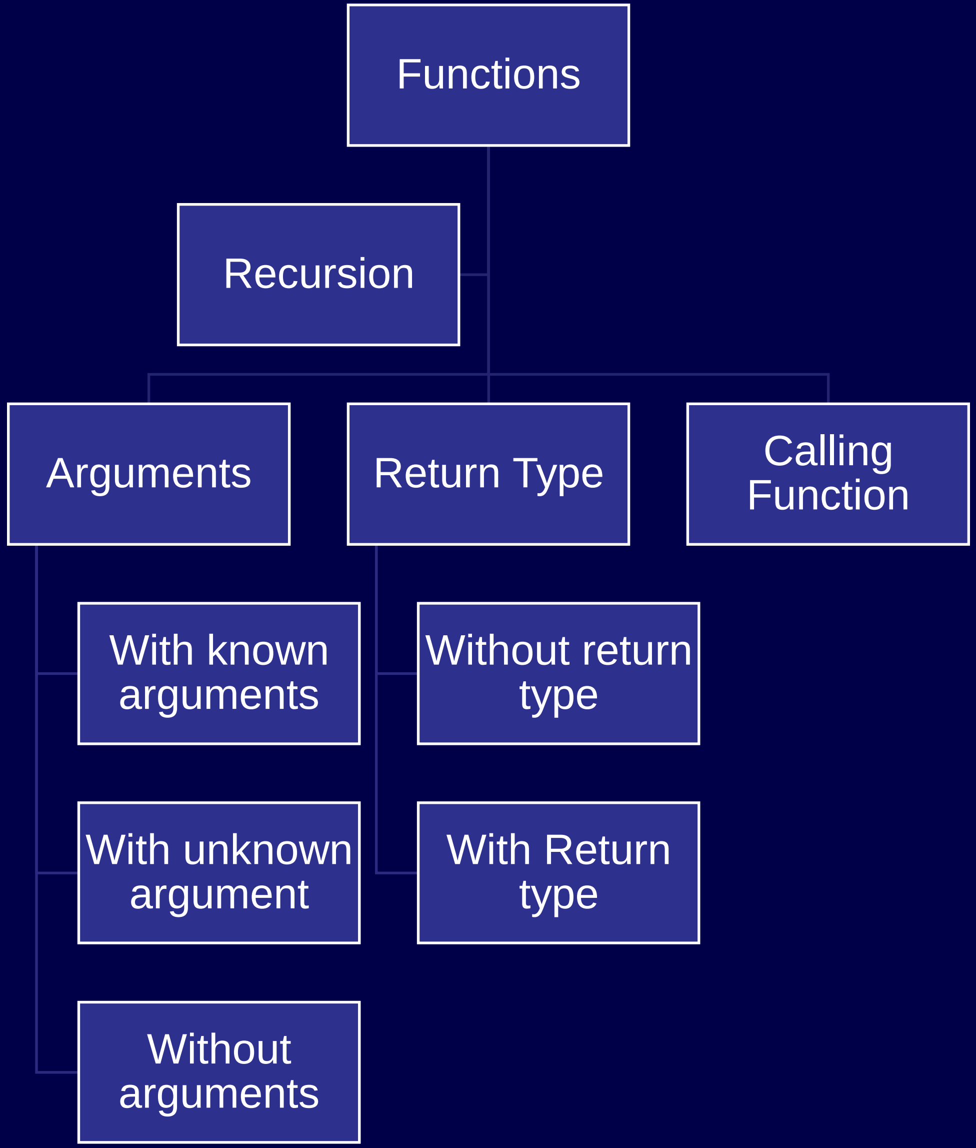
- def my_function(**fname**):

```
    print(fname + " Refsnes")

  my_function("Emil")
  my_function("Tobias")
  my_function("Linus")
```

Recursion

- def tri_recursion(k):

```
    if(k > 0):
      result = k + tri_recursion(k - 1)
      print(result)
    else:
      result = 0
    return result

  print("\n\nRecursion Example Results")
  tri_recursion(6)
```



Modules

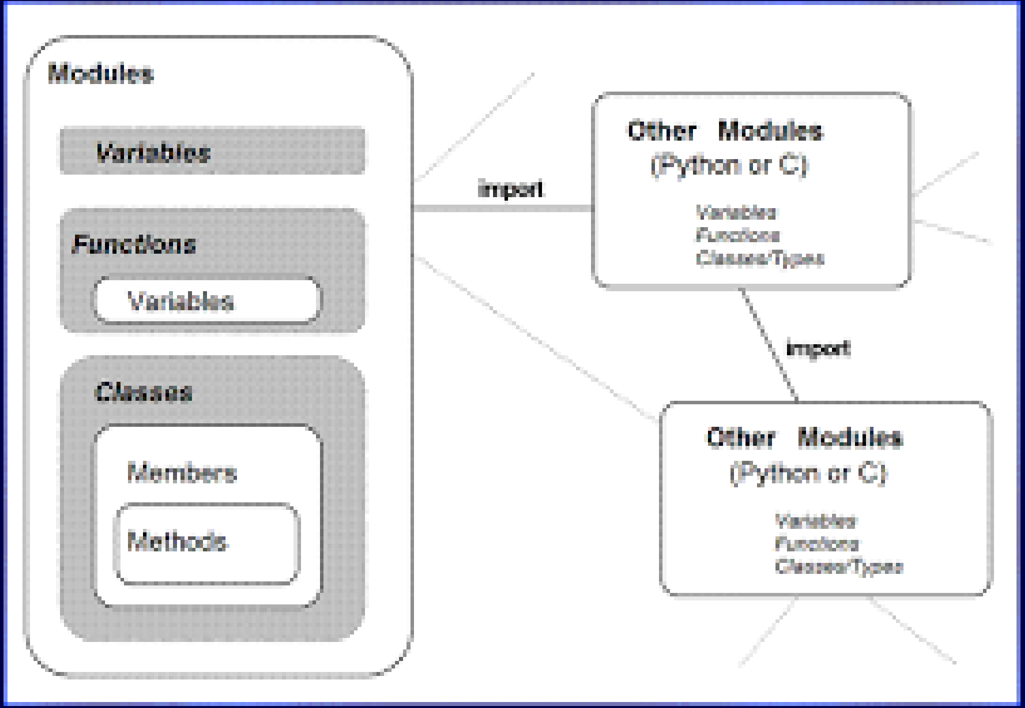
1. Module is intermediate code library that can referenced wherever needed.
2. Module contains built-in, user defined functions, Variables and statements

```
# creating Modules without
functions
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

```
# using the modules in Code
import mymodule

a = mymodule.person1["age"]
print(a)
```

```
# creating Modules with
functions
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

```
# using the modules in Code
from mymodule import person1
from mymodule import person1
as fed
print (person1["age"])
Print ( fed.perosn1[“age”])
```



Comparison

Area	Modules	Functions
Definition	has a bunch of functionalities defined in it that can be imported as a whole file into any application. Has .py as extension	Block of organized, reusable code that is used to perform a single, related action
Accessibility	Stand alone and can be used in another application as well	More specific to a task, to fulfill a functionality while **a module defines classes, functions, attributes, etc.**
Capability	Supports reusability of the code, as well as the scalability	Provides only reusability of the code
Types	User-defined and built-in modules	User-defined and built-in functions

Usage	Import module1()
	Module1.func(“Arguments”)
		def myfunc():
		    <code statements
		myfunc()

Example	import math
	
	print("The value of pi is", math.pi)
		def evenOddFunc(x):
		    if (x % 2 == 0):
		        return "This is even number"
		    else:
		        return "This is odd number“
		evenOddFunc(10)

Working with Files

Containerization is a lightweight alternative to virtualization. This involves encapsulating an application in a container with its own operating environment. Thus, instead of installing an OS for each virtual machine, containers use the host OS.

Overview of File Handling

Directory/File Listing

```
import os
entries =
os.listdir('my_directory/’)
entries =
os.scandir('my_directory/’)

entries = Path('my_directory/')
for entry in entries.iterdir():
    print(entry.name)
```

Retrieve File Properties

os.stat(), os.scandir(), or
pathlib.Path()

Create single/multiple directories

```
os.mkdir()
os.makedirs()

import pathlib

p = pathlib.Path('2018/10/05')
p.mkdir(parents=True)
```

File name Pattern matching

endswith() and startswith()
fnmatch.fnmatch()
glob.glob()
pathlib.Path.glob()

Working with Files

Reading Single file:

```
with open('data.txt', 'r') as f:
    data = f.read()
```

Reading Multiple files:

```
import fileinput
for line in fileinput.input()
    process(line)

f = open("myfile.txt", "a")
f.write("Now the file has more
content!")
f.close()
```

Using ZipFiles

Reading:

```
with zipfile.ZipFile('data.zip', 'r')
as zipobj:
```

Listing

```
with zipfile.ZipFile('data.zip', 'r')
as zipobj:
    zipobj.namelist()
```

Extracting:

.extract() and .extractall().

Writing:

.write(name)

Copy/Move/Rename Files

```
import shutil

src = 'path/to/file.txt'
dst = 'path/to/dest_dir'
shutil.copy(src, dst)
shutil.copytree(src_dir,dst_dir)
shutil.move(src, dst)
os.rename(src, dst)
```

Deleting files

Deleting Files:

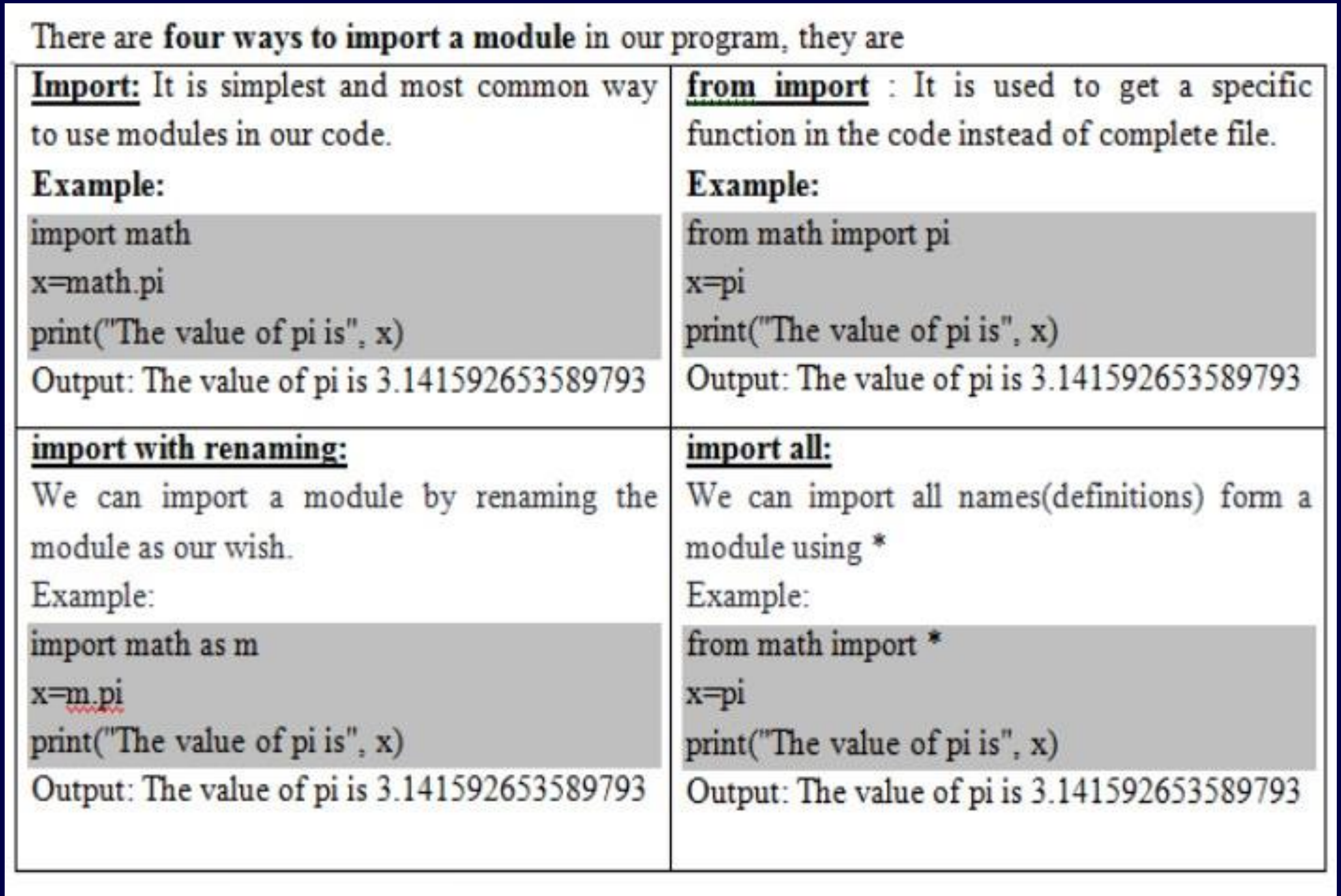
.unlink() or .remove()

Deleting Directories

os.rmdir()
pathlib.Path.rmdir()
shutil.rmtree()

Exception handling in Python

- 2 types of abnormalities handled in Python
  - Errors
    - Logical Errors
    - Syntax errors
  - Exceptions



There are **four ways to import a module** in our program, they are

Import: It is simplest and most common way to use modules in our code. **Example:** import math x=math.pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793	**from import** : It is used to get a specific function in the code instead of complete file. **Example:** from math import pi x=pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793
import with renaming: We can import a module by renaming the module as our wish. Example: import math as m x=m.pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793	**import all:** We can import all names(definitions) form a module using * Example: from math import * x=pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793

Classes and Objects

Class:

- A collection of similar items, logical enteries
- A blueprint to create objects

Objects:

- A instance of your class
- Used to create variables to access the class members

- **Defining class**

```
class pets:
```

- **Adding members for the class**

```
Class pets:
    Count =10
    def showvalues():
        print(“cats” + “Dogs”)
```

- **Creating objects for the class**

```
Class pets:
    Count =10
    def showvalues():
        print(“cats” + “Dogs”)
obj1 = pets()
```

- **Accessing the members of class through object**

```
Class pets:
    Count =10
    def showvalues():
        print(“cats” + “Dogs”)
obj1 = pets()
Print(obj1.count)
Obj1.showvalues
```

Constructors (Self Calling methods)

1. Constructor is a method that is called when an object is created.
2. This method is defined in the class and can be used to initialize basic variables.

```
class Plane:
    def __init__(self):
        self.wings = 2

        # fly
        self.drive()
        self.flaps()
        self.wheels()

    def drive(self):
            print(f'Accelerating {self.wings} wings')

    def flaps(self):
            print(f'Changing flaps to {self.wings + 1}')

    def wheels(self):
            print('Closing wheels')

ba = Plane()
```

Output:

```
Accelerating 2 wings
Changing flaps 3
Closing wheels
```

OOP in Python

Class: A class is a user-defined type that could be used to model object in real world. A class defines the attributes and methods of an object. It serves as the blueprint from which objects are created

Class Variable: This is a variable that belongs to the class. This variable is used by all the objects created from that class

Instance Variable: This is a variable that belongs to one instance of the class. It belongs to the object created from the class and so is also called object variable

Inheritance: A feature that allows a class to inherit the features (variables and methods) of another class

Overloading: A feature that allows two or more functions to have the same name but behave differently depending on the parameters.

Instantiation: Creating an object from a class

Derived Class (Sub-class or child class): A class that inherits from another class.

Inheritance and

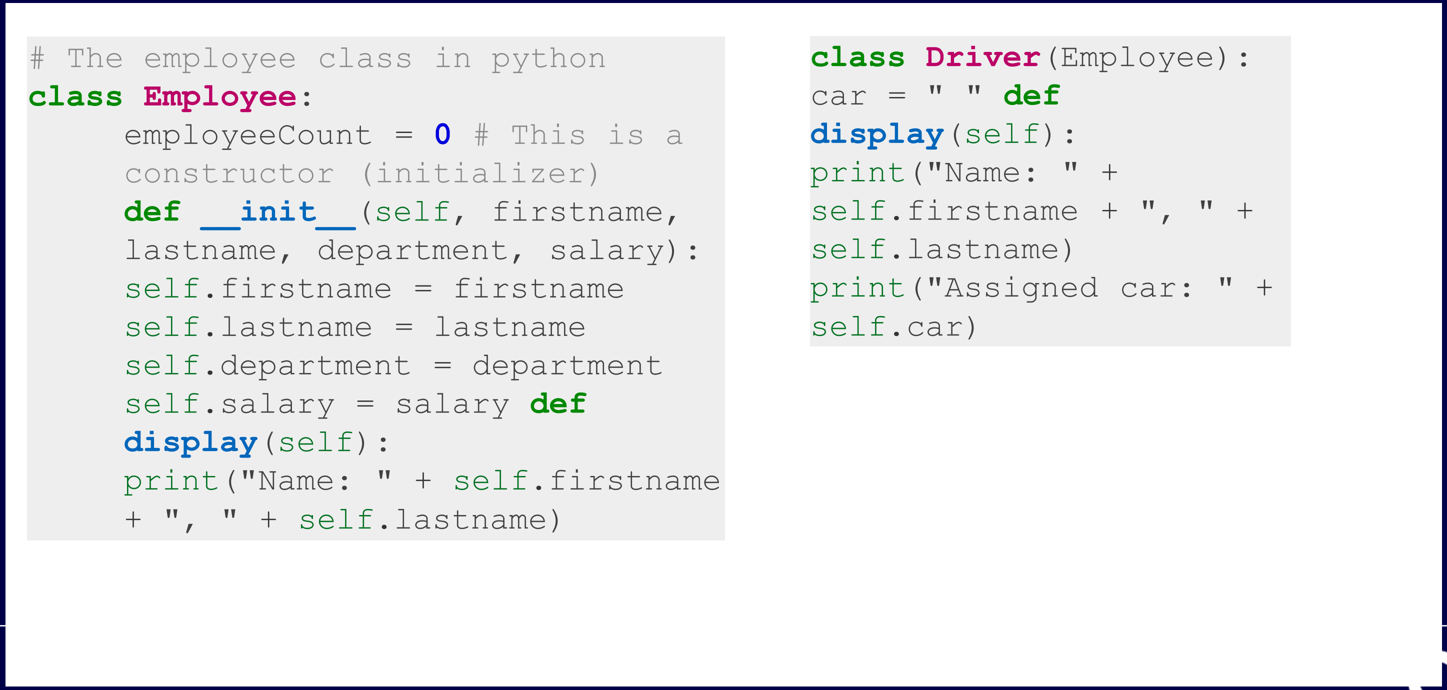
- Inheritance

Creating child classes from parent classes is referred as inheritance

In python, this is achieved by placing the parent class inside parenthesis

- Overriding/Overloading

When a method exactly with the same name as in the parent class present in child class as well, is called method overriding. In this case, the subclass overrides the method in the superclass.



```
# The employee class in python
class Employee:
     employeeCount = 0 # This is a
     constructor (initializer)
     def __init__(self, firstname,
     lastname, department, salary):
     self.firstname = firstname
     self.lastname = lastname
     self.department = department
     self.salary = salary def
     display(self):
     print("Name: " + self.firstname
     + ", " + self.lastname)
```

```
class Driver(Employee):
car = " " def
display(self):
print("Name: " +
self.firstname + ", " +
self.lastname)
print("Assigned car: " +
self.car)
```

Python Libraries

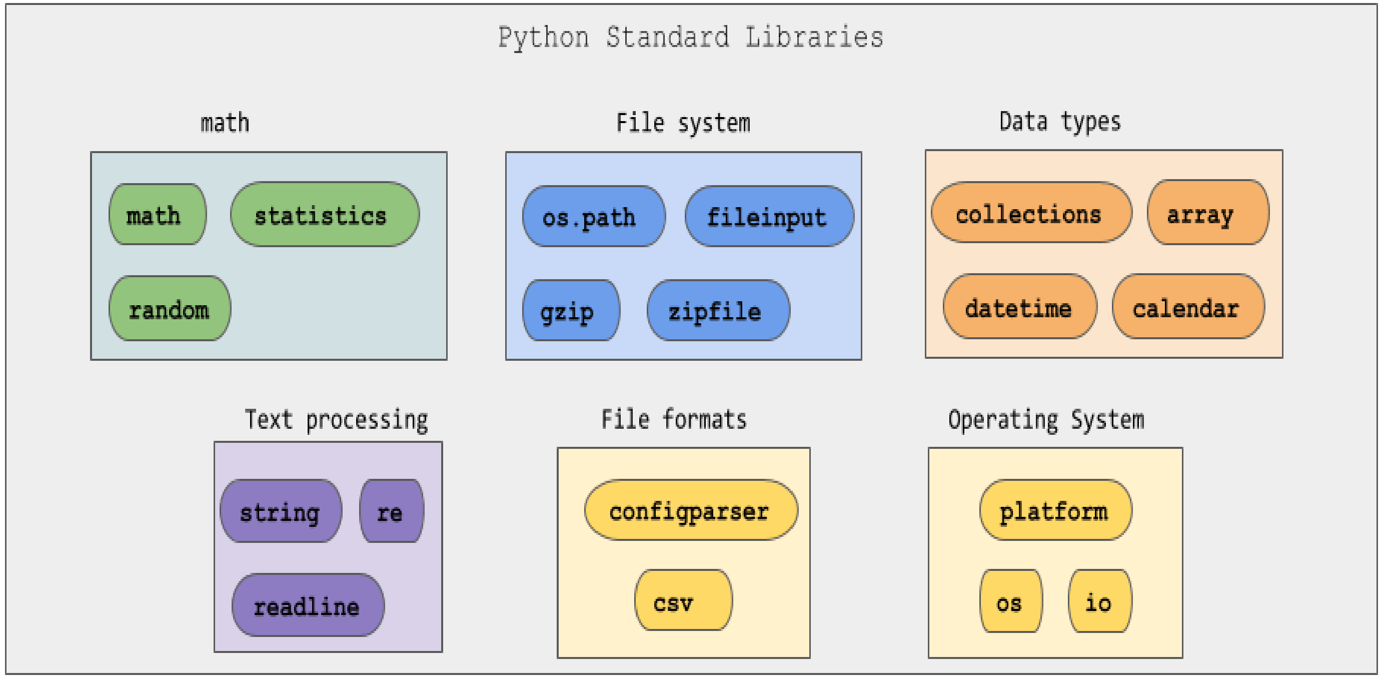
- Collection of related modules.
- Contains bundles of code that can be used repeatedly in different programs.
- Makes Python Programming simpler and convenient for the programmer
- Helps Software platform that is used to build applications based on these library methods
- Standard Libraries

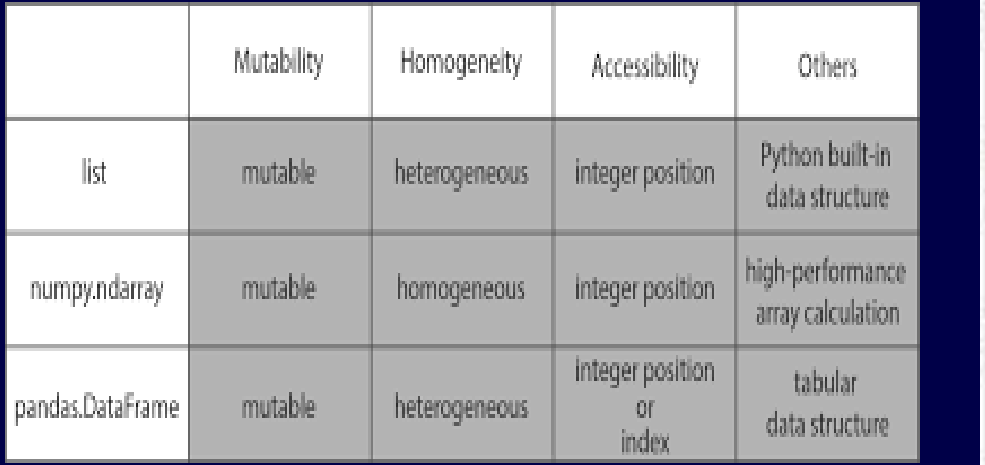
https://docs.python.org/3/library/index.html

- Top 10 Python Libraries:
  - TensorFlow.
  - Scikit-Learn.
  - Numpy.
  - Keras.
  - PyTorch.
  - LightGBM.
  - Pandas.

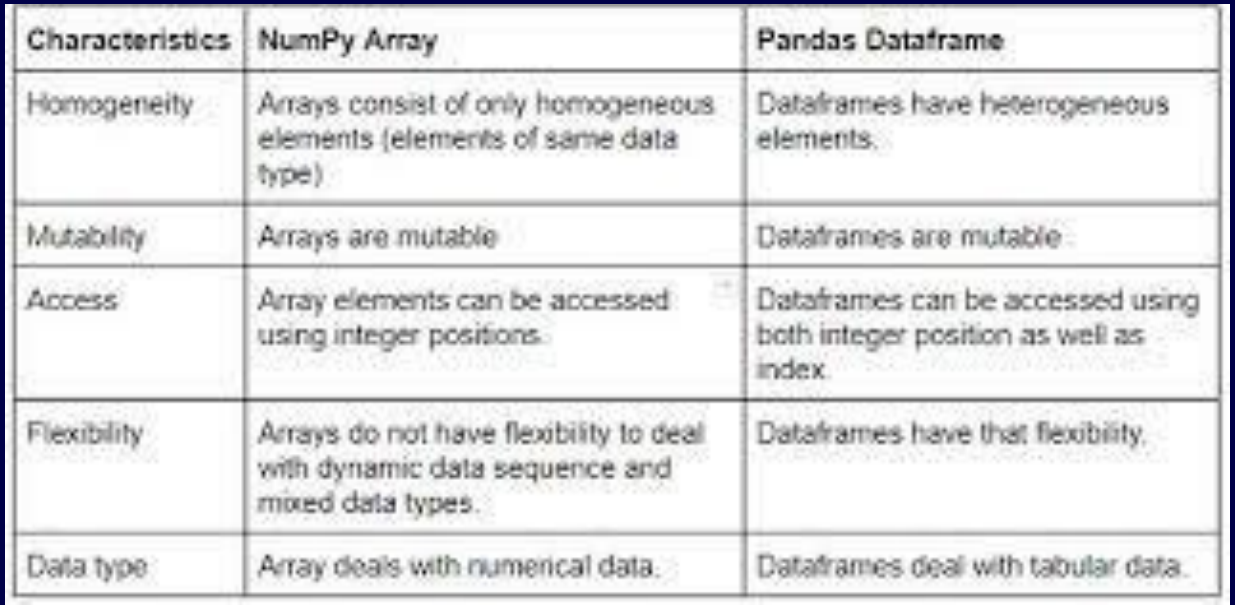
astropy
SunPy
(and many, many more)
PyTables
NetworkX
SM StatsModels Statistics in Python
scikits-image
Bokeh
learn
matplotlib
pandas
PyMC
SciPy
NumPy
Cython
SymPy
DASK
Numba
IP[y]:
IPython
python™
jupyter
ANACONDA

Source: https://pydsc.files.wordpress.com/2017/11/pythonenvironment.png?w=663

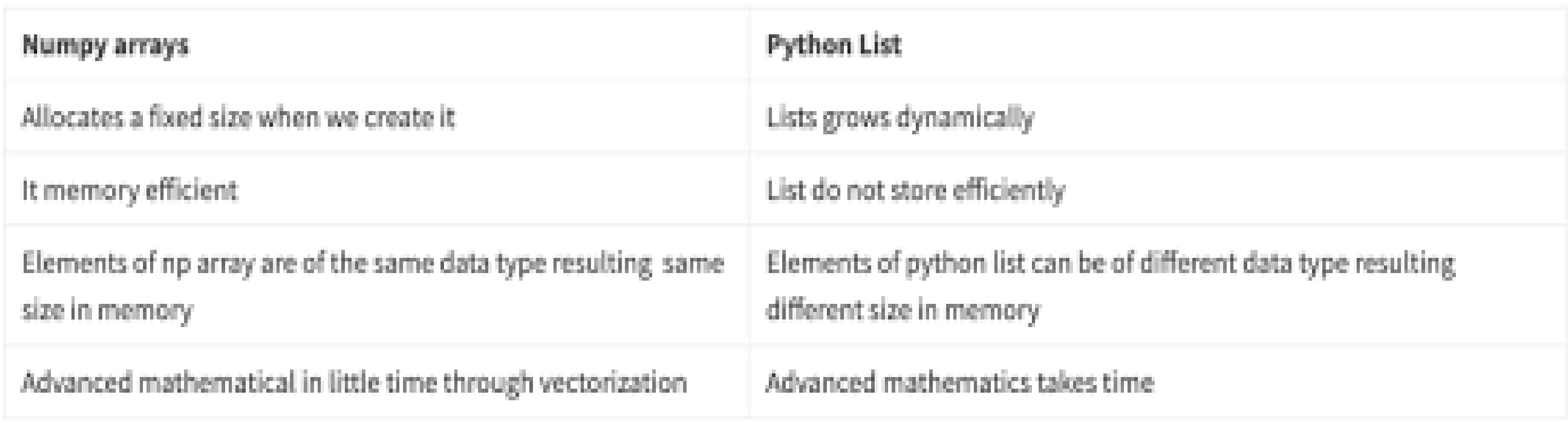

Python Standard Libraries
math
math
statistics
random
File system
os.path
fileinput
gzip
zipfile
Data types
collections
array
datetime
calendar
Text processing
string
re
readline
File formats
configparser
csv
Operating System
platform
os
io



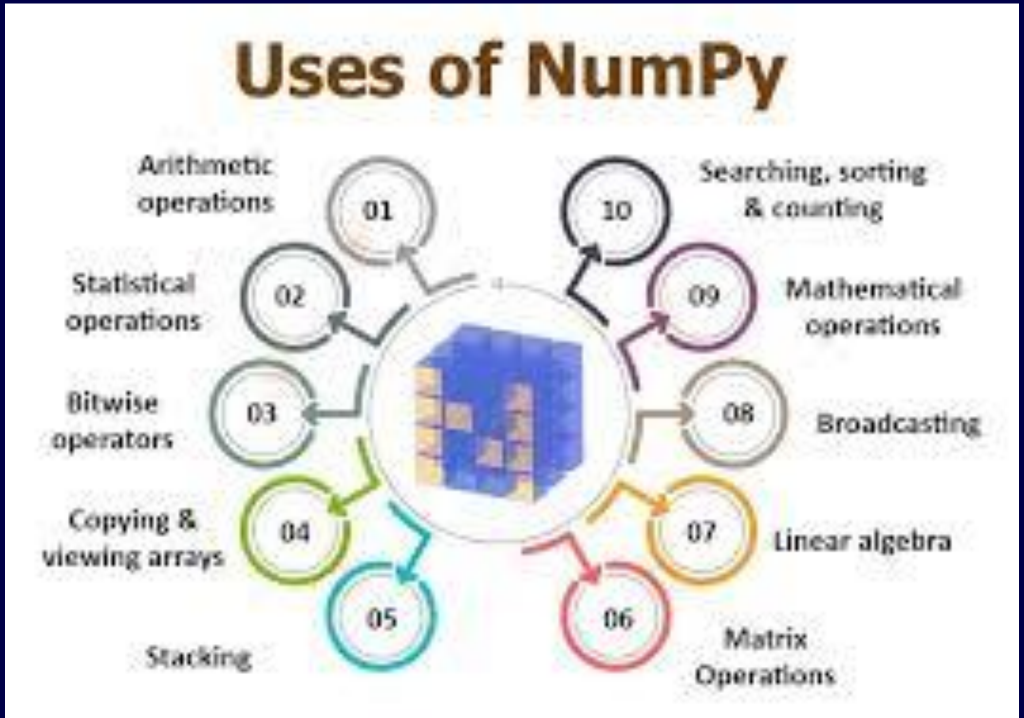
	Mutability	Homogeneity	Accessibility	Others
list	mutable	heterogeneous	integer position	Python built-in data structure
numpy.ndarray	mutable	homogeneous	integer position	high-performance array calculation
pandas.DataFrame	mutable	heterogeneous	integer position or index	tabular data structure

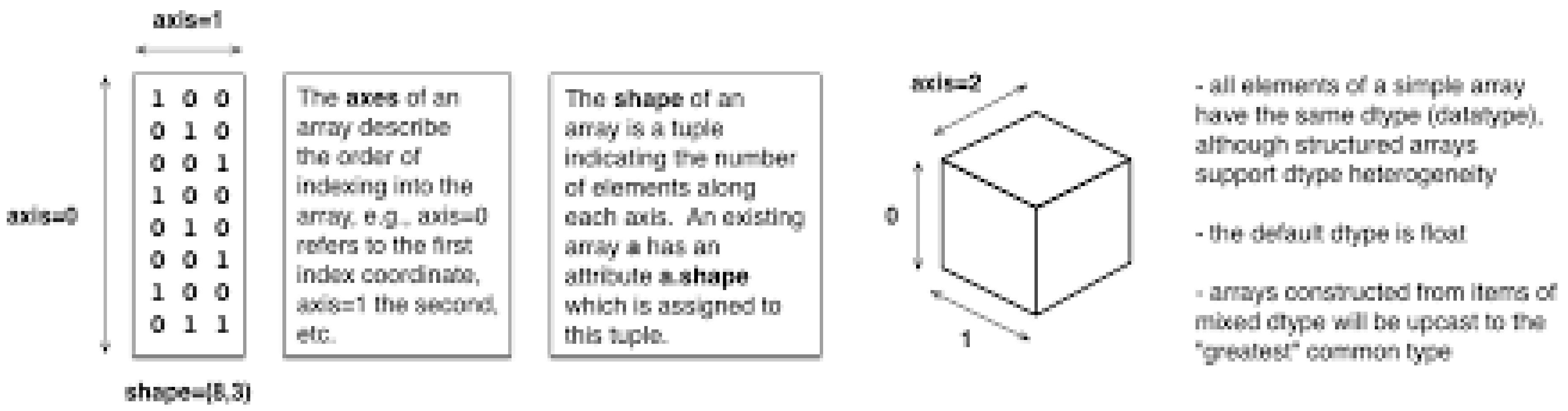


Characteristics	NumPy Array	Pandas Dataframe
Homogeneity	Arrays consist of only homogeneous elements (elements of same data type)	Dataframes have heterogeneous elements.
Mutability	Arrays are mutable	Dataframes are mutable
Access	Array elements can be accessed using integer positions.	Dataframes can be accessed using both integer position as well as index.
Flexibility	Arrays do not have flexibility to deal with dynamic data sequence and mixed data types.	Dataframes have that flexibility.
Data type	Array deals with numerical data.	Dataframes deal with tabular data.



Numpy arrays	Python List
Allocates a fixed size when we create it	Lists grows dynamically
It memory efficient	List do not store efficiently
Elements of np array are of the same data type resulting same size in memory	Elements of python list can be of different data type resulting different size in memory
Advanced mathematical in little time through vectorization	Advanced mathematics takes time


Uses of NumPy
01
Arithmetic operations
02
Statistical operations
03
Bitwise operators
04
Copying & viewing arrays
05
Stacking
06
Matrix Operations
07
Linear algebra
08
Broadcasting
09
Mathematical operations
10
Searching, sorting & counting



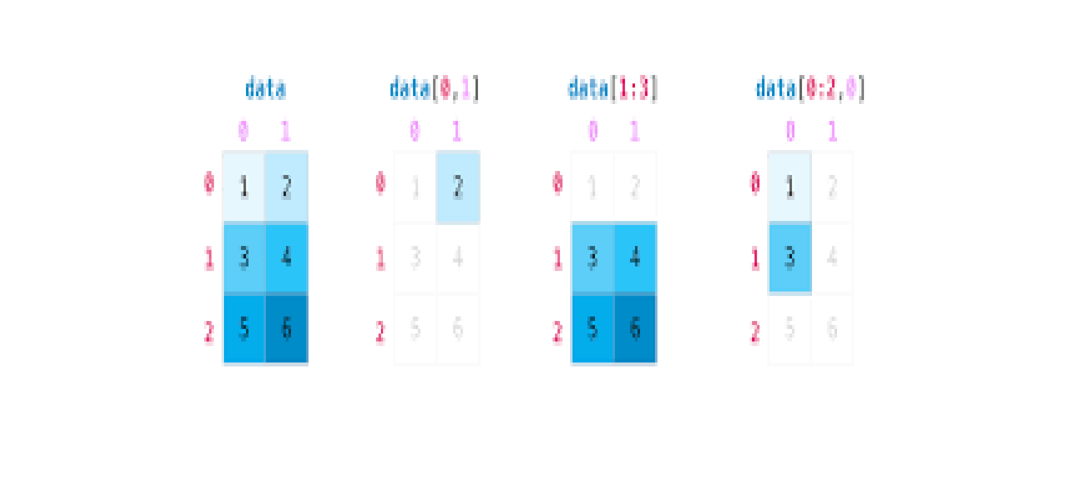
The **axes** of an array describe the order of indexing into the array, e.g., axis=0 refers to the first index coordinate, axis=1 the second, etc.

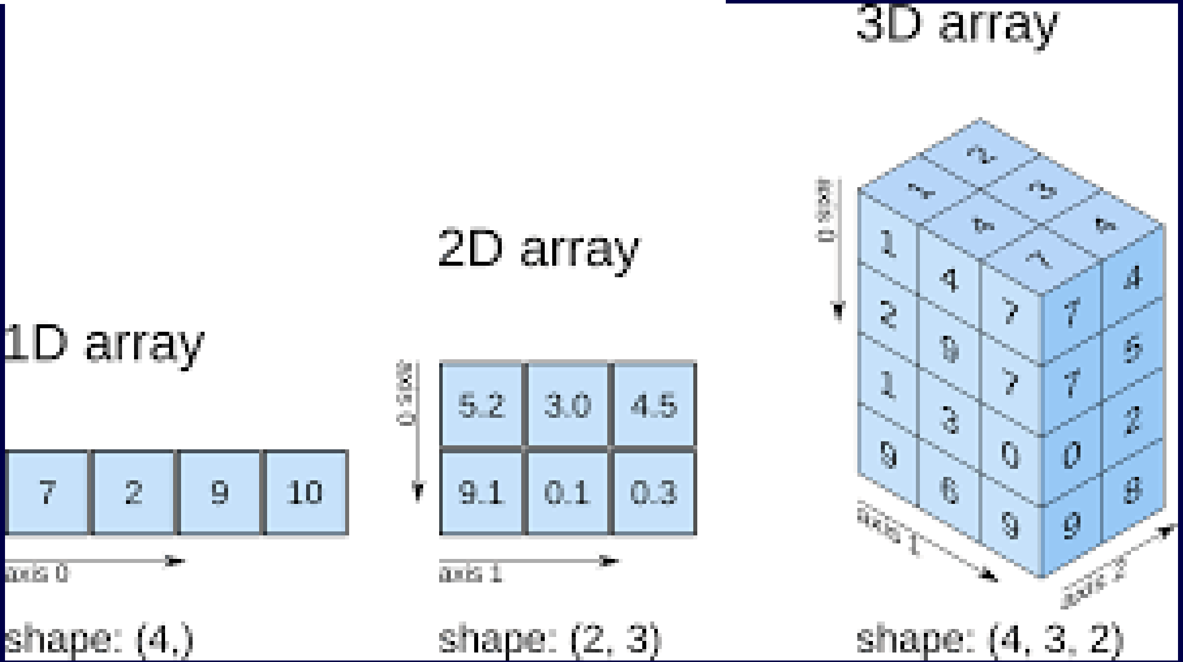
The **shape** of an array is a tuple indicating the number of elements along each axis. An existing array **a** has an attribute **a.shape** which is assigned to this tuple.

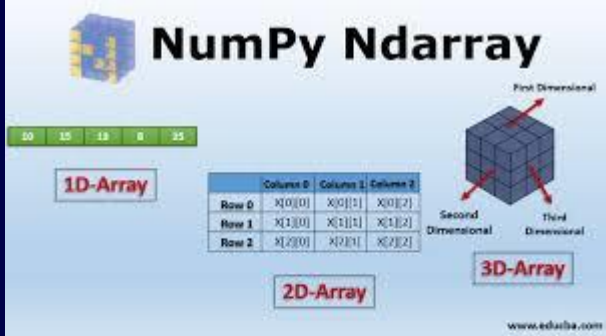
- all elements of a simple array have the same dtype (datatype), although structured arrays support dtype heterogeneity

- the default dtype is float

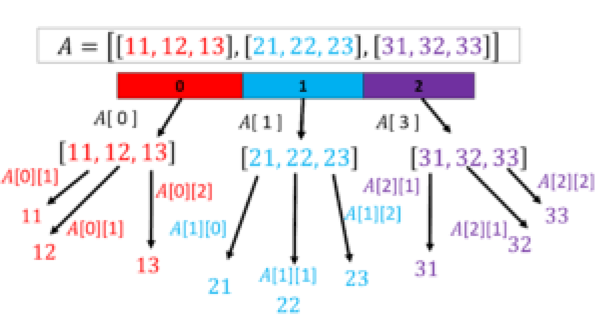
- arrays constructed from items of mixed dtype will be upcast to the "greatest" common type

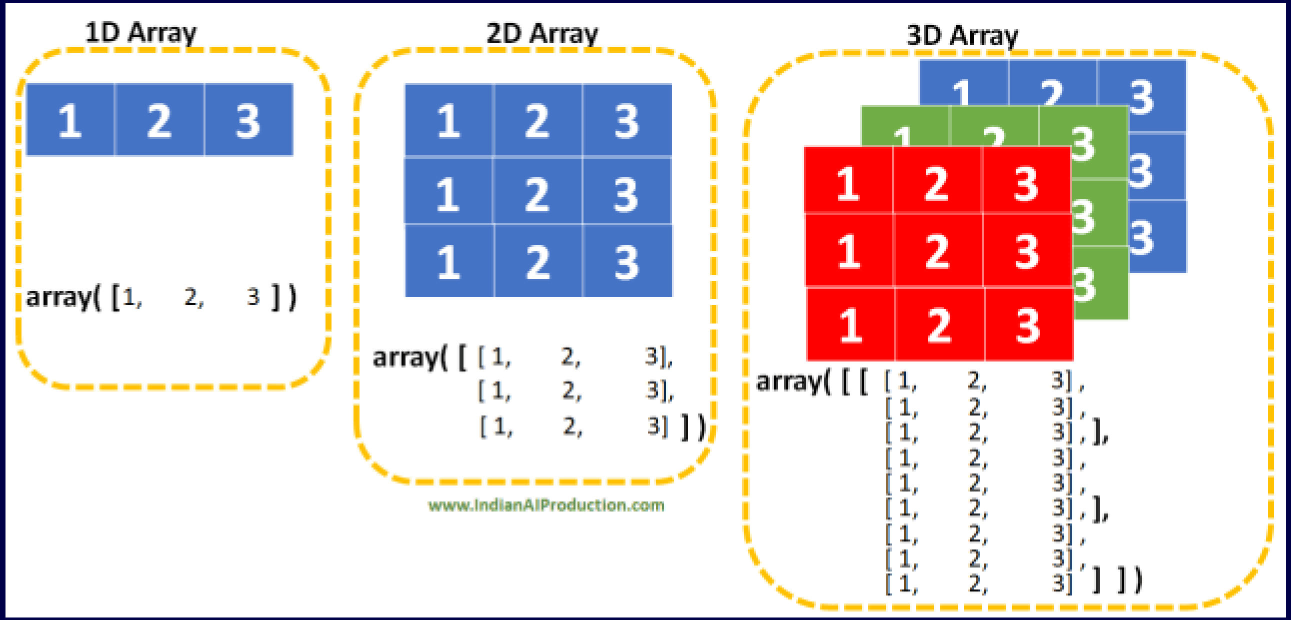






	Column 0	Column 1	Column 2
Row 0	X[0][0]	X[0][1]	X[0][2]
Row 1	X[1][0]	X[1][1]	X[1][2]
Row 2	X[2][0]	X[2][1]	X[2][2]

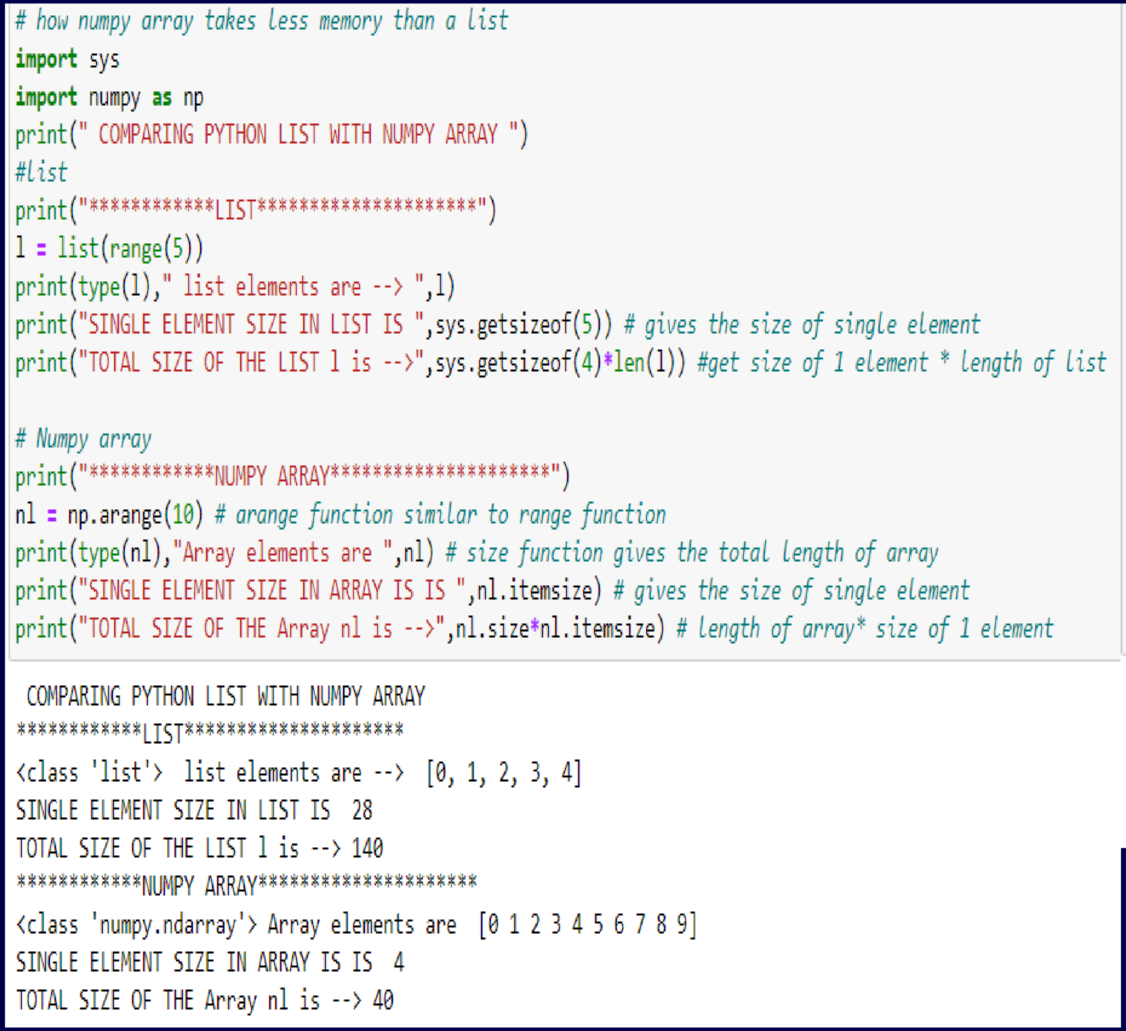

$A = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$



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

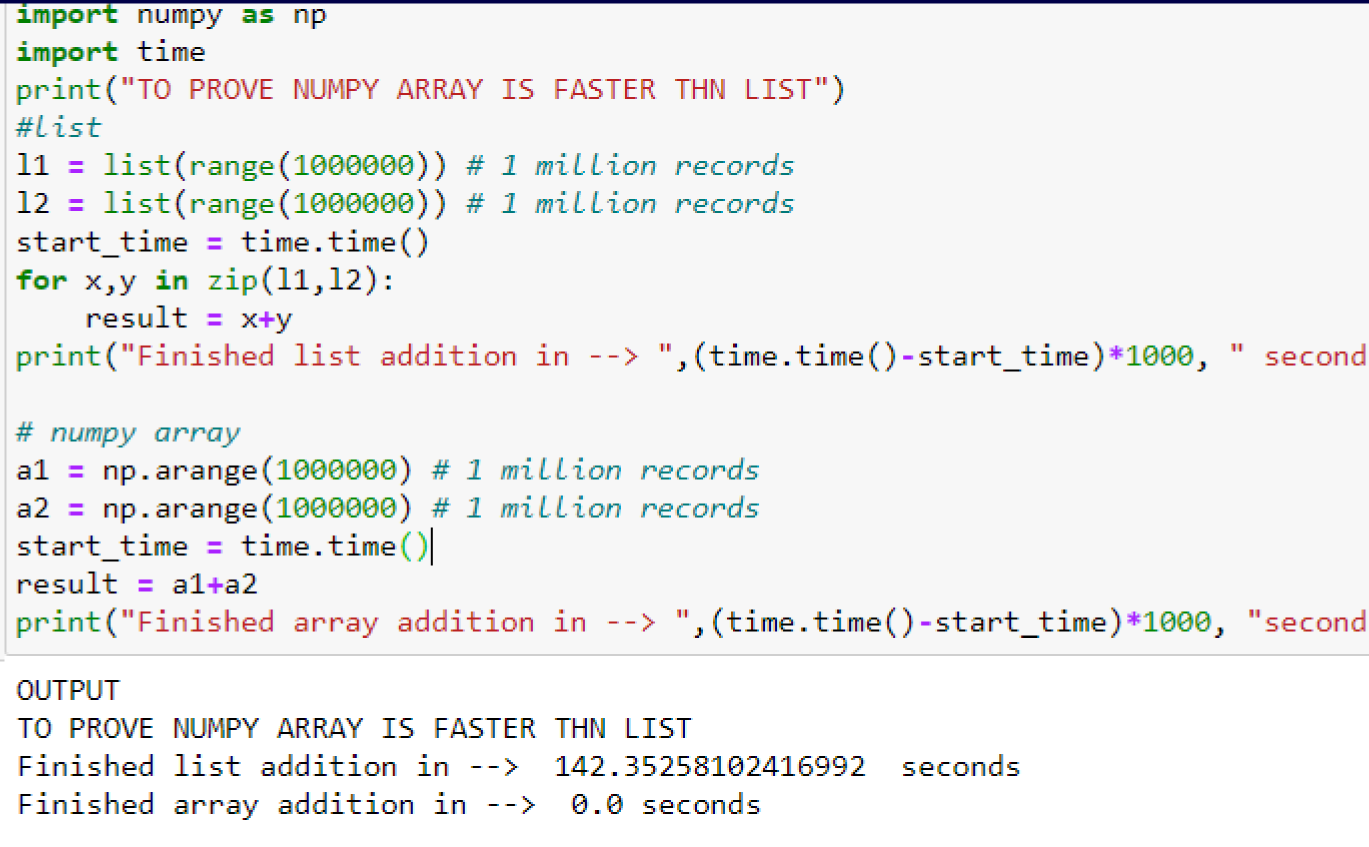
```
array( [ [ 1,     2,      3],
         [ 1,     2,      3],
         [ 1,     2,      3] ] )
```

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



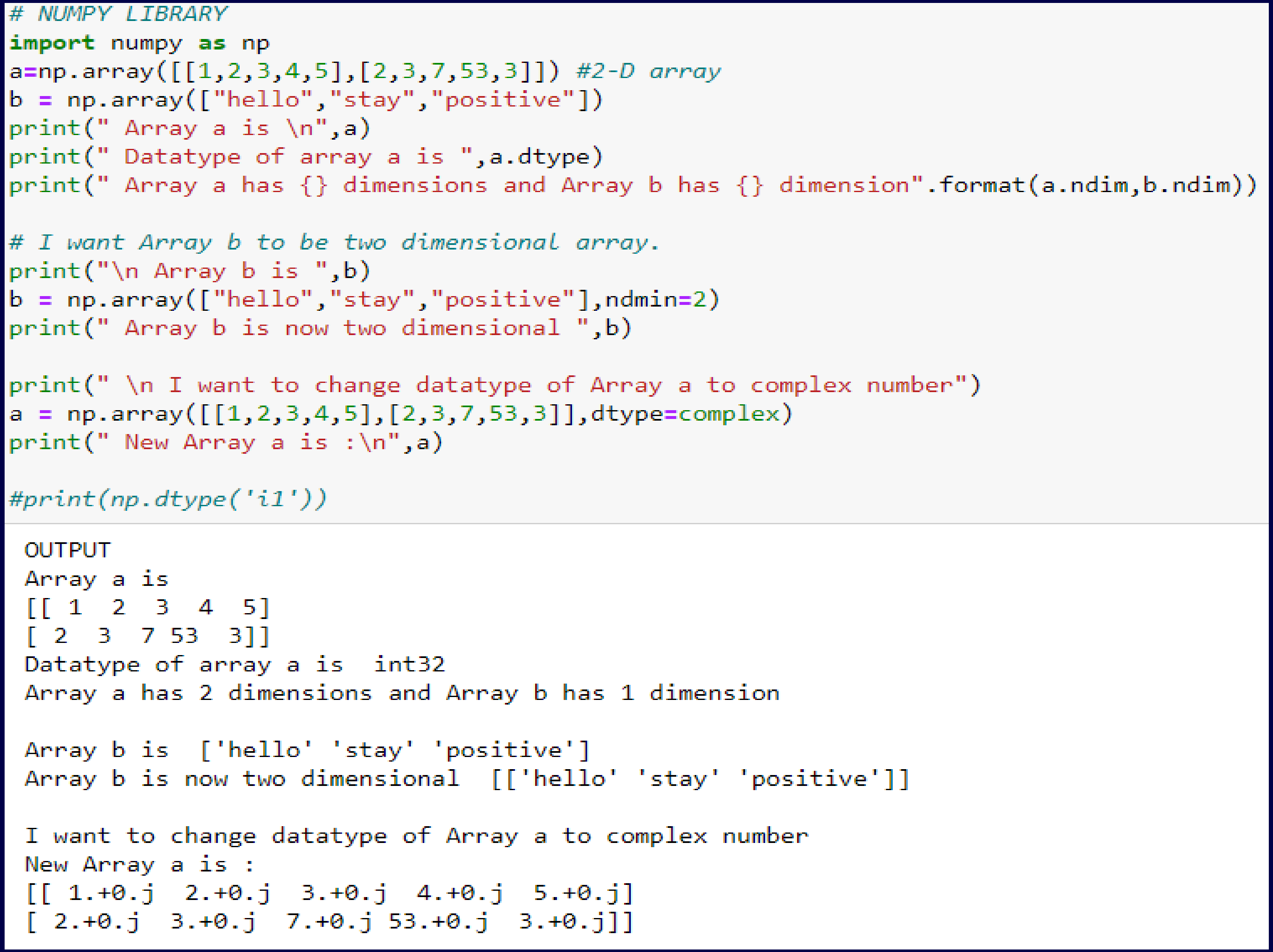
```
# how numpy array takes less memory than a list
import sys
import numpy as np
print(" COMPARING PYTHON LIST WITH NUMPY ARRAY ")
#list
print("************LIST*********************")
l = list(range(5))
print(type(l)," list elements are --> ",l)
print("SINGLE ELEMENT SIZE IN LIST IS ",sys.getsizeof(5)) # gives the size of single element
print("TOTAL SIZE OF THE LIST l is -->",sys.getsizeof(4)*len(l)) #get size of 1 element * length of list

# Numpy array
print("************NUMPY ARRAY*********************")
nl = np.arange(10) # arange function similar to range function
print(type(nl),"Array elements are ",nl) # size function gives the total length of array
print("SINGLE ELEMENT SIZE IN ARRAY IS IS ",nl.itemsize) # gives the size of single element
print("TOTAL SIZE OF THE Array nl is -->",nl.size*nl.itemsize) # length of array* size of 1 element
```

```
 COMPARING PYTHON LIST WITH NUMPY ARRAY
************LIST*********************
<class 'list'>  list elements are -->  [0, 1, 2, 3, 4]
SINGLE ELEMENT SIZE IN LIST IS  28
TOTAL SIZE OF THE LIST l is --> 140
************NUMPY ARRAY*********************
<class 'numpy.ndarray'> Array elements are  [0 1 2 3 4 5 6 7 8 9]
SINGLE ELEMENT SIZE IN ARRAY IS IS  4
TOTAL SIZE OF THE Array nl is --> 40
```



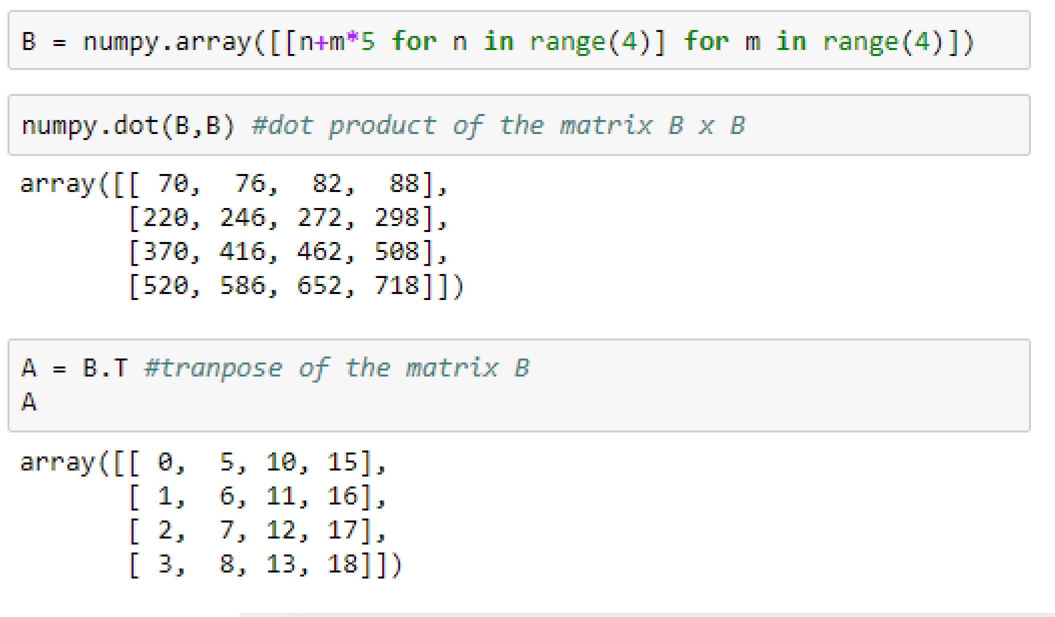
```
import numpy as np
import time
print("TO PROVE NUMPY ARRAY IS FASTER THN LIST")
#list
l1 = list(range(1000000)) # 1 million records
l2 = list(range(1000000)) # 1 million records
start_time = time.time()
for x,y in zip(l1,l2):
    result = x+y
print("Finished list addition in --> ",(time.time()-start_time)*1000, " second

# numpy array
a1 = np.arange(1000000) # 1 million records
a2 = np.arange(1000000) # 1 million records
start_time = time.time()
result = a1+a2
print("Finished array addition in --> ",(time.time()-start_time)*1000, "second
```

```
OUTPUT
TO PROVE NUMPY ARRAY IS FASTER THN LIST
Finished list addition in -->  142.35258102416992  seconds
Finished array addition in -->  0.0 seconds
```



```python
# NUMPY LIBRARY
import numpy as np
a=np.array([[1,2,3,4,5],[2,3,7,53,3]]) #2-D array
b = np.array(["hello","stay","positive"])
print(" Array a is \n",a)
print(" Datatype of array a is ",a.dtype)
print(" Array a has {} dimensions and Array b has {} dimension".format(a.ndim,b.ndim))

# I want Array b to be two dimensional array.
print("\n Array b is ",b)
b = np.array(["hello","stay","positive"],ndmin=2)
print(" Array b is now two dimensional ",b)

print(" \n I want to change datatype of Array a to complex number")
a = np.array([[1,2,3,4,5],[2,3,7,53,3]],dtype=complex)
print(" New Array a is :\n",a)

#print(np.dtype('i1'))
```

```
OUTPUT
Array a is
[[ 1  2  3  4  5]
[ 2  3  7 53  3]]
Datatype of array a is  int32
Array a has 2 dimensions and Array b has 1 dimension

Array b is  ['hello' 'stay' 'positive']
Array b is now two dimensional  [['hello' 'stay' 'positive']]

I want to change datatype of Array a to complex number
New Array a is :
[[ 1.+0.j  2.+0.j  3.+0.j  4.+0.j  5.+0.j]
[ 2.+0.j  3.+0.j  7.+0.j 53.+0.j  3.+0.j]]
```



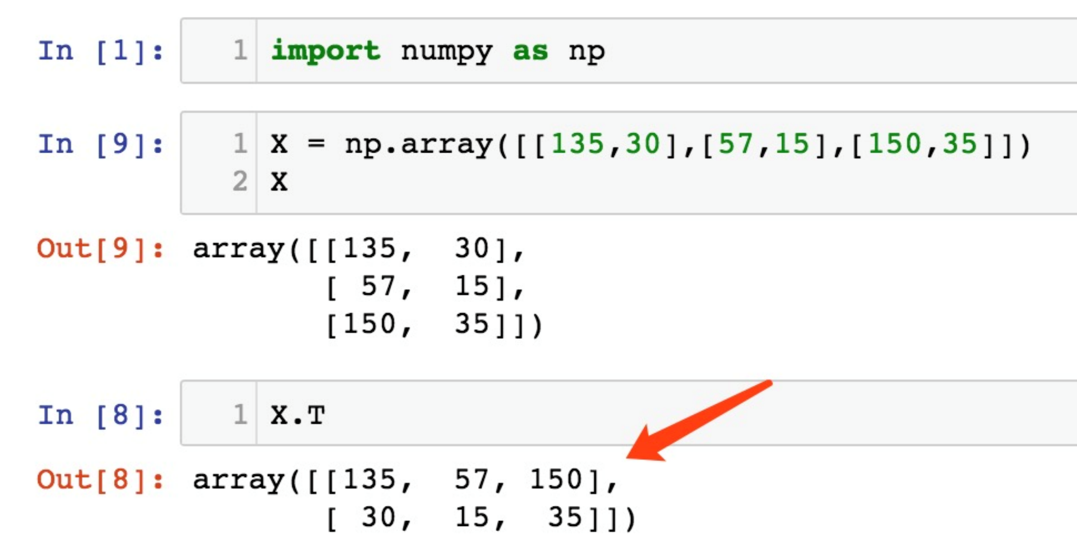
```
B = numpy.array([[n+m*5 for n in range(4)] for m in range(4)])
```

```
numpy.dot(B,B) #dot product of the matrix B x B
```

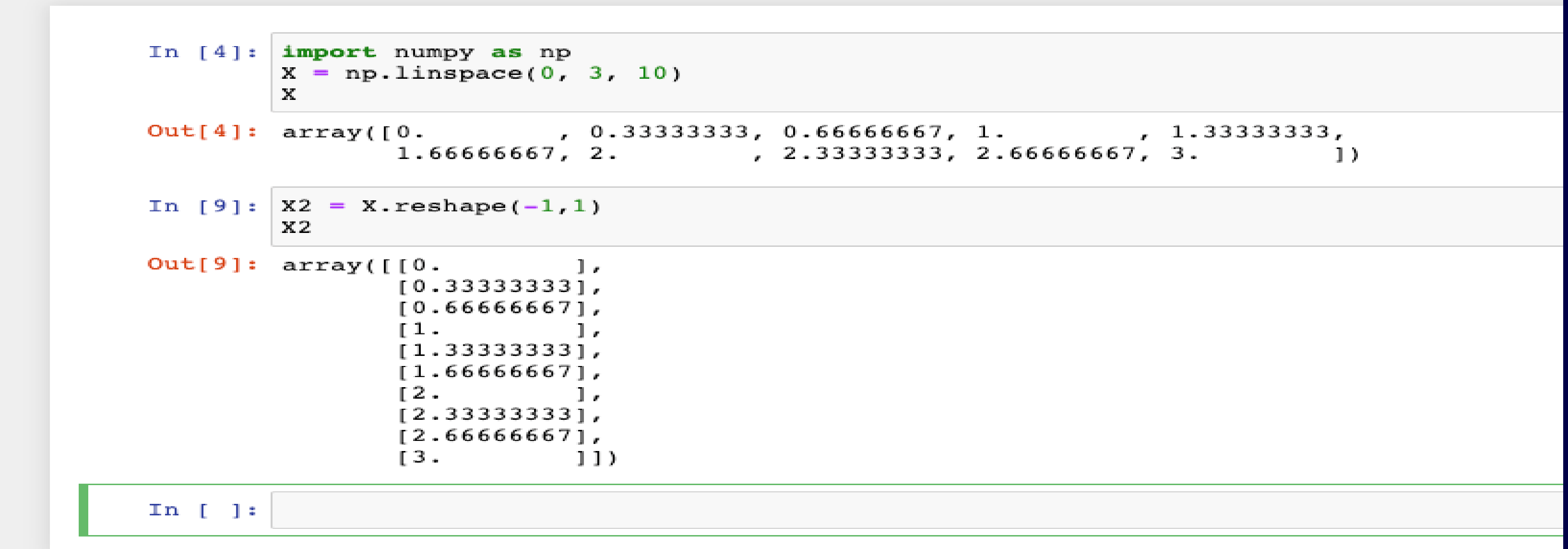
```
array([[ 70,  76,  82,  88],
       [220, 246, 272, 298],
       [370, 416, 462, 508],
       [520, 586, 652, 718]])
```

```
A = B.T #tranpose of the matrix B
A
```

```
array([[ 0,  5, 10, 15],
       [ 1,  6, 11, 16],
       [ 2,  7, 12, 17],
       [ 3,  8, 13, 18]])
```



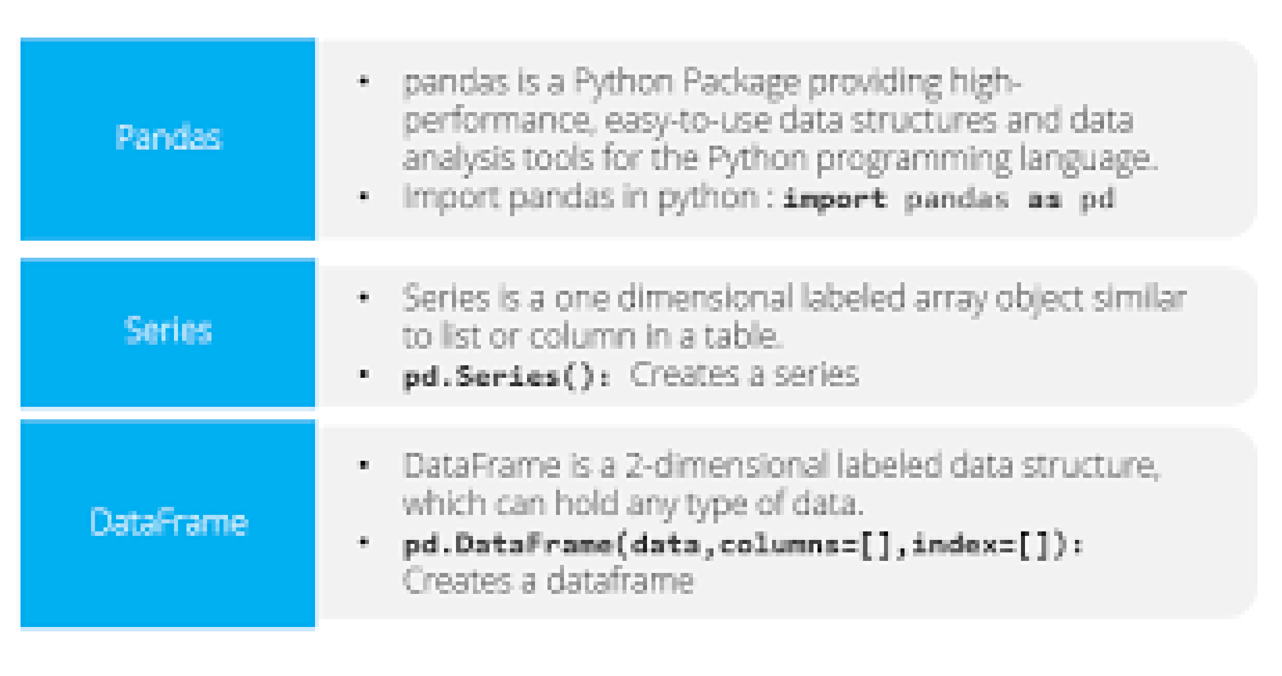
```
In [1]: import numpy as np

In [9]: X = np.array([[135,30],[57,15],[150,35]])
        X

Out[9]: array([[135,  30],
               [ 57,  15],
               [150,  35]])

In [8]: X.T

Out[8]: array([[135,  57, 150],
               [ 30,  15,  35]])
```



```
In [4]: import numpy as np
        X = np.linspace(0, 3, 10)
        X

Out[4]: array([0.        , 0.33333333, 0.66666667, 1.        , 1.33333333,
               1.66666667, 2.        , 2.33333333, 2.66666667, 3.        ])

In [9]: X2 = X.reshape(-1,1)
        X2

Out[9]: array([[0.        ],
               [0.33333333],
               [0.66666667],
               [1.        ],
               [1.33333333],
               [1.66666667],
               [2.        ],
               [2.33333333],
               [2.66666667],
               [3.        ]])

In [ ]:
```

```
import numpy as np

arr1 = np.random.randint(10, 50, size = (5, 8))
print('\n-----Two Dimensional Random Array----')
print(arr1)
print()
print(np.greater(arr1, 30))

arr2 = np.random.randint(1, 20, size = (2, 3, 6))
print('\n-----Three Dimensional Random Array----')
print(arr2)
print()
print(np.greater(arr2, 10))
```

```
import numpy as np

arr = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', arr)
print('Greater Than or Equal to 2 = ', np.greater_equal(arr, 2))

arr1 = np.random.randint(10, 50, size = (5, 8))
print('\n-----Two Dimensional Random Array----')
print(arr1)
print()
print(np.greater_equal(arr1, 25))

arr2 = np.random.randint(1, 15, size = (2, 3, 6))
print('\n-----Three Dimensional Random Array----')
print(arr2)
print()
print(np.greater_equal(arr2, 7))
```



Pandas

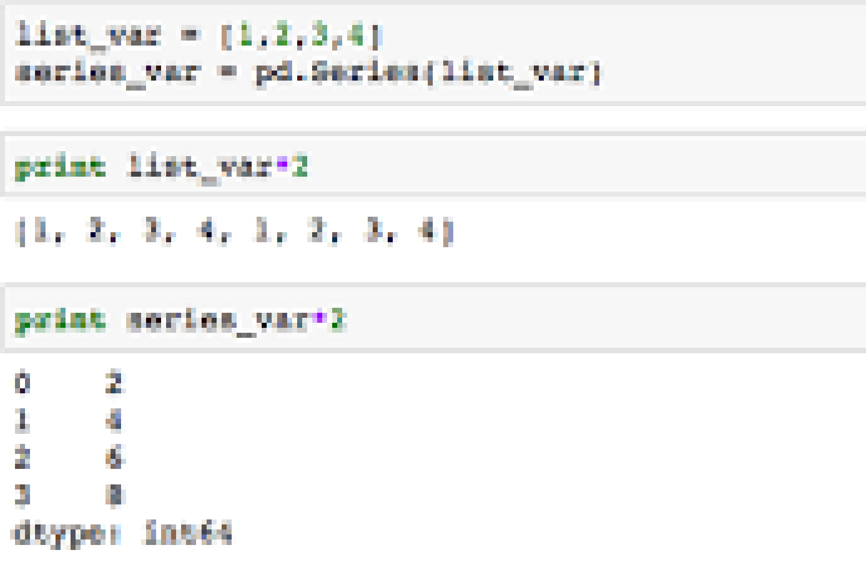
- pandas is a Python Package providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- Import pandas in python : `import pandas as pd`

Series

- Series is a one dimensional labeled array object similar to list or column in a table.
- `pd.Series():` Creates a series

DataFrame

- DataFrame is a 2-dimensional labeled data structure, which can hold any type of data.
- `pd.DataFrame(data,columns=[],index=[]):` Creates a dataframe



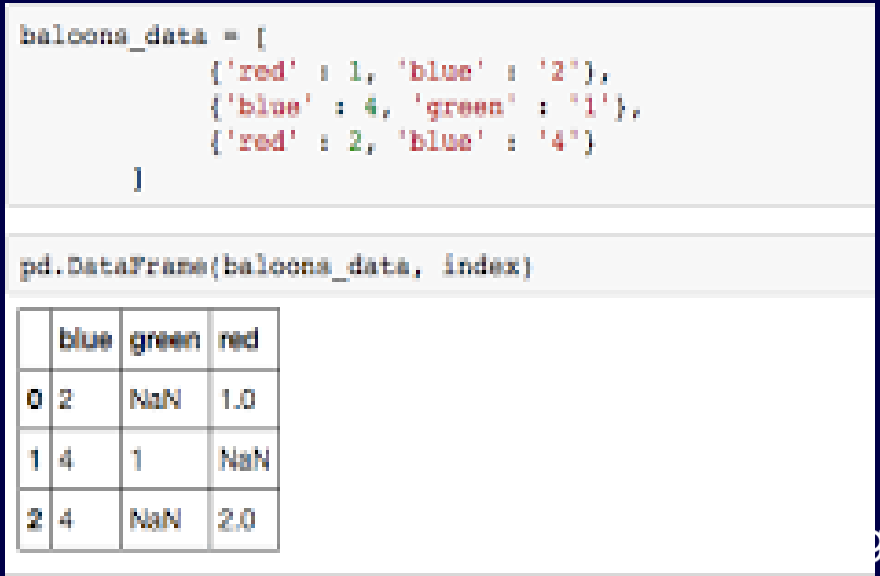
```
list_var = [1,2,3,4]
series_var = pd.Series(list_var)
```

```
print list_var*2
```

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

```
print series_var*2
```

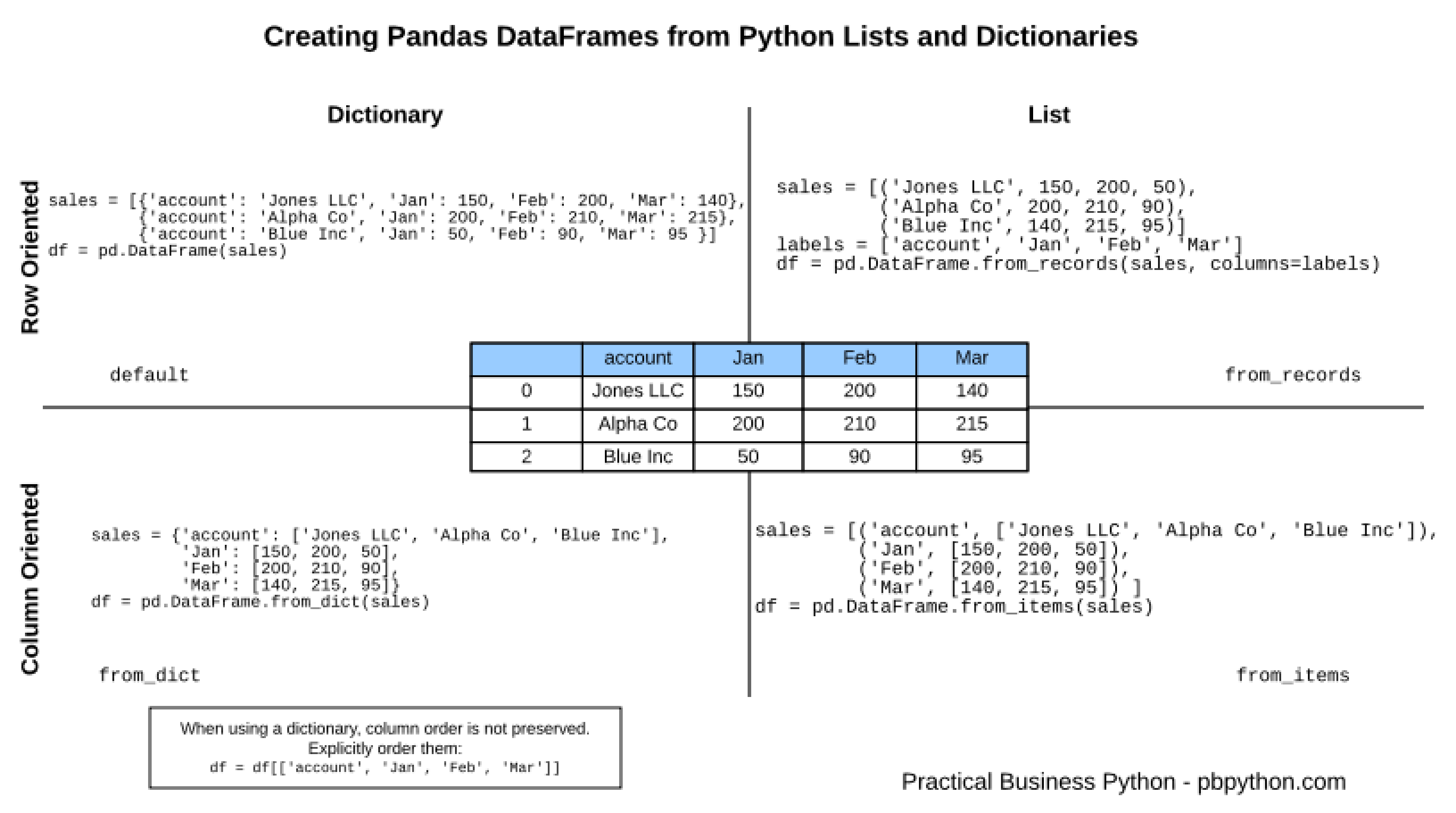
```
0    2
1    4
2    6
3    8
dtype: int64
```



```
baloons_data = [
            {'red' : 1, 'blue' : '2'},
            {'blue' : 4, 'green' : '1'},
            {'red' : 2, 'blue' : '4'}
        ]
```

```
pd.DataFrame(baloons_data, index)
```

	blue	green	red
0	2	NaN	1.0
1	4	1	NaN
2	4	NaN	2.0



Creating Pandas DataFrames from Python Lists and Dictionaries

Dictionary

Row Oriented

```
sales = [{'account': 'Jones LLC', 'Jan': 150, 'Feb': 200, 'Mar': 140},
         {'account': 'Alpha Co', 'Jan': 200, 'Feb': 210, 'Mar': 215},
         {'account': 'Blue Inc', 'Jan': 50, 'Feb': 90, 'Mar': 95 }]
df = pd.DataFrame(sales)
```

default

List

```
sales = [('Jones LLC', 150, 200, 50),
          ('Alpha Co', 200, 210, 90),
          ('Blue Inc', 140, 215, 95)]
labels = ['account', 'Jan', 'Feb', 'Mar']
df = pd.DataFrame.from_records(sales, columns=labels)
```

from_records

	account	Jan	Feb	Mar
0	Jones LLC	150	200	140
1	Alpha Co	200	210	215
2	Blue Inc	50	90	95

Column Oriented

```
sales = {'account': ['Jones LLC', 'Alpha Co', 'Blue Inc'],
         'Jan': [150, 200, 50],
         'Feb': [200, 210, 90],
         'Mar': [140, 215, 95]}
df = pd.DataFrame.from_dict(sales)
```

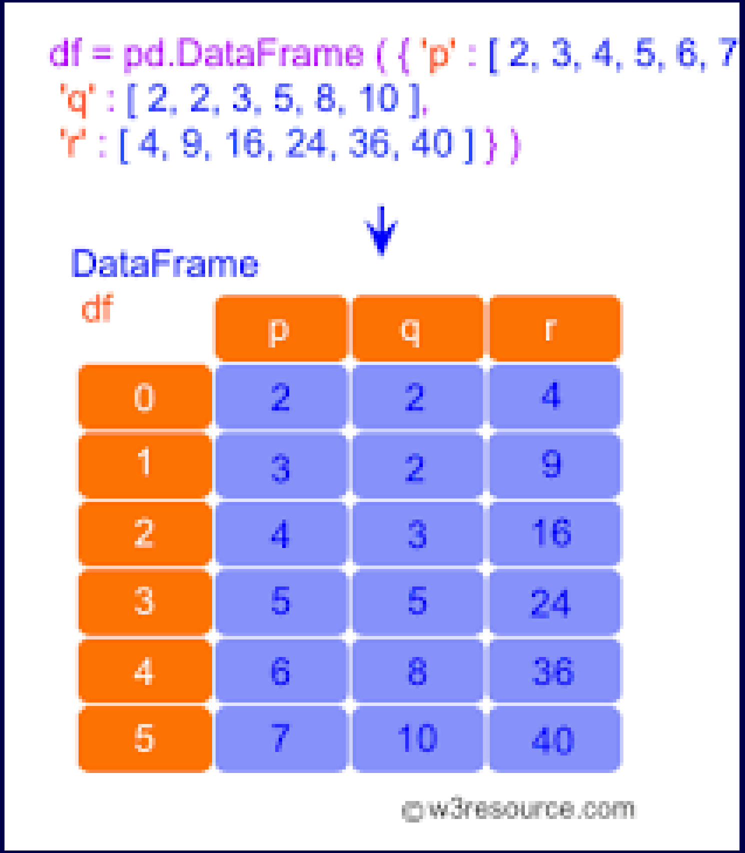
from_dict

```
sales = [('account', ['Jones LLC', 'Alpha Co', 'Blue Inc']),
         ('Jan', [150, 200, 50]),
         ('Feb', [200, 210, 90]),
         ('Mar', [140, 215, 95]) ]
df = pd.DataFrame.from_items(sales)
```

from_items

When using a dictionary, column order is not preserved.
Explicitly order them:
`df = df[['account', 'Jan', 'Feb', 'Mar']]`

Practical Business Python - pbpython.com

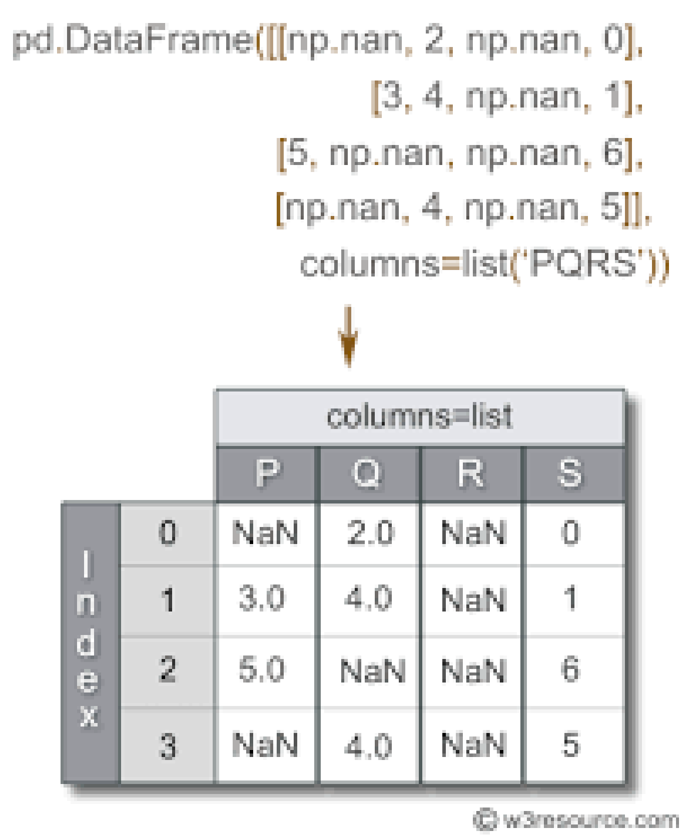


```
df = pd.DataFrame ( { 'p' : [ 2, 3, 4, 5, 6, 7
 'q' : [ 2, 2, 3, 5, 8, 10 ],
 'r' : [ 4, 9, 16, 24, 36, 40 ] } )
```

DataFrame
df

	p	q	r
0	2	2	4
1	3	2	9
2	4	3	16
3	5	5	24
4	6	8	36
5	7	10	40

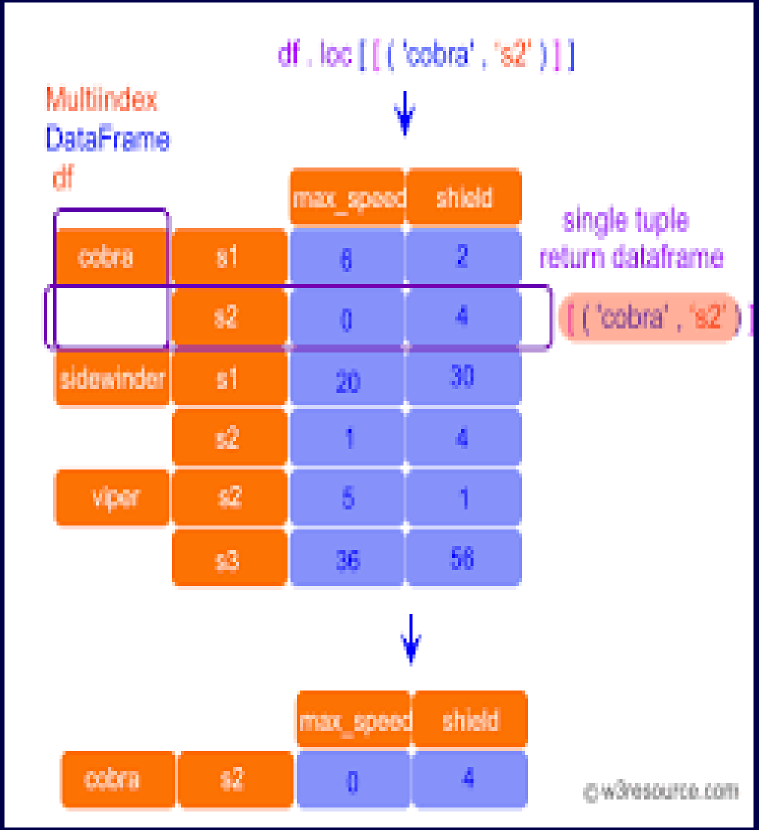
©w3resource.com



```
pd.DataFrame([[np.nan, 2, np.nan, 0],
                        [3, 4, np.nan, 1],
                     [5, np.nan, np.nan, 6],
                     [np.nan, 4, np.nan, 5]],
                      columns=list('PQRS'))
```

Index	columns=list P	 Q	 R	 S
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	5.0	NaN	NaN	6
3	NaN	4.0	NaN	5

© w3resource.com



```
df . loc [ [ ( 'cobra' , 's2' ) ] ]
```

Multiindex
DataFrame
df

		max_speed	shield
cobra	s1	6	2
	s2	0	4
sidewinder	s1	20	30
	s2	1	4
viper	s2	5	1
	s3	36	56

single tuple
return dataframe

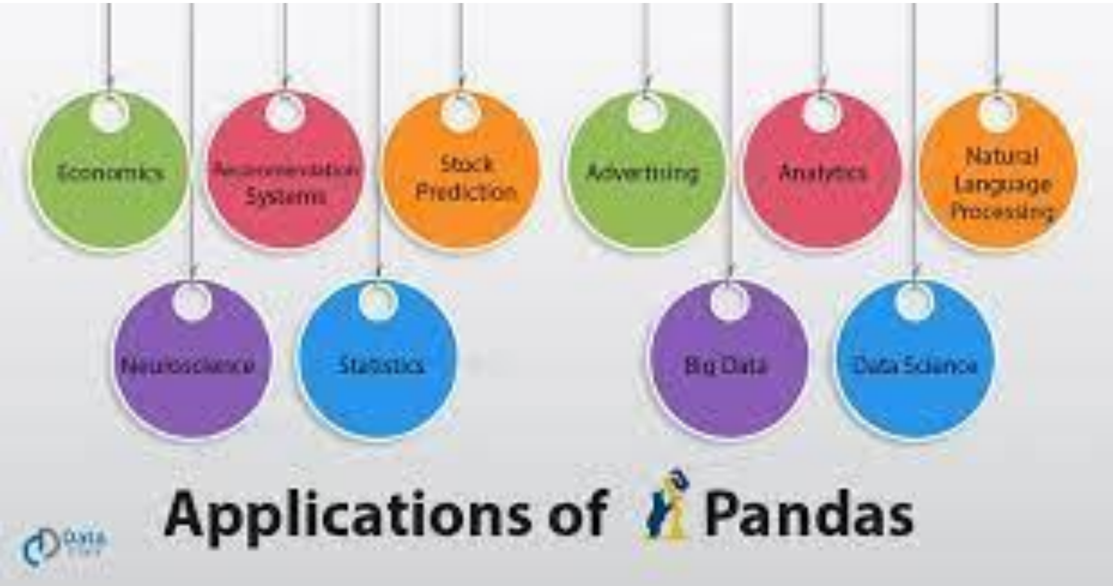
[('cobra' , 's2')]

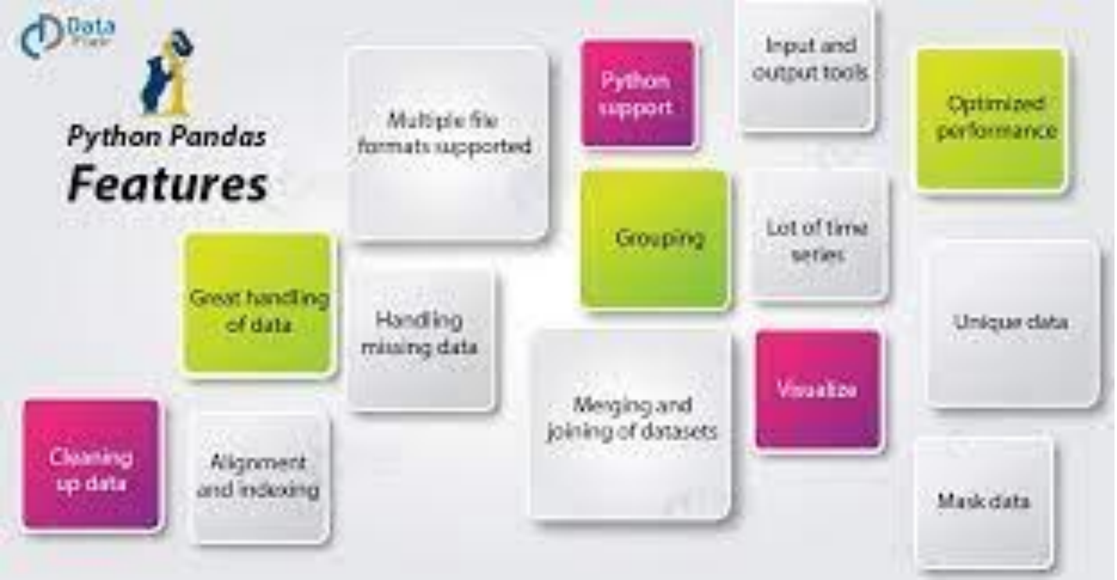
		max_speed	shield
cobra	s2	0	4

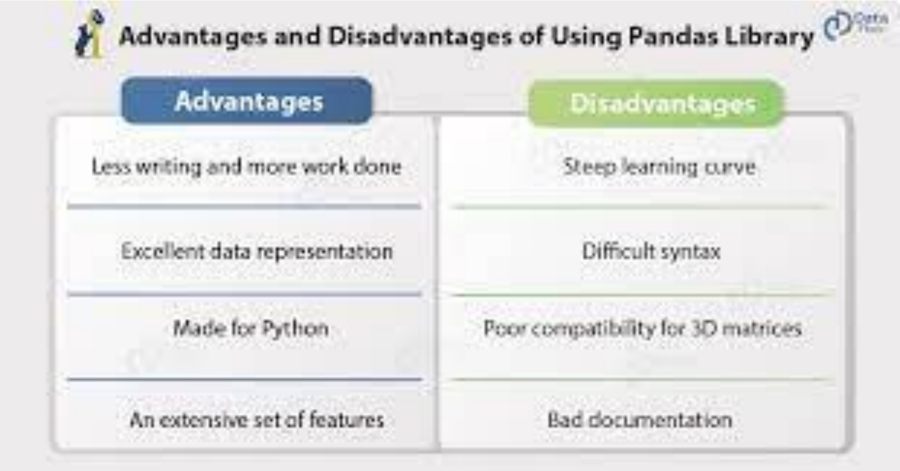
©w3resource.com

https://www.w3resource.com/pandas/series/series-loc.php

Usage of Pandas Library



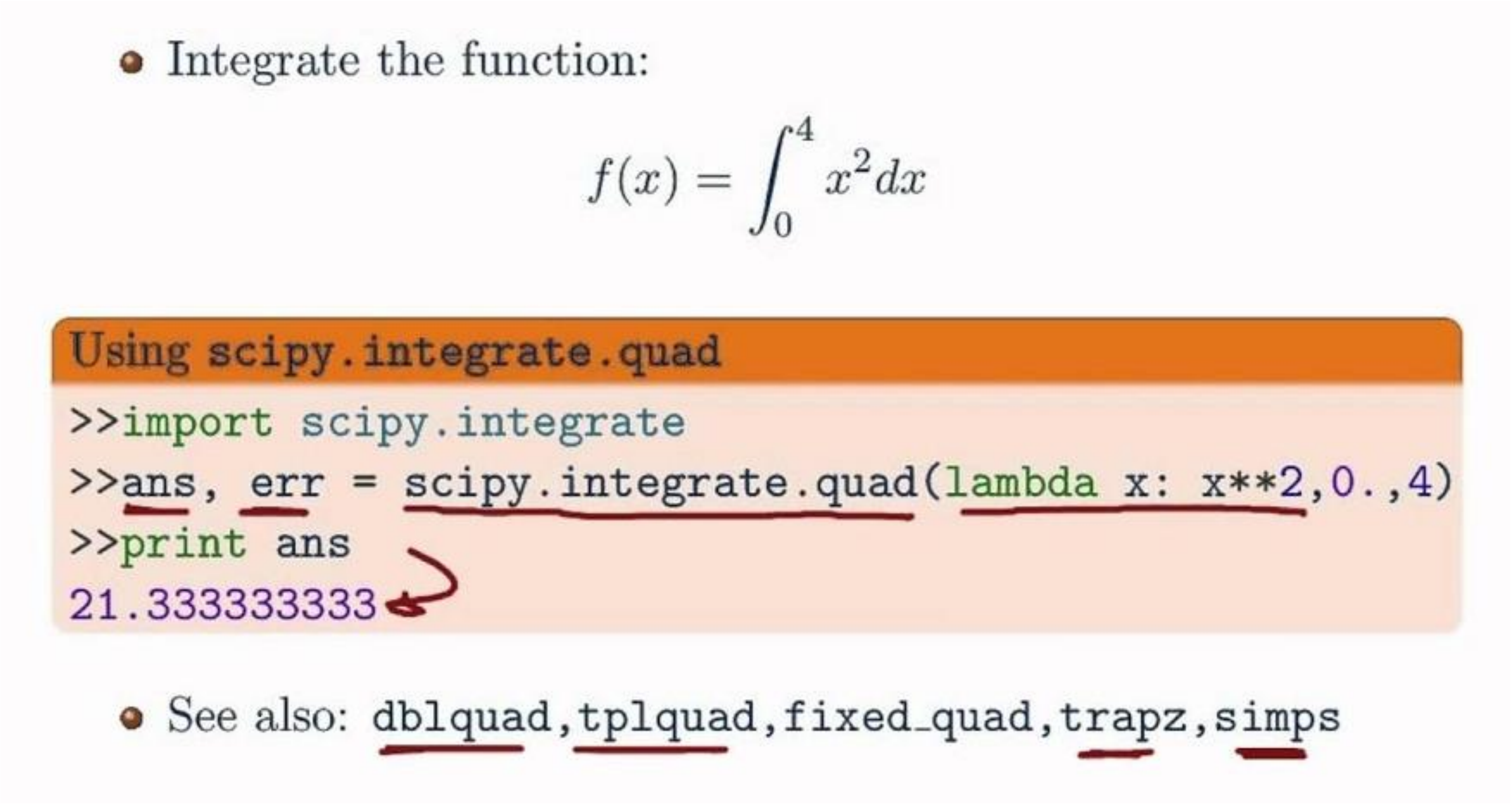




Advantages and Disadvantages of Using Pandas Library

Advantages	Disadvantages
Less writing and more work done	Steep learning curve
Excellent data representation	Difficult syntax
Made for Python	Poor compatibility for 3D matrices
An extensive set of features	Bad documentation

Scipy



- Integrate the function:

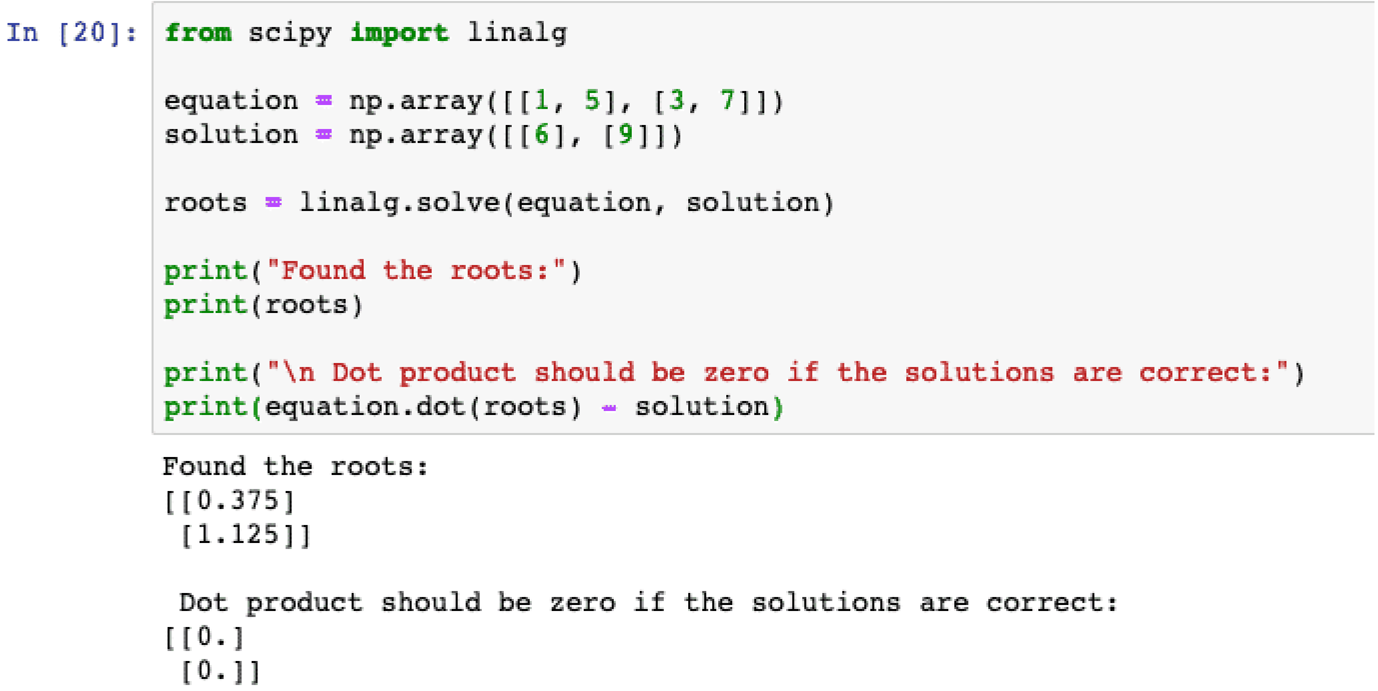
$$f(x) = \int_0^4 x^2 dx$$

Using scipy.integrate.quad

```
>>import scipy.integrate
>>ans, err = scipy.integrate.quad(lambda x: x**2,0.,4)
>>print ans
21.333333333
```

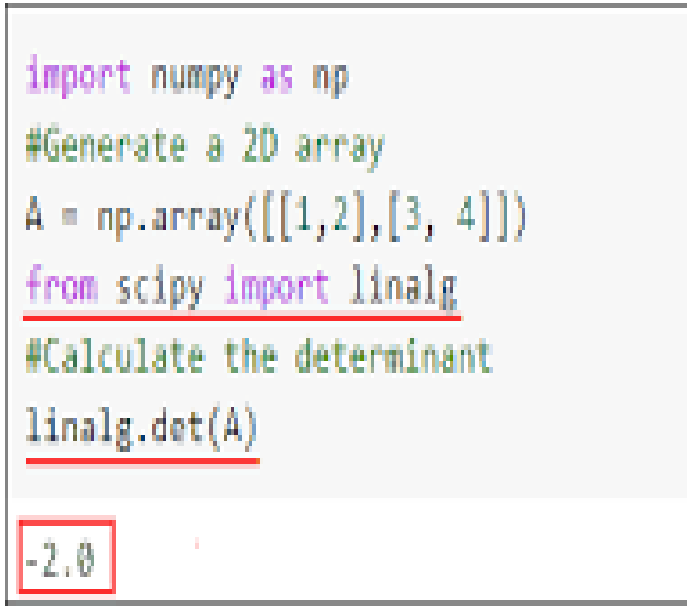
- See also: dblquad, tplquad, fixed_quad, trapz, simps

Scipy Library



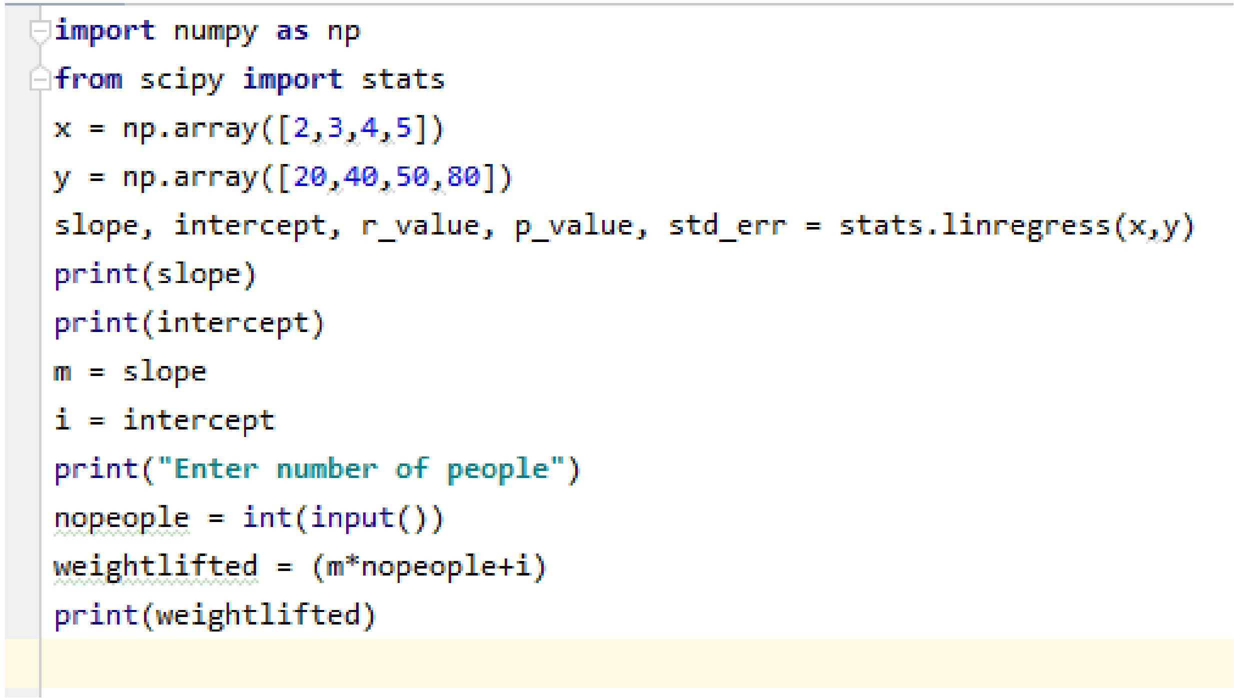
```
In [20]: from scipy import linalg

         equation = np.array([[1, 5], [3, 7]])
         solution = np.array([[6], [9]])

         roots = linalg.solve(equation, solution)

         print("Found the roots:")
         print(roots)

         print("\n Dot product should be zero if the solutions are correct:")
         print(equation.dot(roots) - solution)
```

```
Found the roots:
[[0.375]
 [1.125]]

 Dot product should be zero if the solutions are correct:
[[0.]
 [0.]]
```

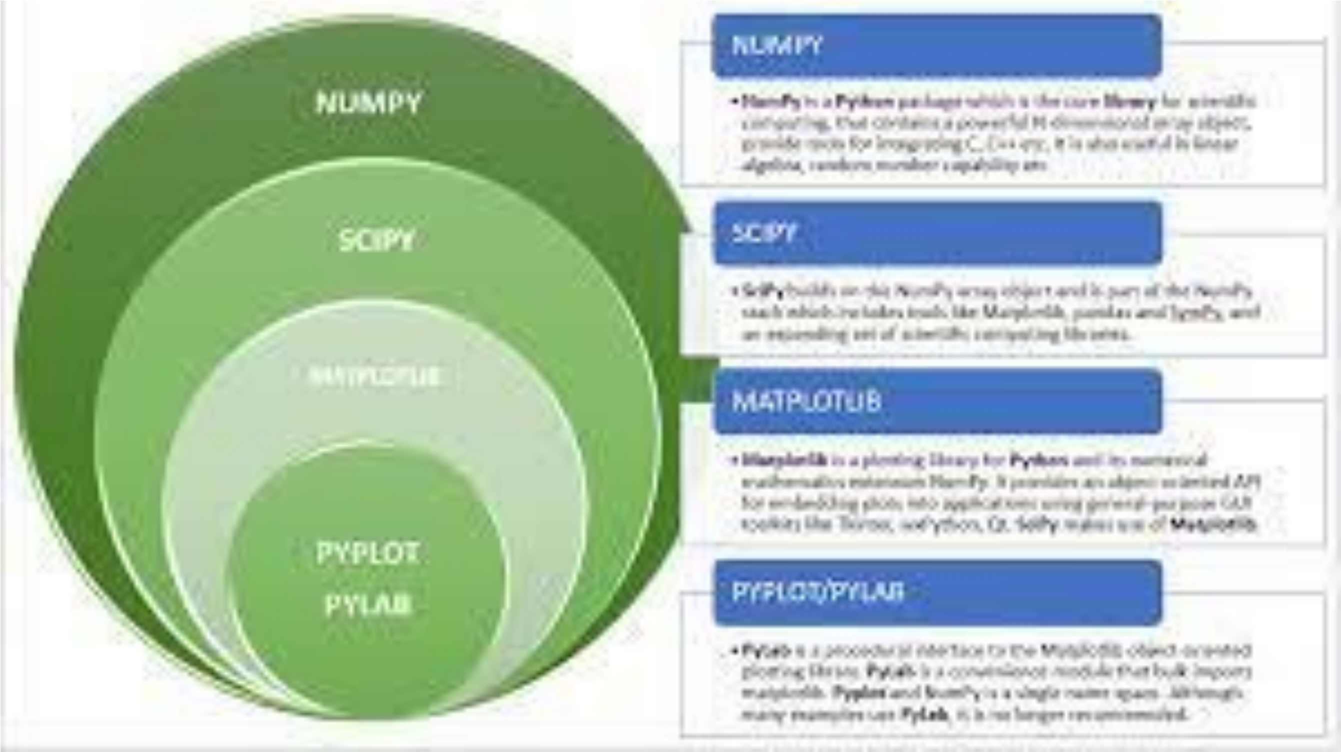


```
import numpy as np
#Generate a 2D array
A = np.array([[1,2],[3, 4]])
from scipy import linalg
#Calculate the determinant
linalg.det(A)
```

```
-2.0
```



```python
import numpy as np
from scipy import stats
x = np.array([2,3,4,5])
y = np.array([20,40,50,80])
slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
print(slope)
print(intercept)
m = slope
i = intercept
print("Enter number of people")
nopeople = int(input())
weightlifted = (m*nopeople+i)
print(weightlifted)
```



NUMPY

- **NumPy** is a **Python** package which is the core **library** for scientific computing, that contains a powerful N-dimensional array object, provide tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc.

SCIPY

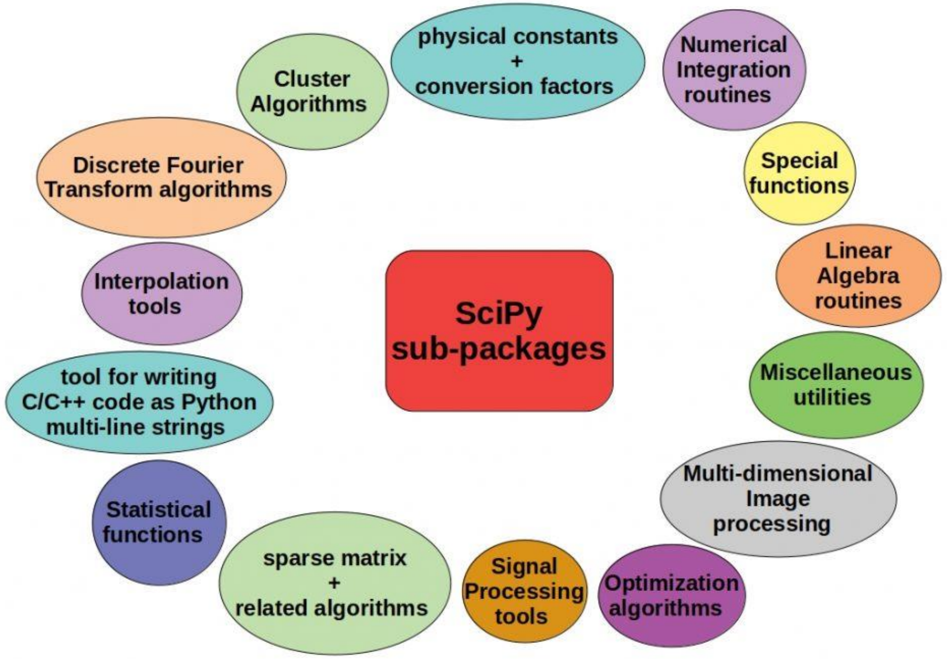
- **SciPy** builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries.

MATPLOTLIB

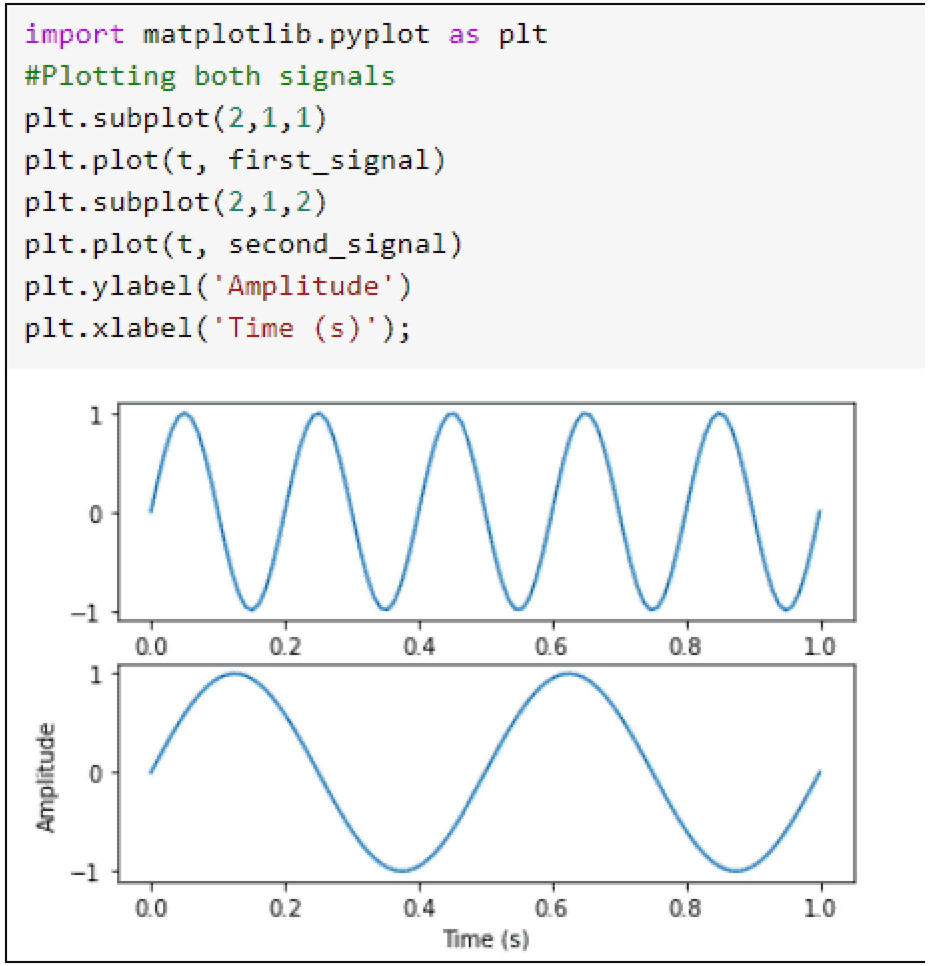
- **Matplotlib** is a plotting library for **Python** and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt. SciPy makes use of **Matplotlib**.

PYPLOT/PYLAB

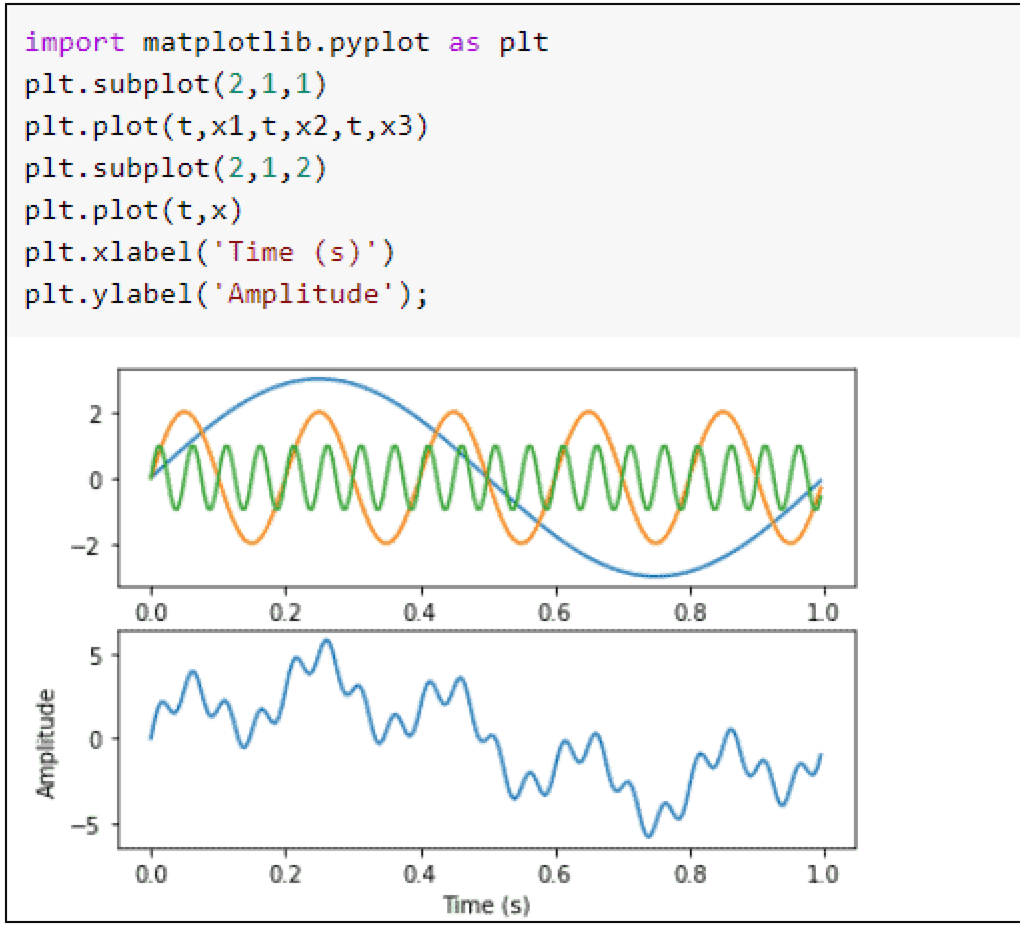
- **Pylab** is a procedural interface to the Matplotlib object-oriented plotting library. **Pylab** is a convenience module that bulk imports matplotlib. **Pyplot** and NumPy in a single name space. Although many examples use **Pylab**, it is no longer recommended.


SciPy sub-packages
Cluster Algorithms
physical constants + conversion factors
Numerical Integration routines
Special functions
Linear Algebra routines
Miscellaneous utilities
Multi-dimensional Image processing
Optimization algorithms
Signal Processing tools
sparse matrix + related algorithms
Statistical functions
tool for writing C/C++ code as Python multi-line strings
Interpolation tools
Discrete Fourier Transform algorithms

Matplot Library



```
import matplotlib.pyplot as plt
#Plotting both signals
plt.subplot(2,1,1)
plt.plot(t, first_signal)
plt.subplot(2,1,2)
plt.plot(t, second_signal)
plt.ylabel('Amplitude')
plt.xlabel('Time (s)');
```



```
import matplotlib.pyplot as plt
plt.subplot(2,1,1)
plt.plot(t,x1,t,x2,t,x3)
plt.subplot(2,1,2)
plt.plot(t,x)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude');
```

Day End

cognizant®

Thank you

Presenter’s name will go here

Contact information will go here