Advanced Programming

Prepared by
Prof. Dr. Gamal M. Behery
Prof. of AI and image processing
Damietta University-Faculty of Computers and AI

Book Part I

Revision

Function Basics

Python Functions

- Defined using keyword "**def**"
  - Followed by the function name, a pair of parentheses, and optional parameters, and the function body .

Return Statement

- Can return any data type, such list
- Return without parameter will return “None”
- Similarly, a function without a return statement will also return “None” when end of function is reached.

General Form for Functions without return

```
def functionname (arg1, arg2,..., argN ):
        # to do
```

Example:

```
# Function definition is here
def printme ( str ):
        print str
```

```
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

General Form for Functions with return

```
def functionname (arg1, arg2,..., argN ):
        # to do
        return Value
```

Example:

```
# Function definition is here
def printme( str ):
        print str
        return
```

```
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

def Executes at Runtime

```
if test:
      def func ():              # Define func this way
            ...
else:
      def func ():              # Or else this way
            ...
...
func()          # Call the version selected and built
```

Example:

```
# Function definition is here
 x=int (input('enter the value of x: '))
if x >= 10:
       def sumfun (x):
          y=x**2+5*x+8
          return y
else:
      def sumfun (x):
         y=4*x**2+7*x+8
         return y

print (sumfun(x))
```

```
enter the value of x: 12
212
enter the value of x: 8
320
```

```
def area_circle(radius):
        pi = 3.14159
        area = pi * radius**2
        return area
```

```
a = area_circle(1)
print(a)
print('A circle with radius 2 has area{:.2f}'. format(area_circle(2)))
r = 75.1
a = area_circle(r)
print("A circle with radius {:.2f} has area {:.2f}".format(r,a))
```

Function Operation

- Standard Function
- Function with a default input value

```
def input_a_number (min=0, max=100):
        …
```

python

Function

```
# a function
def printNameAndYear(name, byear):
        print("Name:  ", name)
        print("Byear: ", byear)

# a function
def printNameAndYear2(name, byear = 2000):
        print("Name:  ", name)
        print("Byear: ", byear)

# a function
def sum(no1, no2):
        result = no1 + no2
        return result
```

Standard Function

Function with a default input value

Function with a return value

Output

```
-
```

python

Function (Continued)

```
print("We execute    printNameAndYear( byear=1998, name=\"Raymond\")")
printNameAndYear( byear=1998, name="Raymond")

print("We execute    printNameAndYear2( byear=1998, name=\"Raymond\")")
printNameAndYear2( byear=1998, name="Raymond")

print("We execute    printNameAndYear2( name=\"Raymond\")")
printNameAndYear2( name="Raymond")
```

The order of input arguments does not matter since we specify the input parameter names

We could decide not to specify "byear" since there is a default value of "byear"

Output

```
We execute    printNameAndYear( byear=1998, name="Raymond")
Name:   Raymond
Byear:  1998

We execute    printNameAndYear2( byear=1998, name="Raymond")
Name:   Raymond
Byear:  1998

We execute    printNameAndYear2( name="Raymond")
Name:   Raymond
Byear:  2000
```

python

Function (Continued)

```
        no1 = 12
        no2 = 15
        print("We execute \"result = sum(no1, no2)\" where no1 and no2 are {} and {}".format(no1,
no2))
        result = sum(no1, no2)
        print("We execute \"print(result)\"")
        print(result)
```

Output

```
We execute "result = sum(no1, no2)" where no1 and no2 are 12 and 15
We execute "print(result)"
27
```

Examples

Ex 1:

Write a function to add 2 numbers and display the result.

Ex 1: Answer

```
def addNumber(n1 , n2=1 ):
    print(n1+n2)

addNumber(5.5,3)
addNumber(3,1)
a=int(input("enter first number"))
b=int(input("enter second number"))
addNumber(a,b)
```

Ex 2:

Write a function to add 2 numbers.

Ex 2: Answer

```
def addNumber(n1 , n2 ):
    return n1+n2

print(addNumber(5.5,3))
print(addNumber(3,1))
a=int(input("enter first number"))
b=int(input("enter second number"))
print(addNumber(a,b))
```

Ex 3:

Write a function to add 2 numbers or increment a given number.

Ex 3: Answer

```
def addNumber(n1 , n2=1 ):
    return n1+n2

print(addNumber(5.5,3))
#print(addNumber(3,1))
print(addNumber(3))
a=int(input("enter first number"))
b=int(input("enter second number"))
print(addNumber(a,b))
```

Ex 4:

Write a function to add 2 numbers.

Write a Program to ask the user to add numbers using the function n times.

Ex 4: Answer

```
def addNumber(n1 , n2=1 ):
    return n1+n2

n=int(input("number of times:"))
for i in range(n):
    a=int(input("enter first number"))
    b=int(input("enter second number"))
    print(addNumber(a,b))
```

Example 5:

Write a function to add 2 numbers.

Ask the user to add numbers using the function n times. But if the user enters a zero value the program terminates sending a notification message “Program Terminates!!” message.

Ex 5: Answer

```
def addNumber(n1 , n2=1 ):
    return n1+n2
n=int(input("number of times:"))
for i in range(n):
    a=int(input("enter first number"))
    b=int(input("enter second number"))
    if(a==0 or b==0):
        break
    print(addNumber(a,b))
if(i<n):
    print("Program Terminates!!")
```

Ex 6:

Write a function to add 2 numbers.

Ask the user to add numbers using the function n times. But if the user enters a zero value the program terminates.

The program sends a notification message “The End of all Trials. Bye!” incase of never entering a zero value.

Ex 6: Answer

```
def addNumber(n1 , n2=1 ):
    return n1+n2

n=int(input("number of times:"))
for i in range(n):
    a=int(input("enter first number"))
    b=int(input("enter second number"))
    if(a==0 or b==0):
        break
    print(addNumber(a,b))

else:
    print("The End of all Trials. Bye!")
```

Assignment

Write a python project using the python functions to compute the following equation:

$$NCR = \frac{N!}{R!\ (N-R)!}$$

```
def fact(x):
    if x == 0 or x ==1 :
        fac = 1
    else:
        fac = 1
        for f in range (2,x+1):
            fac = fac * f
    return fac

N = int(input('enter the value of N: '))
R = int(input('enter the value of R: '))
NCR = fact(N)/(fact(R) * fact(N-R))
print ('the value of NCR = ',NCR)
```

OUTPUT

enter the value of N: 6
enter the value of R: 4
the value of NCR = 15.0

Assignment

Write a python project using the python functions to compute the following equation:

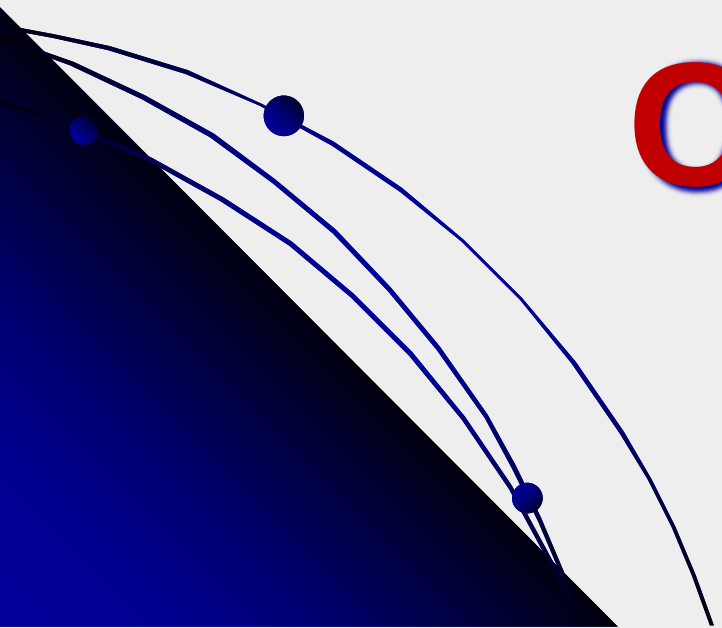
$$NCR = \frac{N! \sum_{i=0}^{i=N-1} i}{R!\ (N-R)!}$$

```
def fact(x):
    if x == 0 or x ==1 :
        fac = 1
    else:
        fac = 1
        for f in range (2,x+1):
            fac = fac * f
    return fac
def sumi(Z):
        s = 0
        for i in range (N) :
                s += i
```

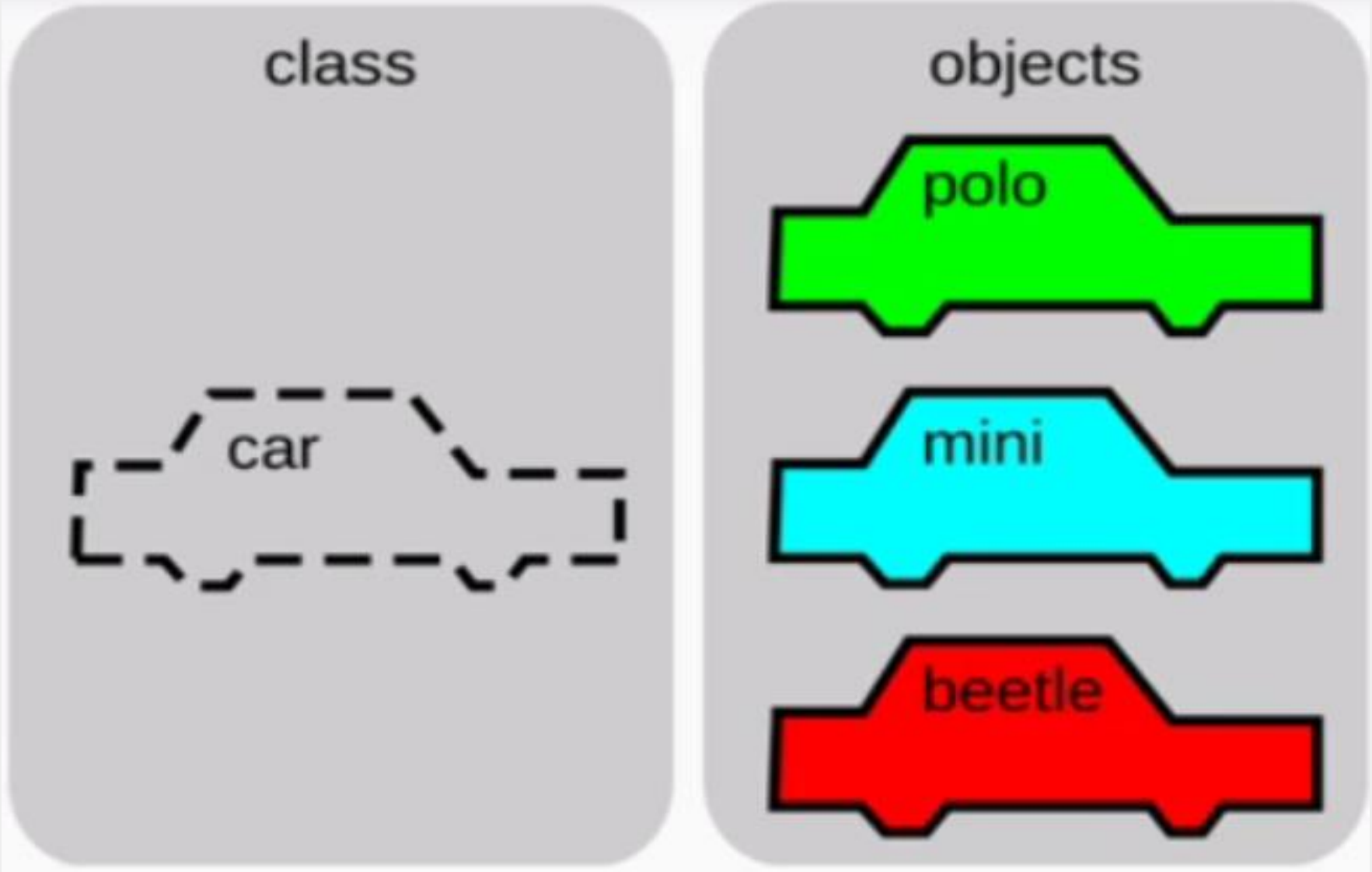
```
N = int(input('enter the value of N: '))
R = int(input('enter the value of R: '))
NCR = (fact(N)*sumi(N-1))/(fact(R) * fact(N-R))
print ('the value of NCR = ',NCR)
```

Object oriented programming

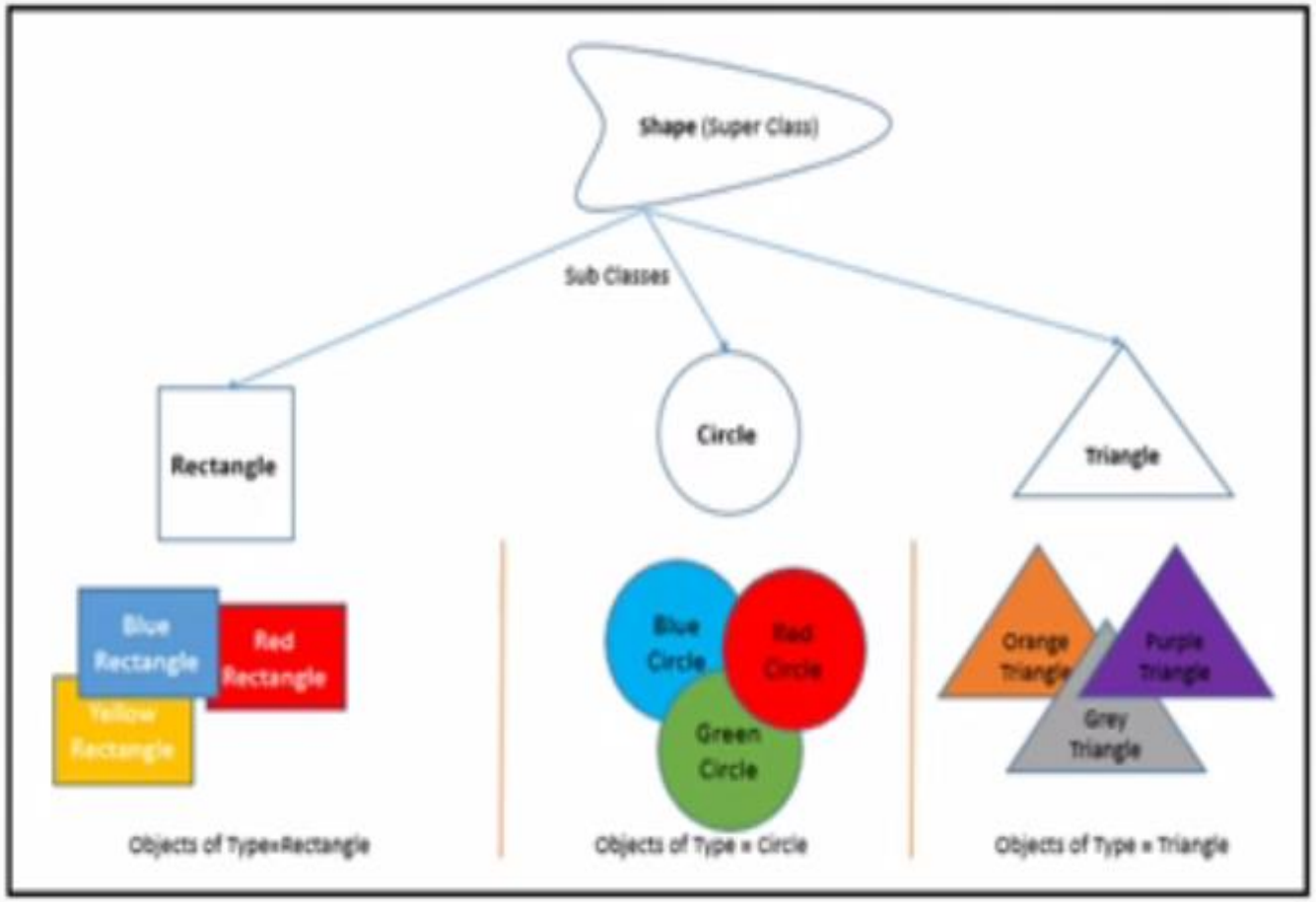
Classes

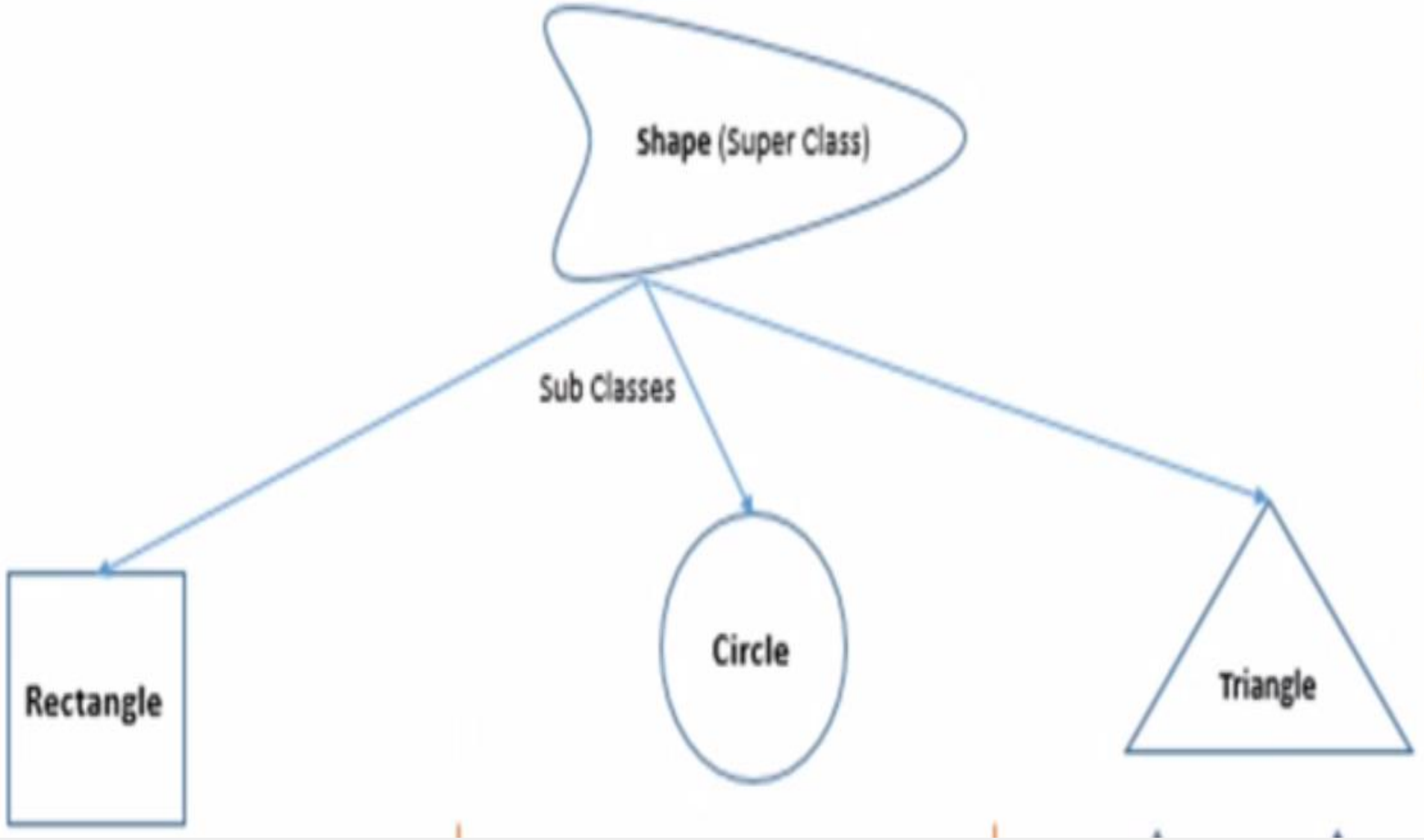


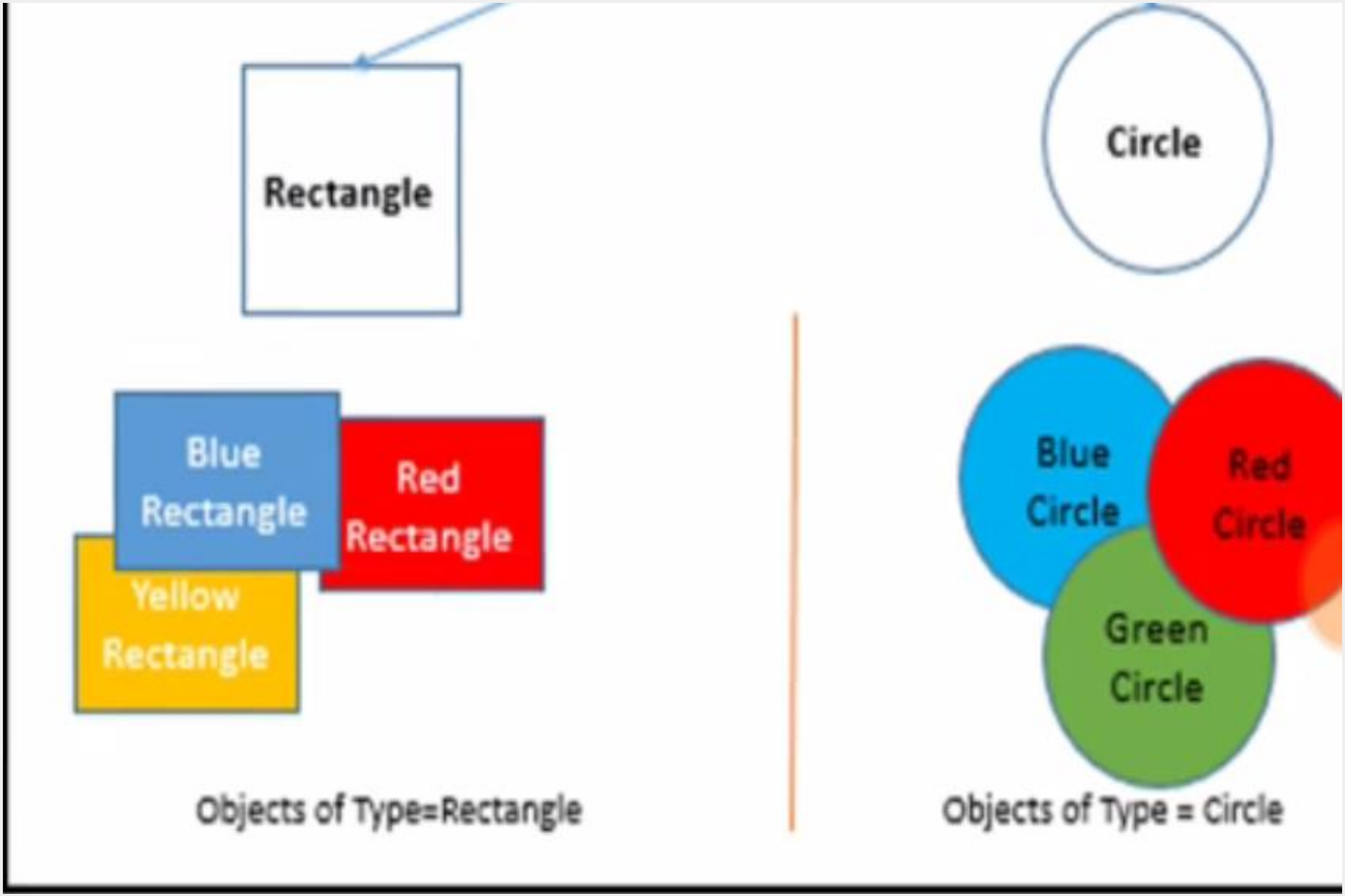
أداة الـ Class

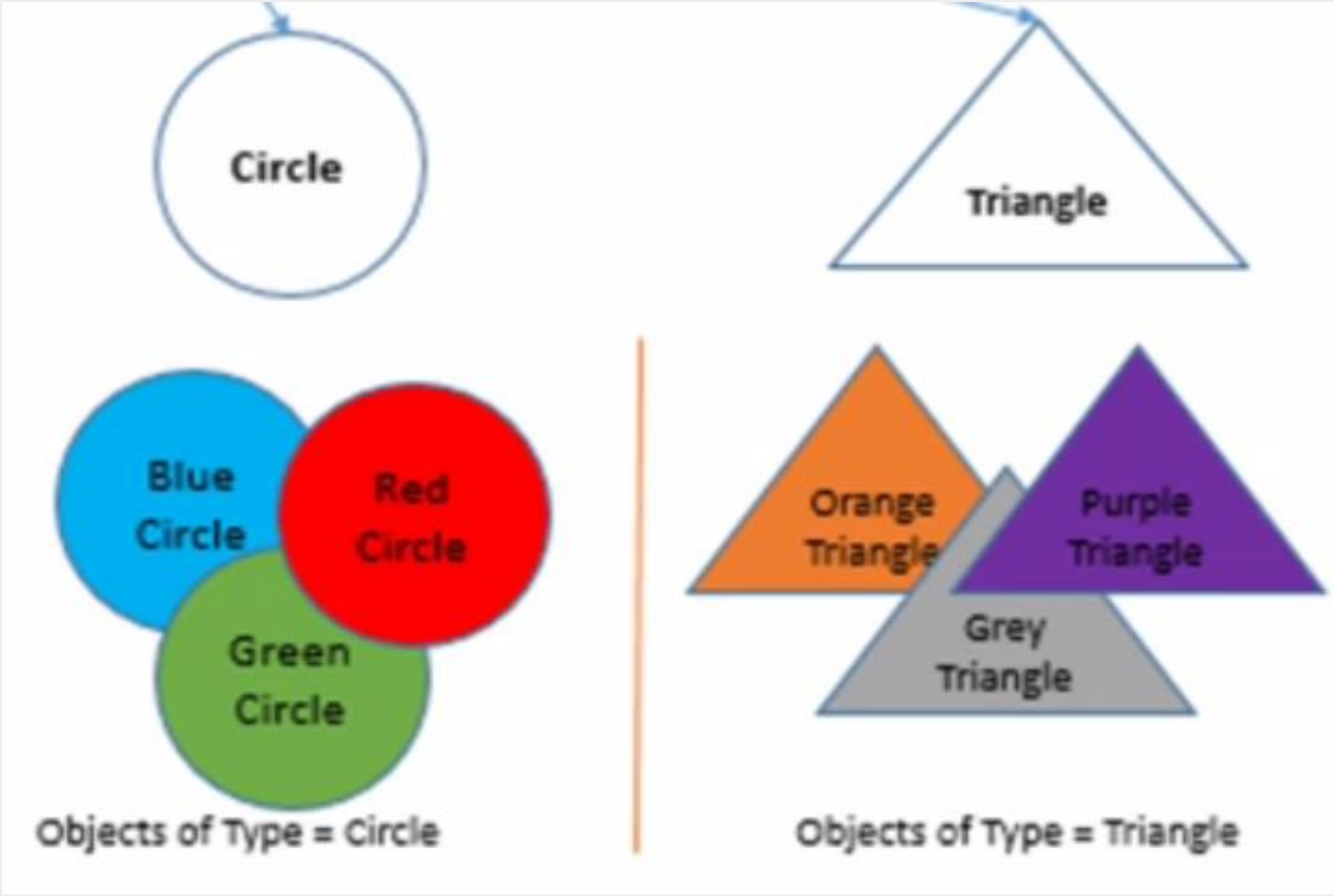


أداة الـ Class

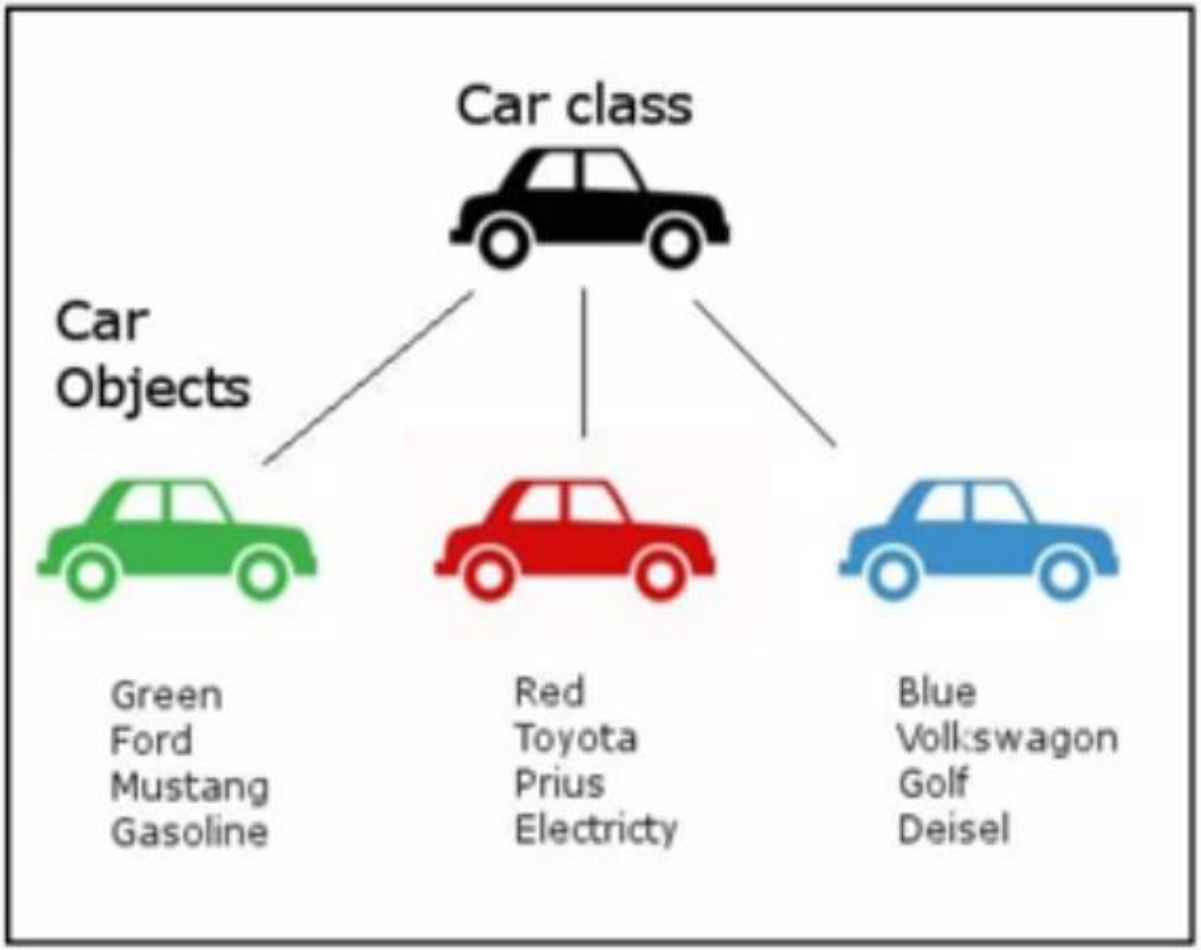



Shape (Super Class)
Sub Classes
Rectangle
Circle
Triangle


Rectangle
Blue Rectangle
Red Rectangle
Yellow Rectangle
Objects of Type=Rectangle
Circle
Blue Circle
Red Circle
Green Circle
Objects of Type = Circle


Circle
Triangle
Blue Circle
Red Circle
Green Circle
Orange Triangle
Purple Triangle
Grey Triangle
Objects of Type = Circle
Objects of Type = Triangle

أداة الـ Class



ممكن عمل كلاس جديدة عن طريق اكتب class بعدها اسمها بعدها نقطتين , ثم المتغيرات او الدوال

untitled6.py*

```
1 class car :
2     length = 15
3     width = 8
4     height = 6
5     color = 'white'
6     volume = length * width * height
7
8 car.
9
```

- color
- height
- length
- volume
- width

```
class car:
    length = 490
    heigh = 160
    width = 150
    color = 'white'
    volume = length*heigh*width
print(car.color)
```

The Output

======= RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl1.py ======

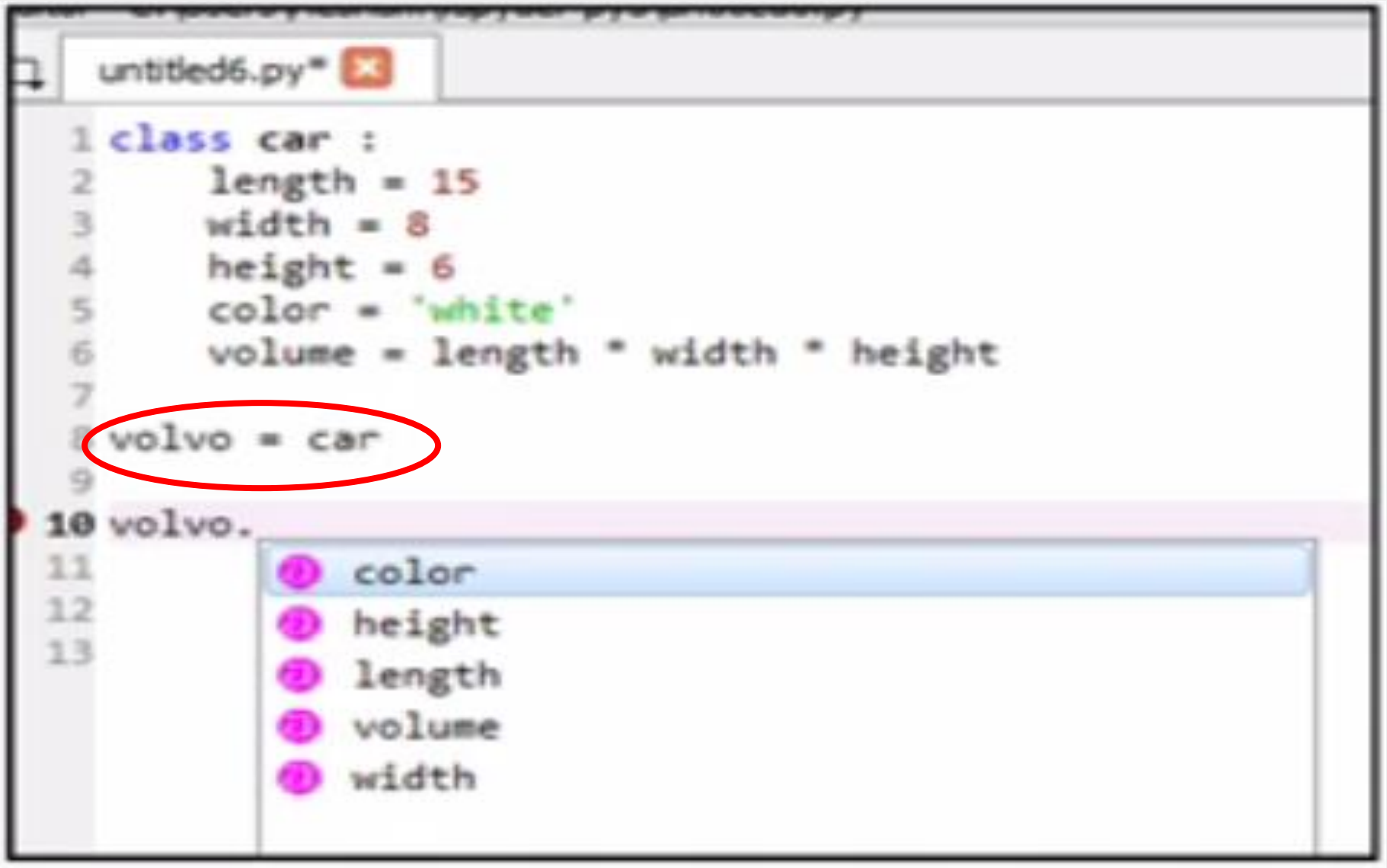
white

```
class car:
    length = 490
    heigh = 160
    width = 150
    color = 'white'
    volume = length*heigh*width
print(car.volume)
```

The Output

```
======= RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl1.py ======
11760000
```

لما اقول ان فيه object يساوي الكلاس , يكون ليه نفس بياناته



```
class car:
    length = 490
    heigh = 160
    width = 150
    color = 'white'
    volume = length*heigh*width
volvo = car
print(volvo.color)
```

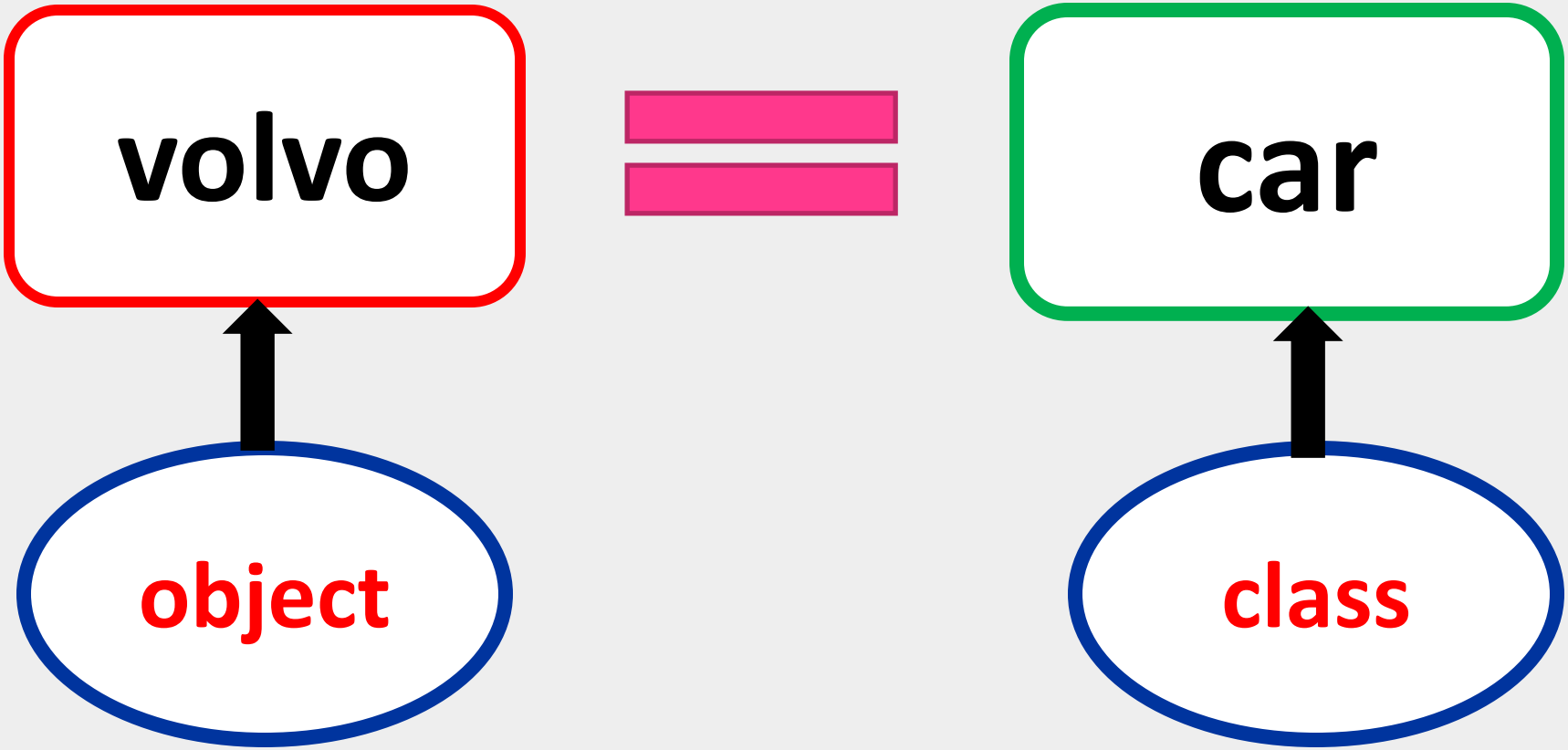
The Output

```
======= RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl1.py ======
white
```

```
class car:
    length = 490
    heigh = 160
    width = 150
    color = 'white'
    volume = length*heigh*width
volvo = car
print(volvo.volume)
```

The Output

```
======= RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl1.py ======
11760000
```


volvo
car
object
class

Note

عندما يكون هناك Object يساوي الـ Class

يكون له نفس بياناته

Note

إذا تم حفظ الـ Class بامتداد py ممكن أن يتم استدعاء واستخدام جميع الدوال والمتغيرات له.

مثال: form myclasses import car

أو استدعاء جميع الـ Classes.

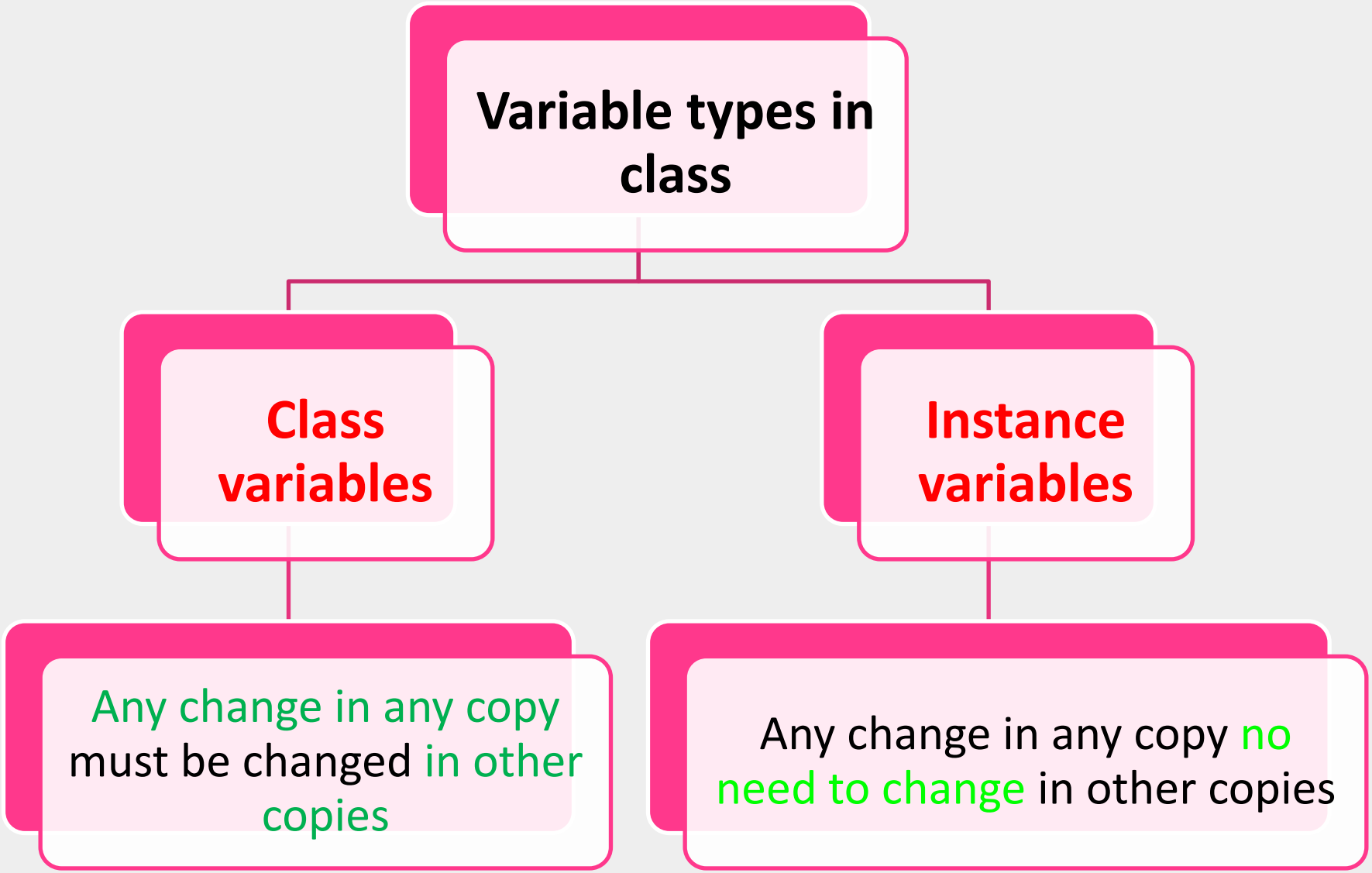
مثال: form myclasses import *

أداة الـ Class

و بشكل عام فيه نوعين من المتغيرات في اي كلاس

الـ Class Variable اللي هو المتغير اللي قيمته بتسمع في كل النسخ اللي بتتاخد منه , فلو اتغير من احد النسخ , قيمته تتغير في الاصل , وبالتالي باقي النسخ تشوف التغيير

الـ Instance variable اللي هو المتغير اللي قيمته في كل نسخة بتكون خاصة بيه و مش بتاثر في قيمته الاصلية او قيمته في باقي النسخ


Variable types in class
Class variables
Instance variables
Any change in any copy must be changed in other copies
Any change in any copy no need to change in other copies

Example for Class variables

```
class car:
      color = 'blue'
volvo = car
nissan = car

volvo.color = 'white'
nissan.color = ‘green’

print(volvo.color)
print(nissan.color)
```

The Output

```
======= RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl1.py ======
green
green
```

Instance Variables

- في حال الرغبة في أن كل سيارة يكون لها لونها الخاص.
- أي ان التغير في قيمة copy لا تستلزم تغيير بقية copies.
- عندها لابد من استخدام Instance Variables.
- يكون ذلك باستخدام الأداة self

```
1 class car() :
2     def __init__(self, color):
3         self.color = color
4
5 volvo = car('white')
6 nissan = car('green')
7
8 print(volvo.color)
9 print(nissan.color)
```

Example for Instance variables

```
class car():
    def __init__(self, color):
        self.color = color

volvo = car('white')
nissan = car('green')

volvo.color = 'white'
nissan.color = 'grren'

print(volvo.color)
print(nissan.color)
```

The Output

======= RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl1.py ======

white

grren

Another example for *self* function

```
1 class calc :
2     def __init__(self,p2,p3) :
3         self.power2 = p2**2
4         self.power3 = p3**3
5
6 a = calc(40,50)
7 print(a.power2)
8 print(a.power3)
```

Another example for *self* function

```
class calc:
    def __init__(self, p2,p3):
        self.power2 = p2**2
        self.power3 = p3**3
a = calc(40,50)

print(a.power2)
print(a.power3)
```

The Output

==== RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl_self_3.py ===

1600

125000

ممكن عمل قيم مفترضة للمعاملات

```
1 class d :
2     def __init__(self,p2 = 10 ,p3 = 20) :
3         self.power2 = p2**2
4         self.power3 = p3**3
5 a = d()
6 print(a.power2)
7 print(a.power3)
```

```
class d:
    def __init__(self, p2 = 10, p3 = 20):
        self.power2 = p2**2
        self.power3 = p3**3

a = d()

print(a.power2)
print(a.power3)
```

The Output

```
==== RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl_self_4.py ===
100
8000
```

وممكن اقوم انا بتعيينها و الغاء القيم المفترضة

untitled6.py

```
1 class d :
2     def __init__(self,p2 = 10 ,p3 = 100) :
3         self.power2 = p2**2
4         self.power3 = p3**3
5 a = d(5,3)
6 print(a.power2)
7 print(a.power3)
8
```

```
In [66]:
Users/Hes
25
27
```

او كدة

untitled6.py*

```
1 class d :
2     def __init__(self,p2 = 10 ,p3 = 100) :
3         self.power2 = p2**2
4         self.power3 = p3**3
5 a = d(p2=5,p3=3)
6 print(a.power2)
7 print(a.power3)
```

```
In [67]:
Users/Hes
25
27
```

```
class d:
    def __init__(self, p2 = 10,p3 = 20):
        self.power2 = p2**2
        self.power3 = p3**3

a = d(p3=100, p2=5)

print(a.power2)
print(a.power3)
```

لاحظ بشدةِ

The Output

```
==== RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl_self_5.py ===
25
1000000
```

طيب لو عايز اعمل تعريف لميثود :

الـ methods هي نفسها الدوال المستخدمة في البرمجة العادية

اكتب اسم الدالة root و يكون فيها عنصر هيتم استخدامه جوة الدالة اعمل الامر اللي عايزه print

من الـ init احدد عنصر يتم ادخاله لاستخدامه في الدالة في الدالة انده عليها اني اكتب roots.self هتساوي كذا

اخيرا انده علي الدالة بعد ما اكون دخلت القيم

```
1  class d :
2      def __init__(self,nn,p2 = 10 ,p3 = 100)
3
4          self.power2 = p2**2
5          self.power3 = p3**3
6          self.roots = nn
7
8      def root(self ) :
9          print (self.roots**0.5)
10
11 a = d(2500)
12 a.root()
```

وممكن التعامل مع الدوال بشكل مباشر من غير المرور بـ init

untitled6.py*

```
1 class d :
2     def square(n) :
3         print (n**2)
4
5     def summ(a,b,c,d,e,f) :
6         return ((a+b-c+e)*(d+f))
7
8
9 a = d
10 a.square(3)
11 print (a.summ(1,2,3,4,5,7))
12
```

```
Users/Hes
9
55
```

وممكن التعامل مع الدوال بشكل مباشر من غير المرور بـ init

```
1  class d :
2      def square(n) :
3          print (n**2)
4
5      def summ(a,b,c,d,e,f) :
6          return ((a+b-c+e)*(d+f))
7
8
9  a = d
10 a.square(3)
11 print (a.summ(1,2,3,4,5,7))
```

```
class d:
    def square(n):
        print(n**2)
    def summ(a,b,c,d,e,f):
        return ((a+b-c+e)*(d+f))

a = d
a.square(3)
print(a.summ(1,2,3,4,5,7))
```

The Output

```
==== RESTART: D:/Courses_2022_2023/Advanced Programming/classes/cl_self_7.py ===
9
55
```

```python
class Account:
    def __init__(self, name, account_number, initial_amount):
        self.name = name
        self.no = account_number
        self.balance = initial_amount
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def show(self):
        s = '%s, %s, balance: %s' % (self.name, self.no, self.balance)
        print (s)

a1 = Account('John Olsson', '19371554951', 20000)
a2 = Account('Liz Olsson', '19371564761', 50000)
print("a1 balance :  " , a1.balance )
print("a2 no      :  " ,   a2.no)

a1.deposit(1000)
a1.withdraw(4000)
a2.withdraw(10500)
a1.withdraw(3500)
print ("a1's balance:", a1.balance)
print ("a2's balance:", a2.balance)

a1.show()
a2.show()
```

```
In [90]: runfile('C:/Users/Hesham/.spyder-
Users/Hesham/.spyder-py3')
a1 balance :   20000
a2 no      :   19371564761
a1's balance: 13500
a2's balance: 39500
John Olsson, 19371554951, balance: 13500
Liz Olsson, 19371564761, balance: 39500
```

```
class account:
    def __init__(self, name, account_number,
initial_amount):
        self.name = name
        self.no = account_number
        self.balance = initial_amount

    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def show(self):
        s = '%s. %s, balance: %s' %(self.name, self.no,
self.balance)
        print (s)
```

```
a1 = account('John Olsson','19371554951',20000)
a2 = account('Liz Olsson','19371564761',50000)
print("a1 balance : = ", a1.balance)
print("a2 no : = ", a2.no)

a1.deposit(1000)
a1.withdraw(4000)
a2.withdraw(10500)
a1.withdraw(3500)

print("a1's balance : = ", a1.balance)
print("a2's balance : = ", a2.balance)

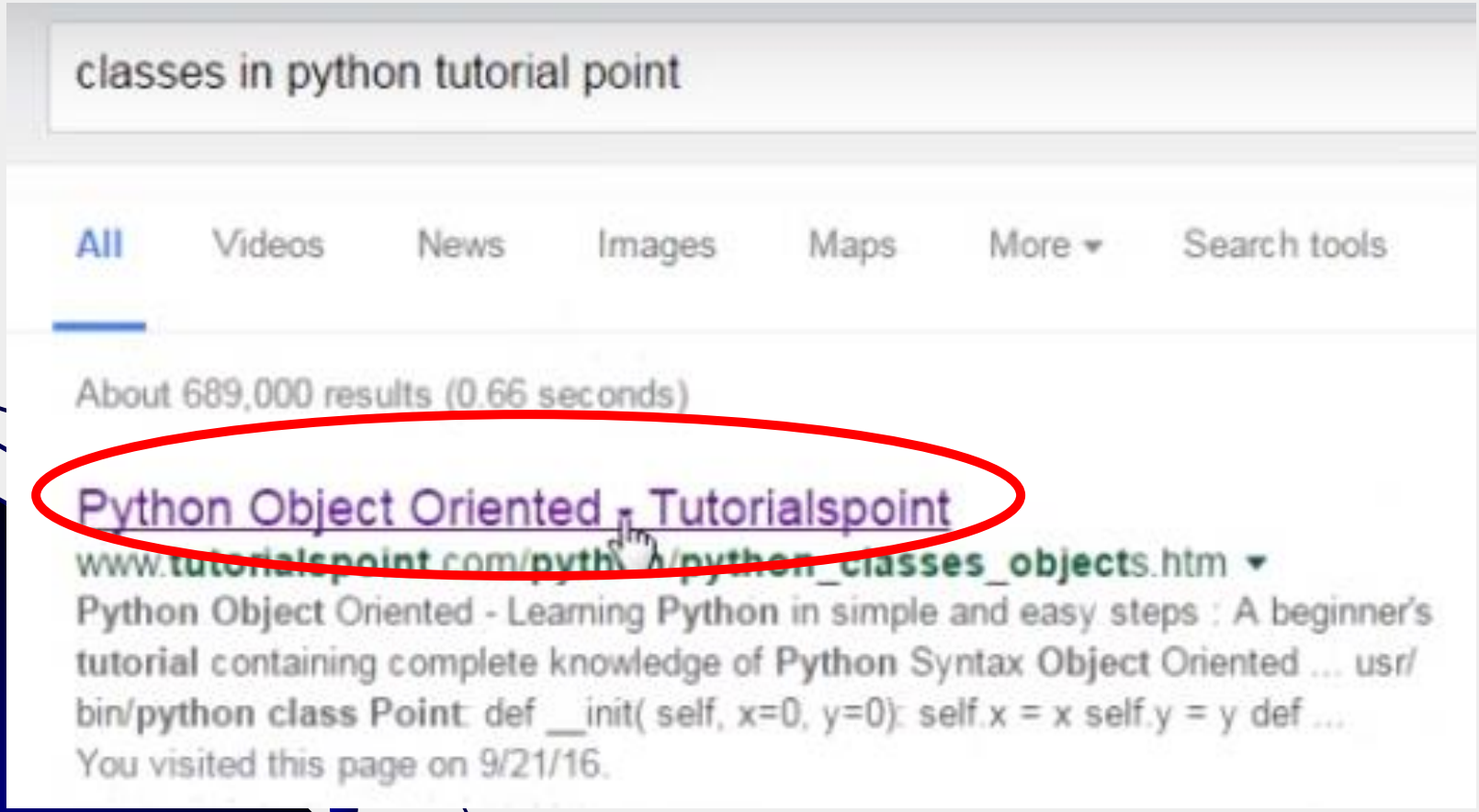
a1.show()
a2.show()
```

```
= RESTART: D:/Courses_2022_2023/Advanced
Programming/classes/cl_self_8_ACCount.py

a1 balance : =  20000
a2 no : =  19371564761
a1's balance : =  13500
a2's balance : =  39500
John Olsson. 19371554951, balance: 13500
Liz Olsson. 19371564761, balance: 39500
```

```
class Fruit:
    name = "apple"
    price = 10
    def eat_fruit(self):
        print("Fruit has been eaten")
                f = Fruit()
                f.eat_fruit()
print(f.name)
print(f.price)
```

من البحث نكتب: classes in python
ثم نختار: tutorial point



Python - Object Oriented

- Python has been an object-oriented language since it existed.
- Because of this, creating and using classes and objects are downright easy.
- If you do not have any previous experience with object-oriented (OO) programming, you may want to consult an introductory course on it or at least a tutorial of some sort so that you have a grasp of the basic concepts.
- However, here is small introduction of Object-Oriented Programming (OOP) to bring you at speed

Overview of OOP Terminology

- **Class** – A user-defined prototype for an object that defines a set of attributes that characterize any object of the class.
- The attributes are **data** members (class variables and instance variables) and **methods**, accessed via dot notation.
- **Class variable** – A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables.

Overview of OOP Terminology

- **Data member** – A class variable or instance variable that holds data associated with a class and its objects.
- **Function overloading** – The assignment of more than one behaviour to a particular function. The operation performed varies by the types of objects or arguments involved.
- **Instance variable** – A variable that is defined inside a method and belongs only to the current instance of a class.

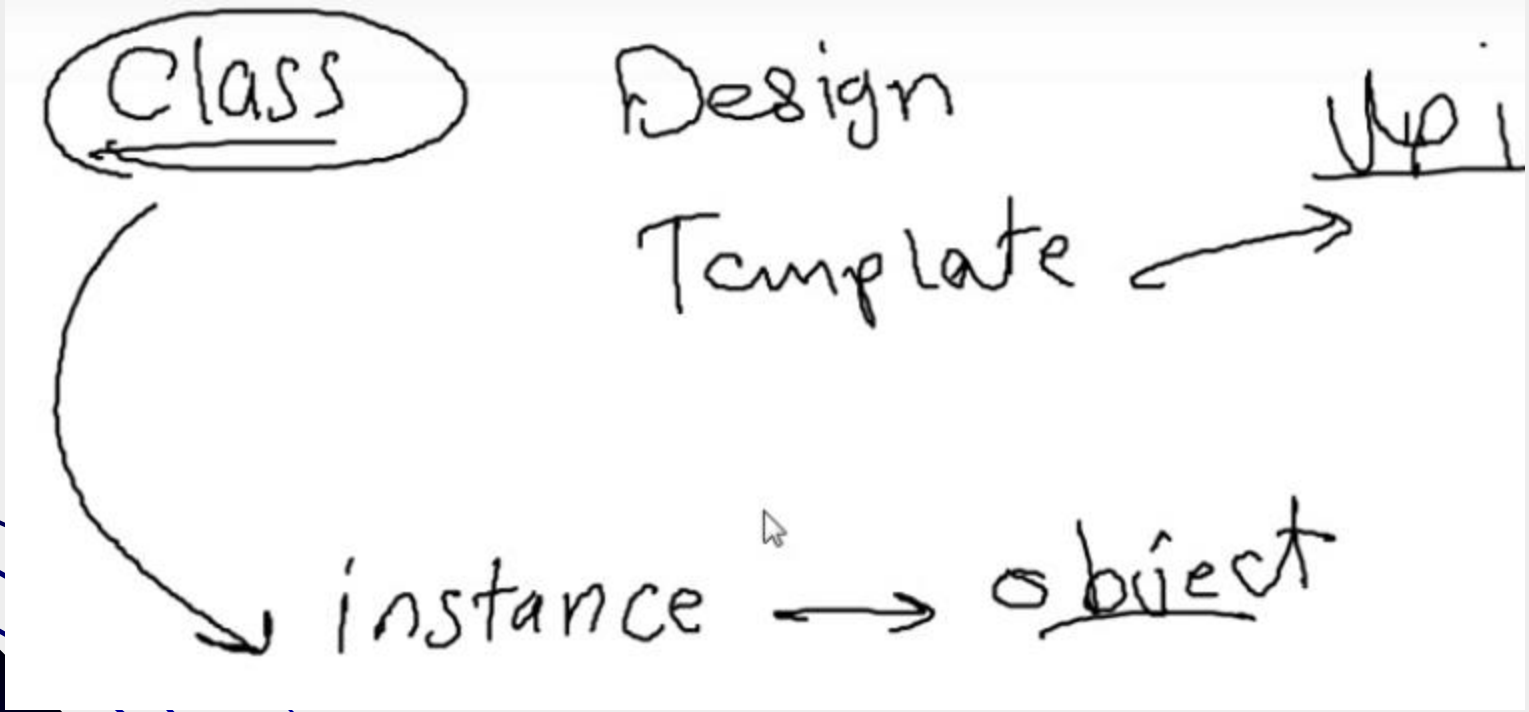
Overview of OOP Terminology

- **Inheritance** − The transfer of the characteristics of a class to other classes that are derived from it.
- **Instance** − An individual object of a certain class. An object ***obj*** that belongs to a class Circle, for example, is an instance of the class Circle.
- **Instantiation** − The creation of an instance of a class.
- **Method** − A special kind of function that is defined in a class definition.

Overview of OOP Terminology

- **Object** − A unique instance of a data structure that's defined by its class.
- object comprises both data members (class variables and instance variables) and methods.
- **Operator overloading** − The assignment of more than one function to a particular operator.

Classes



Creating Classes

- The class statement creates a new class definition. The **name** of the class immediately follows the **keyword class** followed by a **colon** as follows:

```
class ClassName:
   'Optional class documentation string'
   class_suite
```

- The class has a documentation string, which can be accessed via ClassName.__doc__.
- The class_suite consists of all the component statements defining class members, data attributes and functions.

Example: Following is the example of a simple Python class −

```
class Employee:
   'Common base class for all employees'
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
   def displayCount(self):
     print "Total Employee %d" % Employee.empCount
   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary
```

- The variable ***empCount*** is a class variable whose value is shared among all instances of a this class. This can be accessed as ***Employee.empCount*** from inside the class or outside the class.
- The first method ***__init__()*** is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.
- You declare other class methods like normal functions with the exception that the first argument to each method is ***self***. Python adds the ***self*** argument to the list for you; you do not need to include it when you call the methods.

Creating Instance Objects

- To create instances of a class, you call the class using class name and pass in whatever arguments its __init__ method accepts.

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

Accessing Attributes

- You access the object's attributes using the **dot** operator with object.
- Class variable would be accessed using class name as follows :

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

Now, putting all the concepts together:

```
#!/usr/bin/python

class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print ("Total Employee %d" % Employee.empCount)
```

Now, putting all the concepts together:

```
   def displayEmployee(self):
      print ("Name : ", self.name,  ", Salary: ",
self.salary)

"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee
class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" %
Employee.empCount)
```

- When the above code is executed, it produces the following result –

```
Name :  Zara ,Salary:  2000
Name :  Manni ,Salary:  5000
Total Employee 2
```

- You can add, remove, or modify attributes of classes and objects at any time −

```
emp1.age = 7  # Add an 'age' attribute.
emp1.age = 8  # Modify 'age' attribute.
del emp1.age  # Delete 'age' attribute.
```

- Instead of using the normal statements to access attributes, you can use the following functions:

  - The **getattr(obj, name[, default])** – to access the attribute of object.
  - The **hasattr(obj,name)** – to check if an attribute exists or not.
  - The **setattr(obj,name,value)** – to set an attribute. If attribute does not exist, then it would be created.

  - The **delattr(obj, name)** – to delete an attribute.

Built-In Class Attributes

- Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute:
- **__dict__** – Dictionary containing the class's namespace.
- **__doc__** – Class documentation string or none, if undefined.
- **__name__** – Class name.
- **__module__** – Module name in which the class is defined. This attribute is **"__main__"** in interactive mode.

Built-In Class Attributes

- **__bases__** – A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

- For the above class let us try to access all these attributes:

Built-In Class Attributes

```
#!/usr/bin/python

class Employee:
   'Common base class for all employees'
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
```

Built-In Class Attributes

```
    def displayCount(self):
        print "Total Employee %d" % Employee.empCount
    def displayEmployee(self):
        print "Name : ", self.name,  ", Salary: ", self.salary
print "Employee.__doc__:", Employee.__doc__
print "Employee.__name__:", Employee.__name__
print "Employee.__module__:", Employee.__module__
print "Employee.__bases__:", Employee.__bases__
print "Employee.__dict__:", Employee.__dict__
```

مثال مهم جدا على Class

- البرنامج الذي يحتوي على Class يتكون من برنامجين مستقلين بإسمين مختلفين:
- الاسم الأول يحوي **تصميم** الـ Class
- الاسم الثاني يحوي **استدعاء** الـ Class

تصميم الـ Class

```
class Human :
    Number = 0
    def __init__(self) :
        self.Name=None
        self.Age=0
        self.Skin =None
        Human.Number = Human.Number + 1
    def Print (self) :
        print (self.Name,self.Age,self.Skin )
```

تصميم الـ Class

```
def Read (self) :
    name = input ('Plz Enter the name: ')
    self.Name = name
    age = input('Plz Enter the Age: ')
    self.Age = age
    skin = input('Plz Enter the Skin: ')
    self.Skin = skin
```

استدعاء الـ Class

لاحظ أن اسم ملف الـ Class هو Class_1

```
from Class_1 import  *
h1 = Human()
h2 = Human()
h1.Read()
h1.Print()
print (Human.Number)
```

التصميم في حال أكثر من Class

```
class  AAnimal :
    Number = 0
    def __init__(self):
        self.Name = None
        self.Age = 0
        self.Skin = None
        AAnimal.Number = AAnimal.Number + 1
```

التصميم في حال أكثر من Class

```
def Print(self):
    print("Animal Data: ",self.Name, self.Age, self.Skin)
def Read(self):
    name = input('Plz Enter the Animal name: ')
    self.Name = name
    age = input('Plz Enter the Animal Age: ')
    self.Age = age
    skin = input('Plz Enter the Animal Skin: ')
    self.Skin = skin
```

التصميم في حال أكثر من Class

```
class Human :
    Number = 0
    def __init__(self) :
        self.Name=None
        self.Age=0
        self.Skin =None
        Human.Number = Human.Number + 1
    def Print (self) :
        print ("Human Data: ", self.Name,self.Age,self.Skin )
```

التصميم في حال أكثر من Class

```
def Read (self) :
    name = input ('Plz Enter the Human name: ')
    self.Name = name
    age = input('Plz Enter the Human Age: ')
    self.Age = age
    skin = input('Plz Enter the Human Skin: ')
    self.Skin = skin
```

استدعاء الـ Class

لاحظ أن اسم ملف الـ **Class** هو DoubClass

```
from DoubClass import *
h = Human()
a = AAnimal()
h.Read()
a.Read()
h.Print()
a.Print()
print (Human.Number)
print (AAnimal.Number)
```

استدعاء الـ Class

لاحظ أن اسم ملف الـ **Class** هو DoubClass بشكل أدق

```
from DoubClass import *
if __name__ == "__main__"h :
    h = Human()
    a = AAnimal()
    h.Read()
    a.Read()
    h.Print()
    a.Print()
    print (Human.Number)
    print (AAnimal.Number)
```

الشكل الأفضل لتصميم الـ Class

- الأفضل عند تصميم الـ Class استخدام خاصية الـ Inhertance.
- يتم ذلك بتصميم Class واحد على نفس النظام السابق.
- يتم عمل Inhertance لهذا الـ Class بالشكل:

class **New class name (Name of old class) :**

- إضافة المتغيرات الغير موجودة في الـ Class القديم

- أنظر المثال التالي:

مثال

```
class EffClass :
    Number = 0
    def __init__(self) :
        self.Name=None
        self.Age=0
        self.Skin =None
        Human.Number = Human.Number + 1
    def Print (self) :
        print ("Human Data: ",
           self.Name,self.Age,self.Skin )
```

مثال

```
    def Read (self) :
        name = input ('Plz Enter the Human name: ')
        self.Name = name
        age = input('Plz Enter the Human Age: ')
        self.Age = age
        skin = input('Plz Enter the Human Skin: ')
        self.Skin = skin
class AAnimal (EffClass) :
class Human  (EffClass)  :
```

مثال

```
from EffClass import *

if __name__ == "__main__" :
    h = Human()
    a = AAnimal()
    h.Read()
    a.Read()
    h.Print()
    a.Print()
    print (Human.Number)
    print (AAnimal.Number)
```

Numpy Liberariy

Numerical and Python

Libraries in Python

The most important **Libraries** **in python are:**

1- numpy is the best for Mathematics

2- pandas is the best for Data

3- matplotlib is the best for Graphics

4- OpenCV is the best for image processing

What Is NumPy?

- NumPy is one of the most commonly used libraries for numeric and scientific computing.
- NumPy is extremely fast and contains support for multiple mathematical domains such as linear algebra, geometry, etc.
- Therefore, it is extremely important to learn NumPy in case you plan to make a career in data science and data preparation.

A NumPy array advantages

- A NumPy array has many advantages such as:
  - NumPy arrays are much faster for insertion, deletion, updating, and reading of data.
  - NumPy arrays contain advanced broadcasting functionalities compared with regular Python arrays.
  - NumPy arrays come with a lot of methods that support advanced arithmetic and linear algebra options.
  - NumPy provides advanced multi-dimensional array slicing capabilities.

Why Use NumPy?

- in Python we have lists that serve the purpose of arrays, but they are slow to process.

- NumPy aims to provide an array object that is up to **50x faster than** traditional Python lists.

- The **array object** in NumPy is called **ndarray**, it provides a lot of supporting functions that make working with ndarray very easy.

Why Use NumPy?

- **Arrays** are very frequently used in data science, where speed and resources are very important.

- **Data Science:** is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

Why is NumPy Faster Than Lists?

- NumPy arrays are stored at **one continuous place in memory** unlike lists, so processes can access and manipulate them very efficiently.

- This behaviour is called in computer science. **locality of reference**

- This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Data Types in Python

By default Python have these data types:

- strings - used to represent text data, the text is given under quote marks. e.g. "ABCD"
- integer - used to represent integer numbers. e.g. -1, -2, -3
- float - used to represent real numbers. e.g. 1.2, 42.42
- boolean - used to represent True or False.
- complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

NumPy Data Types

Below is a list of all data types in NumPy

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

Installation of NumPy

- If you have Python and PIP already installed on a system, then installation of NumPy is very easy.

- Install it using this command:

```
C:\Users\Your Name>pip install numpy
```

how invoke numpy library?

- We will invoke numpy as follows:
  - **import numpy as np**
    - **With references (Ex. np.abs(x))**
  - **from numpy import ***
    - **Without references (Ex. abs(x))**

EX_1:

```
import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```

EX_2:

```
import numpy as np

print(np.__version__)
```

Create a Numpy ndarray Object

- Numpy is used to work with arrays. The array object in Numpy is called **ndarray**.
- We can create a Numpy ndarray object by using the array() function.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

```
The output:
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

Dimensions in Arrays

- A dimension in arrays is one level of array depth (nested arrays)

Dimensions in Arrays

0-D Arrays

- Create a 0-D array with value 42

```
import numpy as np

arr = np.array(42)

print(arr)
```

Dimensions in Arrays

1-D Arrays

- An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.
- These are the most common and basic arrays.
- Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

Dimensions in Arrays

2-D Arrays

- An array that has 1-D arrays as its elements is called a 2-D array.
- These are often used to represent matrix or 2nd order tensors.
- Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

```
The output:
[[1 2 3]
 [4 5 6]]
```

Dimensions in Arrays

3-D Arrays

- An array that has 2-D arrays (matrices) as its elements is called 3-D array.
- These are often used to represent a 3rd order tensor.
- **Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:**

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3],
[4, 5, 6]]])
print(arr)
```

Check Number of Dimensions?

- NumPy Arrays provides the ***ndim*** attribute that returns an integer that tells us how many dimensions the array have.

Check Number of Dimensions?

- Check how many dimensions the arrays have:
-

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]],
[[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Higher Dimensional Arrays

- Create an array with 5 dimensions and verify that it has 5 dimensions:

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
```

Access Array Elements

- Array indexing is the same as accessing an array element.
- You can access an array element by referring to its index number.
- The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Access Array Elements

- Get the **first** element from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

Access Array Elements

- Get the **second** element from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[1])
```

Access 2-D Arrays

- Access the element on the first row, second column:

- ```
  import numpy as np

  arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

  print('2nd element on 1st row: ', arr[0, 1])
  ```
  ```

Access 2-D Arrays

- Access the element on the 2nd row, 5th column:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd row: ', arr[1, 4])
```

Access 3-D Arrays

- Access the third element of the second array of the first array:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]],
[[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
```

`arr[0, 1, 2]` prints the value `6`
And this is why:

The first number represents the first dimension, which contains two arrays:
[[1, 2, 3], [4, 5, 6]]

and:
[[7, 8, 9], [10, 11, 12]]
Since we selected 0, we are left with the first array:
[[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays:
[1, 2, 3]
and:
[4, 5, 6]
Since we selected 1, we are left with the second array:
[4, 5, 6]

The third number represents the third dimension, which contains three values:

4

5

6

Since we selected 2, we end up with the third value:

6

NumPy Array Slicing

- **Slicing arrays**
- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: [start:end].
- We can also define the step, like this: [start:end:step].
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1

Slicing arrays

- Slice elements from index 1 to index 5 from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

Slicing arrays

- Slice elements from index 4 to the end of the array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

Slicing arrays

- Slice elements from the beginning to index 4 (not included):

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[:4])
```

STEP

- Use the step value to determine the step of the slicing:
- **Exam**:
- Return every other element from index 1 to index 5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

STEP

- **Example**
- Return every other element from the entire array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[::2])
```

Slicing 2-D Arrays

- **Example**
- From the second element, slice elements from index 1 to index 4 (not included):

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5],
[6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

Slicing 2-D Arrays

- From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5],
[6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
```

Mathematical Operations by numpy

By default Python have these data types:

- **strings** - used to represent text data, the text is given under quote marks. e.g. "ABCD"
- **integer** - used to represent integer numbers. e.g. -1, -2, -3
- **float** - used to represent real numbers. e.g. 1.2, 42.42
- **boolean** - used to represent True or False.
- **complex** - used to represent complex numbers.
- e.g. 1.0 + 2.0j, 1.5 + 2.5j

i - integer
b - boolean
u - unsigned integer
f - float
c - complex float
m - timedelta
M - datetime
O - object
S - string
U - unicode string
V - fixed chunk of memory for other type (void)

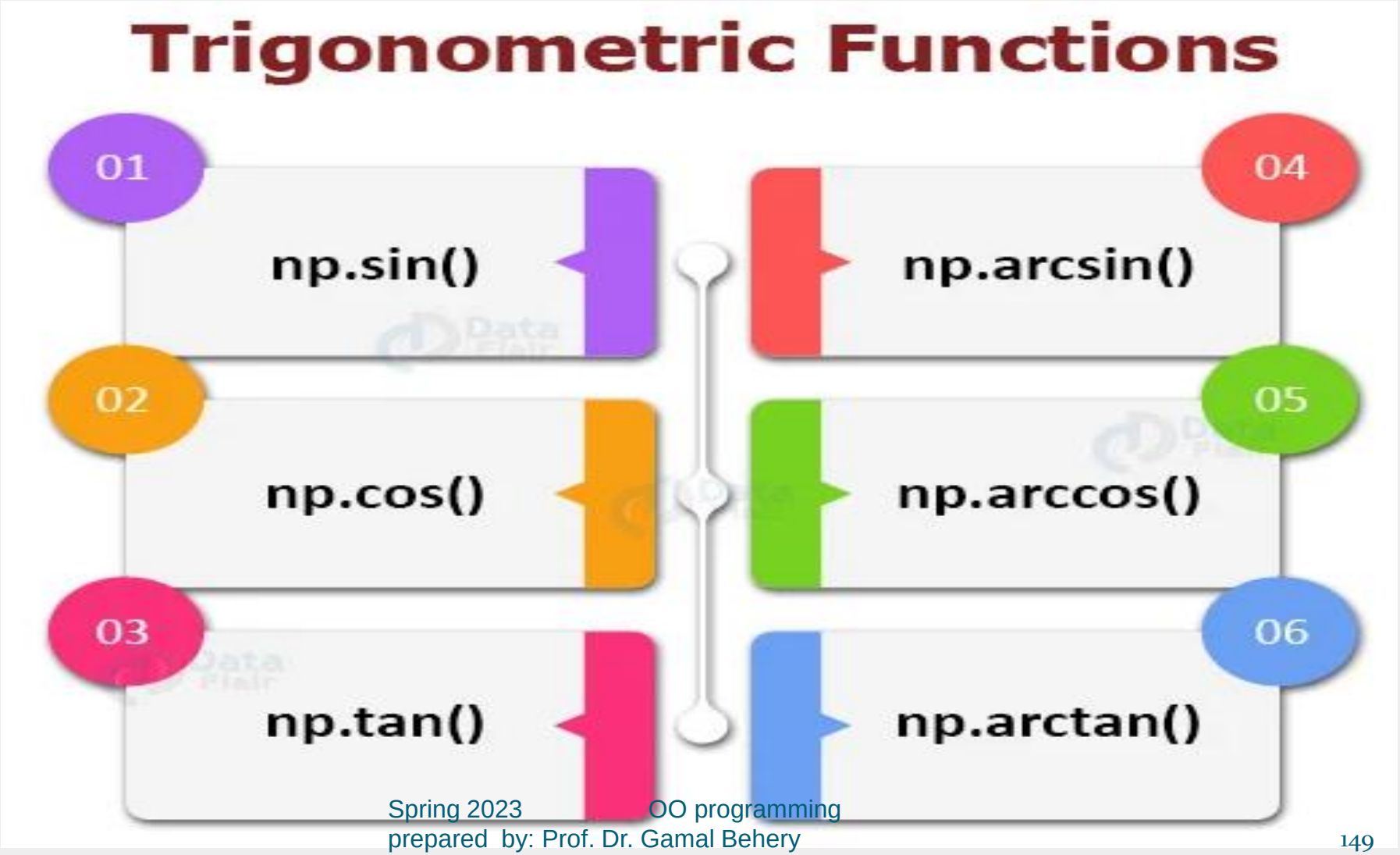
Checking the Data Type of an Array

The NumPy array object has a property called **dtype** that returns the data type of the array:

- Get the data type of an array object:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

1- Trigonometric functions



1- Trigonometric functions

لابد من التعامل بالتقدير الدائري

$$Degree * \frac{pi}{180}$$

```
1 from  numpy import *
2
3 a = sin(30*np.pi/180)
4 b = cos(30*np.pi/180)
5 c = tan(30*np.pi/180)
6
7 print(a)
8 print(b)
9 print(c)
```

```
= sin(deg2rad(30))
= cos(deg2rad(30))
= tan(deg2rad(30))
```

Empty command

- Create an empty array with size nxm:

```
a =empty((4, 3))
```

random.uniform command

- To create only one random number between two digits:

```
a = random.uniform(1,10)
```

To create 20 random numbers between two digits:

```
a = random.uniform(1,10,20)
```

random.random command

- To create an **array** from random numbers:

```
a = random.random((2,3))
```

Note: all created numbers from 0 to 1 only

Ex:

- **Write a program to create an array from random numbers with size 4x5. all random numbers between 10 and 20.**

```
from numpy import *
a = random.random((4,5))
a=a*10+10
print(a)
```

random.normal command

- To Generate n numbers between zero and one and represent a normal distribution:

```
a = random.normal(0, 1, 10)
```

random.randint command

- To create only one integer random number between 0 and n:

```
a = random.randint(140)
```

- To create m integer random numbers between 0 and n:

```
a = random.randint(140, size =10)
```

- To create m integer random numbers between x and y:

```
a = random.randint(140, 190, size =10)
```

- To create an integer random numbers array between x and y of size nxm:

```
a = random.randint(0, 10, (3, 4))
```

random.randint command

- To create an integer random numbers **array** between x and y of **three** dimension L(عمق)xM(صف)xN(عمود):

```
a = random.randint(0, 10, size = (3, 4, 5))
```

- Its OK also without size key:

```
a = random.randint(0, 10, (3, 4, 5))
```

reshape command

- To create an array NxM from List use **reshape** key:

```
a = random.randint(0, 10,25)
b=reshape(a, (5, 5))
```

NOTE: The generated numbers from random.randint should be equal the size of array = NxN

random.choice command

- To choose only one random number or string from List use random.choice command:

```
a = random.randint(0, 10,25)
b=random.choice(a)
```

random.shuffle command

- To reorder the list randomly use random.shuffle command:

```
a = random.randint(0, 20,10)
print(a)
random.shuffle(a)
print(a)
```

random.shuffle command

- To reorder the list randomly use random.shuffle command:

```
a=np.random.randint(0,20,(5,5))
print(a)
np.random.shuffle(a)
print(a)
```

```
The array before using suffle
[[14  10  19   6   0]
 [ 4  11   9  15  16]
 [ 0   1  14   9   2]
 [ 7   7  10  17   0]
 [ 3  16  15  17   2]]
The array after using suffle
[[ 3  16  15  17   2]
 [ 4  11   9  15  16]
 [ 7   7  10  17   0]
 [ 0   1  14   9   2]
 [14  10  19   6   0]]
```

Application for random.shuffle command

- لنفترض أنه تم تجميع بيانات عن مرضى أمراض السكر من محافظات القاهرة، دمياط، الدقهلية، الإسكندرية، أسيوط، بني سويف، الاسماعلية، بورسعيد والشرقية. ثم تم دمج هذه البيانات في مصفوفة واحدة.
- المطلوب اخذ عينة عشوائية من هذه البيانات شرط ان تمثل كل المحافظات المشاركة في العينة. لكي يتم التدريب عليها.
- ممكن استخدام امر copy للنسخة الاصلية من البيانات قبل استخدام أمر shuffle للحفاظ على النسخة الأصلية.

Application for random.shuffle command

- Suppose that data were collected on diabetic patients from the governorates of Cairo, Damietta, Dakahlia, Alexandria, Assiut, Beni Suef, Ismailia, Port Said and Sharkia.
- This data was then combined into a single matrix.
- It is required to take a random sample from this data, provided that it represents all the governorates participating in the sample. to be trained.
- It is possible to use the copy command for the original copy of the data before using the shuffle command to preserve the original copy.

zeros & ones commands

- To create an array in one dimension all of its elements are zeros:

```
a = zeros(8)
```

- To create an array in one dimension all of its elements are ones:

```
b = ones(8)
```

NOTE: may be the array in 2-D or 3-D or more for example:

```
c = ones((3,  5))
```

zeros & ones commands

NOTE: may be the array in 2-D or 3-D or more for example:

2-D:

3-D:

```
c = ones((3, 4, 5))
```

4-D:

```
b3 = ones((4, 5, 5))
```

eye command

- To create an identity array use the eye command:

```
a = eye(8)
```

full command

- To create an array of size MxN, all its elements are x value use full command:

```
a = full((3,5), 40)
```

arange (X).reshape(M, N)

Note: 0:X = NxM

- To create sequence of numbers from 0 to X and organize these numbers in array of size NxM use the command arange (X).reshape(M, N):

(No. of generated elements = NxM)

#2-D

```
a = arange (18).reshape(3, 6)
```

 # 3-D

```
b = arange (27).reshape(3, 3, 3)
```

linspace command

- To generate 50 numbers evenly distributed between two numbers, use the linspace command:

```
a = linspace(4,15)
```

To generate N numbers evenly distributed between two numbers, use the linspace command:

```
b = linspace(4,15, 20)
```

linspace (X, Y, Z).reshape(M, N)

Note: Z = NxM

- To generate N numbers evenly distributed between two numbers and them in array, use the linspace (X, Y, Z).reshape (M, N) command:

#2-D

```
a = linspace(4,15, 18).reshape(3,6)
```

#3-D

```
a1 = linspace(4,15, 27).reshape(3,3, 3)
```

diag(array())

- To create a diagonal array, use the diag(array()):

a=diag(array([1,4,6,7,9,11]))

```
[[ 1  0  0  0  0  0]
 [ 0  4  0  0  0  0]
 [ 0  0  6  0  0  0]
 [ 0  0  0  7  0  0]
 [ 0  0  0  0  9  0]
 [ 0  0  0  0  0 11]]
```

diag(array())

- To create a diagonal array, use the diag(array()):

a=diag(array([1,4,6,7,9,11]), k=3)

أضف ثلاث صفوف وثلاث أعمد ليصبح أبعاد المصفوفة 9×9

```
[[ 0  0  0  1  0  0  0  0  0]
 [ 0  0  0  0  4  0  0  0  0]
 [ 0  0  0  0  0  6  0  0  0]
 [ 0  0  0  0  0  0  7  0  0]
 [ 0  0  0  0  0  0  0  9  0]
 [ 0  0  0  0  0  0  0  0 11]
 [ 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]]
```

Dealing with arrays

- In this part we will use arrays to get some information from the array. For example: determining the number of non-zero elements,

count_nonzero() command

- To determine the number of non-zero elements, use count_nonzero():

```
a=random.randint(0, 10, (3, 3))
b=count_nonzero(a)
```

- To determine the number of elements greater than or less than value X, use count_nonzero():

```
a=random.randint(0, 10, (3, 3))
b=count_nonzero(a > 5)
b=count_nonzero(a <8)
```

count_nonzero() command

- To determine the number of elements greater than or less than value X in each row (البحث يكون صف بصف), use count_nonzero():

```
a=random.randint(0, 10, (3, 3))
b=count_nonzero(a > 5, axis = 1)
b=count_nonzero(a <8, axis = 1)
```

```
[[8 7 9]
 [2 8 7]
 [3 8 9]]
[3 2 2]
[1 2 1]
```

any() command

- To determine if any element in array A greater than or less than value X , use any():

```
a=random.randint(0, 10, (3, 3))
```

```
#in all array
b = any (a > 5)
```

```
#in every row in array
b=any(a > 5, axis = 1)
```

```
[[0 8 6]
 [1 0 2]
 [6 5 1]]
True
[ True False  True]
```

all() command

- To determine if all element in array A greater than or less than value X , use all():

```
a=random.randint(0, 10, (3, 3))
```

```
#in all array
b = all (a > 5)
```

```
#in every row in array
b=all(a < 9, axis = 1)
```

```
[[2 3 8]
 [6 1 1]
 [5 4 3]]
False
[ True  True  True]
```

```
from numpy import *
a=random.randint(5, 20, size=9).reshape(3,3)
b = a > 10
c = a < 15
d = a[b]
e = a[c]
print(a)
print('------------------------------------')
print(b)
print('------------------------------------')
print(c)
print('------------------------------------')
print (d)
print('------------------------------------')
print(e)
print('------------------------------------')
```

```
[[ 8 18 19]
 [19  8  7]
 [14 19 18]]
------------------------------
[[False  True  True]
 [ True False False]
 [ True  True  True]]
------------------------------
[[ True False False]
 [False  True  True]
 [ True False False]]
------------------------------
[18 19 19 14 19 18]
------------------------------
[ 8  8  7 14]
------------------------------
```

isclose(a, b, rtol = c) command

- To measure the difference between two values or arrays, use isclose(a, b, rtol = c) command:

```
d = isclose(a, b, rtol = 0.1)
e = isclose(a, c, rtol = 0.1)
```

إذا كان الفرق بين المصفوفتين (rtol = 0.1) يعطي true

```
from numpy import *

a = arange(9).reshape(3,3)
b = arange(9).reshape(3,3)
c = 2 * b
d = isclose(a, b, rtol = 0.1)
e = isclose(a, c, rtol = 0.1)
print(a)
print('-----------------------------------')
print(b)
print('-----------------------------------')
print(c)
print('-----------------------------------')
print (d)
print('-----------------------------------')
print(e)
print('-----------------------------------')
```

```
[[0  1  2]
 [3  4  5]
 [6  7  8]]
--------------------------
[[0  1  2]
 [3  4  5]
 [6  7  8]]
--------------------------
[[ 0  2  4]
 [ 6  8 10]
 [12 14 16]]
--------------------------
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]]
--------------------------
[[ True False False]
 [False False False]
 [False False False]]
--------------------------
```

multiply() command

- To multiply value in array and put the out put in empty key, use multiply () command:

```
from numpy import *
a = arange(5)
b = empty(5)
multiply(a, 10, out = b)
print(a)
print(b)
```

power() command

- To raise the array to the power of **c**, use power () command:

```
from numpy import *
a = arange(5)
b = empty(5)
Power (a, 4, out = b)
print(a)
print(b)
```

```
[0 1 2 3 4]
[  0.   1.  16.  81. 256.]
```

add. Reduce() command

- To add all elements of array and reduce array dimension , use **add.reduce()** command:

```
from numpy import *
a = arange(2,10)
b = add.reduce(a)
print(a)
print(b)
```

multiply. Reduce() command

- To **multiply** all elements of array and reduce array dimension , use **multiply.reduce()** command:

```
from numpy import *
a = arange(15)
b = multiply.reduce(a)
print(a)
print(b)
```

multiply. outer() command

- To **multiply** all elements of array, use **multiply. outer()** command:

```
from numpy import *
a = arange(2,8)
b = multiply. outer(a, a)
print(a)
print(b)
```

```
[2 3 4 5 6 7]
[[ 4  6  8 10 12 14]
 [ 6  9 12 15 18 21]
 [ 8 12 16 20 24 28]
 [10 15 20 25 30 35]
 [12 18 24 30 36 42]
 [14 21 28 35 42 49]]
```

multiply. accumulate() command

- To **accumulate** all elements of array and reduce array dimension , use **multiply. accumulate()** command:

```
from numpy import *
a = arange(2, 8)
b = multiply. accumulate(a)
print(a)
print(b)
```

```
[2 3 4 5 6 7]
[   2    6   24  120  720 5040]
```

len() command

- To **determine** the number of elements in in 1-D array or no. of raws in 2-D array , use **len()** command:

```
from numpy import *
a = arange(12)
b = len(a)
c = a.reshape(3,4)
d=len(c)
print(a)
print(b)
print(c)
print(d)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
12
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
3
```

shape() command

- To **determine** the dimension of array , use **shape()** command:

```
from numpy import *
a = arange(12)
b = shape(a)
c = a.reshape(3,4)
d=shape(c)
print(a)
print(b)
print(c)
print(d)
```

shape() command

- To **determine** the dimension of array , use **shape()** command:

```
from numpy import *
a = arange(12)
b = shape(a)
c = a.reshape(3,4)
d=shape(c)
print(a)
print(b)
print(c)
print(d)
```


Thank You


السلام عليكم ورحمة الله وبركاته