Functions: (Advantages)

- It is used to make calculations or Perform any task.
- reuse, bcz of reusability the Programmer can avoid redundancy.
- Functions provide modularity for Programming. Programmer divides the main task into smaller sub tasks called modules. To represent each module, the Programmer will develop a function (Solving Problem easily)
- code maintaince will become easy because of function. for new feature new function can be written and integrate in s/w. If we want to remove particular feature corresponding function need to deletes.
- debugging and it reduce length of Program

Diff b/w function and Method:

```
obj.method();
cls.method();
```

Defining a function:

function definition

```
fun header ←  def funname (Para1, Para2...) :
                  """"" fun docstring """""
fun body ←        funstatements.
```

ex:

```
def add (a, b)
    """" To perform addition""""
    c=a+b
    Print(c).
```

- def represents the starting of fun definition.
- : represents the beginning of the fun body.
- fun body contains group of stmnts called 'suite'.
- The string called doc string ~~to~~ should write that gives info abt fun.

''' ___ '''

Calling a function:

```
                   → Formal Parameters
ex:   def sum(a, b):
          ''' add fun '''
          c = a+b
          Print (c)
                     → Actual Parameters
      sum (10, 15)
      sum (10.5, 12.6).
```

- The Parameters do not know what type of values they are going to receive till the values are passed till the time of fun cal. This is called 'dynamic typing' the type of data can determine only during run time. not a compile tim. It is ~~als~~ available only in Python.

Returning result from a function:

```
return c , return 100, return list , return x, y,c
                              ↓                ↓
                             list            tuple.
```

```
ex:   def add (a, b):
          c = a+b
          return c
      x = add (10, 15)   -> Print(x)
      y = add (10.5, 12.6)
      Print (y)
```

Ex:
```
def Prime(n):
    x = 1
    for i in range(2,n):
        if n%i==0:
            x=0
            break
        else:
            x=1
    return x
num = ~~input~~ int(input("enter a number"))
res = Prime(num)
if res==1:
    Print(num,'is Prime')
else:
    Print(num,'is not Prime')
```

Returning Multiple Values from a Fun:

- Fun returns Single value in C, C++, Java but it returns multiple values in python.
- These values are returned by the fun as a tuple.

Scanned by CamScanner

ex :

```
def sum_sub(a,b):
    c = a+b
    d = a-b
    return c,d
x, y = sum_sub(10,5)
Print("sum is", x)
Print("sub is", y).
```

ex :

```
def sum_sub(a,b):
    c = a+b
    d = a-b
    e = a*b
    return c,d,e
t = sum_sub(10,5)
Print(" The result are: ")
for i in t :
    Print(i, end=',')
```

o/p :

```
The result are:
15, 5, 50.
```

Local and Global variables:

Local

ex: 1.
```
def func():
    a=1
    a=a+1
    Print(a)     # display 2
func()
Print (a)     # error.
```

2. Global:

```
a=1
def func():
    b=2
    Print (a)     1
    Print (b)     2
func()
Print (a)  1
Print (b)  error.
```

The Global Keyword:

When the Programmer wants to use the global variable inside a function, he can use the keyword 'global' before the variable in the beginning of the fun body as:

global a.

Scanned by CamScanner

ex: 1.
```
a=1
def fun():
    global a
    Print (a)      #display global var value i.e 1
    a=2            # modify global var to 2.
    Print (a)      # new glo var Value 2.
fun()
Print (a)          # displays new value 2.
```

o/p:
```
1
2
2
```

2. If he use global keyword, then he can accees only global variable and the local variable is no more available.

ex:
```
a=1
def fun()
    a=2
    x=globals()['a']
    Print (x)    #global
    Print (a)    #local
fun()
Print (a)    #global
```

Locall variabler is no more available

- globals() returns a values in the form of dictionary.

o/p:
```
1
2
1
```

Functions are First class objects:

The Following Possibilities are noteworthy

i) It is Possible to assign a function to a variable.

ii) It is Possible to define one function inside another function

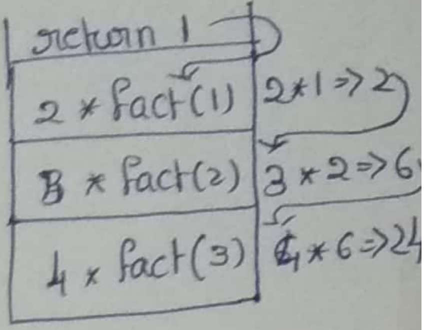
iii) It is Possible to pass a function as Parameter to another fun.

Ex :

```
i) def display(str):
       return 'Hai' + str              => o/p:  Hai Rahul
   x = display ("Rahul")
   Print(x)

ii) def display(str):
        def message():
            return 'How r u'           => o/p:  How r u Rahul
        r = message() + str
        return r
    Print ( display ("Rahul"))

iii) def display (fun):
         return 'Hai' + fun            => o/p : Hi Hw r u
     def message():
         return 'Hw r u'
     Print (display(message()))
```

Recursion :

A function calling itself is called 'Recursion.'



Ex:
```
def fact(n):
    if n==0:
        return 1
    else
        ~~return~~
        res = n * fact(n-1)
    return res

a = fact(4)
Print(a)
```

Anonymous Functions or Lambdas.

A fun without a name is called 'anonymous functions'. They are defined using the Keyword lambda.

```
lambda arg-list : exp
```

ex: lambda x : x * x.

ex: 1. Python Program to create a lambda function that returns a square value of a given no.

```
f = lambda x : x * x
Value = f(5)
Print ('square is', value)
```

lambda x : x * x (5)

o/p: square is 25.

```
2. f = lambda x, y : x + y
   res = f (1.55, 10)
   Print ("add is"; res)
```

o/p: add is 11.55.

∴ lambda function returns a function & hence it is necessary to assigned to a function.

∴ Lambda fun's contain only one exp and they return the result implicitly

-> using lambda functions with filters

using filter(): It is useful to filter out the ele of a seq, depending on the result of a fun.

filter (fun, seq).

ex:

```
lst = [10, 23, 45, 46, 70, 99]
lst1 = list (filter(lambda x : (x%2==0), lst)
Print (lst1)
```

o/p: [10, 46, 70]

Using map() :

It is similar to filter() but it acts on each ele of the seq. and perhaps changes the elements

map (fun, seq).

Ex : 1.
```
list = [1,2,3,4,5]
list1 = list (map (lambda x : x*x, list))
Point (list1)
```
o/p : [1, 4, 9, 16, 25]

```
list = [1,2,3,4 5]
f= lambda x:x*2
y = list (map(f, list))
y
```

ex : 2.
```
list1 = [1, 2, 3, 4, 5]
list 2 = [10, 20, 30, 40, 50]
list3 = list (map(lambda x, y : x*y, list1, list2))
Point (list3).
```
o/p : [10, 40, 90, 160, 250]

reduce() fun : functools

reduce() fun reduces the seq of ele to a single value by processing.

ex :
```
list = [1, 2, 3, 4, 5]
res = reduce (lambda x, y : x*y, list)
Point (res).
```
o/p : 120.

Scanned by CamScanner

Formal & Actual Arguments:

```
                 → formal Arguments.
def sum(a,b):
      c=a+b
      Print(c)
~~Sum~~ :
x=10, y=15
Sum(x,y)  → Actual Arguments.
```

The Actual arguments is of 4 types:

1. Positional Arguments
2. Keyword "
3. Default "
4. Variable length "

1. ~~*~~ Positional Arg :

There are the arg's Passed to a function in correct Positional order. Here, the no. of arguments and their Positions in the fun def should match exactly with the no. ~~&~~ Pos of the arg in the fun call.

```
def attach(s1,s2)
    attach('Hello', 'world')     => attach('world', 'Hello')
o/p: Helloworld.                 o/p: worldHello.
```

Ex:
```
def attach(s1,s2):
    S3 = S1+S2
    Print('Total string :' +S3)
attach('New','York')
```

Scanned by CamScanner

```
ex:   def attach(S1,S2):
          S3 = S1 + S2
          Print ('Total string'+S3)
      attach ('Hello', 'world')     → O/P : Total string : Hello world.
```

2. Keyword Arguments:

Keyword Arguments are arguments that identifies the Parameters by their name.

```
def grocery (item, Price):
```

At the time of calling we have to pass two values & should mention which value for what.

```
grocery (item = 'Sugar', Price = 50.75)
grocery ( Price = 88.00, item = 'oil')
```

```
ex:  def grocery (item, Price):
         Print ('Item = %s' % item)              O/P : Item = Sugar
         Print ('Price = %f' % Price)                  Price = 50.75
     grocery (item = 'Sugar', Price = 50.75)           Item = oil
     grocery (Price = 88.00, item = 'oil')             Price = 88.00.
```

3. Default Arguments:

We can mention some default value for the function parameters in the definition.

Scanned by CamScanner

ex:
```
def grocery(item, Price=40.50)
    Print('Item = %s' % item)
    Print('Price = %f' % Price)
grocery(item='sugar', Price=50.75)
grocery(item='sugar')
```

O/P:
```
Item = sugar
Price = 50.75
Item = Sugar
Price = 40.50.
```

4. **Variable Length Arguments:**

A variable length argument is written with a '*' symbol before it in the function definition as:

def add(farg, *args):

ex:
```
def add(farg, *args):  → it stores the values as taple
    Print('formal arg', farg)
    sum=0
    for i in args:
        sum=sum+i
    Print('sum is', (farg+sum))
add(5,10)
add(5,10,20,30)
```

// If the Programmer wants to develop a function that can accept 'n' arguments it is psble with variable length arguments

O/P:
```
Formal arg = 5
sum = 15
formal arg = 5
sum = 65
```

5. Keyword Variable length Arg:

It is an argument that can accept any no.of values Provided in the format of Keys & values. I ** should be used.

def display(farg, ** kwargs):

It internally represent a dictionary.

display (5, rno =10)

ex:
```
def display(farg, ** kwargs):
        Print ('Formal :', farg)
        For x, y in kwarg .items():
            Print ('Key={}, Value={}'.format(x,y))
        ~~display~~
    display (5, rno = 10)
    Print()
    display (5, rno = 10, name = 'Prakash')
```

o/P:
```
farg = 5
key = rno, value=10
farg = 5
key = name, values = Prakash
key = rno, value = 10.
```

DH 76, 79, 94, B2, C4, D1, D3,

Scanned by CamScanner

Files : 1. Text files

2. Binary files.

- Text file stores the data in the form of characters.
- Binary file stores entire data in the form of bytes.

Opening of File : open() function

~~f~~ ~~co~~ file handler = open ("file name", "open mode")

w - write mode r - read mode a - append mode.

w+ - write + read r+ - read + write a+ - append + read

closing a file : close() function

f.close()

- After entering a string from keyboard using input() function, we store the string into the file using write() method.

f.write(str)

Ex : 1. Python program to create a text file to store strings.

Sol :
```
f= open ("Hi.txt", 'w')
str = input ("enter text")
f.write (str)
f.close()
```

O/P : enter text : This is my first line.

append : f = open ("Hi.txt", 'a')

To read data from a text file, we can use read () method, it returns string.

str = f.read()

we can also use the read() method to read only a specified no. of bytes.

str = f.read(n)

ex :
```
f = open ("Hi.txt", 'r')
    str = f.read()
    print (str)
    f.close()
```
o/p : This is my first line.

```
- str = f.read(4)
```

It display first 4 bytes of the file as This.

Working with Text Files containing multiple strings:

- we have to use write() method. To write the strings in different lines, we are supposed to add "\n" at the end of each string.

Ex : Program to read group of string into a text. file.

Sol :
```
f = open ('Hi.txt', 'w')
print ('enter text")
while str! = '@':
    str = input()
```

```
        if (str! = '@'):
            f.write(str + "\n")
    f.close()
```

o/p : ~~Eh~~ Hi.
Hello.
@

- To read the strings from the file we use read(), it reads all the lines and display them line by line. It returns string

```
Hi
Hello.
```

- readlines() that reads all the lines into a list.

```
f.readlines()
```

[' ~~Eh~~ Hi. \n ', 'Hello.\n']

- If we want to suppress the \n characters.

```
f.read().splitlines()
```

o/p : ['Hi', 'Hello']

Ex : A pgm to read all strings from the text file & display them.

Sol :

```
f = open('Hi.txt', 'r')
print(' The file contents are')
str = f.read()
print(str)
f.close().
```

o/p : Hi.
Hello.

Scanned by CamScanner

```
f=open('Hi.txt', 'a+')

    f.seek(offset, fromwhere)
```

- 'offset' represents how many bytes to move. 'fromwhere' represents from which pos to move. '0' represents beginning of file, 1 means current pos, 2 means end.

f.seek(10, 0) means file handler at 10th byte from the beginning of the file.

ex: pgm to append data to an existing file and then displaying file.

Sol:

```
f=open('Hi.txt','a+')
                    -> print("enter text to append")
while str!='@':
    str=input()
    if (str != '@):
        f.write(str + '\n')
f.seek(0,0)
print('The file contents are:')
str1=f.read()
  print(str1)
f.close()
```

O/P : ~~welcome~~ Enter text to append.

```
welcome.
@
The file contents are:   Hi
                         Hello.
                         welcome.
```

With statement :

The 'with' statement can be used while opening a file. The advantage of 'with' statement is that it will take care of closing a file which is opened by it.

with open ("filename", "openmode") as file object.

Ex : 1. Python Program to use 'with' to open a file & write some Str

Sol with open ('Hi.txt', 'w') as f:

```
        f.write ("I am a learner \n")
        f.write ("python is attractive \n")
```

2. Pgm to read date using 'with'.

Sol : with open ('Hi.txt', 'r') as f:

```
            str = f.read()
            Print (str).
```

- f.tell() : It return the current Position of the file Pointer from the beginning of the file

  n = f.tell()

- Binary Modes : wb, rb, ab, w+b, r+b, a+b

Scanned by CamScanner

Zipping and unzipping Files:

We know that some s/w's like 'winzip' provide zipping and unzipping of file data. In zipping the file contents, following two things could happen:

i) The file contents are compressed and hence the size will be reduced.

ii) The format of data will be changed making it unreadable.

In python, the module zipfile contains ZipFile class that helps us to zip or unzip a file contents.

```
     f = ZipFile('test.zip', 'w', ZIP_DEFLATED)
    /                ↓          ↓        ↓
 zipfile class   zip file name  mode   zipfile Attribute
   obj
```

1. Ex: A python program to compress the contents of file.

Sol:
```
from zipfile import *
f = ZipFile('test.zip', 'w', ZIP_DEFLATED)
f.write('file1.txt')
f.write('file2.txt')
f.write('file3.txt')
print('test.zip file created')
f.close()
```

Scanned by CamScanner

2. Ex: A Python Program to unzip the contents of a file.

Sol:
```
from zipfile import *
z = zipFile('test.zip', 'r')
names = z.namelist()  # extract all file name from zip file.
for fname in names:
    f = z.open(fname)
    str = f.read()
    print(str.decode())  #decode the contents from bytes
f.close()                  to ordinary strings.
```

Scanned by CamScanner

NumPy

- ndarray, a fast and space-efficient multidimensional array providing vectorized arithmetic operations and sophisticated *broadcasting* capabilities.
- Standard mathematical functions for fast operations on entire arrays of data without having to write loops
- Linear algebra, random number generation, and Fourier transform capabilities

```
>>> import numpy as np
>>> x=np.array([2,5,6,7,8])
>>> x
array([2, 5, 6, 7, 8])
>>> x[0]=22
>>> x
array([22,  5,  6,  7,  8])
>>> y=x*2
>>> y
array([44, 10, 12, 14, 16])
>>> y=x+2
>>> y
array([24,  7,  8,  9, 10])
```

```
>>> y.shape
(5,)
>>> y.dtype
dtype('int32')
>>> d=[[2,4,3,6],[2,3,4,5],[8,2,5,7],[9,2,4,6]]
>>> s=np.array(d)
>>> s
array([[2, 4, 3, 6],
       [2, 3, 4, 5],
       [8, 2, 5, 7],
       [9, 2, 4, 6]])
s.shape
(4, 4)
```

```
>>> s.ndim
2
>>> t=np.zeros(10,int)
>>> t
array([0, 0, 0, 0, 0, 0, 0,
0, 0, 0])
>>> t=np.zeros((3,5),int)
>>> t
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
>>> t=np.ones(5)
>>> t
array([1., 1., 1., 1., 1.])
```

```
>>> t=np.arange(2,10,1.3)
>>> t
array([2. , 3.3, 4.6, 5.9, 7.2, 8.5,
9.8])
>>> t=np.logspace(4,5,4)
>>> t
array([ 10000.          ,
21544.34690032,
46415.88833613, 100000.          ])
>>> t=np.linspace(1,10,5)
>>> t
array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
```

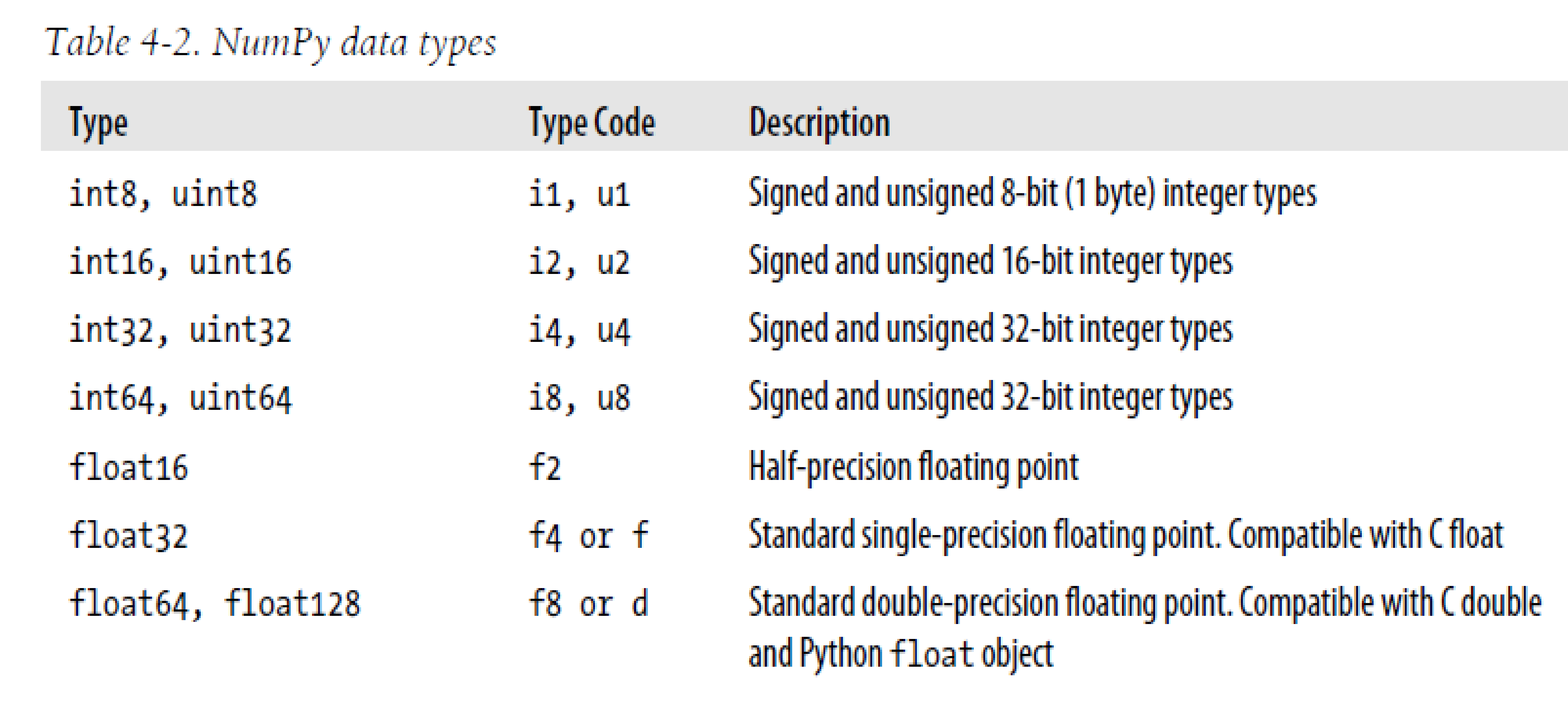
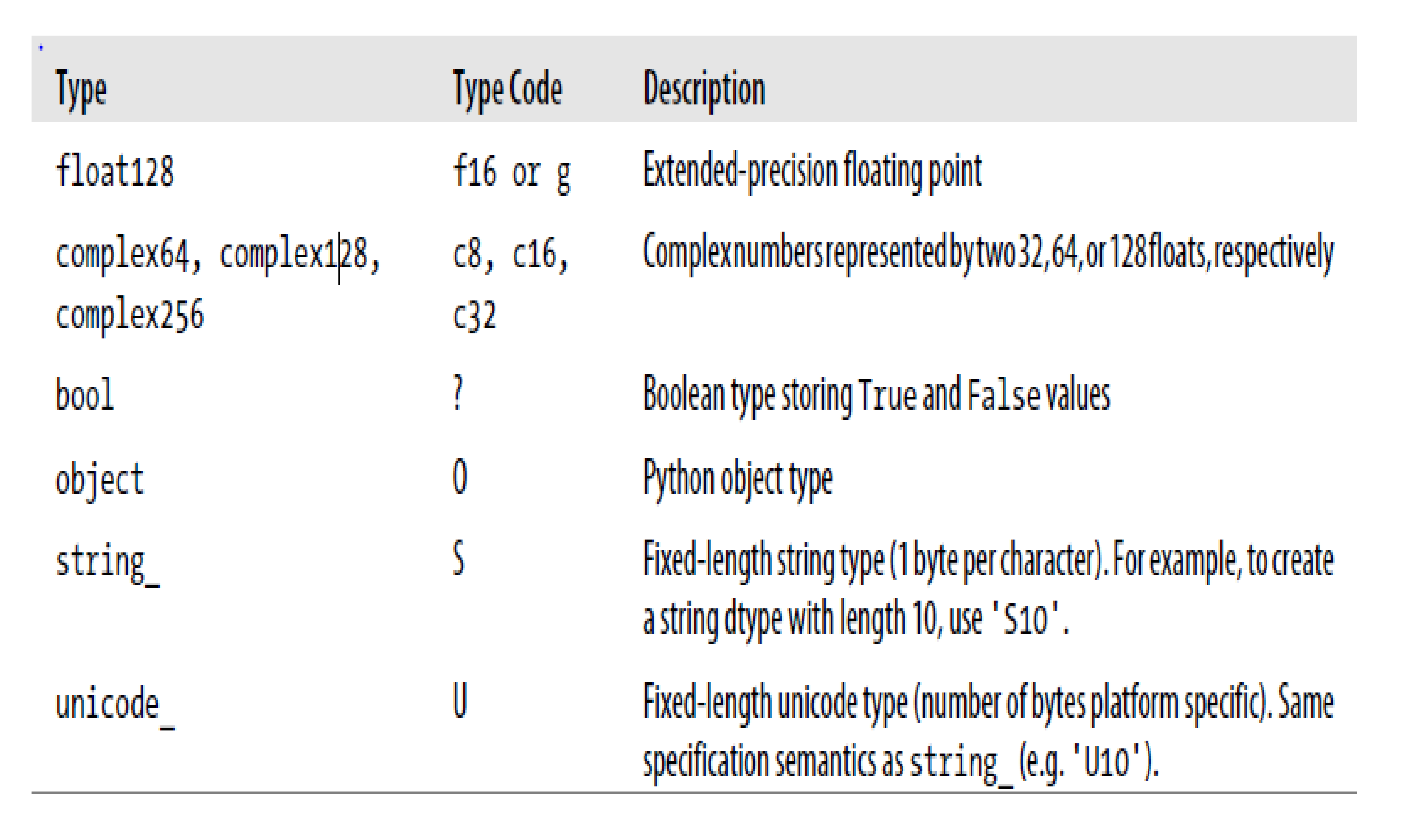


Table 4-2. NumPy data types

Type	Type Code	Description
`int8, uint8`	`i1, u1`	Signed and unsigned 8-bit (1 byte) integer types
`int16, uint16`	`i2, u2`	Signed and unsigned 16-bit integer types
`int32, uint32`	`i4, u4`	Signed and unsigned 32-bit integer types
`int64, uint64`	`i8, u8`	Signed and unsigned 32-bit integer types
`float16`	`f2`	Half-precision floating point
`float32`	`f4 or f`	Standard single-precision floating point. Compatible with C float
`float64, float128`	`f8 or d`	Standard double-precision floating point. Compatible with C double and Python `float` object



Type	Type Code	Description
float128	f16 or g	Extended-precision floating point
complex64, complex128, complex256	c8, c16, c32	Complexnumbersrepresentedbytwo32,64,or128floats,respectively
bool	?	Boolean type storing True and False values
object	O	Python object type
string_	S	Fixed-length string type (1 byte per character). For example, to create a string dtype with length 10, use 'S10'.
unicode_	U	Fixed-length unicode type (number of bytes platform specific). Same specification semantics as string_ (e.g. 'U10').

```
>>> x
array([22,  5,  6,  7,  8])
>>> x.dtype
dtype('int32')
>>> y=x.astype(np.float32)
>>> y
array([22.,  5.,  6.,  7.,  8.], dtype=float32)
>>> a= np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
>>> a
array([ 3.7, -1.2, -2.6,  0.5, 12.9, 10.1])
>>> x=a.astype(np.int32)
>>> x
array([ 3, -1, -2,  0, 12, 10])
```

```
>>> s = np.array(['1.25', '-9.6',
'42'], dtype=np.string_)
>>> y=s.astype(float)
>>> y
array([ 1.25, -9.6 , 42.  ])
>>> a=np.array([2,3,4,5,6,7,8])
>>> a[2:5]
array([4, 5, 6])
>>> x=a[2:5]
>>> x[:]=25
>>> a
array([ 2,  3, 25, 25, 25,  7,  8])
```

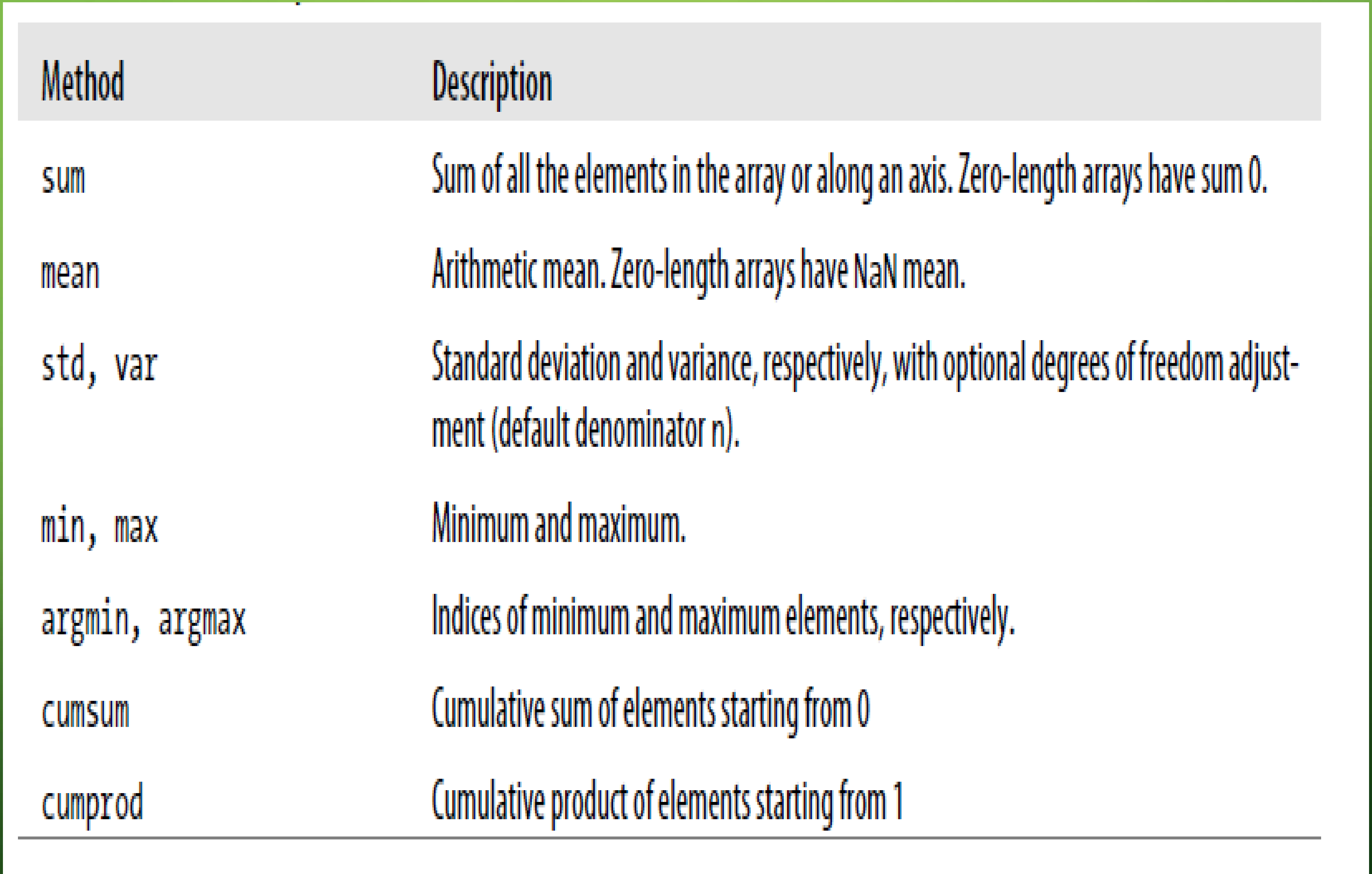
```
>>> a= np.array([[1, 2, 3],
[4, 5, 6], [7, 8, 9]])
>>> a
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> a[1][1]
5
>>> a[1,1]
5
>>> a[0:2,0:2]
array([[1, 2],
       [4, 5]])
```

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

       [[ 7,  8,  9],
        [10, 11, 12]]])
>>> a.shape
(2, 2, 3)
>>> a[0,1,1]
5
>>> x[:,:]=5
>>> x
array([[5, 5, 5],
       [5, 5, 5]])
```

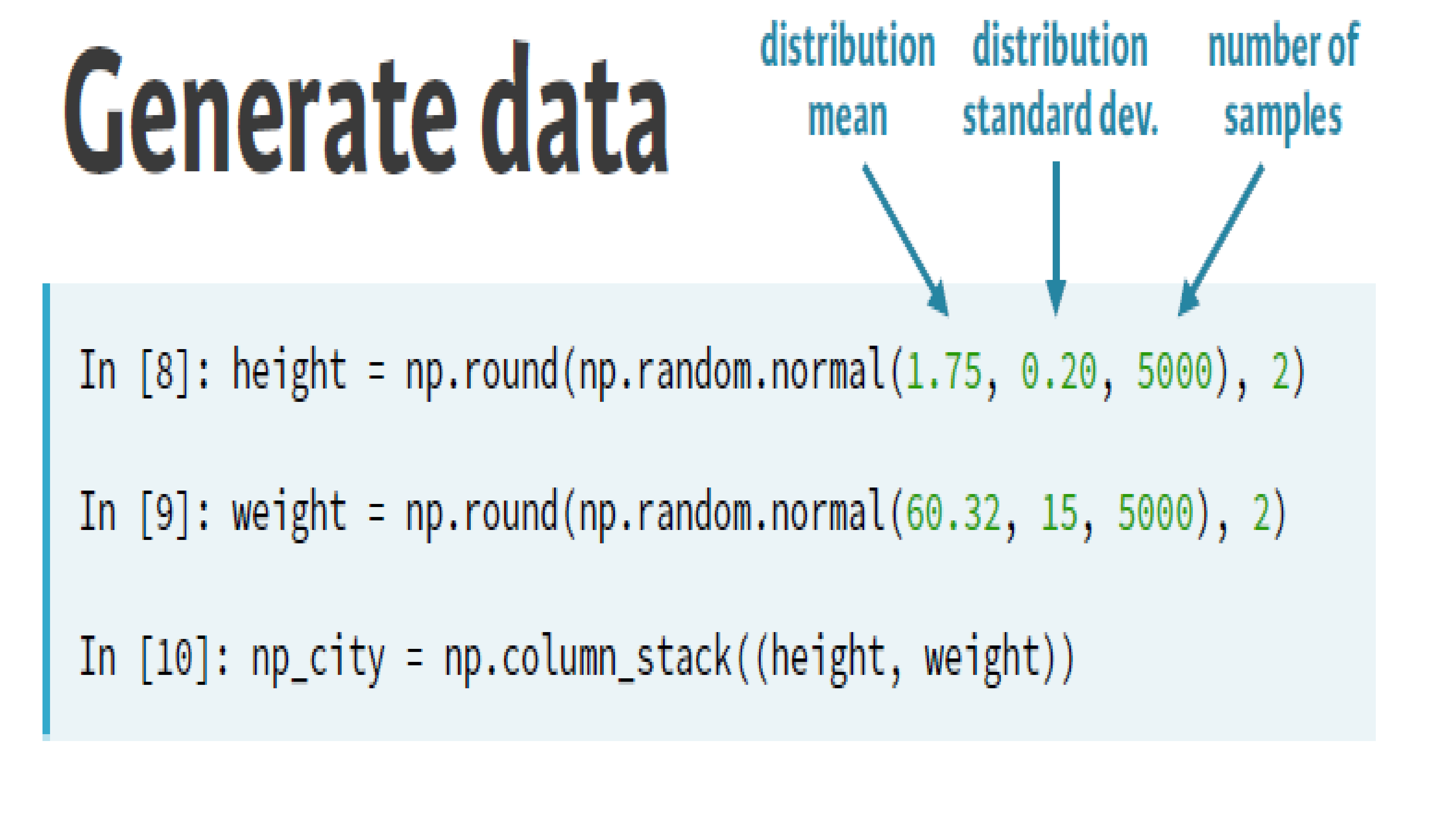
```
>>>
a=np.array([2,3,4,5,6,7])
>>> x=a%2==0
>>> x
array([ True, False,  True, False,
True, False])
>>> a[x]
array([2, 4, 6])
>>> x[[3,1,2]]
array([[4, 4, 4],
       [2, 2, 2],
       [3, 3, 3]])
```

```
x=np.arange(15).reshape((3, 5))
>>> x
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> y=x.reshape((5,3))
>>> y
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

```
>>> x=np.array([[1,2,3],[1,2,3]])
>>> y=np.array([[1,2,3],[1,2,3]])
>>> yt=y.T
>>> yt
array([[1, 1],
       [2, 2],
       [3, 3]])
>>> z=np.dot(x,yt)
>>> z
array([[14, 14],
       [14, 14]])

x=np.meshgrid(p,p)
```



Method	Description
sum	Sum of all the elements in the array or along an axis. Zero-length arrays have sum 0.
mean	Arithmetic mean. Zero-length arrays have NaN mean.
std, var	Standard deviation and variance, respectively, with optional degrees of freedom adjustment (default denominator n).
min, max	Minimum and maximum.
argmin, argmax	Indices of minimum and maximum elements, respectively.
cumsum	Cumulative sum of elements starting from 0
cumprod	Cumulative product of elements starting from 1



Generate data

distribution mean

distribution standard dev.

number of samples

```
In [8]: height = np.round(np.random.normal(1.75, 0.20, 5000), 2)

In [9]: weight = np.round(np.random.normal(60.32, 15, 5000), 2)

In [10]: np_city = np.column_stack((height, weight))
In [24]: np.logical_and(bmi > 21, bmi < 22)
Out[24]: array([ True, False, True, False, True], dtype=bool)
```

logical_and()

logical_or()

logical_not()

```
In [25]: bmi[np.logical_and(bmi > 21, bmi < 22)]
Out[25]: array([ 21.852, 21.75, 21.441])
```

In [1]:

```
import numpy as np
x=np.array([2,5,6,7,8])
print(x)
```

```
[2 5 6 7 8]
```

In [2]:

```
x[0]=12
print(x)
```

```
[12  5  6  7  8]
```

In [3]:

```
y=x*2
print(y)
```

```
[24 10 12 14 16]
```

In [4]:

```
y=x+2
print(y)
```

```
[14  7  8  9 10]
```

In [5]:

```
y=x**2
print(y)
```

```
[144  25  36  49  64]
```

In [6]:

```
print(x.shape)
print(y.shape)
```

```
(5,)
(5,)
```

In [7]:

```
d=[[2,4,3,6],[2,3,4,5],[8,2,5,7],[9,2,4,6]]
s=np.array(d)
print(s)
print(s.shape)
```

```
[[2 4 3 6]
 [2 3 4 5]
 [8 2 5 7]
 [9 2 4 6]]
(4, 4)
```

In [49]:

```
print(s.ndim)
```

```
1
```

In [9]:

```
t=np.zeros(10,int)
print(t)
```

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

In [10]:

```
t=np.zeros((3,5),int)
print(t)
```

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

In [11]:

```
t=np.ones(5)
print(t)
t=np.ones((5,4),int)
print(t)
```

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

In [12]:

```
t=np.arange(2,10,1.3)
print(t)
```

```
[2.  3.3 4.6 5.9 7.2 8.5 9.8]
```

In [13]:

```
t=np.logspace(4,5,4)
print(t)
```

```
[ 10000.           21544.34690032  46415.88833613 100000.        ]
```

In [14]:

```
t=np.linspace(1,10,5)
print(t)
```

```
[ 1.    3.25  5.5   7.75 10.  ]
```

In [15]:

```
x=np.array([2,3,4,5,6,7])
y=x.astype(np.float32)
print(x)
print(y)
```

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

In [16]:

```
a= np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
x=a.astype(np.int32)
print(x)
```

```
[ 3 -1 -2  0 12 10]
```

In [17]:

```
s = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
y=s.astype(float)
print(y)
```

```
[ 1.25 -9.6  42.  ]
```

In [18]:

```
a=np.array([2,4,6,5,7,4,3,2,1])
print(a[2:5])
```

```
[6 5 7]
```

In [19]:

```
x=a[2:5]
x[:]=12
print(x)
print(a)
```

```
[12 12 12]
[ 2  4 12 12 12  4  3  2  1]
```

In [20]:

```
a= np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
a[0:2,0:2]
```

Out[20]:

```
array([[1, 2],
       [4, 5]])
```

In [21]:

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

```
(2, 2, 3)
```

In [22]:

```
print(a[0,1,1])
a[:,:]=5
```

```
5
```

In [23]:

```
print(a)
```

```
[[[5 5 5]
  [5 5 5]]

 [[5 5 5]
  [5 5 5]]]
```

In [24]:

```
a=np.array([2,3,4,5,6,7])
x=a%2==0
print(x)
```

```
[ True False  True False  True False]
```

In [25]:

```
x=np.arange(15).reshape((3, 5))
y=x.reshape((5,3))
```

In [26]:

```
x=np.array([[1,2,3],[1,2,3]])
y=np.array([[1,2,3],[1,2,3]])
yt=y.T
print(y)
z=np.dot(x,yt)
print(z)
```

```
[[1 2 3]
 [1 2 3]]
[[14 14]
 [14 14]]
```

In [27]:

```
print(x)
x.sum()
```

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

Out[27]:

```
12
```

In [28]:

```
x.sum(axis=1)
```

Out[28]:

```
array([6, 6])
```

In [29]:

```
x.sum(axis=0)
```

Out[29]:

```
array([2, 4, 6])
```

```
In [30]: x.mean(axis=0)

Out[30]: array([1., 2., 3.])
```

```
In [31]: x.std(axis=0)

Out[31]: array([0., 0., 0.])
```

```
In [32]: h=np.round(np.random.normal(1.75,0.20,5000),2)
         w=np.round(np.random.normal(60.32,15,5000),2)
         bmi = w /h ** 2
         print(bmi)
         print(bmi[bmi>40])

         [20.51104174 26.45726808 15.10062358 ... 17.34601573 24.78266745
          13.26905418]
         [42.04888889 46.57210402 40.11697117 43.12444444 43.22059115 43.20743119
          46.9138057  53.06995885 44.32074312 42.63565891 49.45616249 54.27548257
          59.25645359 45.99072939 40.0567542  41.46731312 46.41836735 51.58847737
          41.22781636 48.32520173 46.59209157 40.03675652 52.17234541 60.11352539
          47.05944129 42.64034914 41.09898807 44.30178326 44.07187068 46.97976592
          43.296      50.54103186 60.91874423 42.82578875 68.63905325 40.35811569
          41.4738341  41.16493134 47.6308876  42.45689655 49.49221802 47.110152
          45.91676576 40.61121613 41.28191001 44.49562652 50.53950084 49.71451184
          40.43579102 40.76678084 52.17759311 43.81253887 46.70237345 40.57988166
          63.75597149 45.66207076 47.65673582 42.64061359 43.09997313 41.88924098
          67.09917355 50.12725045 40.4869732  47.38049476 42.47669113 51.20654984
          43.86418922 46.17170958 48.3264     61.75746384 40.8622449  41.34183673
          46.98400136 47.23789448 40.02008766 43.51714678 45.16129032 42.68877551
          46.55556131 43.52987732 44.20289855 47.12757202 42.76444847 40.45857988
          40.77452743]
```

```
In [33]: np.logical_and(h > 1.2, h < 1.4)

Out[33]: array([False, False, False, ..., False, False, False])
```

logical_and()

logical_or()

logical_not()

In [34]:

```
import numpy as np
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
meas = np.array([np_height, np_weight])
print(meas)
for val in np.nditer(meas):
    print(val)
```

```
[[ 1.73  1.68  1.71  1.89  1.79]
 [65.4  59.2  63.6  88.4  68.7 ]]
1.73
1.68
1.71
1.89
1.79
65.4
59.2
63.6
88.4
68.7
```

In [35]:

```
print(np.random.rand())
```

```
0.809297109750017
```

In [36]:

```
np.random.seed(123)
```

In [37]:

```
np.random.rand()
```

Out[37]:

```
0.6964691855978616
```

In [38]:

```
np.random.rand()
```

Out[38]:

```
0.28613933495037946
```

In [39]:

```
np.random.seed(123)
np.random.rand()
```

Out[39]:

```
0.6964691855978616
```

In [40]:

```
np.random.seed(int(input("enter seed")))
np.random.randint(0,6)
```

```
enter seed6
```

Out[40]:

```
2
```

In [41]:

```
np.random.randint(0,2)
```

Out[41]:

```
1
```

In [42]:

```
import numpy as np
np.random.seed(123)
outcomes = []
for x in range(10) :
    coin = np.random.randint(0, 2)
    if coin == 0 :
        outcomes.append("heads")
    else :
        outcomes.append("tails")
print(outcomes)
```

```
['heads', 'tails', 'heads', 'heads', 'heads', 'heads', 'heads', 'tails', 'tails', 'h
eads']
```

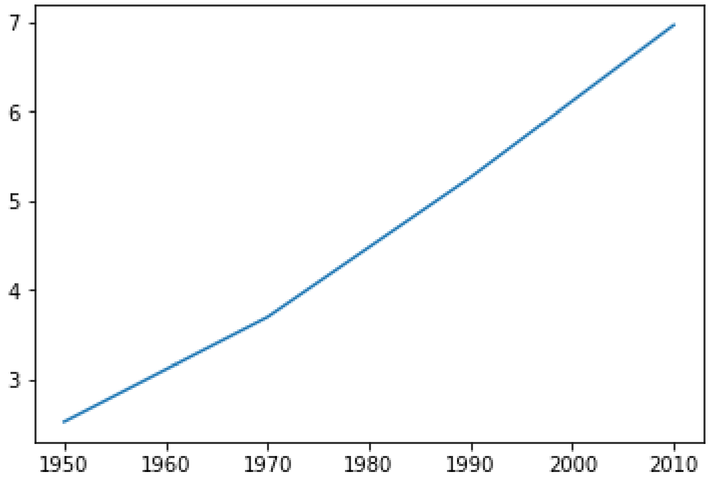
In [ ]:

In [43]:

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

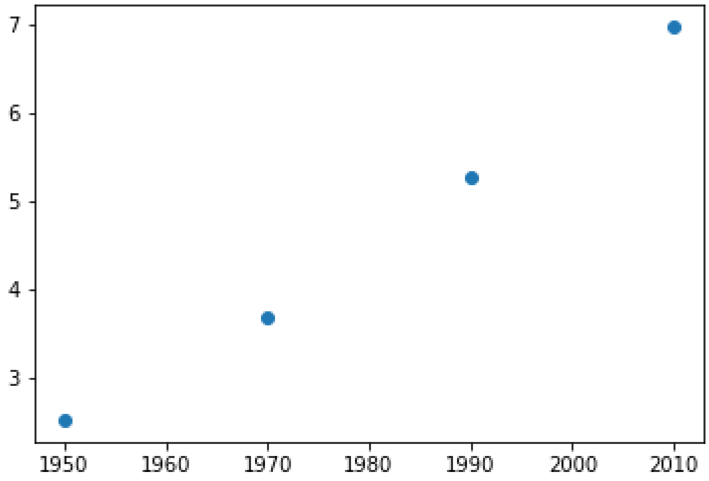
In [44]:

```
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
plt.plot(year,pop)
plt.show()
```



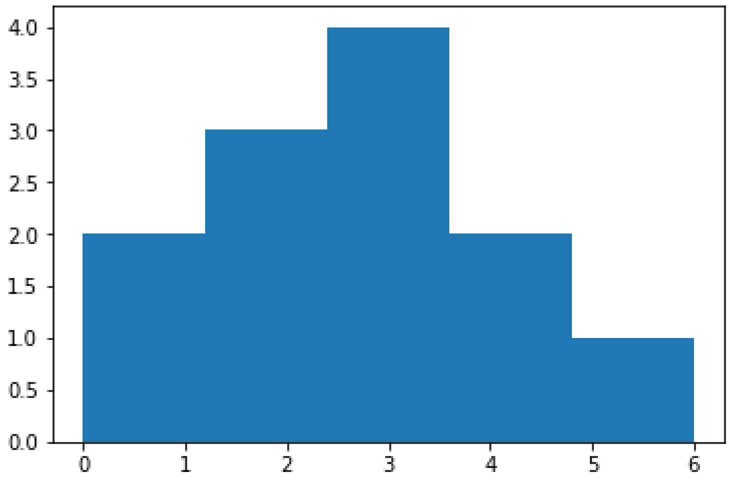
In [45]:

```
plt.scatter(year, pop)
plt.show()
```



In [46]:

```
x= [0,0.6,1.4,1.6,2.2,2.5,2.6,3.2,3.5,3.9,4.2,6]
plt.hist(x, bins = 5)
plt.show()
```

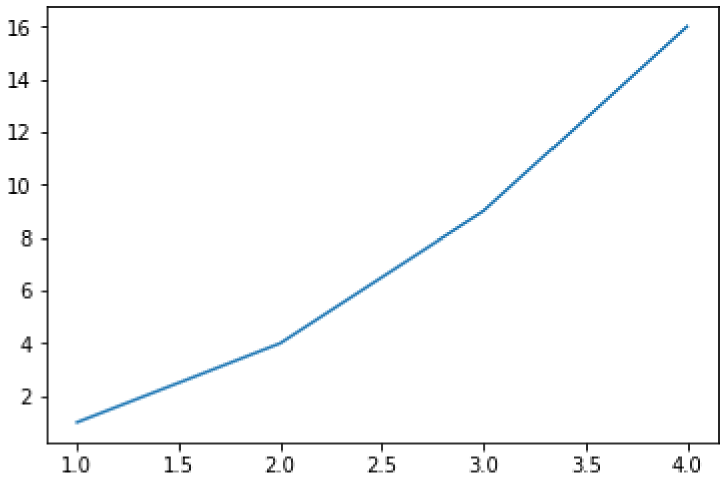


In [47]:

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show
```

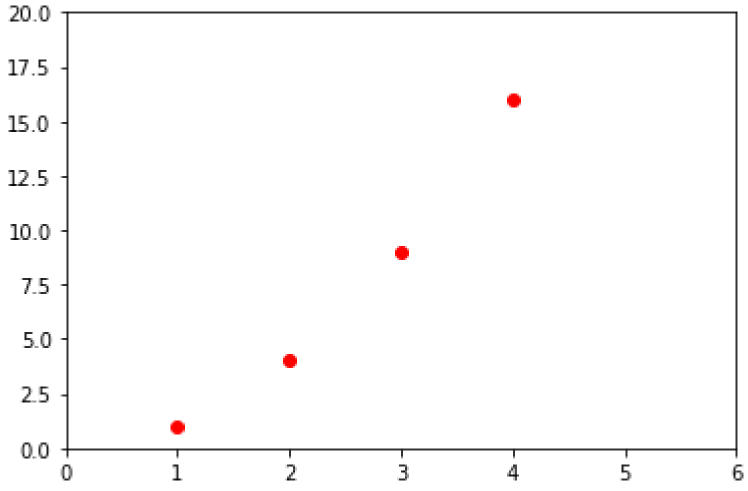
Out[47]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [48]:

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



In [ ]:

MATH MODULE

The **math** module is used to access mathematical functions in the Python. All methods of this function are used for integer or real type objects, not for complex numbers.

To use this module, we should import that module into our code.

```
import math
```

Some Constants

These constants are used to put them into our calculations.

Sr.No.	Constants & Description
1	**pi** Return the value of pi: 3.141592
2	**E** Return the value of natural base e. e is 0.718282
3	**tau** Returns the value of tau. tau = 6.283185
4	**inf** Returns the infinite
5	**nan** Not a number type.

Numbers and Numeric Representation

These functions are used to represent numbers in different forms. The methods are like below −

Sr.No.	Function & Description
1	**ceil(x)** Return the Ceiling value. It is the smallest integer, greater or equal to the number x.
2	**copysign(x, y)** It returns the number x and copy the sign of y to x.
3	**fabs(x)** Returns the absolute value of x.
4	**factorial(x)** Returns factorial of x. where x ≥ 0
5	**floor(x)** Return the Floor value. It is the largest integer, less or equal to the number x.
6	**fsum(iterable)** Find sum of the elements in an iterable object
7	**gcd(x, y)** Returns the Greatest Common Divisor of x and y
8	**isfinite(x)** Checks whether x is neither an infinity nor nan.
9	**isinf(x)** Checks whether x is infinity
10	**isnan(x)**

	Checks whether x is not a number.
11	**remainder(x, y)** Find remainder after dividing x by y.

Example Code

```
import math

print('The Floor and Ceiling value of 23.56 are: ' +
str(math.ceil(23.56)) + ', ' + str(math.floor(23.56)))

x = 10

y = -15

print('The value of x after copying the sign from y is: ' +
str(math.copysign(x, y)))

print('Absolute value of -96 and 56 are: ' + str(math.fabs(-96)) +
', ' + str(math.fabs(56)))

my_list = [12, 4.25, 89, 3.02, -65.23, -7.2, 6.3]

print('Sum of the elements of the list: ' +
str(math.fsum(my_list)))

print('The GCD of 24 and 56 : ' + str(math.gcd(24, 56)))

x = float('nan')

if math.isnan(x):

    print('It is not a number')

x = float('inf')

y = 45

if math.isinf(x):

    print('It is Infinity')

print(math.isfinite(x)) #x is not a finite number

print(math.isfinite(y)) #y is a finite number
```

Output

```
The Floor and Ceiling value of 23.56 are: 24, 23
The value of x after copying the sign from y is: -10.0
Absolute value of -96 and 56 are: 96.0, 56.0
Sum of the elements of the list: 42.13999999999999
The GCD of 24 and 56 : 8
It is not a number
It is Infinity
False
True
```

Power and Logarithmic Functions

These functions are used to calculate different power related and logarithmic related tasks.

Sr.No.	Function & Description
1	**pow(x, y)** Return the x to the power y value.
2	**sqrt(x)** Finds the square root of x
3	**exp(x)** Finds xe, where e = 2.718281
4	**log(x[, base])** Returns the Log of x, where base is given. The default base is e
5	**log2(x)** Returns the Log of x, where base is 2
6	**log10(x)** Returns the Log of x, where base is 10

Example Code

```
import math

print('The value of 5^8: ' + str(math.pow(5, 8)))

print('Square root of 400: ' + str(math.sqrt(400)))

print('The value of 5^e: ' + str(math.exp(5)))

print('The value of Log(625), base 5: ' + str(math.log(625, 5)))

print('The value of Log(1024), base 2: ' + str(math.log2(1024)))

print('The value of Log(1024), base 10: ' + str(math.log10(1024)))
```

Output

```
The value of 5^8: 390625.0
Square root of 400: 20.0
The value of 5^e: 148.4131591025766
The value of Log(625), base 5: 4.0
The value of Log(1024), base 2: 10.0
The value of Log(1024), base 10: 3.010299956639812
```

Trigonometric & Angular Conversion Functions

These functions are used to calculate different trigonometric operations.

Sr.No.	Function & Description
1	**sin(x)** Return the sine of x in radians
2	**cos(x)** Return the cosine of x in radians
3	**tan(x)** Return the tangent of x in radians
4	**asin(x)** This is the inverse operation of the sine, there are acos, atan also.

5	**degrees(x)** Convert angle x from radian to degrees
6	**radians(x)** Convert angle x from degrees to radian

Example Code

```
import math

print('The value of Sin(60 degree): ' +
str(math.sin(math.radians(60))))

print('The value of cos(pi): ' + str(math.cos(math.pi)))

print('The value of tan(90 degree): ' + str(math.tan(math.pi/2)))

print('The angle of sin(0.8660254037844386): ' +
str(math.degrees(math.asin(0.8660254037844386))))
```

Output

```
The value of Sin(60 degree): 0.8660254037844386
The value of cos(pi): -1.0
The value of tan(90 degree): 1.633123935319537e+16
The angle of sin(0.8660254037844386): 59.99999999999999
```