NIELIT ML course

July 5, 2021

```
[1]: x=65
     type(x)
```

```
[1]: int
```

```
[2]: x=2
     y=4
     x+y
```

```
[2]: 6
```

```
[8]: x=500.6
     y=1900.9
     x-y
     round(x-y,1)
```

```
[8]: -1400.3
```

```
[3]: l=[10.13,20,"30",[10,20.9,"ram"]] #lists can have different data types
     l
```

```
[3]: [10.13, 20, '30', [10, 20.9, 'ram']]
```

```
[4]: l[2]=14 #lists can be indexed with forward and reverse indexing
     l
```

```
[4]: [10.13, 20, 14, [10, 20.9, 'ram']]
```

```
[5]: l[0:5:2] #we can access different elements of list at different separation. This␣
      ↪method is called 'Forward indexing slicing'
```

```
[5]: [10.13, 14]
```

```
[6]: l[-1] #this is reverse indexing
```

```
[6]: [10, 20.9, 'ram']
```

```
[10]: l #this is the list
```

```
[10]: [10.13, 20, 14, [10, 20.9, 'ram']]
```

```
[11]: l.index(10.13) #this is how we get index of a particular element
```

```
[11]: 0
```

```
[14]: l.index(14) #getting index of element 14
```

```
[14]: 2
```

```
[15]: l[-3:-1] #this is reverse index slicing
```

```
[15]: [20, 14]
```

```
[16]: l[-3:-1:-1] # this gives us an empty list as we are incrementing by -1 whereas␣
       ↪it should increment y 1
```

```
[16]: []
```

```
[17]: l[:] #gives full list
```

```
[17]: [10.13, 20, 14, [10, 20.9, 'ram']]
```

```
[20]: l[::-1] #gi
```

```
[20]: [[10, 20.9, 'ram'], 14, 20, 10.13]
```

```
[19]: l[:3]
```

```
[19]: [10.13, 20, 14, [10, 20.9, 'ram']]
```

```
[26]: list(range(1,100,2)) #this gives you all odd  numbers list
```

```
[26]: [1,
       3,
       5,
       7,
       9,
       11,
       13,
       15,
       17,
       19,
       21,
       23,
       25,
       27,
 29,
 31,
 33,
 35,
 37,
 39,
 41,
 43,
 45,
 47,
 49,
 51,
 53,
 55,
 57,
 59,
 61,
 63,
 65,
 67,
 69,
 71,
 73,
 75,
 77,
 79,
 81,
 83,
 85,
 87,
 89,
 91,
 93,
 95,
 97,
 99]
```

[28]:
```
x=list(range(0,100,2))#and this gives a list of all even numbers
len(x) #this gives the number of elements in the list of even numbers
```

[28]:
```
50
```

[29]:
```
l
```

[29]:
```
[10.13, 20, 14, [10, 20.9, 'ram']]
```

```
[32]: l[0:2]=list(range(152,156,1)) #here we changed the first 3 elements and put␣
      ↪values between 152 to 155 with a separation of 1
      l #this is the new list
```

```
[32]: [152, 153, 154, 155, 154, 155, 14, [10, 20.9, 'ram']]
```

```
[33]: type(l) #how to know the type of a list
```

```
[33]: list
```

```
[35]: l[7][2] #how to calculate elements of a sublist
```

```
[35]: 'ram'
```

```
[36]: type(l[7][2]) #the type of 'ram' is string
```

```
[36]: str
```

```
[37]: l.append("DRDO") #appending an element to the list, more than one element can␣
      ↪not be appended at the same time
      l
```

```
[37]: [152, 153, 154, 155, 154, 155, 14, [10, 20.9, 'ram'], 'DRDO']
```

```
[38]: l.pop() #popping up the last element
```

```
[38]: 'DRDO'
```

```
[39]: l.pop(2) #2nd element popped
```

```
[39]: 154
```

```
[2]: l=[20,30,50,70]
     l
```

```
[2]: [20, 30, 50, 70]
```

```
[3]: l.append(100)
     l
```

```
[3]: [20, 30, 50, 70, 100]
```

```
[4]: l.pop() # pop will show last element and remove it from the list
     l
```

```
[4]: [20, 30, 50, 70]
```

```
[5]: l.pop(1) # popping element at position 1
```

```
[5]: 30
```

```
[6]: l.insert(1,1000) #inserting an element 1000 at position 1
     l
```

```
[6]: [20, 1000, 50, 70]
```

```
[7]: l.clear() #clear method creates an empty list. There will be no element after␣
      ↪clear method
     l
```

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

```
[8]: l1=[10,20,30] #adding two lists together
     l2=[50,60,70]
     l=l1+l2
     l
```

```
[8]: [10, 20, 30, 50, 60, 70]
```

```
[29]: l1=[10,20,30]
      l2=[50,60,70]
      x=l2+l1
```

```
[10]: m=l1+2*l2 #ok this is nice, 2nd list is getting added twice
      m
```

```
[10]: [10, 20, 30, 50, 60, 70, 50, 60, 70]
```

```
[14]: l1.extend(l2) #this thing is not working , need to be checked
```

```
[16]: 2
      l
```

```
[16]: [10, 20, 30, 50, 60, 70]
```

```
[17]: l.reverse()
```

```
[18]: l1
```

```
[18]: [10, 20, 30, 50, 60, 70, 50, 60, 70, 50, 60, 70]
```

```
[19]: l1.reverse() #this reverses the elements in the list
```

```
[20]: l1
```

```
[20]: [70, 60, 50, 70, 60, 50, 70, 60, 50, 30, 20, 10]
```

```
[25]: i=2
       for i<10:
         i++
      print(i)
```

```
  File "<ipython-input-25-02359d58472f>", line 2
    for i<10:
    ^
IndentationError: unexpected indent
```

```
[30]: l1=[10,20,30]
      l2=[50,60,70]
      x=l2+l1
```

```
[31]: p=52
      p
```

```
[31]: 52
```

```
[35]: l1=[5,6,7]
      m=l1*3
      l1
      m
```

```
[35]: [5, 6, 7, 5, 6, 7, 5, 6, 7]
```

```
[48]: k=[]
      l=[[1,2,3],[2,3,4],[5,6,8]]
      for i in l:
          (i.reverse())
          k.append(i)
      k
```

```
[48]: [[3, 2, 1], [4, 3, 2], [8, 6, 5]]
```

```
[46]: l=[[10,20,30],[40,20,50],[84,90,28]] #this is a list, its each element need to␣
       ↪be reversed
      k=[]
      for i in l:
          i.reverse()
          k.append(i)
      print(k)
```

```
[[30, 20, 10], [50, 20, 40], [28, 90, 84]]
```

[51]:
```
# making a list from user input
l1=[]
for i in list(range(1,7)):
   l1.append(input("enter your name:"))
l1
```

```
enter your name:a
enter your name:b
enter your name:c
enter your name:d
enter your name:e
enter your name:f
```

[51]:
```
['a', 'b', 'c', 'd', 'e', 'f']
```

[52]:
```
l=[10,20,30,40,50,60,70]
l[-3:-1]
```

[52]:
```
[50, 60]
```

[53]:
```
l[:-2]
```

[53]:
```
[10, 20, 30, 40, 50]
```

[54]:
```
l[: :2]
```

[54]:
```
[10, 30, 50, 70]
```

[56]:
```
l=list(range(0,20,5))
l
```

[56]:
```
[0, 5, 10, 15]
```

[58]:
```
l=[]
l.append(300)
l.append('python')
l.append([])
l
```

[58]:
```
[300, 'python', []]
```

[61]:
```
l1=[20,40]
l2=[60,70]
l3=l1.extend(l2)
l3
```

[ ]:
```

```

```
[1]: print('hello world')
```

```
hello world
```

```
[3]: print('From here we start Tuple. Tuple is written inside curved braces ()')
```

```
From here we start Tuple. Tuple is written inside curved braces ()
```

```
[4]: t=(1,2,3)
     type(t)
```

```
[4]: tuple
```

```
[5]: t=(1,2,3,"ram")
     type(t)
```

```
[5]: tuple
```

```
[6]: t=([1,2,3],2,3.89) #tuples can contain lists, int, strings or floats
     type(t)
```

```
[6]: tuple
```

```
[7]: t=(20, ) #it is important to put a , after int if you are putting only 1␣
      ↪integer, otherwise it will take the type as int only. This does not happen for␣
      ↪list. This is a difference between list and tuple
     type(t)
```

```
[7]: tuple
```

```
[8]: a=1,2,3 #tuple can be written without brackets
     type (a)
```

```
[8]: tuple
```

```
[13]: x,y,z=10,20,30
      a=x,y,z
      print(a )
      print(type(a))
```

```
(10, 20, 30)
<class 'tuple'>
```

```
[15]: t=(20, 40,50) #accessing a specific element from tuple
      t[1]
```

```
[15]: 40
```

[16]:
```
#in list you can update any value but in tuple you can not contain basic data␣
 ↪types- int, float, string.
t=(2,3,4,[4,5,6])
t[3][1]=40
t
```

[16]:
```
(2, 3, 4, [4, 40, 6])
```

[18]:
```
l=[2*i for i in range(1,4)] #operations ar not possible at tupple
print(l)
t=(2*i for i in range(1,4))
print(t)
```

```
[2, 4, 6]
<generator object <genexpr> at 0x000001E99AB4BF90>
```

[23]:
```
t=(3,3,3,3,4,5,6,7) #count how many times 3 is coming
a=t.count(3)
print(a)
b=t.index(3) #if multiple instances are there then the index is taken to be 0
print(b)
l=list(t) #converting tuple to a list
print(l)
x=tuple(l)
print(x)
```

```
4
0
[3, 3, 3, 3, 4, 5, 6, 7]
(3, 3, 3, 3, 4, 5, 6, 7)
```

[27]:
```
#here we calculate whether tuples are faster than list,
import time #time module imported
begin=time.time() #we note beginning time
t=(3,3,3,3,4,5,6,7)
print(t)
time.sleep(1) #here we stop the timer for begin time
end=time.time() # we note the end time here
print(end-begin)
```

```
(3, 3, 3, 3, 4, 5, 6, 7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-27-c700cce8611c> in <module>
      6 time.sleep(1) #here we stop the timer for begin time
      7 end=time.time() # we note the end time here
----> 8 print(" time required for processing tuple is "+end-begin)
```

```
TypeError: can only concatenate str (not "float") to str
```

[26]:
```
#Similarly for list, the time duration is calculated, we note from the output␣
↪that the list is slower than tuple
import time
begin=time.time()
t=[3,3,3,3,4,5,6,7]
print(t)
time.sleep(1)
end=time.time()
print(end-begin)
```

```
[3, 3, 3, 3, 4, 5, 6, 7]
1.0054833889007568
```

[29]:
```
#Similar way to calculate processing time for lists
begin=time.time()
l=list(range(1,10))
print(l)
l1=[i*i for i in l]
print(l1)
time.sleep(1)
end=time.time()
print(end-begin)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 4, 9, 16, 25, 36, 49, 64, 81]
1.007704734802246
```

[30]:
```
t=(1,2,3)*3 #here the tuple operation is created because till this line the␣
↪tuple is not created and interpreter will calculate once the tuple is created
t
```

[30]:
```
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

[1]:
```
#Dictionary
d={}
type(d)
```

[1]:
```
dict
```

[2]:
```
#a dictionary has a key part  and a value. Key contains int, string, float,␣
↪tuple. Value can contain  all kind of data. For example
d={1:2}
print(len(d))
e={'a':'b','c':'d'}
```

```
print(len(e))
```

```
1
2
```

[15]:
```
o=list(range(1,101,2))
o
```

[15]:
```
[1,
 3,
 5,
 7,
 9,
 11,
 13,
 15,
 17,
 19,
 21,
 23,
 25,
 27,
 29,
 31,
 33,
 35,
 37,
 39,
 41,
 43,
 45,
 47,
 49,
 51,
 53,
 55,
 57,
 59,
 61,
 63,
 65,
 67,
 69,
 71,
 73,
 75,
 77,
 79,
 81,
 83,
 85,
 87,
 89,
 91,
 93,
 95,
 97,
 99]
```

```
[16]: e=list(range(2,100,2))
      e
```

```
[16]: [2,
 4,
 6,
 8,
 10,
 12,
 14,
 16,
 18,
 20,
 22,
 24,
 26,
 28,
 30,
 32,
 34,
 36,
 38,
 40,
 42,
 44,
 46,
 48,
 50,
 52,
 54,
 56,
 58,
 60,
 62,
 64,
 66,
 68,
 70,
 72,
 74,
 76,
 78,
 80,
 82,
 84,
 86,
 88,
 90,
 92,
 94,
 96,
 98]
```

```
[18]: d={'odd':o,'even':e}
      d['odd'] #accessing one key value from a list
```

```
[18]: [1,
       3,
       5,
       7,
       9,
       11,
       13,
       15,
       17,
       19,
       21,
       23,
       25,
       27,
       29,
       31,
       33,
       35,
       37,
       39,
       41,
       43,
       45,
       47,
       49,
       51,
 53,
 55,
 57,
 59,
 61,
 63,
 65,
 67,
 69,
 71,
 73,
 75,
 77,
 79,
 81,
 83,
 85,
 87,
 89,
 91,
 93,
 95,
 97,
 99]
```

```
[20]: d['three']=list(range(3,99,3))
      d
```

```
[20]: {'odd': [1,
        3,
        5,
        7,
        9,
        11,
        13,
        15,
        17,
        19,
        21,
        23,
        25,
        27,
        29,
        31,
        33,
        35,
        37,
  39,
  41,
  43,
  45,
  47,
  49,
  51,
  53,
  55,
  57,
  59,
  61,
  63,
  65,
  67,
  69,
  71,
  73,
  75,
  77,
  79,
  81,
  83,
  85,
  87,
  89,
  91,
  93,
  95,
  97,
  99],
 'even': [2,
  4,
  6,
  8,
  10,
  12,
  14,
  16,
  18,
  20,
  22,
  24,
  26,
  28,
  30,
  32,
  34,
  36,
  38,
  40,
  42,
  44,
  46,
  48,
  50,
  52,
  54,
  56,
  58,
  60,
  62,
  64,
  66,
  68,
  70,
  72,
  74,
  76,
  78,
  80,
  82,
  84,
  86,
  88,
  90,
  92,
  94,
  96,
  98],
 'three': [3,
  6,
  9,
  12,
  15,
  18,
  21,
  24,
  27,
  30,
  33,
  36,
  39,
  42,
  45,
  48,
  51,
  54,
  57,
  60,
  63,
  66,
  69,
  72,
  75,
  78,
  81,
  84,
  87,
  90,
  93,
  96]}
```

[21]:
```
d.keys() #accessing all the keys in dictionary
```

[21]:
```
dict_keys(['odd', 'even', 'three'])
```

[22]:
```
d.values() #accessing all the values in dictionary
```

[22]:
```
dict_values([[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35,
37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75,
77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99], [2, 4, 6, 8, 10, 12, 14, 16,
18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56,
58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96,
98], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57,
60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96]])
```

[23]:
```
b={'a':[1,2,3,[4,5,6]]} #dictionaries can contain different lists or dictionaries
b
```

[23]:
```
{'a': [1, 2, 3, [4, 5, 6]]}
```

[24]:
```
b['a'][3]
```

[24]:
```
[4, 5, 6]
```

[25]:
```
b['a'][3][0]=500 #accessing and changing one element in dictionary
b
```

[25]:
```
{'a': [1, 2, 3, [500, 5, 6]]}
```

```
[27]: t={'a':{},'b':[],'c':()}
      type (t['a'])
```

```
[27]: dict
```

```
[29]: #sets in python
      s={6,1,1,3,2,4,2,5}
      s
```

```
[29]: {1, 2, 3, 4, 5, 6}
```

```
[30]: s={6,1,2,"debajyoti",3,2,1}
      s
```

```
[30]: {1, 2, 3, 6, 'debajyoti'}
```

```
[33]: s='IITM Online Degree Program'
      print(list(s))
      print(tuple(s))
      print(set(s))
```

```
['I', 'I', 'T', 'M', ' ', 'O', 'n', 'l', 'i', 'n', 'e', ' ', 'D', 'e', 'g', 'r',
'e', 'e', ' ', 'P', 'r', 'o', 'g', 'r', 'a', 'm']
('I', 'I', 'T', 'M', ' ', 'O', 'n', 'l', 'i', 'n', 'e', ' ', 'D', 'e', 'g', 'r',
'e', 'e', ' ', 'P', 'r', 'o', 'g', 'r', 'a', 'm')
{'r', 'g', 'M', 'I', 'P', 'i', 'o', 'D', 'e', 'n', ' ', 'm', 'T', 'a', 'O', 'l'}
```

```
[34]: s='IITM Online Degree $ Program'
      s.split('$') #splitting at a particular point of a string
```

```
[34]: ['IITM Online Degree ', ' Program']
```

```
[1]: #Array starts here. Array is a homogeneous datatype
     import numpy as np
```

```
[2]: np.array([1,2,3]) # Array contains homogeneous type of data
```

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

```
[3]: np.array([1,2.6,3]) # If heterogeneous data is given, numpy converts full data␣
      ↪to a particular data type
```

```
[3]: array([1. , 2.6, 3. ])
```

```
[5]: c=np.array(('1',2,3)) #array changes a tuple into a list and changed all list␣
      ↪elements to string data type
     c
```

```
[5]: array(['1', '2', '3'], dtype='<U1')
```

```
[7]: b=np.arrange(1,100,2)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-7-76c8a71c2c3b> in <module>
----> 1 b=np.arrange(1,100,2)

C:\ProgramData\Anaconda3\lib\site-packages\numpy\__init__.py in __getattr__(attr)
    212                 return Tester
    213             else:
--> 214                 raise AttributeError("module {!r} has no attribute "

    215                                      "{!r}".format(__name__, attr))
    216 

AttributeError: module 'numpy' has no attribute 'arrange'
```

```
[8]: b=np.array(list(range(1,10,1)))
     b
```

```
[8]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
[9]: c=np.array(list(range(100,1,-1)))
     c
```

```
[9]: array([100,  99,  98,  97,  96,  95,  94,  93,  92,  91,  90,  89,  88,
             87,  86,  85,  84,  83,  82,  81,  80,  79,  78,  77,  76,  75,
             74,  73,  72,  71,  70,  69,  68,  67,  66,  65,  64,  63,  62,
             61,  60,  59,  58,  57,  56,  55,  54,  53,  52,  51,  50,  49,
             48,  47,  46,  45,  44,  43,  42,  41,  40,  39,  38,  37,  36,
             35,  34,  33,  32,  31,  30,  29,  28,  27,  26,  25,  24,  23,
             22,  21,  20,  19,  18,  17,  16,  15,  14,  13,  12,  11,  10,
              9,   8,   7,   6,   5,   4,   3,   2])
```

```
[10]: list(c) #arrays can be converted to lists
```

```
[10]: [100,
       99,
       98,
       97,
       96,
       95,
       94,
       93,
       92,
91,
90,
89,
88,
87,
86,
85,
84,
83,
82,
81,
80,
79,
78,
77,
76,
75,
74,
73,
72,
71,
70,
69,
68,
67,
66,
65,
64,
63,
62,
61,
60,
59,
58,
57,
56,
55,
54,
53,
52,
51,
50,
49,
48,
47,
46,
45,
 44,
 43,
 42,
 41,
 40,
 39,
 38,
 37,
 36,
 35,
 34,
 33,
 32,
 31,
 30,
 29,
 28,
 27,
 26,
 25,
 24,
 23,
 22,
 21,
 20,
 19,
 18,
 17,
 16,
 15,
 14,
 13,
 12,
 11,
 10,
 9,
 8,
 7,
 6,
 5,
 4,
 3,
 2]
```

[12]:
```
d=[b,c]
d
```

```
[12]: [array([1, 2, 3, 4, 5, 6, 7, 8, 9]),
        array([100,  99,  98,  97,  96,  95,  94,  93,  92,  91,  90,  89,  88,
                87,  86,  85,  84,  83,  82,  81,  80,  79,  78,  77,  76,  75,
                74,  73,  72,  71,  70,  69,  68,  67,  66,  65,  64,  63,  62,
                61,  60,  59,  58,  57,  56,  55,  54,  53,  52,  51,  50,  49,
                48,  47,  46,  45,  44,  43,  42,  41,  40,  39,  38,  37,  36,
                35,  34,  33,  32,  31,  30,  29,  28,  27,  26,  25,  24,  23,
                22,  21,  20,  19,  18,  17,  16,  15,  14,  13,  12,  11,  10,
                 9,   8,   7,   6,   5,   4,   3,   2])]
```

```
[13]: b
```

```
[13]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
[15]: f=np.exp(b) #we can do mathematical operations like exponentiation on array
      f
```

```
[15]: array([2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01,
             1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
             8.10308393e+03])
```

```
[17]: g=np.sqrt(b)
      g
```

```
[17]: array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
             2.44948974, 2.64575131, 2.82842712, 3.        ])
```

```
[18]: x={'number': b, 'sqr root': g}
      x
```

```
[18]: {'number': array([1, 2, 3, 4, 5, 6, 7, 8, 9]),
        'sqr root': array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
               2.44948974, 2.64575131, 2.82842712, 3.        ])}
```

```
[20]: import math #in list you can do math operations by importing math library
      for i in b:
          print(math.sqrt(i))
```

```
1.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
```

```
[25]: a=np.linspace(1,50,25) #dividing a particular interval 1 to 50 into 25␣
       ↪equidistant intervals
      a
```

```
[25]: array([ 1.        ,  3.04166667,  5.08333333,  7.125     ,  9.16666667,
             11.20833333, 13.25      , 15.29166667, 17.33333333, 19.375     ,
             21.41666667, 23.45833333, 25.5       , 27.54166667, 29.58333333,
             31.625     , 33.66666667, 35.70833333, 37.75      , 39.79166667,
             41.83333333, 43.875     , 45.91666667, 47.95833333, 50.        ])
```

```
[24]: print(np.array([ 1., 50.]))
```

```
[ 1. 50.]
```

```
[27]: c=np.array([[2,5],[3,5]]) #list inside list inside an array gives a matrix
      c
```

```
[27]: array([[2, 5],
             [3, 5]])
```

```
[28]: d=np.array([[102,105],[103,105]])
      d
```

```
[28]: array([[102, 105],
             [103, 105]])
```

```
[30]: e=c+d #adding two matrices
      e
```

```
[30]: array([[104, 110],
             [106, 110]])
```

```
[31]: e.ndim #calculating dimension of matrices
```

```
[31]: 2
```

```
[32]: e.shape #calculating shape of matrices
```

```
[32]: (2, 2)
```

```
[37]: a=np.array([[1,2,3],[3,4,5],[5,6,7]]) #3 dimensional matrices
      a
      b=a[0:2,0:2] #selecting a part of a bigger matrix, we can do negative indexing␣
       ↪also
      print(b)
```

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

[34]:
```
a=np.array([[1,2,3],[3,4,5]])
a
```

[34]:
```
array([[1, 2, 3],
       [3, 4, 5]])
```

[35]:
```
b=a.reshape(3,2) #reshaping a matrix to another dimension
b
```

[35]:
```
array([[1, 2],
       [3, 3],
       [4, 5]])
```

[1]:
```
import numpy as np #numpy library imported
```

[5]:
```
a=np.arange(1,100,3) #note the spelling, single r in arange!
a
```

[5]:
```
array([ 1,  4,  7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49,
       52, 55, 58, 61, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97])
```

[6]:
```
a>12 #checking conditions for an array
```

[6]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True])
```

[8]:
```
a[a>12] #boolean indexing
```

[8]:
```
array([13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58, 61,
       64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97])
```

[9]:
```
np.random.randn(30)
```

[9]:
```
array([-0.21736238,  0.58801713,  0.44569635, -1.51256734, -1.38311457,
        0.96540824,  0.02392428, -0.55798878, -0.16103693, -1.15804543,
        0.49044418,  0.62156642, -1.41194199,  0.17553799,  1.20913104,
        0.59764153,  2.62183412,  0.70155829,  0.04045968, -0.78109838,
       -1.87788615,  0.82661885, -0.81758845,  0.07144211,  0.0852401 ,
       -1.72898045, -1.15590602, -0.25728389, -0.61859594,  0.45355869])
```

[15]:
```
for i in np.arange(1,5):
    print(np.array(np.random.randn(5)))
```

```
[ 0.74582788 -0.91602147 -0.04534665  0.89502734 -0.13903417]
[ 0.6221388  -0.89161572  0.30030766  0.05529125 -0.14480873]
[ 1.58185318  0.79653392  1.69908975  0.31189824 -0.37950026]
[-0.71339163  0.74268355  0.81077402  0.7220466  -0.763921  ]
```

[17]:
```
a=np.array([[1,2],[3,4]]) #generate a matrix
a
```

[17]:
```
array([[1, 2],
       [3, 4]])
```

[18]:
```
a.flatten() #write all the elements of matrix in one row
```

[18]:
```
array([1, 2, 3, 4])
```

[19]:
```
a.ravel() #another method to write all the elements of matrix in one row
```

[19]:
```
array([1, 2, 3, 4])
```

[20]:
```
np.zeros((4,5))
```

[20]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

[21]:
```
np.ones((6,6))
```

[21]:
```
array([[1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.]])
```

[25]:
```
np.random.randn(3,3) #generating a matrix of random numbers
```

[25]:
```
array([[-1.88370907, -0.62071936, -1.49411219],
       [-0.24046853,  0.04642171, -0.86755667],
       [ 1.3905141 ,  0.77033659,  0.48828997]])
```

[3]:
```
#array concatenation
import numpy as np
arr1=np.array([1,2,3])
arr2=np.array([4,5,6])
arr=np.concatenate((arr1,arr2))
arr
```

[3]:
```
array([1, 2, 3, 4, 5, 6])
```

[4]:
```
a2=np.arange(51,101)
a2
```

[4]:
```
array([ 51,  52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,
        64,  65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,
        77,  78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,
        90,  91,  92,  93,  94,  95,  96,  97,  98,  99, 100])
```

[7]:
```
a=np.concatenate((arr,a2))
a
```

[7]:
```
array([  1,   2,   3,   4,   5,   6,  51,  52,  53,  54,  55,  56,  57,
        58,  59,  60,  61,  62,  63,  64,  65,  66,  67,  68,  69,  70,
        71,  72,  73,  74,  75,  76,  77,  78,  79,  80,  81,  82,  83,
        84,  85,  86,  87,  88,  89,  90,  91,  92,  93,  94,  95,  96,
        97,  98,  99, 100])
```

[17]:
```
a1=np.array([10,20,30])
a2=np.array([40,50,70])
a3=np.array([5,6,7])
arr1=np.stack((a1,a2,a3),axis=0) #horizontal stacking of elements over each other
print(arr1)
arr2=np.stack((a1,a2,a3), axis=1)#vertical stacking of elements over each other
print(arr2)
```

```
[[10 20 30]
 [40 50 70]
 [ 5  6  7]]
[[10 40  5]
 [20 50  6]
 [30 70  7]]
```

[22]:
```
a1=([1,1,1],[2,2,2])
a2=([3,3,3],[4,4,4])
a3=([5,5,5],[6,6,6])
ap=np.stack((a1,a2,a3)) #normal stacking
print(ap)
ax=np.hstack((a1,a2,a3)) #horizontal stacking
print(ax)
ay=np.vstack((a1,a2,a3)) #vertical stacking
print(ay)
```

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

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

[24]:
```
a=np.arange(0,1000,2)
a
```

[24]:
```
array([  0,   2,   4,   6,   8,  10,  12,  14,  16,  18,  20,  22,  24,
        26,  28,  30,  32,  34,  36,  38,  40,  42,  44,  46,  48,  50,
        52,  54,  56,  58,  60,  62,  64,  66,  68,  70,  72,  74,  76,
        78,  80,  82,  84,  86,  88,  90,  92,  94,  96,  98, 100, 102,
       104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128,
       130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154,
       156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180,
       182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 202, 204, 206,
       208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232,
       234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254, 256, 258,
       260, 262, 264, 266, 268, 270, 272, 274, 276, 278, 280, 282, 284,
       286, 288, 290, 292, 294, 296, 298, 300, 302, 304, 306, 308, 310,
       312, 314, 316, 318, 320, 322, 324, 326, 328, 330, 332, 334, 336,
       338, 340, 342, 344, 346, 348, 350, 352, 354, 356, 358, 360, 362,
       364, 366, 368, 370, 372, 374, 376, 378, 380, 382, 384, 386, 388,
       390, 392, 394, 396, 398, 400, 402, 404, 406, 408, 410, 412, 414,
       416, 418, 420, 422, 424, 426, 428, 430, 432, 434, 436, 438, 440,
       442, 444, 446, 448, 450, 452, 454, 456, 458, 460, 462, 464, 466,
       468, 470, 472, 474, 476, 478, 480, 482, 484, 486, 488, 490, 492,
       494, 496, 498, 500, 502, 504, 506, 508, 510, 512, 514, 516, 518,
       520, 522, 524, 526, 528, 530, 532, 534, 536, 538, 540, 542, 544,
       546, 548, 550, 552, 554, 556, 558, 560, 562, 564, 566, 568, 570,
       572, 574, 576, 578, 580, 582, 584, 586, 588, 590, 592, 594, 596,
       598, 600, 602, 604, 606, 608, 610, 612, 614, 616, 618, 620, 622,
       624, 626, 628, 630, 632, 634, 636, 638, 640, 642, 644, 646, 648,
       650, 652, 654, 656, 658, 660, 662, 664, 666, 668, 670, 672, 674,
       676, 678, 680, 682, 684, 686, 688, 690, 692, 694, 696, 698, 700,
       702, 704, 706, 708, 710, 712, 714, 716, 718, 720, 722, 724, 726,
       728, 730, 732, 734, 736, 738, 740, 742, 744, 746, 748, 750, 752,
       754, 756, 758, 760, 762, 764, 766, 768, 770, 772, 774, 776, 778,
       780, 782, 784, 786, 788, 790, 792, 794, 796, 798, 800, 802, 804,
       806, 808, 810, 812, 814, 816, 818, 820, 822, 824, 826, 828, 830,
       832, 834, 836, 838, 840, 842, 844, 846, 848, 850, 852, 854, 856,
       858, 860, 862, 864, 866, 868, 870, 872, 874, 876, 878, 880, 882,
       884, 886, 888, 890, 892, 894, 896, 898, 900, 902, 904, 906, 908,
       910, 912, 914, 916, 918, 920, 922, 924, 926, 928, 930, 932, 934,
       936, 938, 940, 942, 944, 946, 948, 950, 952, 954, 956, 958, 960,
       962, 964, 966, 968, 970, 972, 974, 976, 978, 980, 982, 984, 986,
       988, 990, 992, 994, 996, 998])
```

[31]:
```
x=np.array_split(a,4)
x
```

[31]:
```
[array([  0,   2,   4,   6,   8,  10,  12,  14,  16,  18,  20,  22,  24,
         26,  28,  30,  32,  34,  36,  38,  40,  42,  44,  46,  48,  50,
         52,  54,  56,  58,  60,  62,  64,  66,  68,  70,  72,  74,  76,
         78,  80,  82,  84,  86,  88,  90,  92,  94,  96,  98, 100, 102,
        104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128,
        130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154,
        156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180,
        182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 202, 204, 206,
        208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232,
        234, 236, 238, 240, 242, 244, 246, 248]),
 array([250, 252, 254, 256, 258, 260, 262, 264, 266, 268, 270, 272, 274,
        276, 278, 280, 282, 284, 286, 288, 290, 292, 294, 296, 298, 300,
        302, 304, 306, 308, 310, 312, 314, 316, 318, 320, 322, 324, 326,
        328, 330, 332, 334, 336, 338, 340, 342, 344, 346, 348, 350, 352,
        354, 356, 358, 360, 362, 364, 366, 368, 370, 372, 374, 376, 378,
        380, 382, 384, 386, 388, 390, 392, 394, 396, 398, 400, 402, 404,
        406, 408, 410, 412, 414, 416, 418, 420, 422, 424, 426, 428, 430,
        432, 434, 436, 438, 440, 442, 444, 446, 448, 450, 452, 454, 456,
        458, 460, 462, 464, 466, 468, 470, 472, 474, 476, 478, 480, 482,
        484, 486, 488, 490, 492, 494, 496, 498]),
 array([500, 502, 504, 506, 508, 510, 512, 514, 516, 518, 520, 522, 524,
        526, 528, 530, 532, 534, 536, 538, 540, 542, 544, 546, 548, 550,
        552, 554, 556, 558, 560, 562, 564, 566, 568, 570, 572, 574, 576,
        578, 580, 582, 584, 586, 588, 590, 592, 594, 596, 598, 600, 602,
        604, 606, 608, 610, 612, 614, 616, 618, 620, 622, 624, 626, 628,
        630, 632, 634, 636, 638, 640, 642, 644, 646, 648, 650, 652, 654,
        656, 658, 660, 662, 664, 666, 668, 670, 672, 674, 676, 678, 680,
        682, 684, 686, 688, 690, 692, 694, 696, 698, 700, 702, 704, 706,
        708, 710, 712, 714, 716, 718, 720, 722, 724, 726, 728, 730, 732,
        734, 736, 738, 740, 742, 744, 746, 748]),
 array([750, 752, 754, 756, 758, 760, 762, 764, 766, 768, 770, 772, 774,
        776, 778, 780, 782, 784, 786, 788, 790, 792, 794, 796, 798, 800,
        802, 804, 806, 808, 810, 812, 814, 816, 818, 820, 822, 824, 826,
        828, 830, 832, 834, 836, 838, 840, 842, 844, 846, 848, 850, 852,
        854, 856, 858, 860, 862, 864, 866, 868, 870, 872, 874, 876, 878,
        880, 882, 884, 886, 888, 890, 892, 894, 896, 898, 900, 902, 904,
        906, 908, 910, 912, 914, 916, 918, 920, 922, 924, 926, 928, 930,
        932, 934, 936, 938, 940, 942, 944, 946, 948, 950, 952, 954, 956,
        958, 960, 962, 964, 966, 968, 970, 972, 974, 976, 978, 980, 982,
        984, 986, 988, 990, 992, 994, 996, 998])]
```

```
[32]: len(x[0])
```

```
[32]: 125
```

```
[33]: arr=np.array([1,2,3,4,5])
      np.where(arr%2==0)
```

```
[33]: (array([1, 3], dtype=int64),)
```

```
[34]: arr%2==0
```

```
[34]: array([False,  True, False,  True, False])
```

```
[36]: np.where(a>=225)
```

```
[36]: (array([113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
        126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138,
        139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151,
        152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
        165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177,
        178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
        191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203,
        204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216,
        217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229,
        230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242,
        243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255,
        256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268,
        269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281,
        282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294,
        295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307,
        308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320,
        321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333,
        334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346,
        347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359,
        360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372,
        373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385,
        386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398,
        399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411,
        412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424,
        425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437,
        438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450,
        451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463,
        464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476,
        477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489,
        490, 491, 492, 493, 494, 495, 496, 497, 498, 499], dtype=int64),)
```

```
[ ]:
```