Numpy Experiment ¶

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

```
In [2]:  a = [1,2,3,4,5,6,7,8]
         print(a)
         print(type(a))
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
<class 'list'>
```

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

```
[1 2 3 4 5 6 7 8]
<class 'numpy.ndarray'>
```

ndim

```
In [4]:  a.ndim
```

```
Out[4]: 1
```

```
In [5]:  arr_2D = np.array([[1,2,3],[4,5,6],[7,8,9]])
         print(arr_2D)
         print(type(arr_2D))
         print(arr_2D.ndim)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
<class 'numpy.ndarray'>
2
```

```
In [13]:  arr_3D = np.array([[[1,2,3],[4,5,6],[7,8,9]],[[1,2,3],[4,5,6],[7,8,9]]])
          print(arr_3D)
          print(type(arr_3D))
          print(arr_3D.ndim)
```

```
[[[1 2 3]
  [4 5 6]
  [7 8 9]]

 [[1 2 3]
  [4 5 6]
  [7 8 9]]]
<class 'numpy.ndarray'>
3
```

```
In [26]:  arr_5D = np.array([[1,1,1,1,1],[2,2,2,2,2],[3,3,3,3,3],[4,4,4,4,4],[5,5,5,
          print(arr_5D)
          print(type(arr_5D))
          print(arr_5D.ndim)  # 5 X 5 matrix
```

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

```
In [27]:  arr = np.array([[[0,0],[1,1]],[[2,2],[3,3]],[[4,4],[5,5]]])
          print(arr)
          print(type(arr))
          print(arr.ndim)    #  '3 X 2 X 2'  matrix
```

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

 [[2 2]
  [3 3]]

 [[4 4]
  [5 5]]]
<class 'numpy.ndarray'>
3
```

size(show element no)

```
In [30]:  arr_2D.size
```

```
Out[30]: 9
```

```
In [31]:  arr.size
```

```
Out[31]: 12
```

shape show no of rows

```
In [35]:  arr_2D.shape
```

```
Out[35]: (3, 3)
```

```
In [36]:  arr_5D.shape
```

```
Out[36]: (5, 5)
```

```
In [34]: arr.shape
```

```
Out[34]: (3, 2, 2)
```

dtype :- show data type of array

```
In [37]: arr.dtype
```

```
Out[37]: dtype('int32')
```

```
In [38]: a1 = np.array([1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8])
         print(a1)
         print(a1.dtype)
```

```
[1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8]
float64
```

```
In [43]: a1 = np.array([[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1]])
         print(a1)
         print(a1.dtype)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
int32
```

```
In [44]: a2 = np.array([[0,0,0],[0,0,0],[0,0,0]])
         print(a2)
         print(a2.dtype)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
int32
```

ones() and zeros() with dtype int str

```
In [45]: b1 = np.ones((4,4))
         print(b1)
         print(b1.dtype)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
float64
```

```
In [48]: b1 = np.zeros((3,3),dtype=int)
         print(b1)
         print(b1.dtype)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
int32
```

```
In [51]: b1 = np.ones((4,4),dtype=str)
         print(b1)
         print(b1.dtype)
```

```
[['1' '1' '1' '1']
 ['1' '1' '1' '1']
 ['1' '1' '1' '1']
 ['1' '1' '1' '1']]
<U1
```

```
In [49]: b1 = np.zeros((3,3),dtype=str)
         print(b1)
         print(b1.dtype)
```

```
[['' '' '']
 ['' '' '']
 ['' '' '']]
<U1
```

```
In [53]: b1 = np.ones((4,4),dtype=bool)
         print(b1)
         print(b1.dtype)
```

```
[[ True  True  True  True]
 [ True  True  True  True]
 [ True  True  True  True]
 [ True  True  True  True]]
bool
```

```
In [52]: b1 = np.zeros((3,3),dtype=bool)
         print(b1)
         print(b1.dtype)
```

```
[[False False False]
 [False False False]
 [False False False]]
bool
```

empty

```
In [55]: em_mx = np.empty((4,4))  # it give garbage value
         em_mx

Out[55]: array([[1., 1., 1., 1.],
                [1., 1., 1., 1.],
                [1., 1., 1., 1.],
                [1., 1., 1., 1.]])
```

arange()

```
In [ ]: # np.arange(start_value,end_value,step)

In [56]: ar_1d = np.arange(13)
         ar_1d

Out[56]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])

In [57]: ar_1d = np.arange(2,13)
         ar_1d

Out[57]: array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

linespace()

```
In [58]: # np.arange(start_value,end_value,no of value b\t start to end)

In [61]: np.linspace(1,10,5)

Out[61]: array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])

In [62]: np.linspace(100,150,3)

Out[62]: array([100., 125., 150.])
```

reshape()

```
In [65]: a = np.linspace(1,10,25)
         a

Out[65]: array([ 1.   ,  1.375,  1.75 ,  2.125,  2.5  ,  2.875,  3.25 ,  3.625,
                 4.   ,  4.375,  4.75 ,  5.125,  5.5  ,  5.875,  6.25 ,  6.625,
                 7.   ,  7.375,  7.75 ,  8.125,  8.5  ,  8.875,  9.25 ,  9.625,
                10.   ])
```

```
In [67]: arr1 = a.reshape(5,5)
         arr1

Out[67]: array([[ 1.   ,  1.375,  1.75 ,  2.125,  2.5  ],
                [ 2.875,  3.25 ,  3.625,  4.   ,  4.375],
                [ 4.75 ,  5.125,  5.5  ,  5.875,  6.25 ],
                [ 6.625,  7.   ,  7.375,  7.75 ,  8.125],
                [ 8.5  ,  8.875,  9.25 ,  9.625, 10.   ]])

In [76]: arr2 = np.arange(1,13).reshape(3,2,2)
         arr2

Out[76]: array([[[ 1,  2],
                 [ 3,  4]],

                [[ 5,  6],
                 [ 7,  8]],

                [[ 9, 10],
                 [11, 12]]])
```

ravel()

```
In [77]: arr2 = np.arange(1,13).reshape(3,2,2)
         arr2

Out[77]: array([[[ 1,  2],
                 [ 3,  4]],

                [[ 5,  6],
                 [ 7,  8]],

                [[ 9, 10],
                 [11, 12]]])

In [78]: arr2.ravel()

Out[78]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

transpose() or T

```
In [79]: arr = np.linspace(1,10,25).reshape(5,5)
         arr

Out[79]: array([[ 1.   ,  1.375,  1.75 ,  2.125,  2.5  ],
                [ 2.875,  3.25 ,  3.625,  4.   ,  4.375],
                [ 4.75 ,  5.125,  5.5  ,  5.875,  6.25 ],
                [ 6.625,  7.   ,  7.375,  7.75 ,  8.125],
                [ 8.5  ,  8.875,  9.25 ,  9.625, 10.   ]])
```

```
In [80]: arr.transpose()  # it convert row to colum

Out[80]: array([[ 1.   ,  2.875,  4.75 ,  6.625,  8.5  ],
                [ 1.375,  3.25 ,  5.125,  7.   ,  8.875],
                [ 1.75 ,  3.625,  5.5  ,  7.375,  9.25 ],
                [ 2.125,  4.   ,  5.875,  7.75 ,  9.625],
                [ 2.5  ,  4.375,  6.25 ,  8.125, 10.   ]])
```

```
In [81]: arr.T   # it convert row to colum

Out[81]: array([[ 1.   ,  2.875,  4.75 ,  6.625,  8.5  ],
                [ 1.375,  3.25 ,  5.125,  7.   ,  8.875],
                [ 1.75 ,  3.625,  5.5  ,  7.375,  9.25 ],
                [ 2.125,  4.   ,  5.875,  7.75 ,  9.625],
                [ 2.5  ,  4.375,  6.25 ,  8.125, 10.   ]])
```

mathematical operation using numpy

```
In [82]: arr2 = np.arange(1,10).reshape(3,3)
         arr3 = np.arange(1,10).reshape(3,3)
         print(arr2)
         print(arr3)

[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
In [83]: print(arr2 + arr3)

[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

```
In [86]: print(arr2 - arr3)

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

```
In [84]: print(arr2 * arr3)

[[ 1  4  9]
 [16 25 36]
 [49 64 81]]
```

```
In [85]: print(arr2 ** arr3)

[[        1         4        27]
 [      256      3125     46656]
 [   823543  16777216 387420489]]
```

```
In [87]: print(arr2 / arr3)

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

```
In [88]: print(arr2 @ arr3) # matrix multification

[[ 30  36  42]
 [ 66  81  96]
 [102 126 150]]
```

```
In [90]: arr2.dot(arr3)

Out[90]: array([[ 30,  36,  42],
                [ 66,  81,  96],
                [102, 126, 150]])
```

```
In [92]: np.add(arr2,arr3)

Out[92]: array([[ 2,  4,  6],
                [ 8, 10, 12],
                [14, 16, 18]])
```

```
In [94]: np.subtract(arr2,arr3)

Out[94]: array([[0, 0, 0],
                [0, 0, 0],
                [0, 0, 0]])
```

```
In [96]: np.multiply(arr2,arr3)

Out[96]: array([[ 1,  4,  9],
                [16, 25, 36],
                [49, 64, 81]])
```

```
In [97]: np.divide(arr2,arr3)

Out[97]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])
```

Maximum and Minimum

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

```
In [2]: arr1 = np.arange(1,10).reshape(3,3)
        arr1

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

```
In [3]:  arr1.max()  # it return maximum value

Out[3]:  9

In [4]:  arr1.argmax()  # it return maximum value index

Out[4]:  8

In [5]:  arr1.max(axis=0)  # it return maximum row

         # NOTE :- row denote as a 1 and col denote as 0

Out[5]:  array([7, 8, 9])

In [6]:  arr1.max(axis=1)  # it return maximum col

         # NOTE :- row denote as a 1 and col denote as 0

Out[6]:  array([3, 6, 9])

In [7]:  arr1.argmax(axis=0)

Out[7]:  array([2, 2, 2], dtype=int64)

In [9]:  arr1.argmax(axis=1)

Out[9]:  array([2, 2, 2], dtype=int64)

In [10]: np.sum(arr1)

Out[10]: 45

In [12]: np.sum(arr1, axis=0)

Out[12]: array([12, 15, 18])

In [13]: np.sum(arr1, axis=1)

Out[13]: array([ 6, 15, 24])
```

mean median std

```
In [14]: np.mean(arr1)

Out[14]: 5.0

In [17]: np.median(arr1)

Out[17]: 5.0
```

```
In [18]: np.sqrt(arr1)

Out[18]: array([[1.        , 1.41421356, 1.73205081],
                [2.        , 2.23606798, 2.44948974],
                [2.64575131, 2.82842712, 3.        ]])

In [19]: np.std(arr1)

Out[19]: 2.581988897471611

In [20]: np.exp(arr1)

Out[20]: 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]])

In [21]: np.log(arr1)

Out[21]: array([[0.        , 0.69314718, 1.09861229],
                [1.38629436, 1.60943791, 1.79175947],
                [1.94591015, 2.07944154, 2.19722458]])

In [22]: np.log(arr1)

Out[22]: array([[0.        , 0.69314718, 1.09861229],
                [1.38629436, 1.60943791, 1.79175947],
                [1.94591015, 2.07944154, 2.19722458]])

In [23]: np.log10(arr1)

Out[23]: array([[0.        , 0.30103   , 0.47712125],
                [0.60205999, 0.69897   , 0.77815125],
                [0.84509804, 0.90308999, 0.95424251]])

In [ ]:  np.max()

         # NOTE :- shift + tap press button to show suggestion
```

numpyarray slicing(:)

```
In [27]: mx = np.arange(1,101).reshape(10,10)
         mx

Out[27]: array([[  1,   2,   3,   4,   5,   6,   7,   8,   9,  10],
                [ 11,  12,  13,  14,  15,  16,  17,  18,  19,  20],
                [ 21,  22,  23,  24,  25,  26,  27,  28,  29,  30],
                [ 31,  32,  33,  34,  35,  36,  37,  38,  39,  40],
                [ 41,  42,  43,  44,  45,  46,  47,  48,  49,  50],
                [ 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]])

In [28]: mx[0,0]

Out[28]: 1

In [29]: mx[4,5]

Out[29]: 46

In [32]: mx[:,2:5]

Out[32]: array([[ 3,  4,  5],
                [13, 14, 15],
                [23, 24, 25],
                [33, 34, 35],
                [43, 44, 45],
                [53, 54, 55],
                [63, 64, 65],
                [73, 74, 75],
                [83, 84, 85],
                [93, 94, 95]])

In [34]: mx[:,6:8]

Out[34]: array([[ 7,  8],
                [17, 18],
                [27, 28],
                [37, 38],
                [47, 48],
                [57, 58],
                [67, 68],
                [77, 78],
                [87, 88],
                [97, 98]])

In [35]: mx[1:3,:]

Out[35]: array([[11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
                [21, 22, 23, 24, 25, 26, 27, 28, 29, 30]])
```

```
In [36]: mx[7:9,:]

Out[36]: array([[71, 72, 73, 74, 75, 76, 77, 78, 79, 80],
                [81, 82, 83, 84, 85, 86, 87, 88, 89, 90]])

In [37]: mx[2:4,2:4]

Out[37]: array([[23, 24],
                [33, 34]])

In [43]: mx[5:8,5:8]

Out[43]: array([[56, 57, 58],
                [66, 67, 68],
                [76, 77, 78]])

In [56]: mat = np.arange(1,41).reshape(5,8)
         mat

Out[56]: array([[ 1,  2,  3,  4,  5,  6,  7,  8],
                [ 9, 10, 11, 12, 13, 14, 15, 16],
                [17, 18, 19, 20, 21, 22, 23, 24],
                [25, 26, 27, 28, 29, 30, 31, 32],
                [33, 34, 35, 36, 37, 38, 39, 40]])

In [58]: add = mat[:,0:4] + mat[:,4:]
         add

Out[58]: array([[ 6,  8, 10, 12],
                [22, 24, 26, 28],
                [38, 40, 42, 44],
                [54, 56, 58, 60],
                [70, 72, 74, 76]])
```

random

```
In [64]: import random

In [66]: np.random.random()  # by default genrate float value

Out[66]: 0.3666755530593553

In [67]: np.random.random(5)  # by default genrate float value

Out[67]: array([0.54956409, 0.78147967, 0.76356517, 0.70406575, 0.58331084])

In [69]: np.random.random(5)  # by default genrate float value

Out[69]: array([0.63157526, 0.20080261, 0.15914704, 0.75795035, 0.99109785])
```

```
In [70]: np.random.random((3,3))
```

```
Out[70]: array([[0.06832737, 0.8216853 , 0.77001425],
                [0.47662683, 0.34516421, 0.12139191],
                [0.51613595, 0.7938053 , 0.07044807]])
```

randint

```
In [73]: np.random.randint(5)   # it genrate only one value and value comes b/t 0 t
```

```
Out[73]: 4
```

```
In [75]: np.random.randint(20,25)   # it genrate only one value and value comes b/t
```

```
Out[75]: 23
```

```
In [77]: np.random.randint(20,25, 3)   # 3rd argument denotes that how many value n
```

```
Out[77]: array([21, 24, 20])
```

```
In [78]: np.random.randint(20,25, (4,4))  # it create matix
```

```
Out[78]: array([[22, 23, 20, 21],
                [22, 22, 23, 22],
                [22, 20, 21, 21],
                [21, 20, 24, 23]])
```

```
In [79]: np.random.randint(20,25, (2,4,4))  # it create matix and 2 denote as how m
```

```
Out[79]: array([[[23, 22, 24, 22],
                 [23, 23, 21, 23],
                 [22, 24, 24, 21],
                 [22, 20, 24, 24]],

                [[22, 21, 22, 23],
                 [21, 21, 21, 22],
                 [21, 22, 24, 23],
                 [23, 22, 21, 23]]])
```

choice

```
In [82]: p = [1,2,3,4,5,6,7,8,9]
         p
```

```
Out[82]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [86]: np.random.choice(p)  # it genrate one value from list p
```

```
Out[86]: 3
```

```
In [87]: np.random.choice(p)  # it genrate one value from list p
```

```
Out[87]: 4
```

```
In [89]: np.random.choice(p,4)  # it genrate 4 value from list p
```

```
Out[89]: array([5, 9, 8, 6])
```

permutation

```
In [90]: k = [1,2,3]
         k
```

```
Out[90]: [1, 2, 3]
```

```
In [91]: np.random.permutation(k)
```

```
Out[91]: array([2, 3, 1])
```

concatenate

```
In [99]: mat1 = np.arange(1,17).reshape(4,4)
         print(mat1)

         print()

         mat2 = np.arange(16,32).reshape(4,4)
         print(mat2)
```

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

[[16 17 18 19]
 [20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]]
```

```
In [100]: np.concatenate((mat1,mat2))
```

```
Out[100]: array([[ 1,  2,  3,  4],
                 [ 5,  6,  7,  8],
                 [ 9, 10, 11, 12],
                 [13, 14, 15, 16],
                 [16, 17, 18, 19],
                 [20, 21, 22, 23],
                 [24, 25, 26, 27],
                 [28, 29, 30, 31]])
```

```
In [101]: np.concatenate((mat1,mat2), axis=1)

Out[101]: array([[ 1,  2,  3,  4, 16, 17, 18, 19],
                 [ 5,  6,  7,  8, 20, 21, 22, 23],
                 [ 9, 10, 11, 12, 24, 25, 26, 27],
                 [13, 14, 15, 16, 28, 29, 30, 31]])
```

split

```
In [102]: mat1

Out[102]: array([[ 1,  2,  3,  4],
                 [ 5,  6,  7,  8],
                 [ 9, 10, 11, 12],
                 [13, 14, 15, 16]])

In [106]: np.split(mat1,2)

Out[106]: [array([[1, 2, 3, 4],
                  [5, 6, 7, 8]]),
           array([[ 9, 10, 11, 12],
                  [13, 14, 15, 16]])]

In [107]: x,y = np.split(mat1,2)

In [108]: print(x)
          print()
          print(y)

          [[1 2 3 4]
           [5 6 7 8]]

          [[ 9 10 11 12]
           [13 14 15 16]]
```

Genrate Data for house price using numpy

```
In [111]: # area , valkini, bhk, houseprice

          area = np.random.randint(700,22000,100)
          valkini = np.random.randint(1,6,100)
          bhk = np.random.randint(1,6,100)
          houseprice = np.random.randint(450000,1500000,100)

In [120]: housedata = np.concatenate((area,valkini,bhk,houseprice))
```

```
In [122]: housedata
```

```
Out[122]: array([  16144,   20123,    9923,    2128,    7061,   13603,   12575,
                   15325,    3432,   15080,   13614,   21965,    1061,   11897,
                   14683,    5666,     956,    9305,   20490,    2758,   19849,
                    1267,   10234,    7536,    8305,   20927,   13450,   10314,
                    7080,    3524,    3077,   11226,   20783,   18336,   12091,
                   20621,   16511,    3185,   11335,    3999,   21436,   17228,
                   18768,    7646,    1848,     952,   18564,    5531,   10655,
                    5503,    6009,    8692,    5415,    9192,   12156,   19729,
                    7768,   13066,    3704,    9781,   11417,   11494,   21179,
                   11726,   11194,    3796,   15104,    3026,    2424,    5819,
                   19601,    7315,    4570,   18535,   21744,     758,    1002,
                   20029,    8951,   12558,   11741,   19433,    3368,   21567,
                   18705,   13902,    1555,    5077,   12294,    6556,   20717,
                   16803,   15162,    4779,   18488,    2163,   21017,    4110,
                   15409,   13293,       5,       4,       2,       2,       2,
                       1,       1,       1,       1,       5,       1,       2,
                       4,       1,       2,       5,       1,       5,       1,
                       2,       4,       1,       3,       4,       3,       2,
                       1,       5,       5,       4,       3,       4,       4,
                       1,       3,       2,       2,       1,       1,       3,
                       3,       3,       2,       4,       4,       5,       4,
                       3,       3,       3,       1,       2,       5,       5,
                       1,       2,       5,       5,       5,       1,       1,
                       1,       3,       1,       1,       3,       2,       4,
                       2,       4,       2,       5,       3,       5,       1,
                       5,       1,       2,       2,       3,       3,       4,
                       1,       3,       3,       2,       1,       1,       4,
                       5,       3,       1,       4,       2,       3,       5,
                       1,       4,       1,       2,       4,       4,       1,
                       5,       3,       2,       4,       1,       2,       3,
                       3,       2,       5,       2,       5,       2,       2,
                       4,       2,       3,       1,       5,       3,       2,
                       5,       2,       3,       5,       2,       3,       5,
                       5,       5,       4,       3,       3,       4,       5,
                       1,       5,       3,       5,       4,       2,       1,
                       4,       5,       3,       4,       1,       5,       2,
                       1,       4,       2,       2,       3,       1,       4,
                       1,       3,       3,       2,       5,       4,       4,
                       3,       5,       3,       2,       5,       1,       1,
                       5,       2,       5,       1,       2,       4,       2,
                       3,       2,       2,       2,       3,       3,       2,
                       1,       5,       3,       5,       5,       3,       3,
                       3,       2,       2,       4,       4,       5,  489994,
                  658199, 1434425, 1305067,  778568, 1194223, 1281034, 1422287,
                 1308608, 1239615, 1433582, 1225716,  691713, 1496791,  920295,
                  591526,  810716,  601277,  717810, 1275430, 1122832,  805515,
                  485549, 1116145, 1338333, 1498938,  812290,  610400,  894126,
                  589729,  575464, 1227393, 1082075, 1233808,  783724, 1185270,
                 1476286,  512455,  907216, 1102940,  914530,  580238,  999449,
                 1021847, 1287071,  971358,  565094, 1216886, 1190383,  960219,
                 1100261,  805417, 1147063, 1358002, 1254804, 1406764, 1424880,
                  952718,  845699,  978472,  787256, 1168306,  803040,  505798,
                  701161,  893571, 1039644, 1200139, 1183244,  934472,  990947,
                  969101, 1210602,  915457,  760803, 1304502, 1105464, 1124958,
                 1125965,  696589,  486153, 1472589, 1409445, 1296762,  709068,
                 1101713, 1457585,  674601, 1002915,  794049,  855168, 1055984,
```

```
                  985688,  786497, 1338663,  774115,  465723, 1117498, 1196818,
                 1342711])
```

```
In [ ]:  1
```

```
In [ ]:  1
```