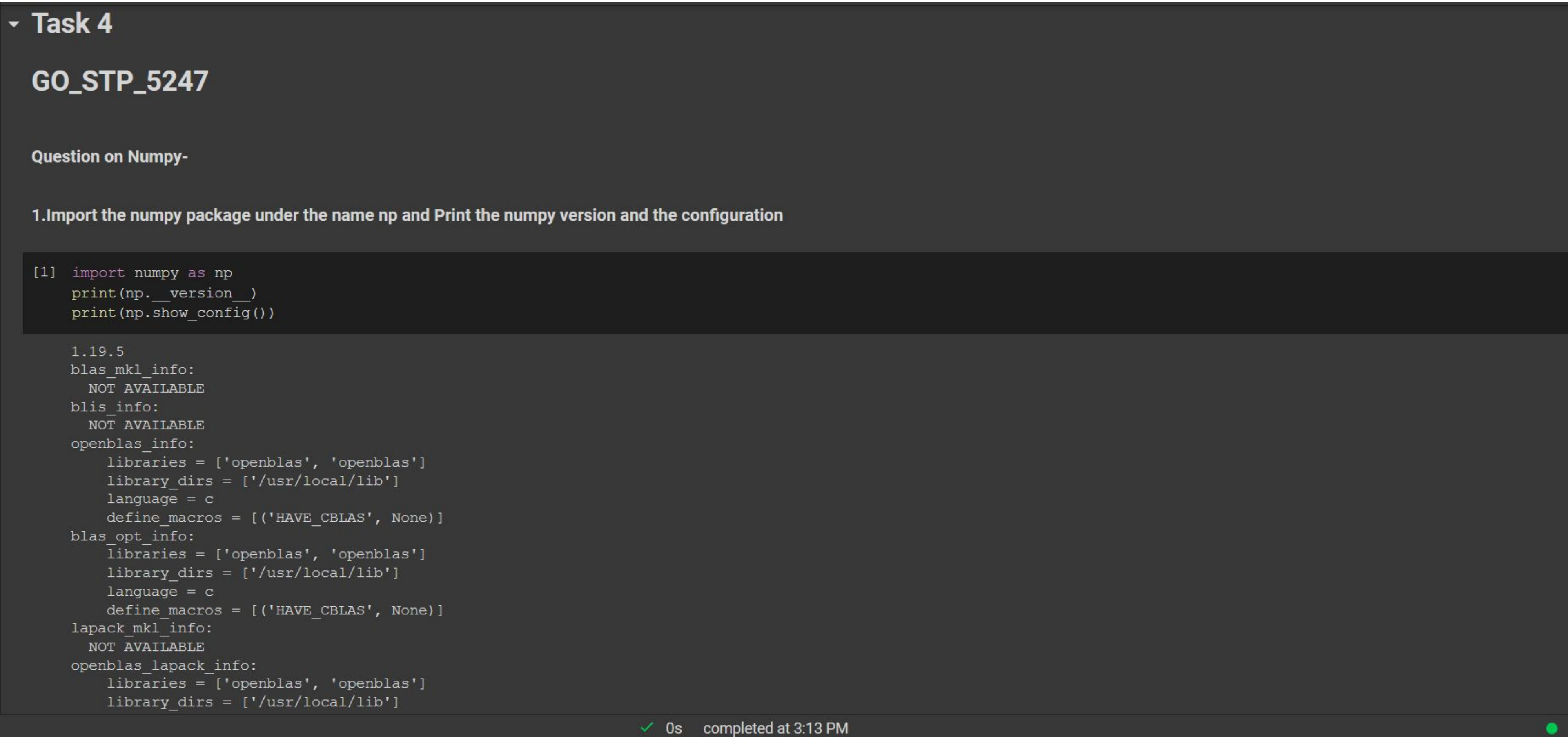


Task 4

GO_STP_5247

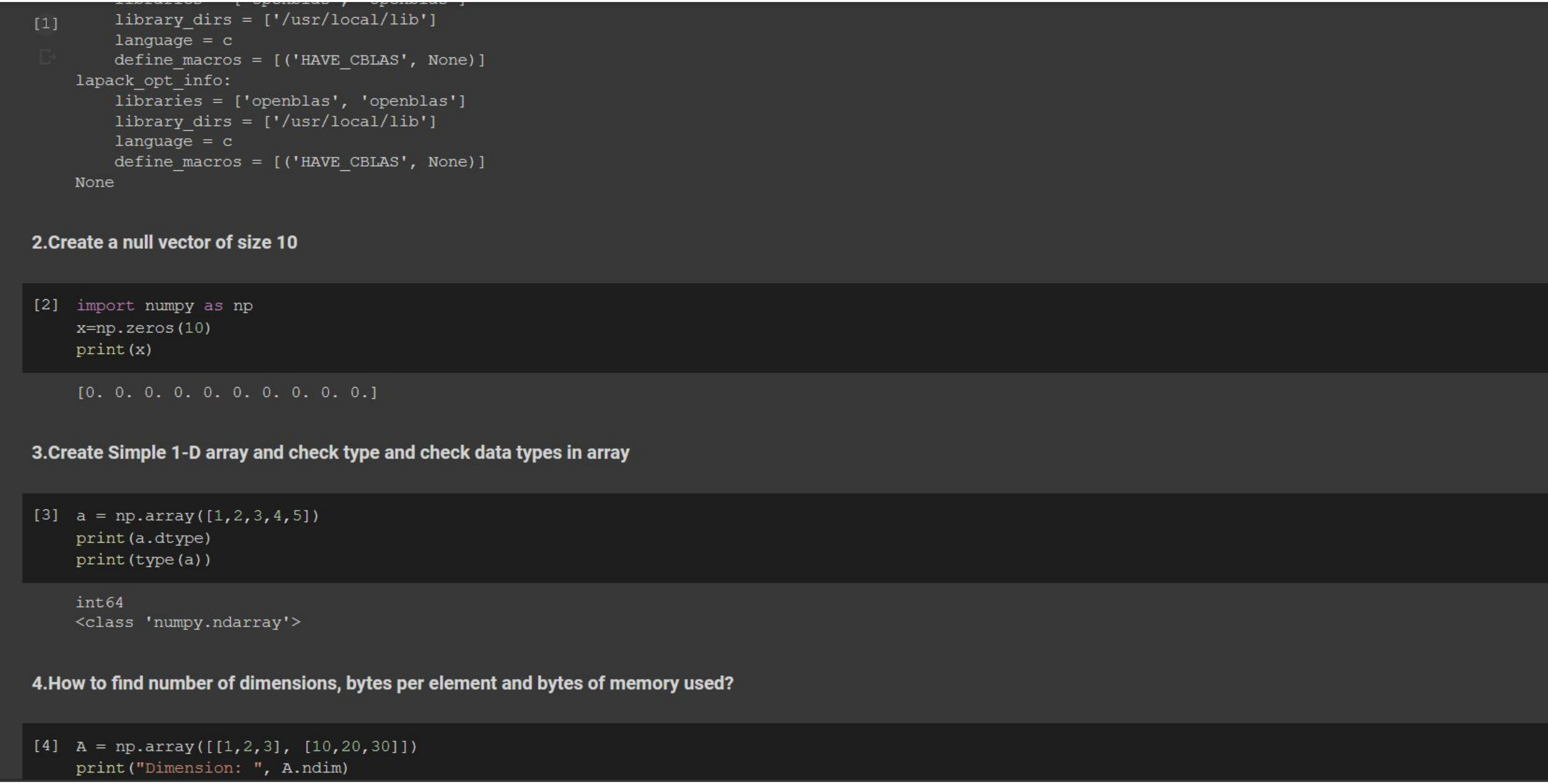
Question on Numpy-

1.Import the numpy package under the name np and Print the numpy version and the configuration

```
[1] import numpy as np
    print(np.__version__)
    print(np.show_config())
```

```
1.19.5
blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
blas_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
```

✓ 0s completed at 3:13 PM



```
[1]     library_dirs = ['/usr/local/lib']
        language = c
        define_macros = [('HAVE_CBLAS', None)]
    lapack_opt_info:
        libraries = ['openblas', 'openblas']
        library_dirs = ['/usr/local/lib']
        language = c
        define_macros = [('HAVE_CBLAS', None)]
    None
```

2.Create a null vector of size 10

```
[2] import numpy as np
    x=np.zeros(10)
    print(x)
```

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

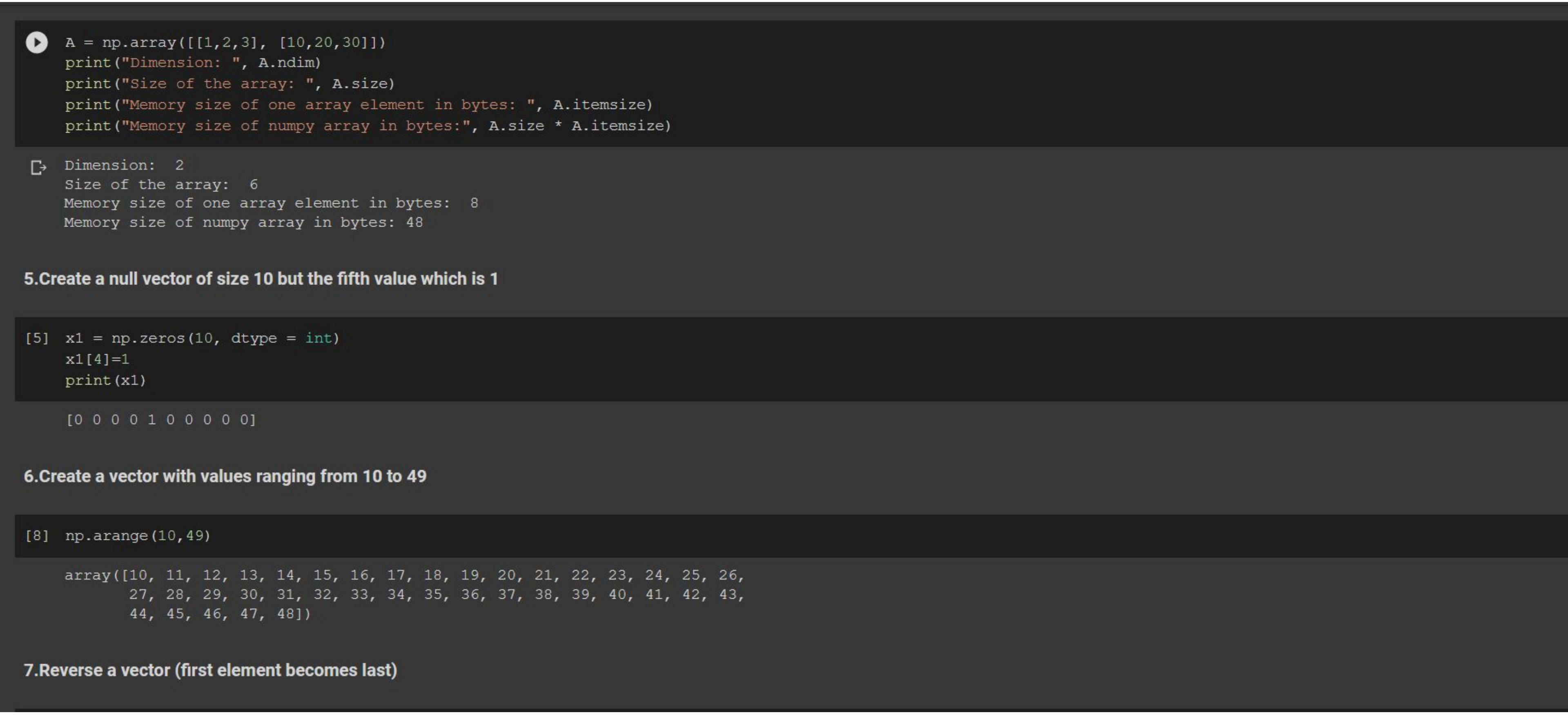
3.Create Simple 1-D array and check type and check data types in array

```
[3] a = np.array([1,2,3,4,5])
    print(a.dtype)
    print(type(a))
```

```
    int64
    <class 'numpy.ndarray'>
```

4.How to find number of dimensions, bytes per element and bytes of memory used?

```
[4] A = np.array([[1,2,3], [10,20,30]])
    print("Dimension: ", A.ndim)
```



```
A = np.array([[1,2,3], [10,20,30]])
print("Dimension: ", A.ndim)
print("Size of the array: ", A.size)
print("Memory size of one array element in bytes: ", A.itemsize)
print("Memory size of numpy array in bytes:", A.size * A.itemsize)
```

```
Dimension:  2
Size of the array:  6
Memory size of one array element in bytes:  8
Memory size of numpy array in bytes: 48
```

5.Create a null vector of size 10 but the fifth value which is 1

```
[5] x1 = np.zeros(10, dtype = int)
    x1[4]=1
    print(x1)
```

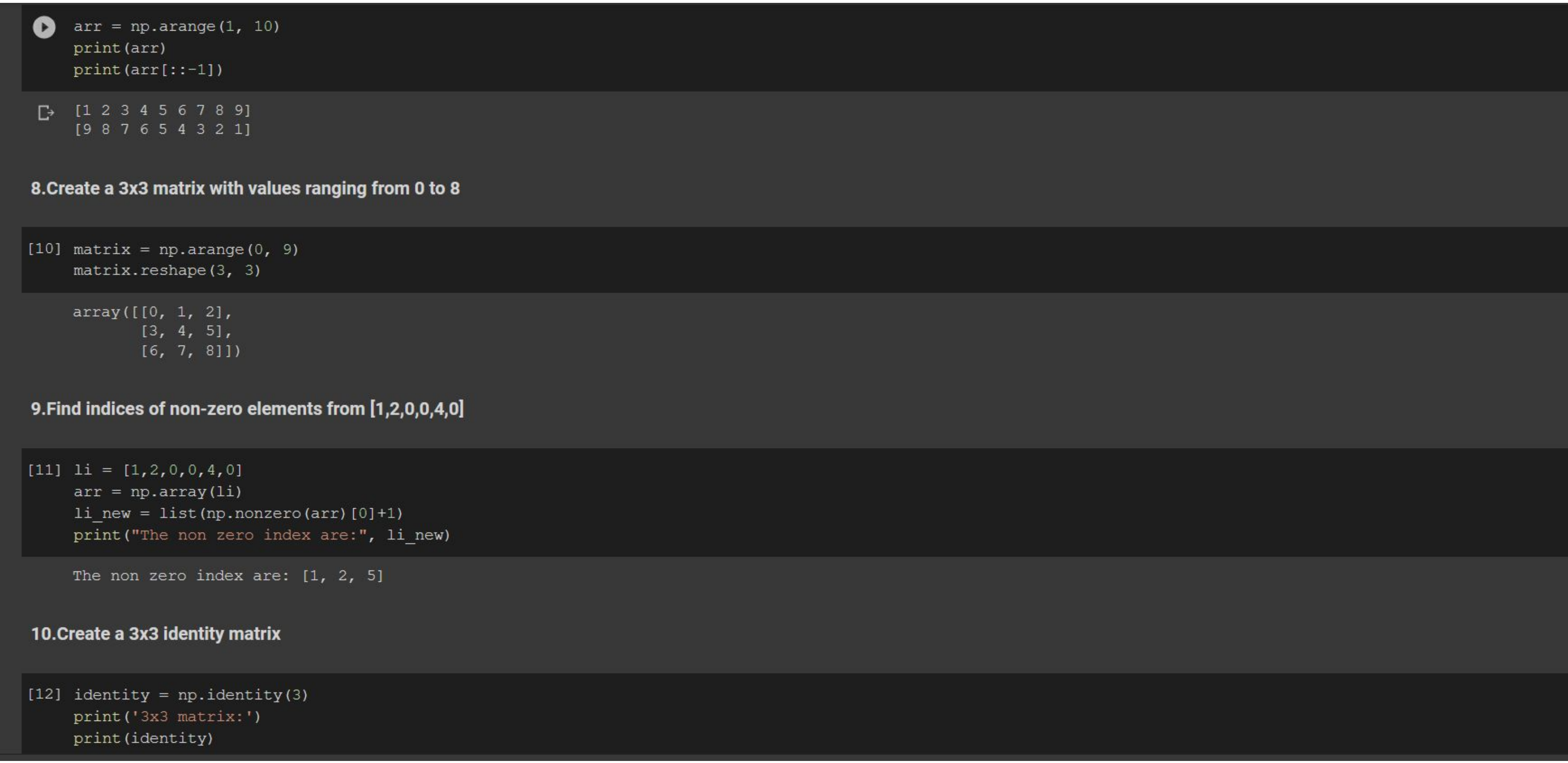
```
[0 0 0 0 1 0 0 0 0 0]
```

6.Create a vector with values ranging from 10 to 49

```
[8] np.arange(10,49)
```

```
array([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])
```

7.Reverse a vector (first element becomes last)



```
arr = np.arange(1, 10)
print(arr)
print(arr[::-1])
```

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

8.Create a 3x3 matrix with values ranging from 0 to 8

```
[10] matrix = np.arange(0, 9)
     matrix.reshape(3, 3)
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

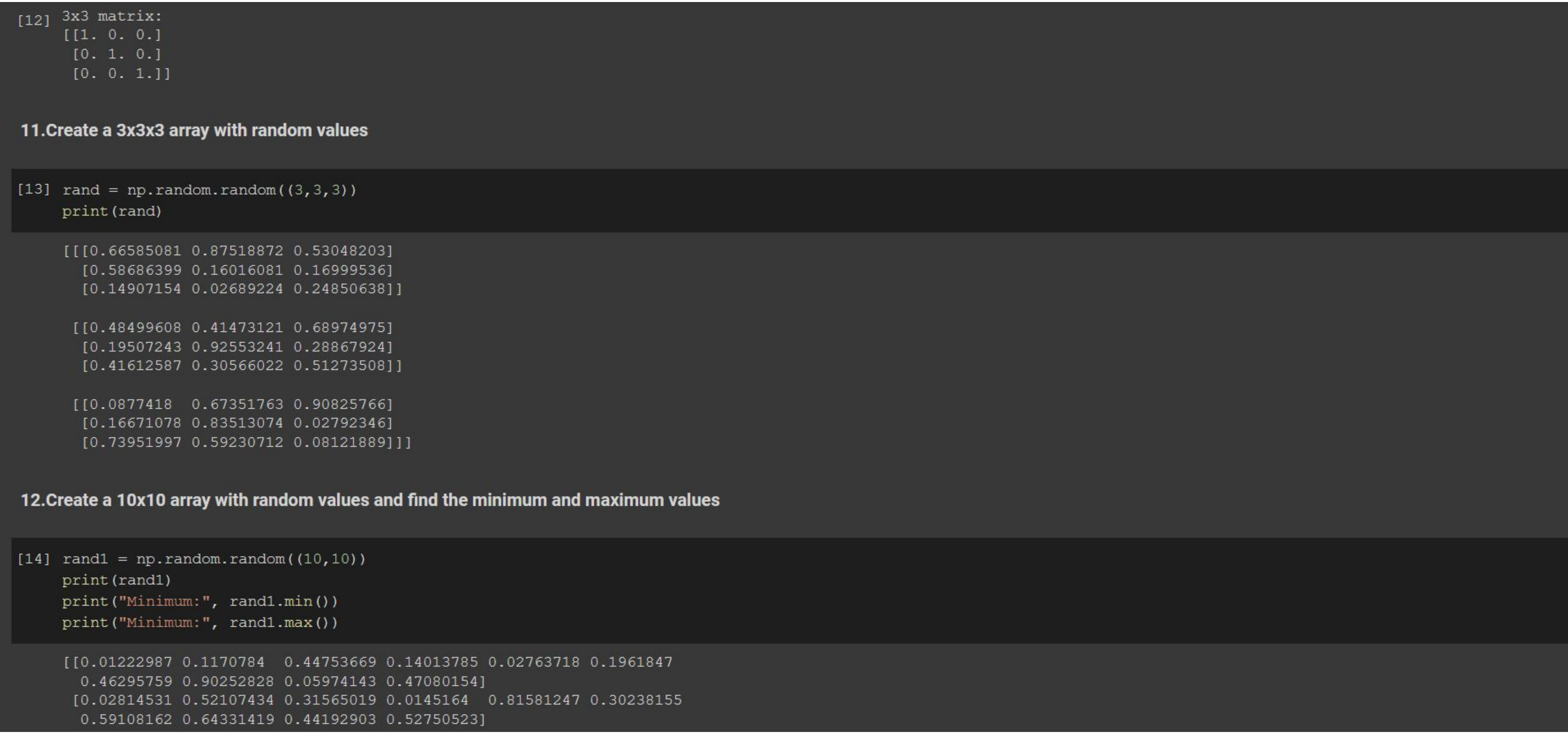
9.Find indices of non-zero elements from [1,2,0,0,4,0]

```
[11] li = [1,2,0,0,4,0]
     arr = np.array(li)
     li_new = list(np.nonzero(arr)[0]+1)
     print("The non zero index are:", li_new)
```

```
The non zero index are: [1, 2, 5]
```

10.Create a 3x3 identity matrix

```
[12] identity = np.identity(3)
     print('3x3 matrix:')
     print(identity)
```



```
[12] 3x3 matrix:
     [[1. 0. 0.]
      [0. 1. 0.]
      [0. 0. 1.]]
```

11.Create a 3x3x3 array with random values

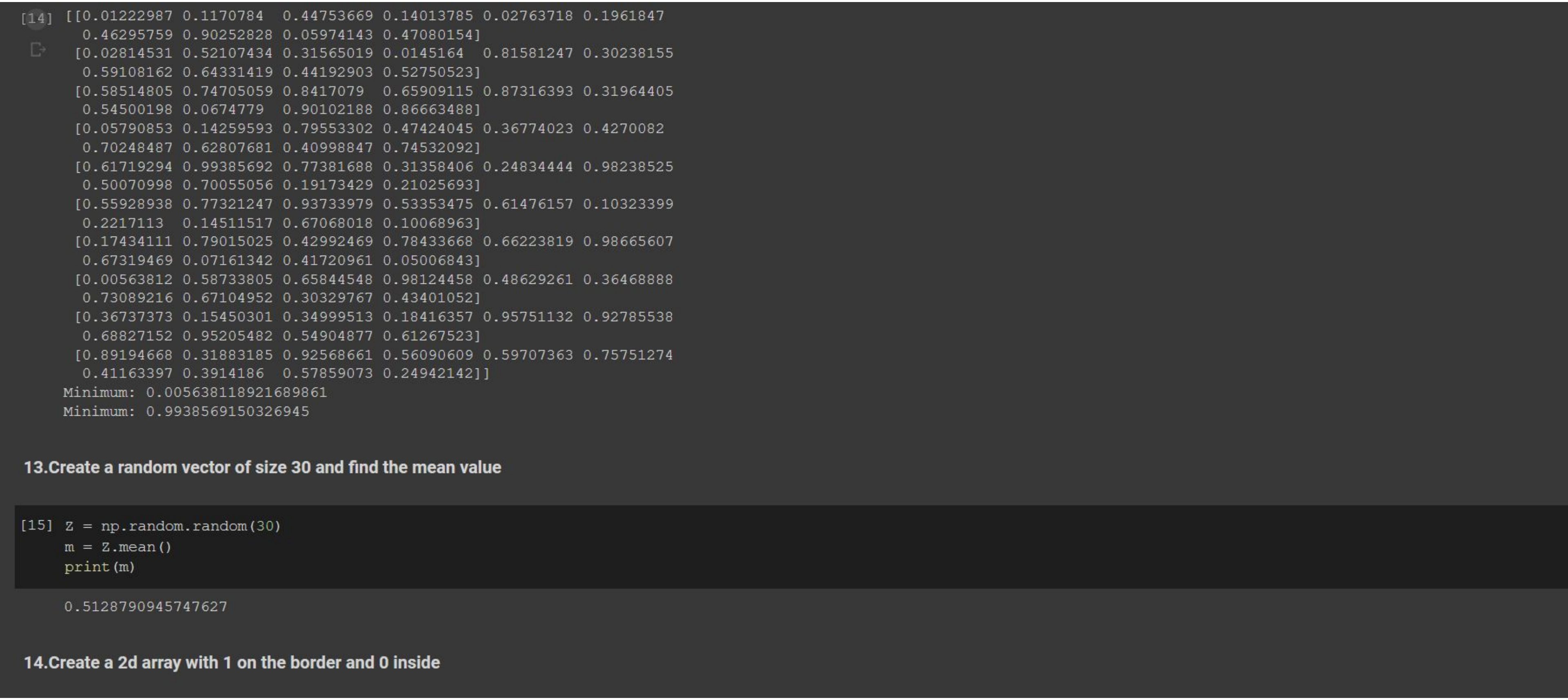
```
[13] rand = np.random.random((3,3,3))
     print(rand)
```

```
     [[[0.66585081 0.87518872 0.53048203]
       [0.58686399 0.16016081 0.16999536]
       [0.14907154 0.02689224 0.24850638]]

      [[0.48499608 0.41473121 0.68974975]
       [0.19507243 0.92553241 0.28867924]
       [0.41612587 0.30566022 0.51273508]]

      [[0.0877418  0.67351763 0.90825766]
       [0.16671078 0.83513074 0.02792346]
       [0.73951997 0.59230712 0.08121889]]]
```

12.Create a 10x10 array with random values and find the minimum and maximum values

```
[14] rand1 = np.random.random((10,10))
     print(rand1)
     print("Minimum:", rand1.min())
     print("Minimum:", rand1.max())
```

```
     [[0.01222987 0.1170784  0.44753669 0.14013785 0.02763718 0.1961847
       0.46295759 0.90252828 0.05974143 0.47080154]
      [0.02814531 0.52107434 0.31565019 0.0145164  0.81581247 0.30238155
       0.59108162 0.64331419 0.44192903 0.52750523]
```



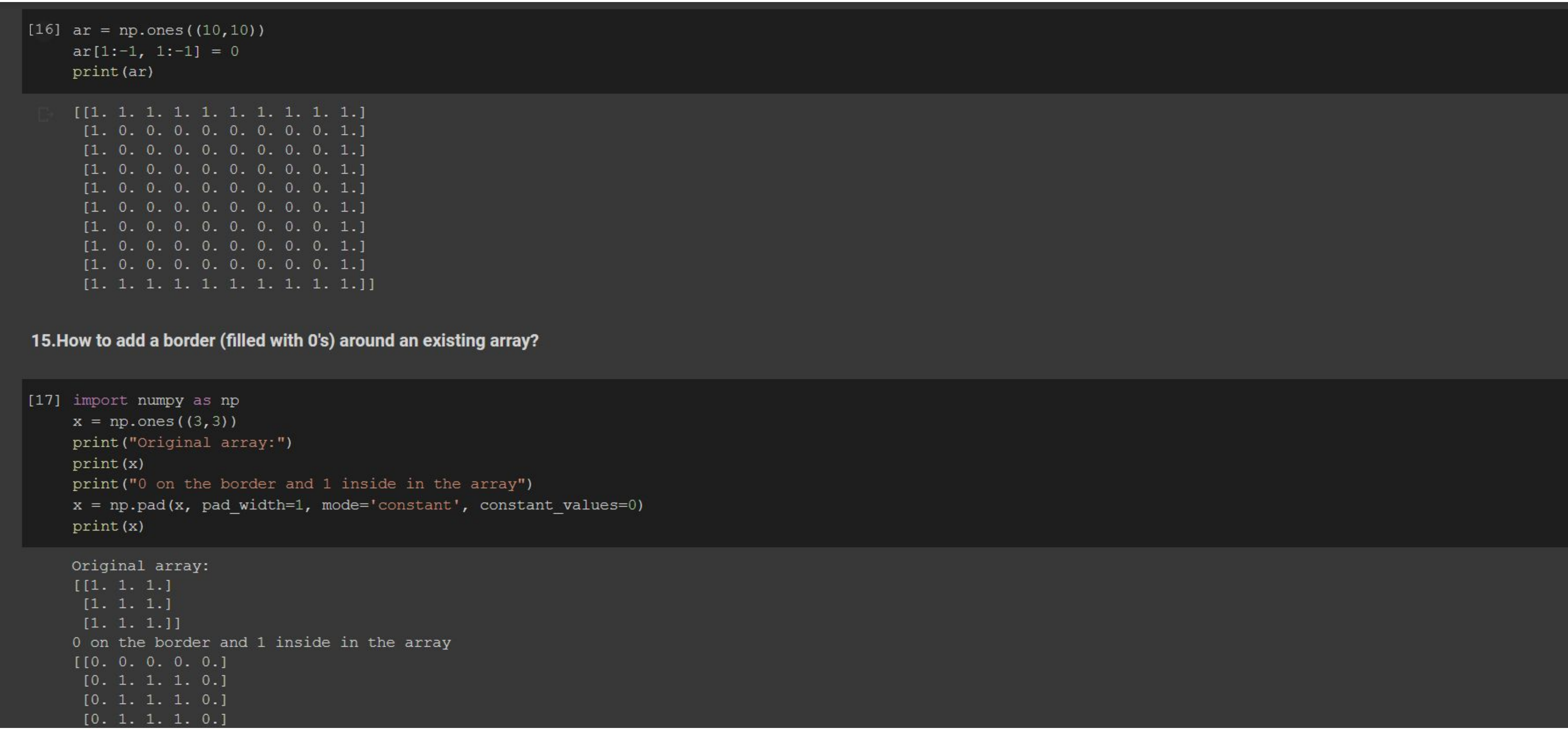
```
[14] [[0.01222987 0.1170784  0.44753669 0.14013785 0.02763718 0.1961847
       0.46295759 0.90252828 0.05974143 0.47080154]
      [0.02814531 0.52107434 0.31565019 0.0145164  0.81581247 0.30238155
       0.59108162 0.64331419 0.44192903 0.52750523]
      [0.58514805 0.74705059 0.8417079  0.65909115 0.87316393 0.31964405
       0.54500198 0.0674779  0.90102188 0.86663488]
      [0.05790853 0.14259593 0.79553302 0.47424045 0.36774023 0.4270082
       0.70248487 0.62807681 0.40998847 0.74532092]
      [0.61719294 0.99385692 0.77381688 0.31358406 0.24834444 0.98238525
       0.50070998 0.70055056 0.19173429 0.21025693]
      [0.55928938 0.77321247 0.93733979 0.53353475 0.61476157 0.10323399
       0.2217113  0.14511517 0.67068018 0.10068963]
      [0.17434111 0.79015025 0.42992469 0.78433668 0.66223819 0.98665607
       0.67319469 0.07161342 0.41720961 0.05006843]
      [0.00563812 0.58733805 0.65844548 0.98124458 0.48629261 0.36468888
       0.73089216 0.67104952 0.30329767 0.43401052]
      [0.36737373 0.15450301 0.34999513 0.18416357 0.95751132 0.92785538
       0.68827152 0.95205482 0.54904877 0.61267523]
      [0.89194668 0.31883185 0.92568661 0.56090609 0.59707363 0.75751274
       0.41163397 0.3914186  0.57859073 0.24942142]]
     Minimum: 0.005638118921689861
     Minimum: 0.9938569150326945
```

13.Create a random vector of size 30 and find the mean value

```
[15] Z = np.random.random(30)
     m = Z.mean()
     print(m)
```

```
     0.5128790945747627
```

14.Create a 2d array with 1 on the border and 0 inside



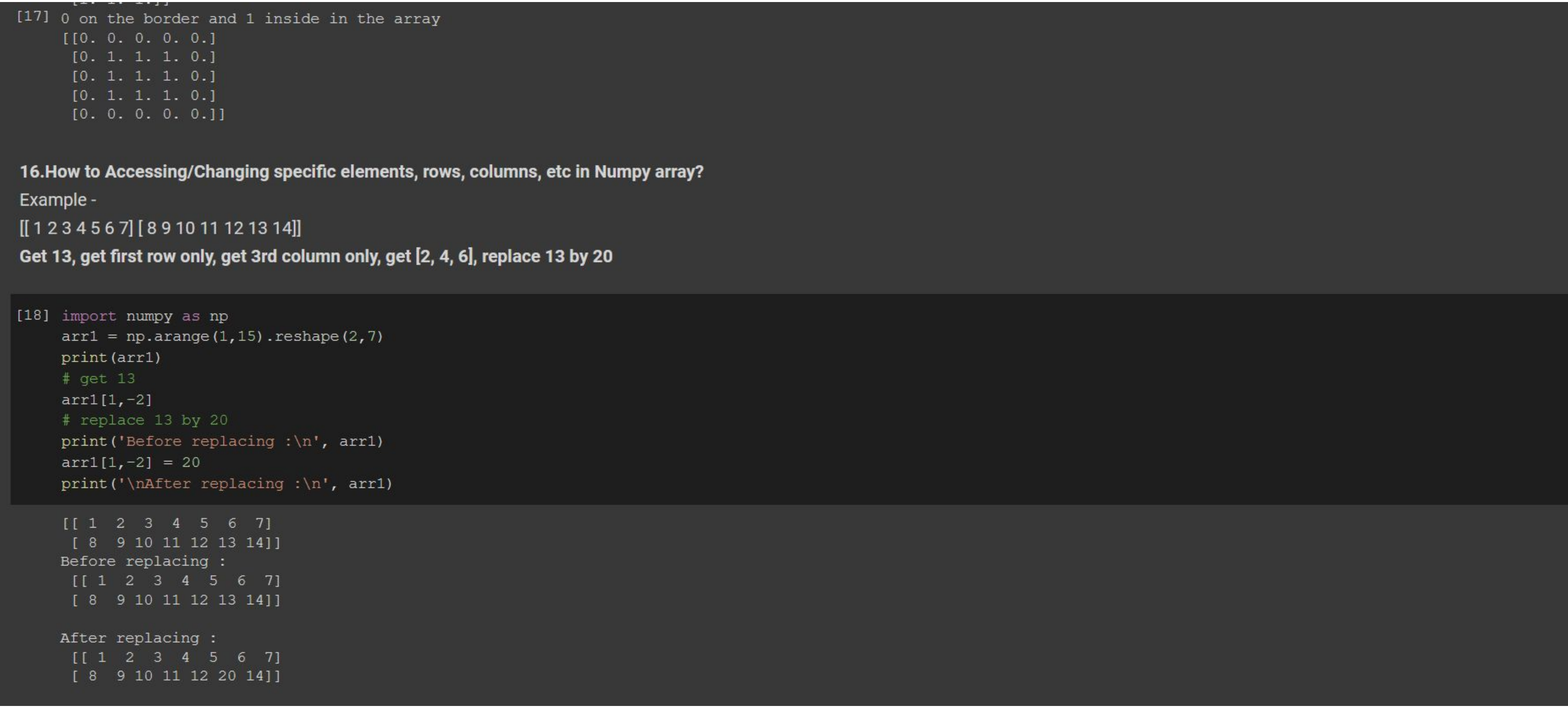
```
[16] ar = np.ones((10,10))
     ar[1:-1, 1:-1] = 0
     print(ar)
```

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

15.How to add a border (filled with 0's) around an existing array?

```
[17] import numpy as np
     x = np.ones((3,3))
     print("Original array:")
     print(x)
     print("0 on the border and 1 inside in the array")
     x = np.pad(x, pad_width=1, mode='constant', constant_values=0)
     print(x)
```

```
Original array:
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
0 on the border and 1 inside in the array
[[0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 0.]
 [0. 1. 1. 1. 0.]
 [0. 1. 1. 1. 0.]
```



```
[17] 0 on the border and 1 inside in the array
     [[0. 0. 0. 0. 0.]
      [0. 1. 1. 1. 0.]
      [0. 1. 1. 1. 0.]
      [0. 1. 1. 1. 0.]
      [0. 0. 0. 0. 0.]]
```

16.How to Accessing/Changing specific elements, rows, columns, etc in Numpy array?

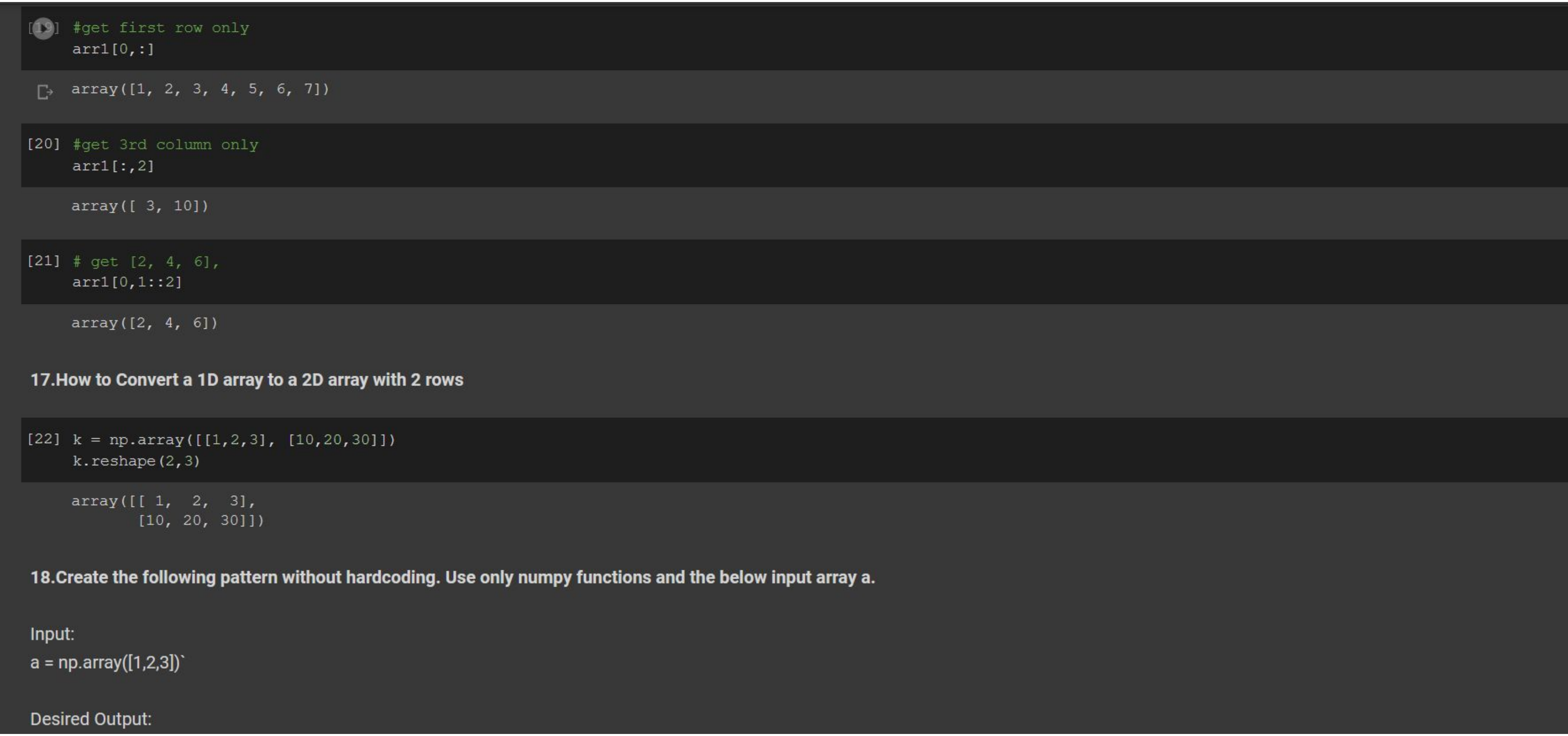
Example -

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

Get 13, get first row only, get 3rd column only, get [2, 4, 6], replace 13 by 20

```
[18] import numpy as np
     arr1 = np.arange(1,15).reshape(2,7)
     print(arr1)
     # get 13
     arr1[1,-2]
     # replace 13 by 20
     print('Before replacing :\n', arr1)
     arr1[1,-2] = 20
     print('\nAfter replacing :\n', arr1)
```

```
     [[ 1  2  3  4  5  6  7]
      [ 8  9 10 11 12 13 14]]
     Before replacing :
      [[ 1  2  3  4  5  6  7]
      [ 8  9 10 11 12 13 14]]

     After replacing :
      [[ 1  2  3  4  5  6  7]
      [ 8  9 10 11 12 20 14]]
```



```
[19] #get first row only
     arr1[0,:]
```

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

```
[20] #get 3rd column only
     arr1[:,2]
```

```
array([ 3, 10])
```

```
[21] # get [2, 4, 6],
     arr1[0,1::2]
```

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

17.How to Convert a 1D array to a 2D array with 2 rows

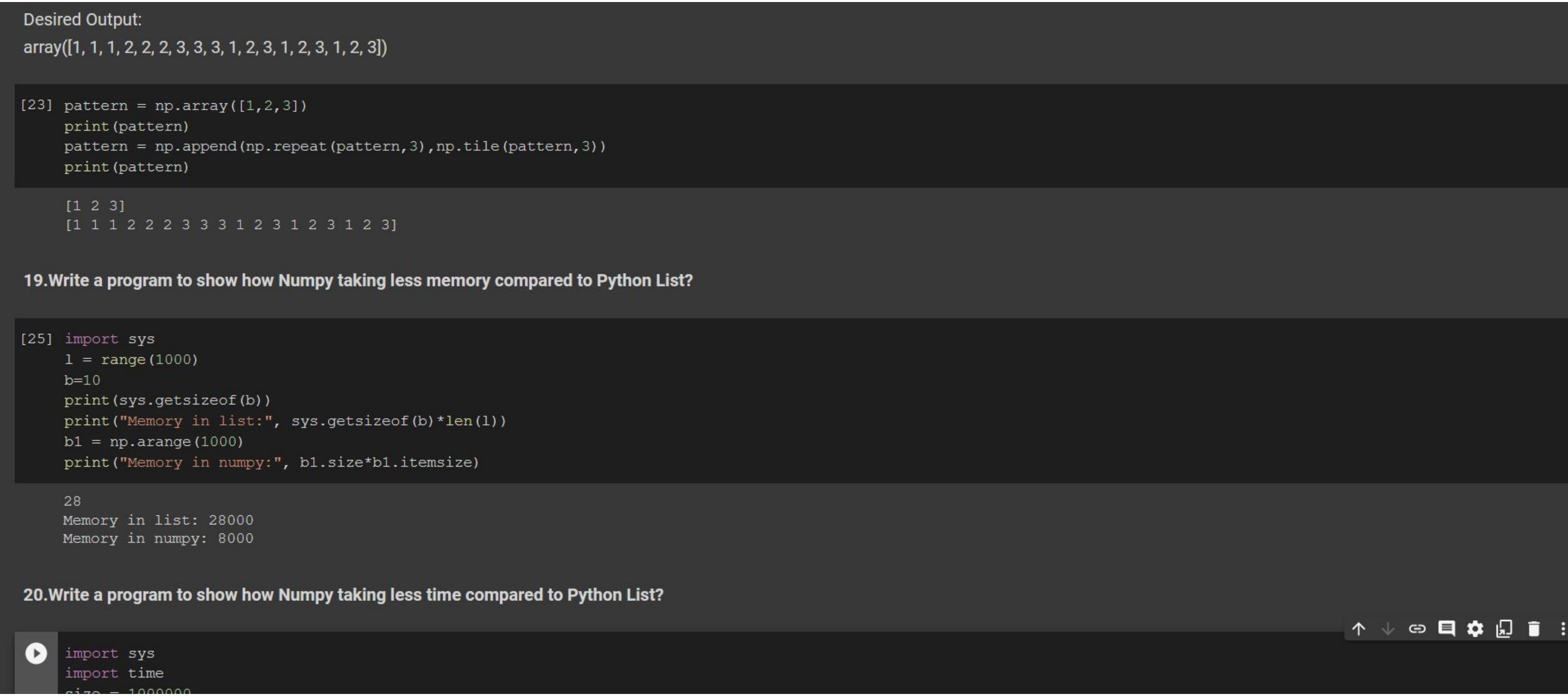
```
[22] k = np.array([[1,2,3], [10,20,30]])
     k.reshape(2,3)
```

```
array([[ 1,  2,  3],
       [10, 20, 30]])
```

18.Create the following pattern without hardcoding. Use only numpy functions and the below input array a.

Input:
a = np.array([1,2,3])`

Desired Output:



Desired Output:

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

```
[23] pattern = np.array([1,2,3])
     print(pattern)
     pattern = np.append(np.repeat(pattern,3),np.tile(pattern,3))
     print(pattern)
```

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

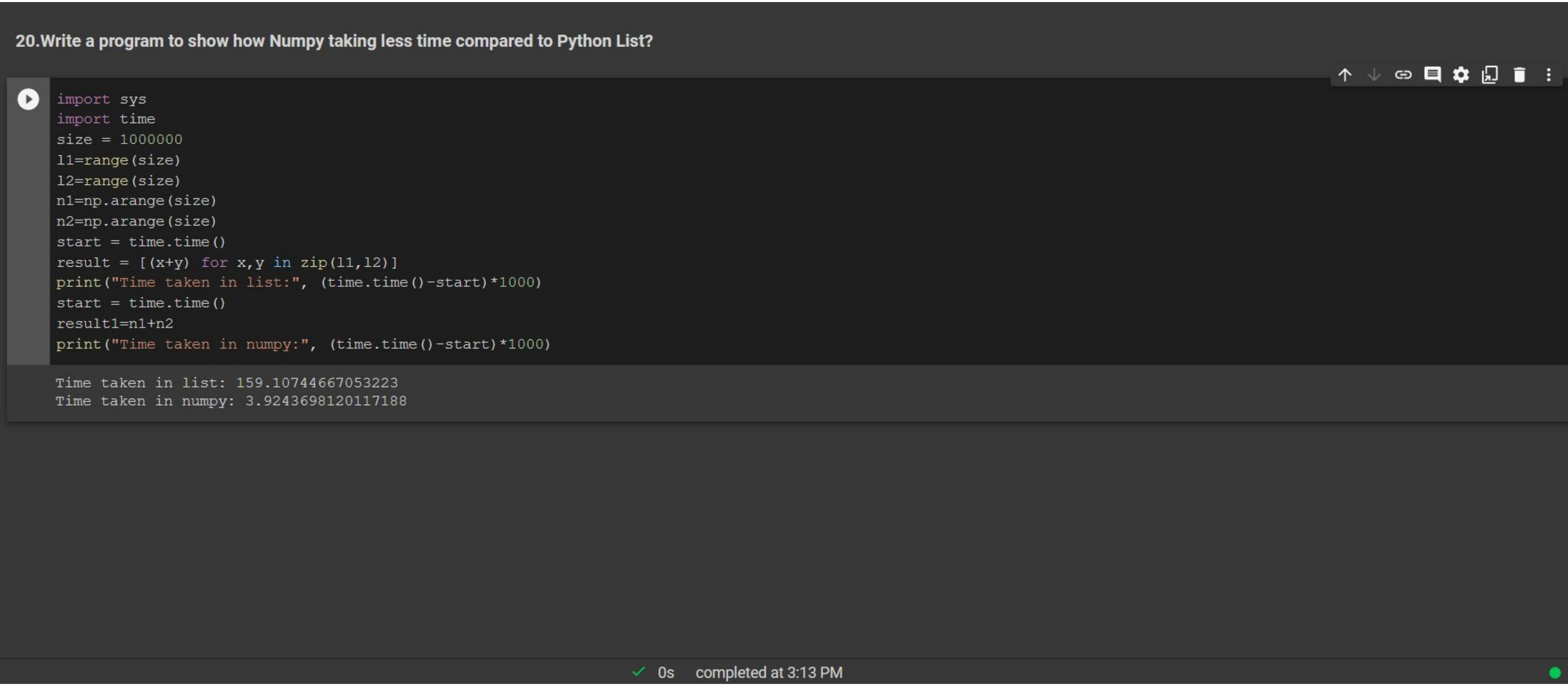
19.Write a program to show how Numpy taking less memory compared to Python List?

```
[25] import sys
     l = range(1000)
     b=10
     print(sys.getsizeof(b))
     print("Memory in list:", sys.getsizeof(b)*len(l))
     b1 = np.arange(1000)
     print("Memory in numpy:", b1.size*b1.itemsize)
```

```
28
Memory in list: 28000
Memory in numpy: 8000
```

20.Write a program to show how Numpy taking less time compared to Python List?

```
import sys
import time
size = 1000000
```



20.Write a program to show how Numpy taking less time compared to Python List?

```
import sys
import time
size = 1000000
l1=range(size)
l2=range(size)
n1=np.arange(size)
n2=np.arange(size)
start = time.time()
result = [(x+y) for x,y in zip(l1,l2)]
print("Time taken in list:", (time.time()-start)*1000)
start = time.time()
result1=n1+n2
print("Time taken in numpy:", (time.time()-start)*1000)
```

```
Time taken in list: 159.10744667053223
Time taken in numpy: 3.9243698120117188
```

✓ 0s completed at 3:13 PM