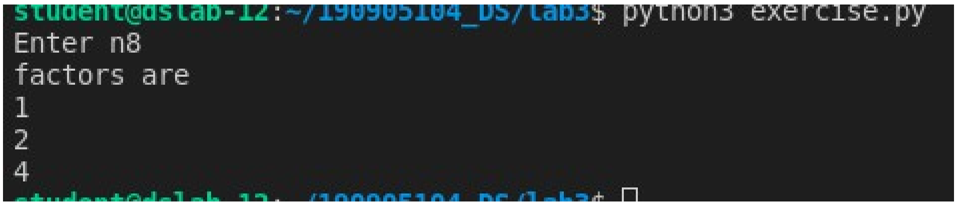
Parth Shukla
190905104
Lab 3

Exercise

1)

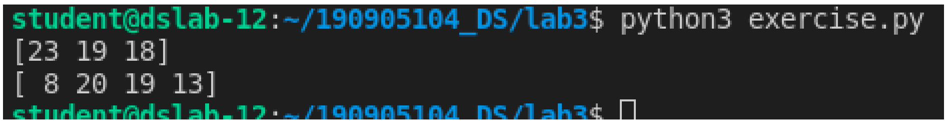
```
# Write a program to find the factors of a given number (get input from user) using for loop.

factors = []
n = int(input('Enter n'))

for i in range(1, n):
if(n%i == 0):
factors.append(i)


print('factors are')
for i in factors:
print(i)
```



```
student@dslab-12:~/190905104_DS/lab3$ python3 exercise.py
Enter n8
factors are
1
2
4
student@dslab-12:~/190905104_DS/lab3$ 
```

2)

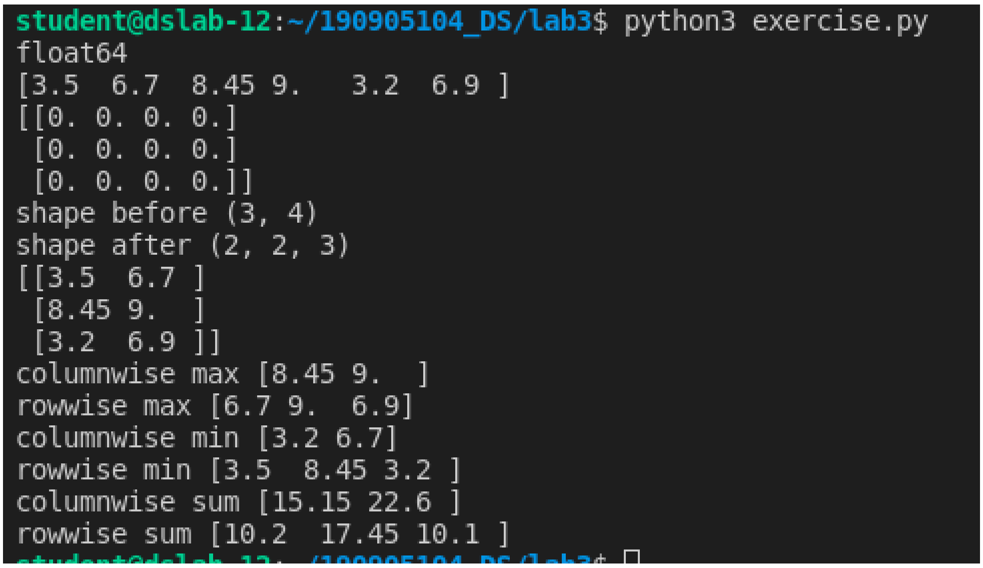
```
# Find the sum of columns and rows using axis.
import numpy as np

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

print(a.sum(axis=0))
print(a.sum(axis=1))
```



```
student@dslab-12:~/190905104_DS/lab3$ python3 exercise.py
[23 19 18]
[ 8 20 19 13]
student@dslab-12:~/190905104_DS/lab3$ 
```

3)

```
#Operations on Arrays (use numpy wherever required):
# Create array from list with type float
# Create array from tuple
# Creating a 3X4 array with all zeros
# Create a sequence of integers from 0 to 20 with steps of 5
# Reshape 3X4 array to 2X2X3 array
# Find maximum and minimum element of array, Row wise max and min, column wise
max and min and sum of elements. (Use functions max(), min(), sum())

import numpy as np

li = [3.5, 6.7, 8.45, 9.0, 3.2]

a = np.array(li)
print(a.dtype)
tu = (3.5, 6.7, 8.45, 9.0, 3.2, 6.9)
b = np.asarray(tu)
print(b)

z = np.zeros((3, 4))
print(z)

seq = np.arange(0, 20, 5)
print('shape before', z.shape)
z = z.reshape((2, 2, 3))
print('shape after', z.shape)

b = b.reshape((3, 2))

print(b)
print('columnwise max', b.max(axis = 0))
print('rowwise max', b.max(axis = 1))

print('columnwise min', b.min(axis = 0))
print('rowwise min', b.min(axis = 1))

print('columnwise sum', b.sum(axis = 0))
print('rowwise sum', b.sum(axis = 1))
```

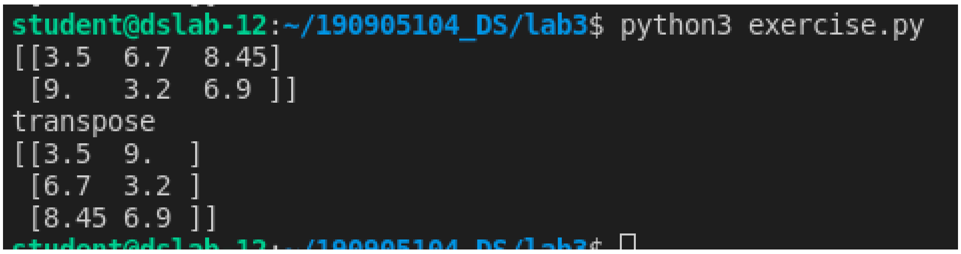


```
student@dslab-12:~/190905104_DS/lab3$ python3 exercise.py
float64
[3.5  6.7  8.45 9.   3.2  6.9 ]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
shape before (3, 4)
shape after (2, 2, 3)
[[3.5  6.7 ]
 [8.45 9.  ]
 [3.2  6.9 ]]
columnwise max [8.45 9.  ]
rowwise max [6.7  9.   6.9]
columnwise min [3.2 6.7]
rowwise min [3.5  8.45 3.2 ]
columnwise sum [15.15 22.6 ]
rowwise sum [10.2  17.45 10.1 ]
student@dslab-12:~/190905104_DS/lab3$ 
```

4)

```
# Write a program to transpose a given matrix.
import numpy as np
```

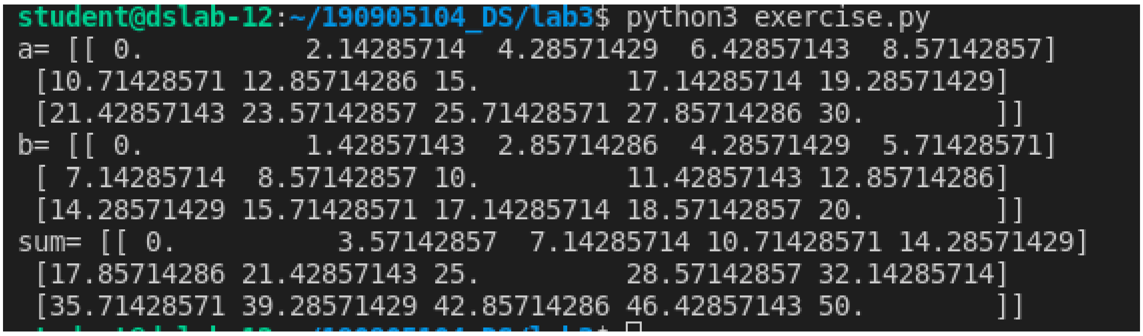
```
tu = (3.5, 6.7, 8.45, 9.0, 3.2, 6.9)
b = np.asarray(tu)
b = b.reshape((2, 3))
print(b)
print('transpose')
print(b.T)
```



```
student@dslab-12:~/190905104_DS/lab3$ python3 exercise.py
[[3.5  6.7  8.45]
 [9.   3.2  6.9 ]]
transpose
[[3.5  9.  ]
 [6.7  3.2 ]
 [8.45 6.9 ]]
student@dslab-12:~/190905104_DS/lab3$ 
```

5)

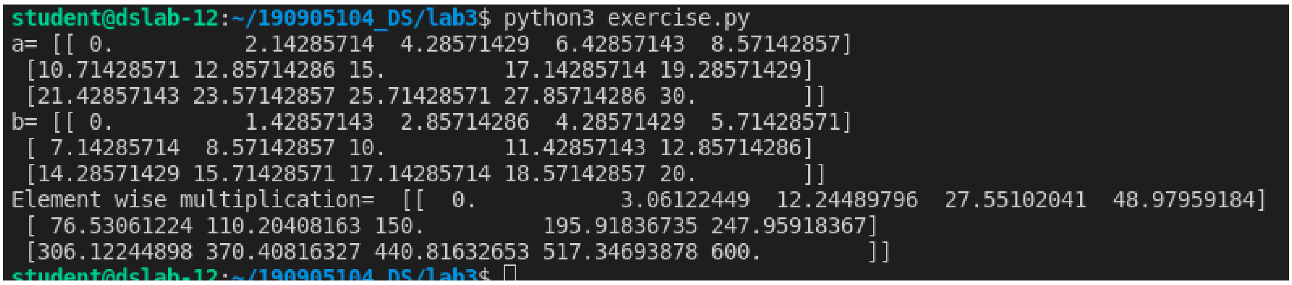
```
# Write a program to add two matrices

import numpy as np

a = np.linspace(0,30,15).reshape((3, 5))
b = np.linspace(0,20,15).reshape((3, 5))
print('a=', a)
print('b=', b)
print('sum=', a+b)
```



```
student@dslab-12:~/190905104_DS/lab3$ python3 exercise.py
a= [[ 0.          2.14285714  4.28571429  6.42857143  8.57142857]
 [10.71428571 12.85714286 15.         17.14285714 19.28571429]
 [21.42857143 23.57142857 25.71428571 27.85714286 30.        ]]
b= [[ 0.          1.42857143  2.85714286  4.28571429  5.71428571]
 [ 7.14285714  8.57142857 10.         11.42857143 12.85714286]
 [14.28571429 15.71428571 17.14285714 18.57142857 20.        ]]
sum= [[ 0.          3.57142857  7.14285714 10.71428571 14.28571429]
 [17.85714286 21.42857143 25.         28.57142857 32.14285714]
 [35.71428571 39.28571429 42.85714286 46.42857143 50.        ]]
student@dslab-12:~/190905104_DS/lab3$ 
```

6)

```
# Write a program to find element wise product between two matrices.
import numpy as np
a = np.linspace(0,30,15).reshape((3, 5))
b = np.linspace(0,20,15).reshape((3, 5))
print('a=', a)
print('b=', b)

print('Element wise multiplication= ', np.multiply(a, b))
```

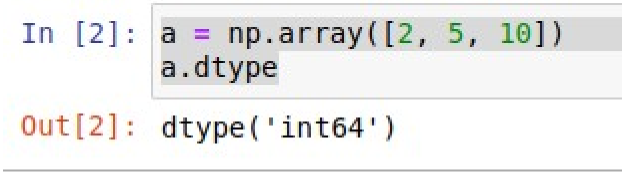


```
student@dslab-12:~/190905104_DS/lab3$ python3 exercise.py
a= [[ 0.          2.14285714  4.28571429  6.42857143  8.57142857]
 [10.71428571 12.85714286 15.         17.14285714 19.28571429]
 [21.42857143 23.57142857 25.71428571 27.85714286 30.        ]]
b= [[ 0.          1.42857143  2.85714286  4.28571429  5.71428571]
 [ 7.14285714  8.57142857 10.         11.42857143 12.85714286]
 [14.28571429 15.71428571 17.14285714 18.57142857 20.        ]]
Element wise multiplication=  [[  0.           3.06122449  12.24489796  27.55102041  48.97959184]
 [ 76.53061224 110.20408163 150.         195.91836735 247.95918367]
 [306.12244898 370.40816327 440.81632653 517.34693878 600.        ]]
student@dslab-12:~/190905104_DS/lab3$ 
```

Practice

Array creation

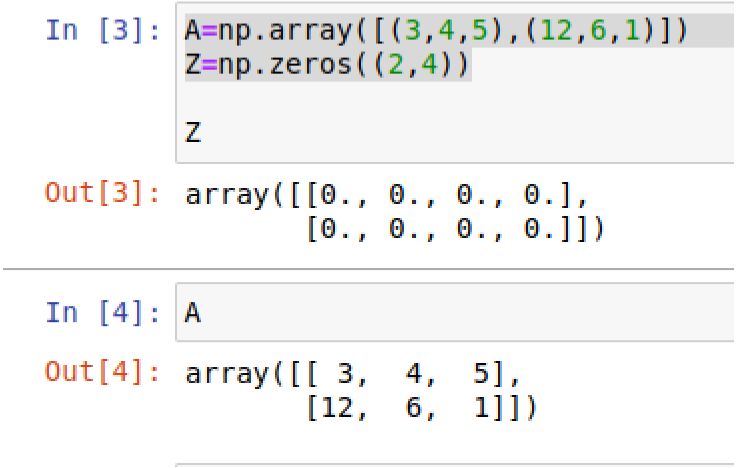
a = np.array([2, 5, 10])
a.dtype



```
In [2]: a = np.array([2, 5, 10])
        a.dtype

Out[2]: dtype('int64')
```

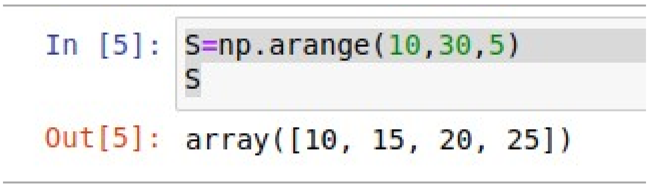
2-D array

A=np.array([(3,4,5),(12,6,1)])
Z=np.zeros((2,4))



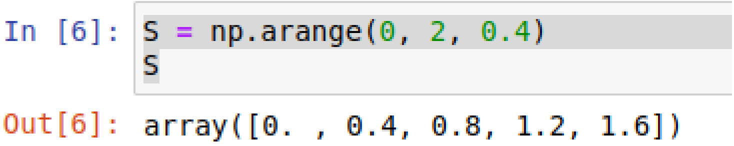
```
In [3]: A=np.array([(3,4,5),(12,6,1)])
        Z=np.zeros((2,4))

        Z

Out[3]: array([[0., 0., 0., 0.],
               [0., 0., 0., 0.]])

In [4]: A

Out[4]: array([[ 3,  4,  5],
               [12,  6,  1]])
```

Numpy Arange
S=np.arange(10,30,5)
S



```
In [5]: S=np.arange(10,30,5)
        S

Out[5]: array([10, 15, 20, 25])
```

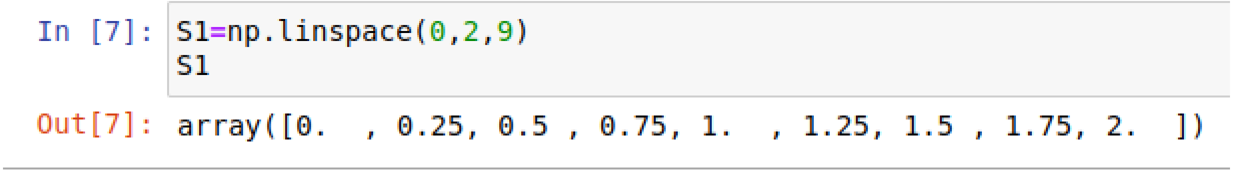
Numpy Arange with float argument
S = np.arange(0, 2, 0.4)
S



```
In [6]: S = np.arange(0, 2, 0.4)
        S

Out[6]: array([0. , 0.4, 0.8, 1.2, 1.6])
```

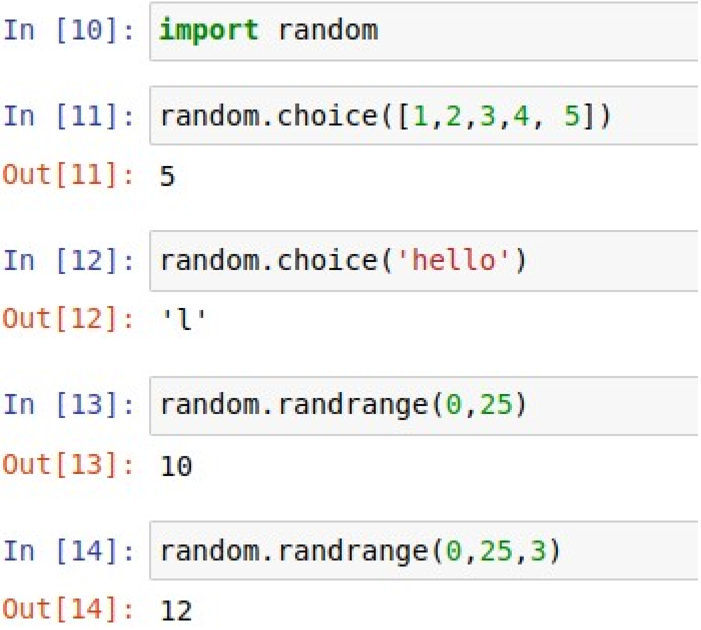
Specify number of elements in the array

S1=np.linspace(0,2,9)
S1

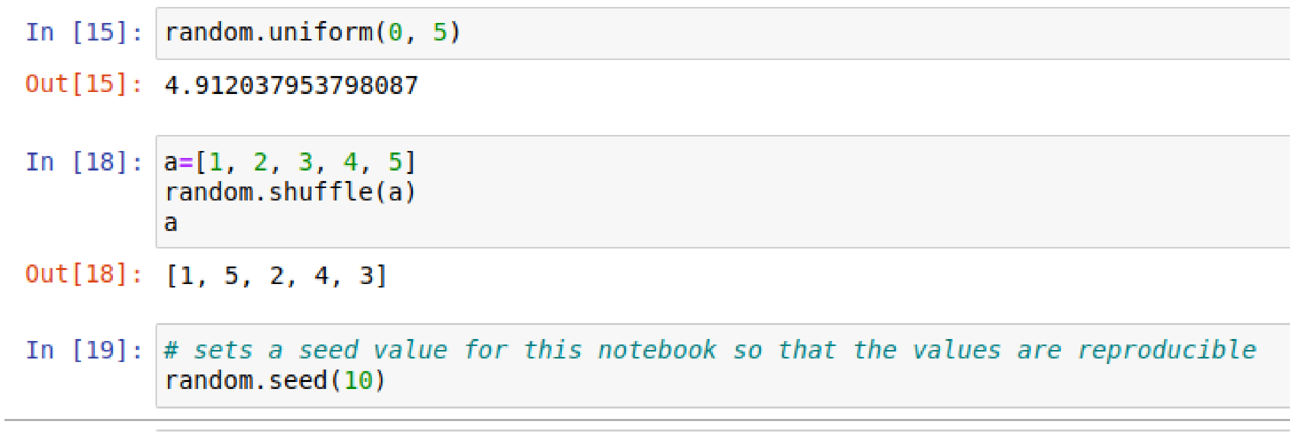


```
In [7]: S1=np.linspace(0,2,9)
        S1

Out[7]: array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
```

Random

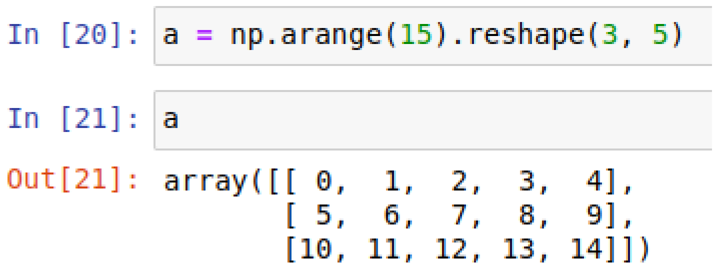


```
In [10]: import random

In [11]: random.choice([1,2,3,4, 5])

Out[11]: 5

In [12]: random.choice('hello')

Out[12]: 'l'

In [13]: random.randrange(0,25)

Out[13]: 10

In [14]: random.randrange(0,25,3)

Out[14]: 12
```

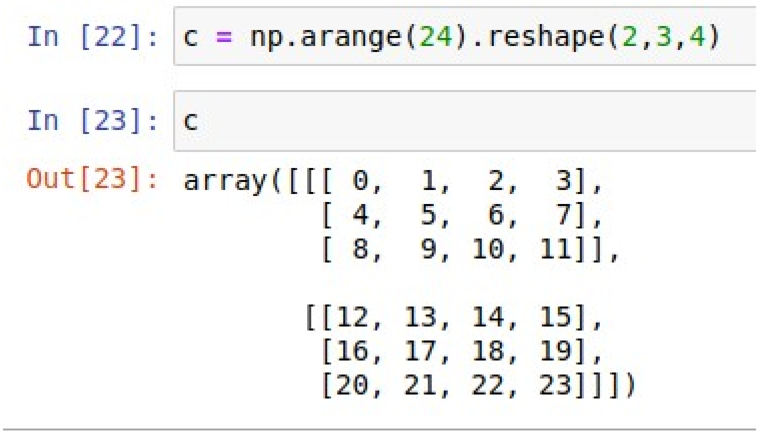


```
In [15]: random.uniform(0, 5)

Out[15]: 4.912037953798087

In [18]: a=[1, 2, 3, 4, 5]
         random.shuffle(a)
         a

Out[18]: [1, 5, 2, 4, 3]

In [19]: # sets a seed value for this notebook so that the values are reproducible
         random.seed(10)
```

Reshape using numpy

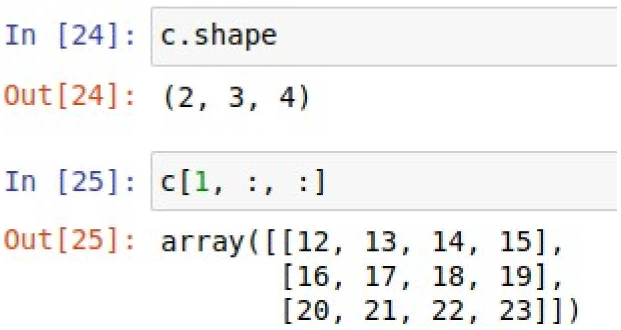


```
In [20]: a = np.arange(15).reshape(3, 5)

In [21]: a

Out[21]: array([[ 0,  1,  2,  3,  4],
                [ 5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14]])
```



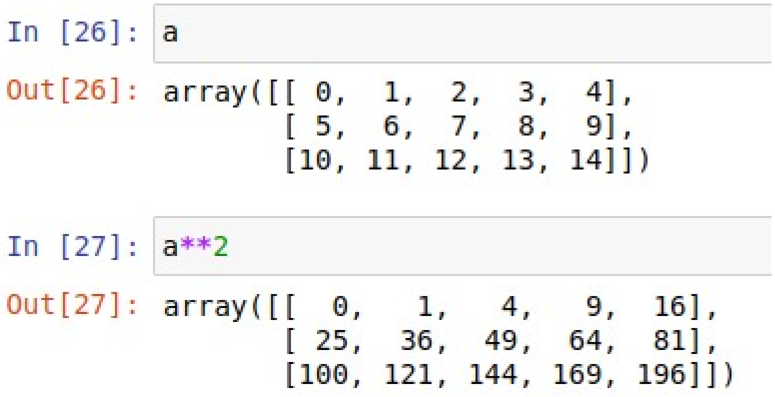
```
In [22]: c = np.arange(24).reshape(2,3,4)

In [23]: c

Out[23]: array([[[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]],

                [[12, 13, 14, 15],
                 [16, 17, 18, 19],
                 [20, 21, 22, 23]]])
```

Slicing the array

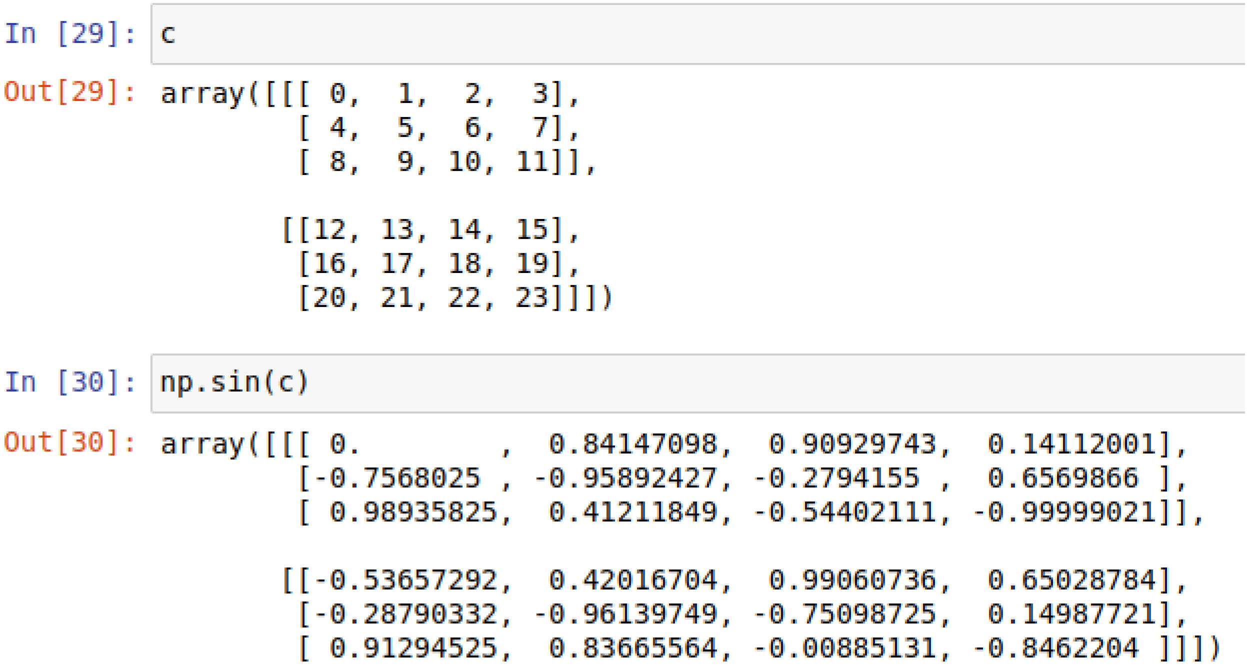


```
In [24]: c.shape

Out[24]: (2, 3, 4)

In [25]: c[1, :, :]

Out[25]: array([[12, 13, 14, 15],
                [16, 17, 18, 19],
                [20, 21, 22, 23]])
```

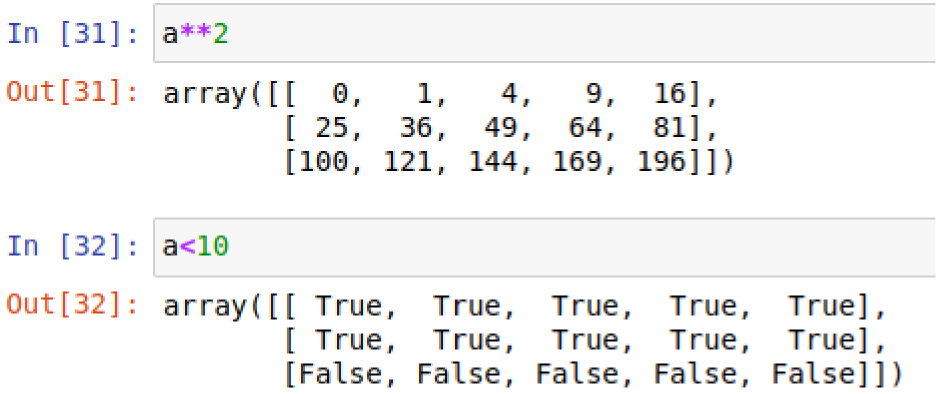
Array operations



```
In [26]: a

Out[26]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14]])

In [27]: a**2

Out[27]: array([[  0,   1,   4,   9,  16],
               [ 25,  36,  49,  64,  81],
               [100, 121, 144, 169, 196]])
```

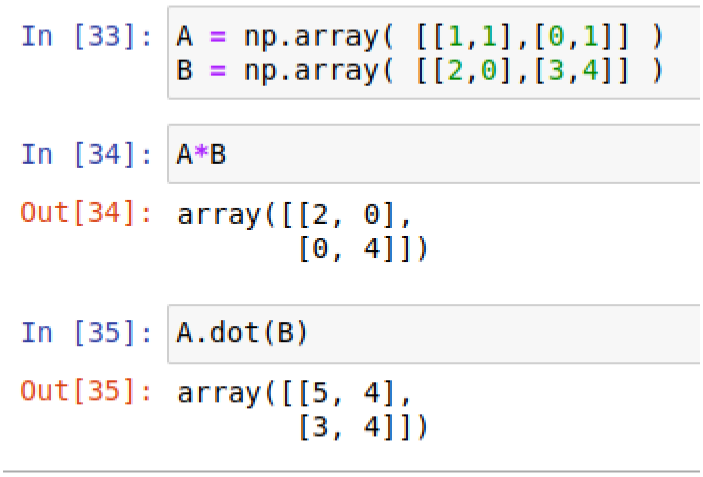


```
In [29]: c

Out[29]: array([[[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11]],

               [[12, 13, 14, 15],
                [16, 17, 18, 19],
                [20, 21, 22, 23]]])

In [30]: np.sin(c)

Out[30]: array([[[ 0.        ,  0.84147098,  0.90929743,  0.14112001],
                [-0.7568025 , -0.95892427, -0.2794155 ,  0.6569866 ],
                [ 0.98935825,  0.41211849, -0.54402111, -0.99999021]],

               [[-0.53657292,  0.42016704,  0.99060736,  0.65028784],
                [-0.28790332, -0.96139749, -0.75098725,  0.14987721],
                [ 0.91294525,  0.83665564, -0.00885131, -0.8462204 ]]])
```

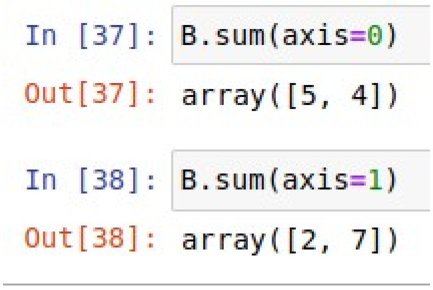


```
In [31]: a**2

Out[31]: array([[  0,   1,   4,   9,  16],
               [ 25,  36,  49,  64,  81],
               [100, 121, 144, 169, 196]])

In [32]: a<10

Out[32]: array([[ True,  True,  True,  True,  True],
               [ True,  True,  True,  True,  True],
               [False, False, False, False, False]])
```

Matrix operations

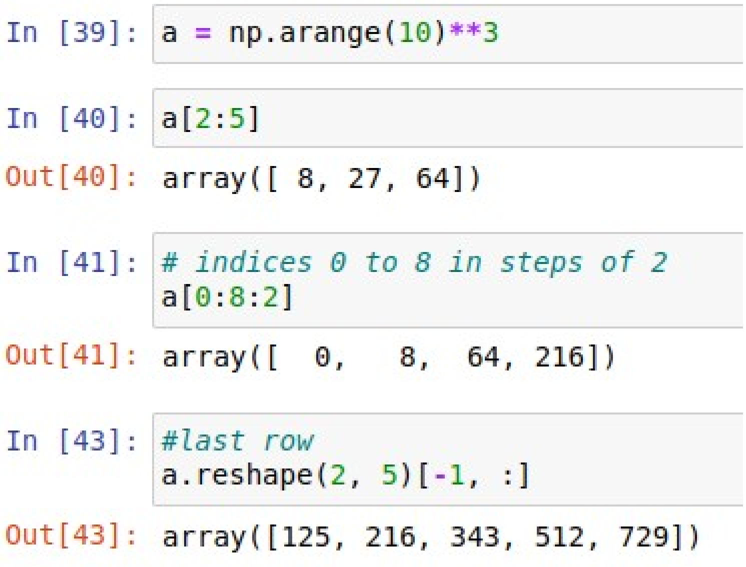


```
In [33]: A = np.array( [[1,1],[0,1]] )
         B = np.array( [[2,0],[3,4]] )

In [34]: A*B

Out[34]: array([[2, 0],
                [0, 4]])

In [35]: A.dot(B)

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

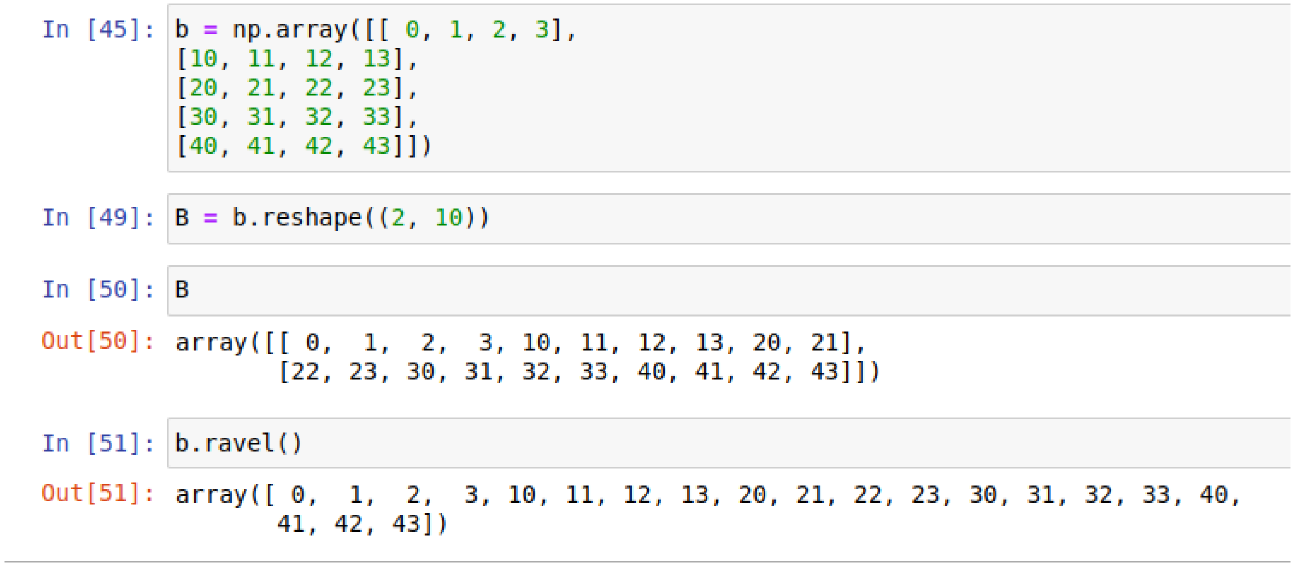


```
In [37]: B.sum(axis=0)

Out[37]: array([5, 4])

In [38]: B.sum(axis=1)

Out[38]: array([2, 7])
```

Indexing and slicing array

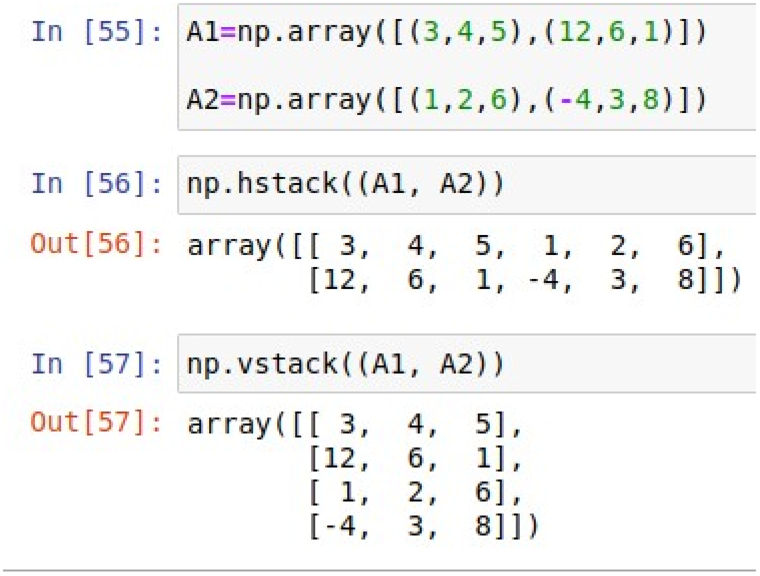


```
In [39]: a = np.arange(10)**3

In [40]: a[2:5]

Out[40]: array([ 8, 27, 64])

In [41]: # indices 0 to 8 in steps of 2
         a[0:8:2]

Out[41]: array([  0,   8,  64, 216])

In [43]: #last row
         a.reshape(2, 5)[-1, :]

Out[43]: array([125, 216, 343, 512, 729])
```

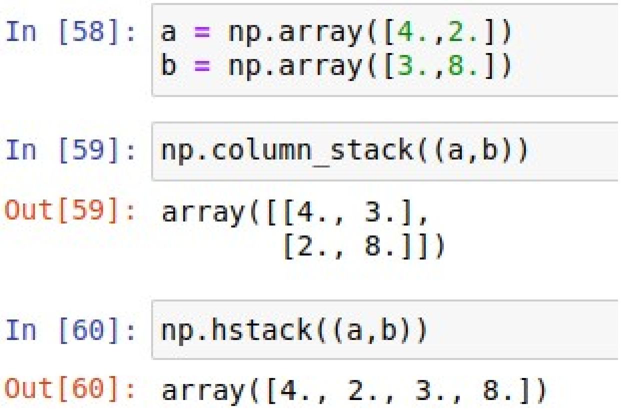


```
In [45]: b = np.array([[ 0, 1, 2, 3],
         [10, 11, 12, 13],
         [20, 21, 22, 23],
         [30, 31, 32, 33],
         [40, 41, 42, 43]])

In [49]: B = b.reshape((2, 10))

In [50]: B

Out[50]: array([[ 0,  1,  2,  3, 10, 11, 12, 13, 20, 21],
                [22, 23, 30, 31, 32, 33, 40, 41, 42, 43]])

In [51]: b.ravel()

Out[51]: array([ 0,  1,  2,  3, 10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33, 40,
                41, 42, 43])
```

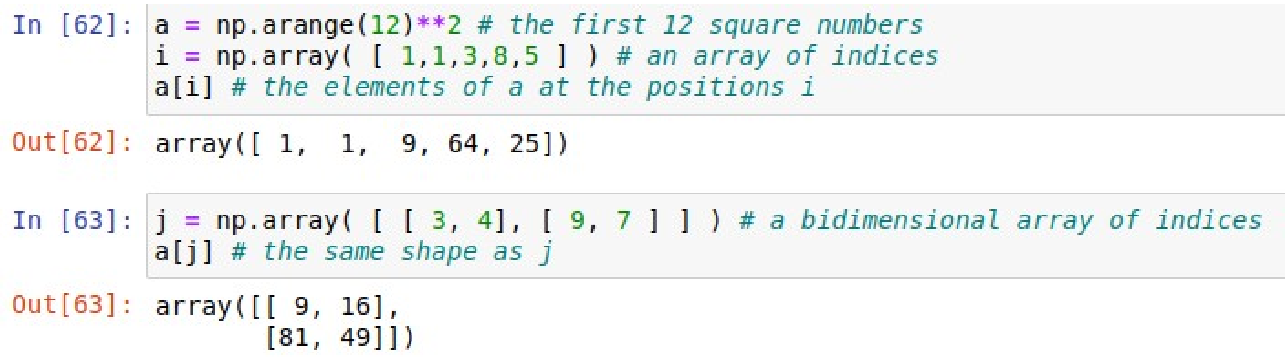
Stacking arrays on each other



```
In [55]: A1=np.array([(3,4,5),(12,6,1)])

         A2=np.array([(1,2,6),(-4,3,8)])

In [56]: np.hstack((A1, A2))

Out[56]: array([[ 3,  4,  5,  1,  2,  6],
                [12,  6,  1, -4,  3,  8]])

In [57]: np.vstack((A1, A2))

Out[57]: array([[ 3,  4,  5],
                [12,  6,  1],
                [ 1,  2,  6],
                [-4,  3,  8]])
```

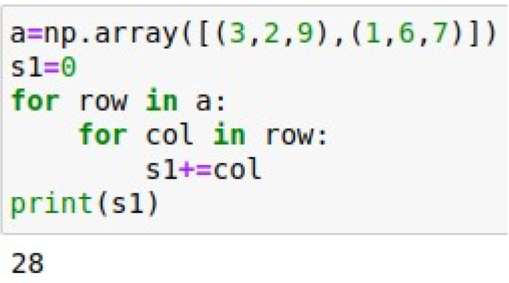


```
In [58]: a = np.array([4.,2.])
         b = np.array([3.,8.])

In [59]: np.column_stack((a,b))

Out[59]: array([[4., 3.],
                [2., 8.]])

In [60]: np.hstack((a,b))

Out[60]: array([4., 2., 3., 8.])
```



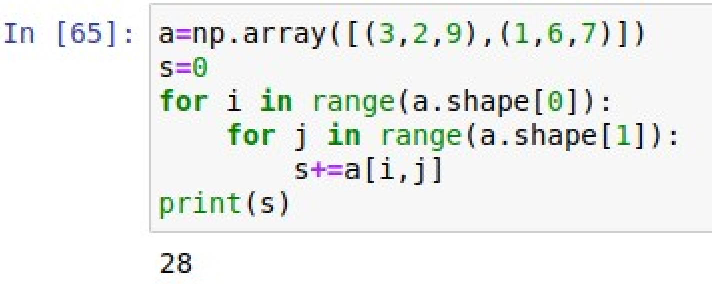
```
In [62]: a = np.arange(12)**2 # the first 12 square numbers
         i = np.array( [ 1,1,3,8,5 ] ) # an array of indices
         a[i] # the elements of a at the positions i

Out[62]: array([ 1,  1,  9, 64, 25])

In [63]: j = np.array( [ [ 3, 4], [ 9, 7 ] ] ) # a bidimensional array of indices
         a[j] # the same shape as j

Out[63]: array([[ 9, 16],
                [81, 49]])
```

Mapping by value in for loops



```
a=np.array([(3,2,9),(1,6,7)])
s1=0
for row in a:
    for col in row:
        s1+=col
print(s1)

28
```

Mapping by index in for loops



```
In [65]: a=np.array([(3,2,9),(1,6,7)])
         s=0
         for i in range(a.shape[0]):
             for j in range(a.shape[1]):
                 s+=a[i,j]
         print(s)

         28
```