Type on command line

```
#check python version
py –version
# check pip version
Py –m pip –version
#insatll numpy
py -m pip install numpy
# to upgrade pip
py -m pip install --upgrade pip

# to install pandas
py -m pip install pandas
# to install matplotlib
py -m pip install matplotlib
```

NUMPY

Numpy Exercise 1:

```
>>>import numpy as np
>>> k=np.array([[1,2,3],[2,4,5],[6,7,8]])
>>>print(k)
>>>print(k.shape)

# Basic Indexing
```

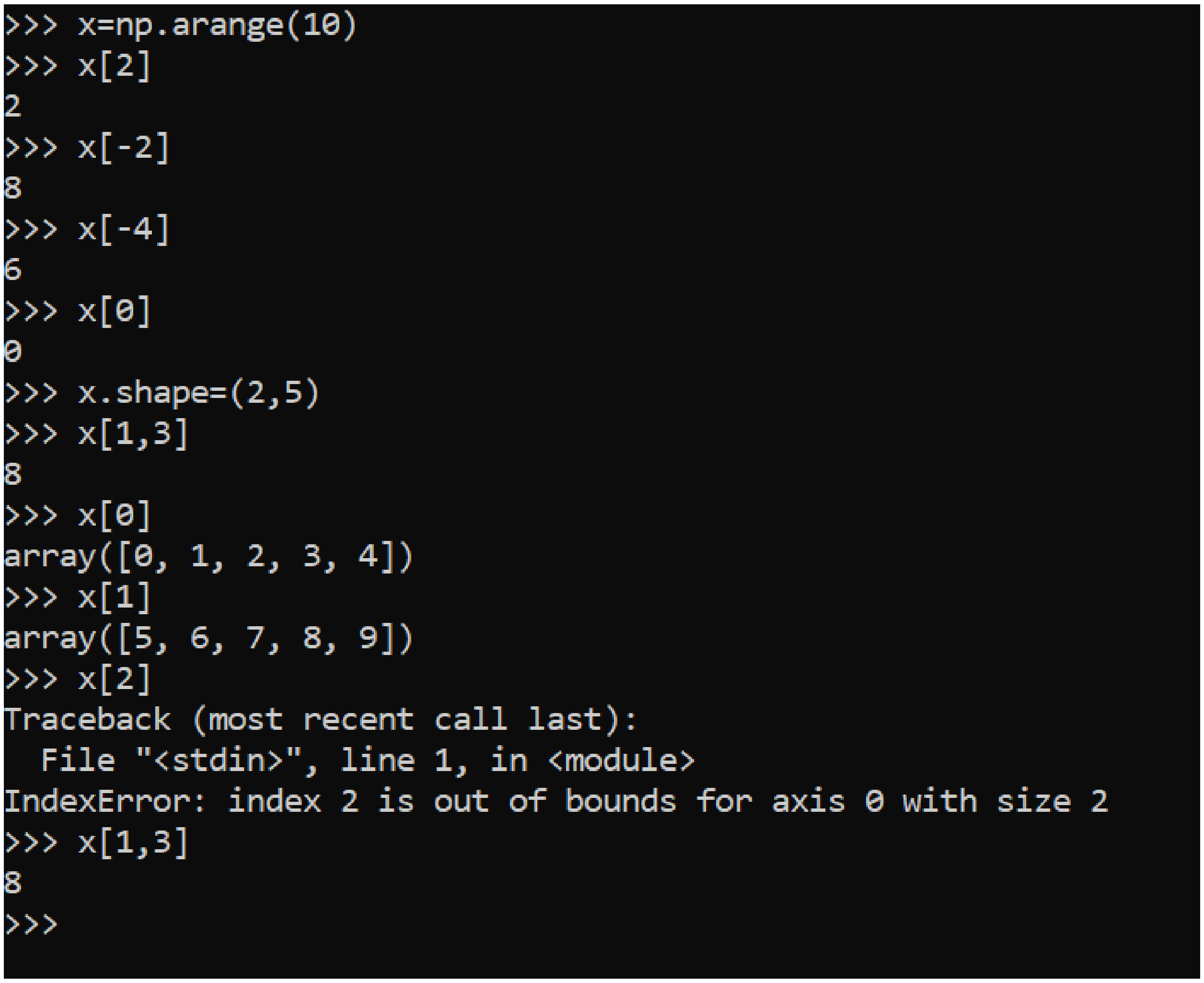
Single element indexing

Single element indexing works exactly like that for other standard Python sequences. It is 0-based, and accepts negative indices for indexing from the end of the array.

```
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
```

It is not necessary to separate each dimension’s index into its own set of square brackets.

```
>>> x.shape = (2, 5)  # now x is 2-dimensional
>>> x[1, 3]
8
>>> x[1, -1]
9
```



```
>>> x=np.arange(10)
>>> x[2]
2
>>> x[-2]
8
>>> x[-4]
6
>>> x[0]
0
>>> x.shape=(2,5)
>>> x[1,3]
8
>>> x[0]
array([0, 1, 2, 3, 4])
>>> x[1]
array([5, 6, 7, 8, 9])
>>> x[2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: index 2 is out of bounds for axis 0 with size 2
>>> x[1,3]
8
>>>
```

Numpy Exercise 2:

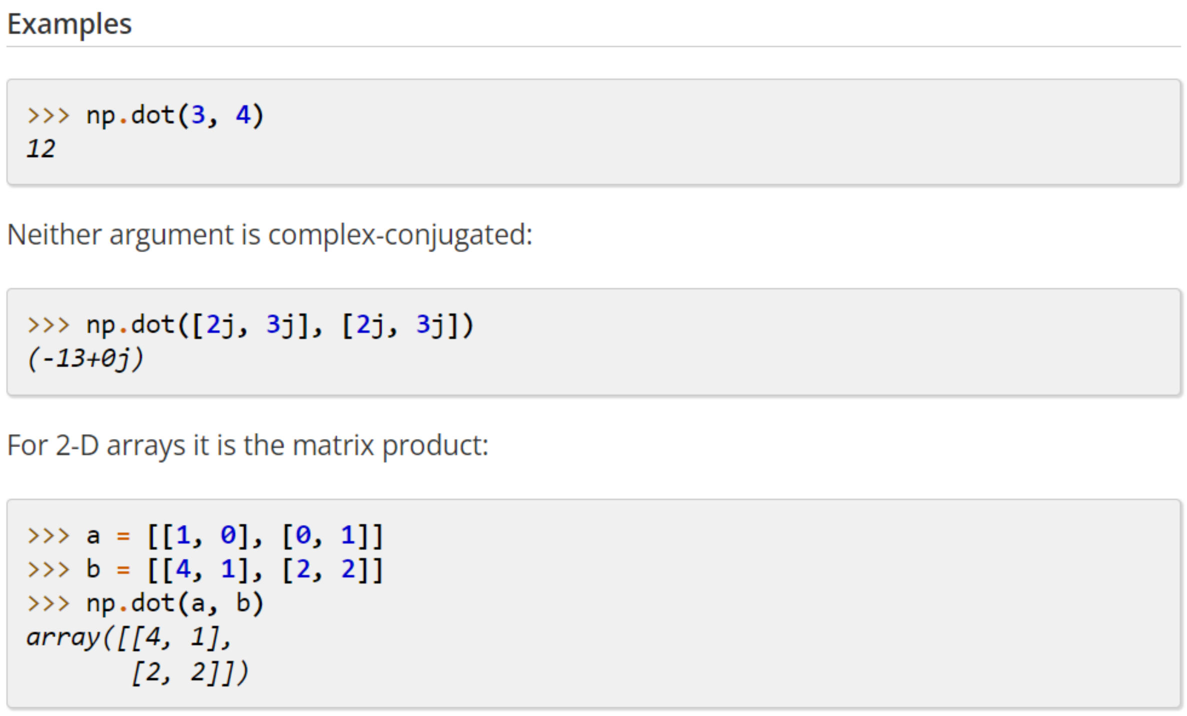
```python
import numpy as np

arr1 = np.array([10, 11, 12, 13, 14, 15])
arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = np.add(arr1, arr2)

print(newarr)
```

Numpy Exercise 3:

```python
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = np.subtract(arr1, arr2)

print(newarr)
```

Ref examples:
https://www.w3schools.com/python/numpy/numpy_ufunc_simple_arithmetic.asp

#Numpy Dot Product:



Examples

```
>>> np.dot(3, 4)
12
```

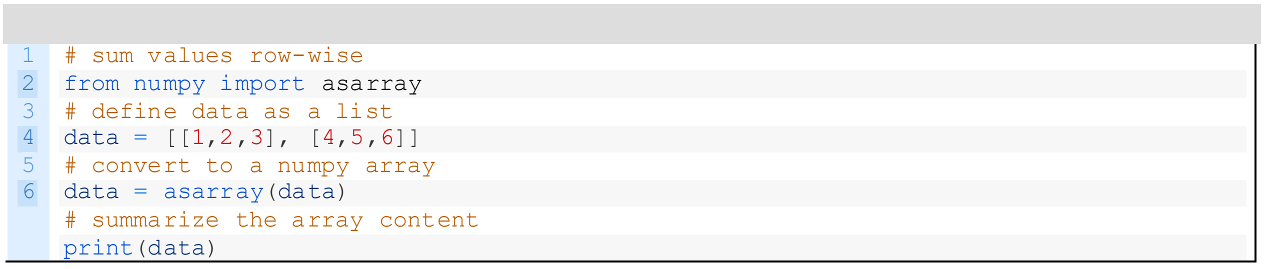
Neither argument is complex-conjugated:

```
>>> np.dot([2j, 3j], [2j, 3j])
(-13+0j)
```

For 2-D arrays it is the matrix product:

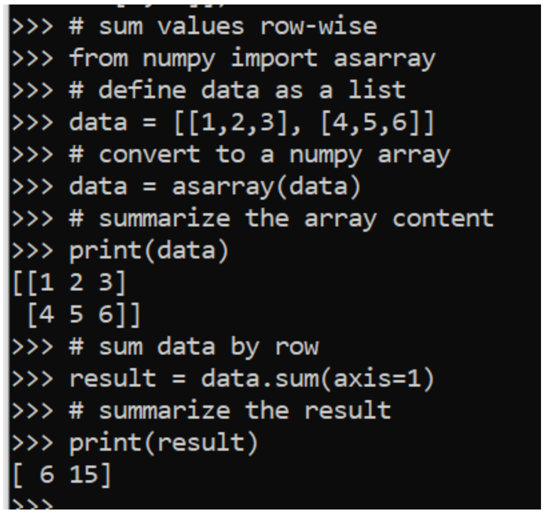
```
>>> a = [[1, 0], [0, 1]]
>>> b = [[4, 1], [2, 2]]
>>> np.dot(a, b)
array([[4, 1],
       [2, 2]])
```

# Axis Operations:



```python
# sum values row-wise
from numpy import asarray
# define data as a list
data = [[1,2,3], [4,5,6]]
# convert to a numpy array
data = asarray(data)
# summarize the array content
print(data)
```

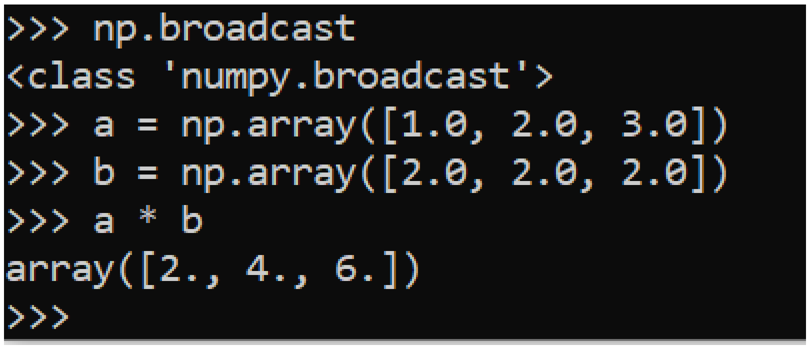
```
# sum data by row
result = data.sum(axis=1)
# summarize the result
print(result)
```



```
>>> # sum values row-wise
>>> from numpy import asarray
>>> # define data as a list
>>> data = [[1,2,3], [4,5,6]]
>>> # convert to a numpy array
>>> data = asarray(data)
>>> # summarize the array content
>>> print(data)
[[1 2 3]
 [4 5 6]]
>>> # sum data by row
>>> result = data.sum(axis=1)
>>> # summarize the result
>>> print(result)
[ 6 15]
>>>
```

#Broadcast:

```
a = np.array([1.0, 2.0, 3.0])
>>> b = np.array([2.0, 2.0, 2.0])
>>> a * b
```



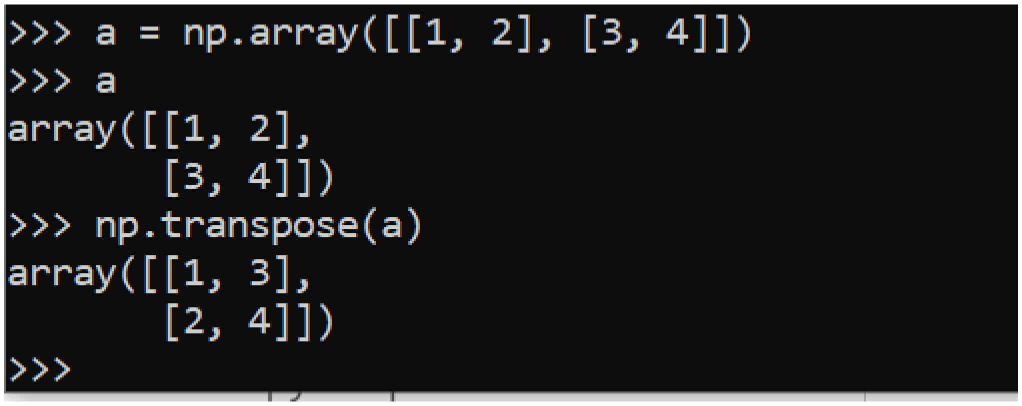
```
>>> np.broadcast
<class 'numpy.broadcast'>
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = np.array([2.0, 2.0, 2.0])
>>> a * b
array([2., 4., 6.])
>>>
```

Numpy Broadcast exercise:

```
a = np.array([1.0, 2.0, 3.0])
>>> b = 2.0
>>> a * b
```

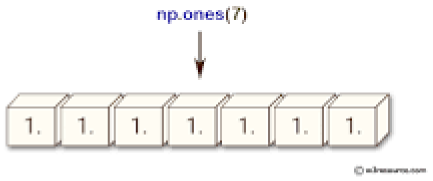
```
#transpose a matrix
```

```
a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> np.transpose(a)
```

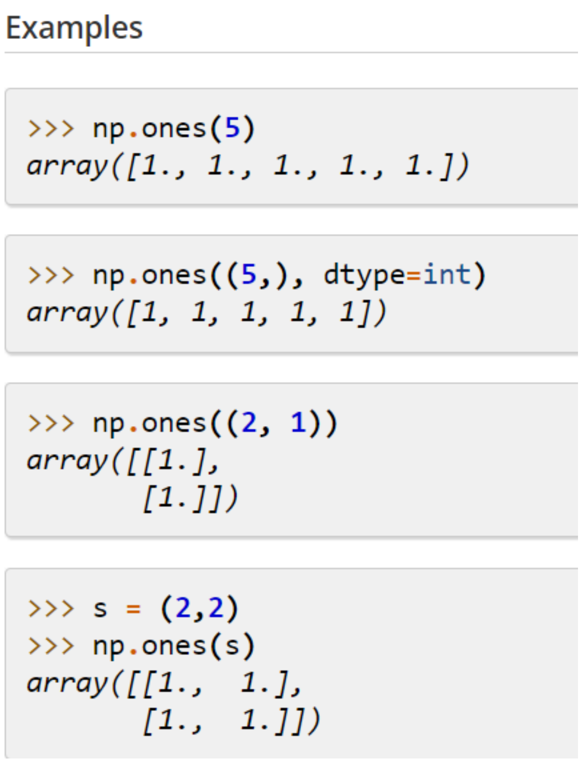


```
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> np.transpose(a)
array([[1, 3],
       [2, 4]])
>>>
```

What is NP ones in Python?



The numpy. ones() function is used to create a new array of given shape and type, filled with ones. The ones() function is useful in situations where we need to create an array of ones with a specific shape and data type, for example in matrix operations or in initializing an array with default values.


Examples

```
>>> np.ones(5)
array([1., 1., 1., 1., 1.])
```

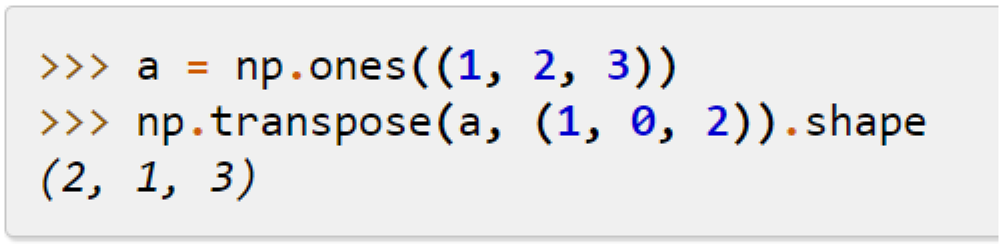
```
>>> np.ones((5,), dtype=int)
array([1, 1, 1, 1, 1])
```

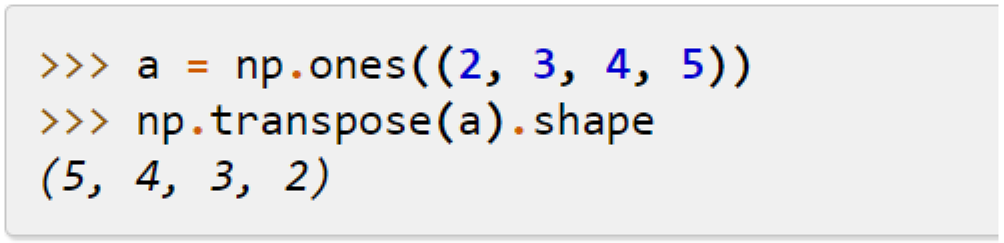
```
>>> np.ones((2, 1))
array([[1.],
       [1.]])
```

```
>>> s = (2,2)
>>> np.ones(s)
array([[1.,  1.],
       [1.,  1.]])
```

Let’s try:

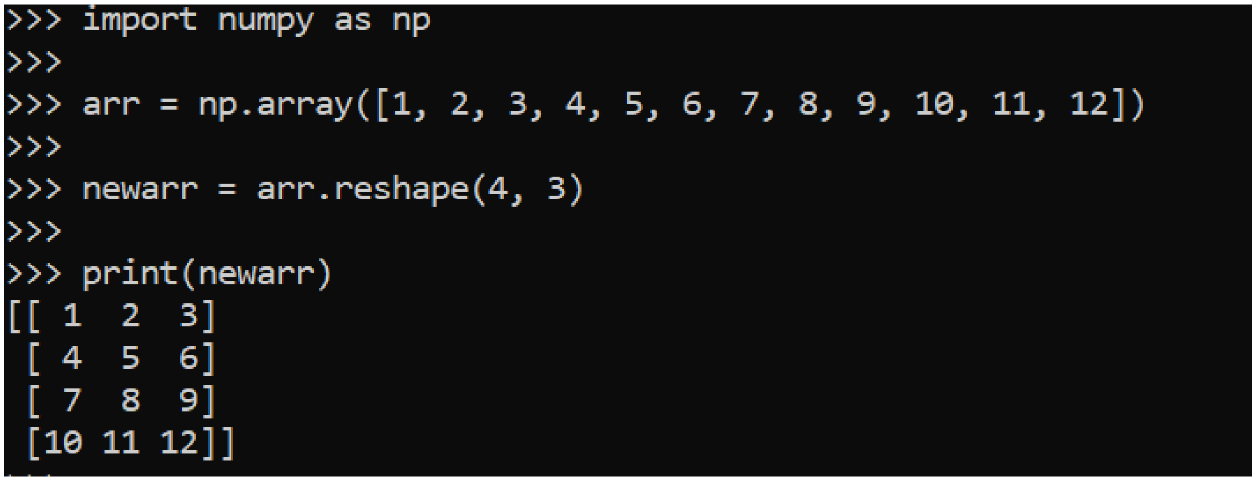
```
a = np.ones((1, 2, 3))
>>> np.transpose(a, (1, 0, 2)).shape
```


```
>>> a = np.ones((1, 2, 3))
>>> np.transpose(a, (1, 0, 2)).shape
(2, 1, 3)
```


```
>>> a = np.ones((2, 3, 4, 5))
>>> np.transpose(a).shape
(5, 4, 3, 2)
```

# reshape

```
import numpy as np

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

newarr = arr.reshape(4, 3)

print(newarr)
```



```
>>> import numpy as np
>>>
>>> arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
>>>
>>> newarr = arr.reshape(4, 3)
>>>
>>> print(newarr)
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

# joining:

Ex1:

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

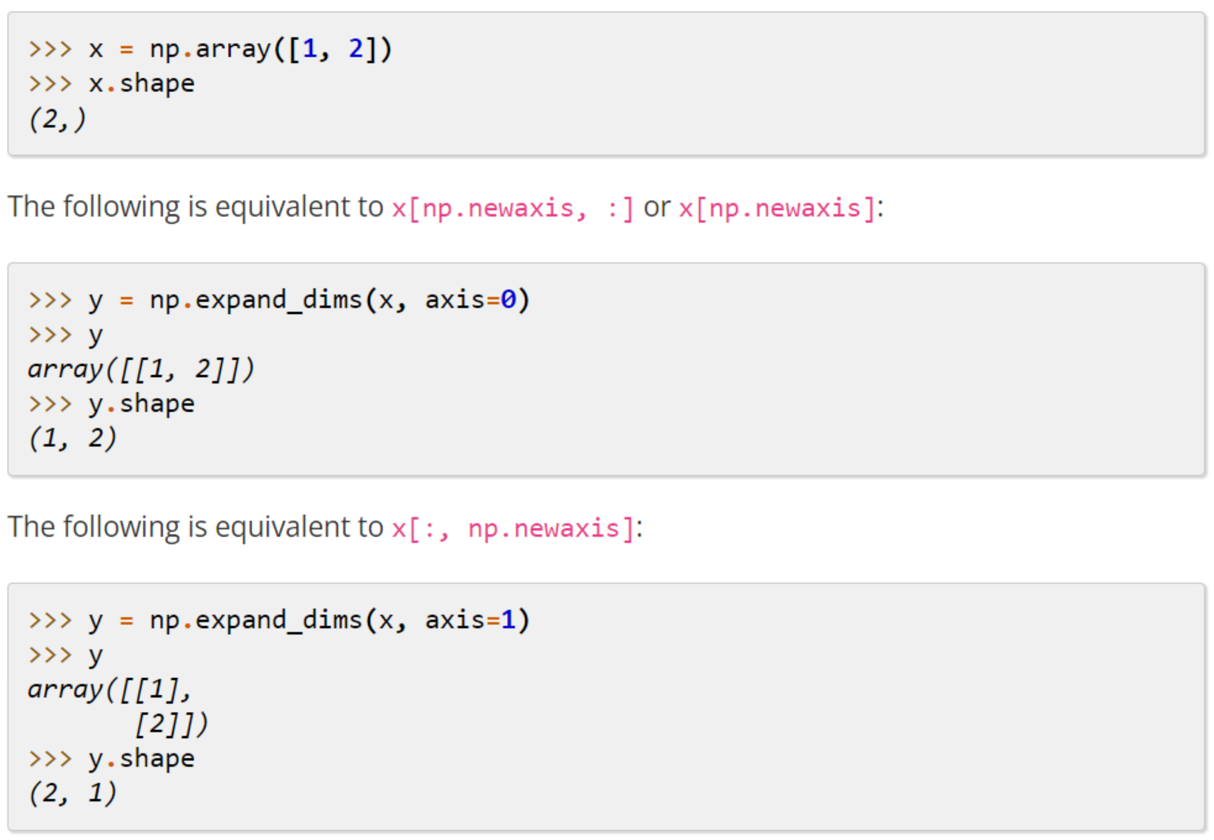
Ex2:

```
import numpy as np

arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])
```

```
arr = np.concatenate((arr1, arr2), axis=1)

print(arr)
```

# expand:

#Try on your own:



```
>>> x = np.array([1, 2])
>>> x.shape
(2,)
```

The following is equivalent to `x[np.newaxis, :]` or `x[np.newaxis]`:

```
>>> y = np.expand_dims(x, axis=0)
>>> y
array([[1, 2]])
>>> y.shape
(1, 2)
```

The following is equivalent to `x[:, np.newaxis]`:

```
>>> y = np.expand_dims(x, axis=1)
>>> y
array([[1],
       [2]])
>>> y.shape
(2, 1)
```

#reduce:

```
np.multiply.reduce([2,3,5])
```

To more in detail:
https://numpy.org/doc/stable/reference/generated/numpy.ufunc.reduce.html

Official Documentation:

https://numpy.org/doc/stable/user/index.html