NumPy

Introduction

- NumPy (**Numerical Python**) is an open source Python library.
- Work with numerical data in Python.
- Contains multidimensional array and matrix data structures.
- provides **ndarray**, a homogeneous n-dimensional array object, with methods to efficiently operate on it.
- Used to perform a wide variety of mathematical operations on arrays.

Installation of NumPy

- conda install numpy

  or

- pip install numpy

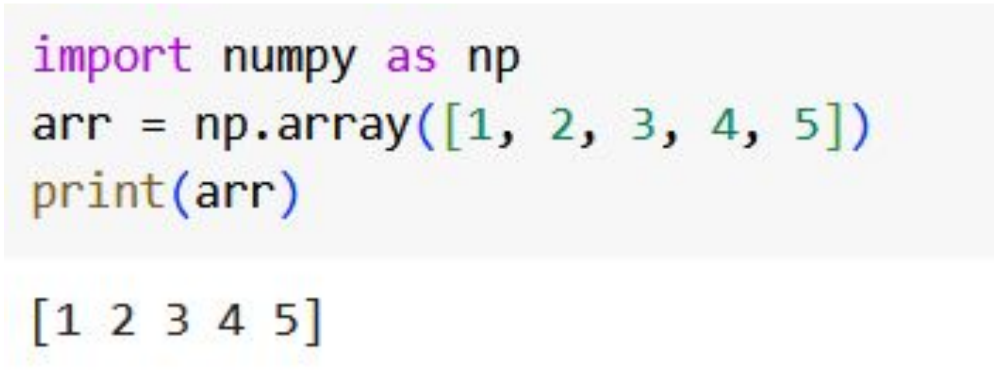
How to import NumPy

- To access NumPy and its functions import it in your Python code like this:

```
import numpy as np
```

- We shorten the imported name to np for better readability of code using NumPy.

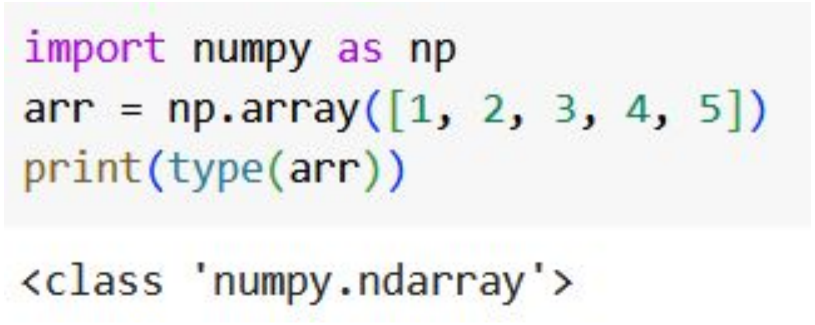
Create a NumPy ndarray Object



```python
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

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

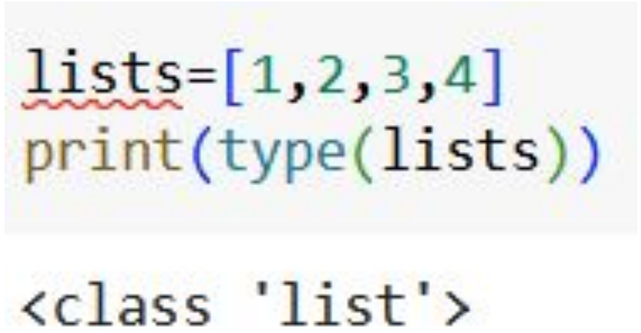
Array vs List



```python
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(type(arr))
```

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

Array

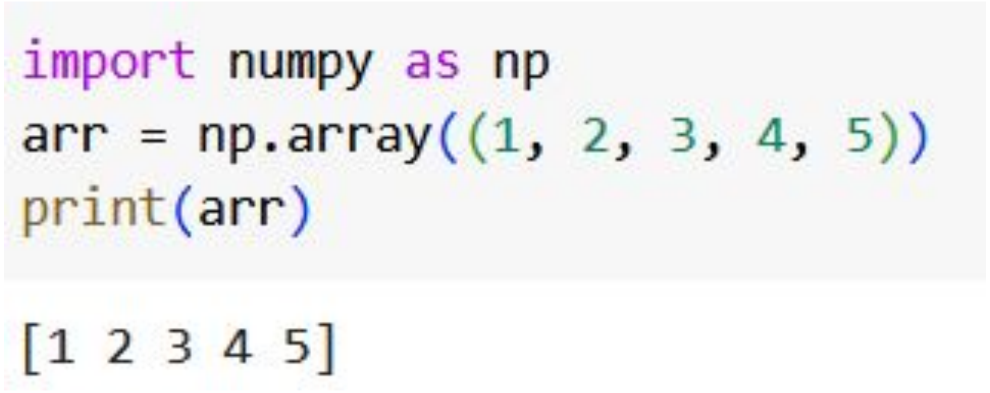


```python
lists=[1,2,3,4]
print(type(lists))
```

```
<class 'list'>
```

List

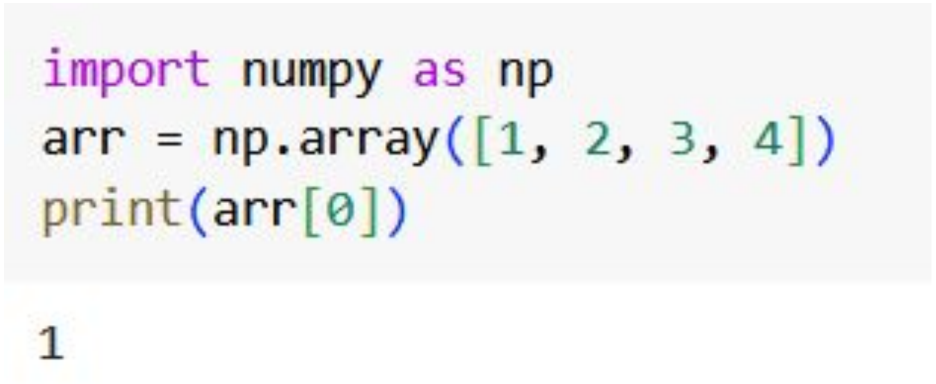
NumPy array using Tuple



```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))
print(arr)
```

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

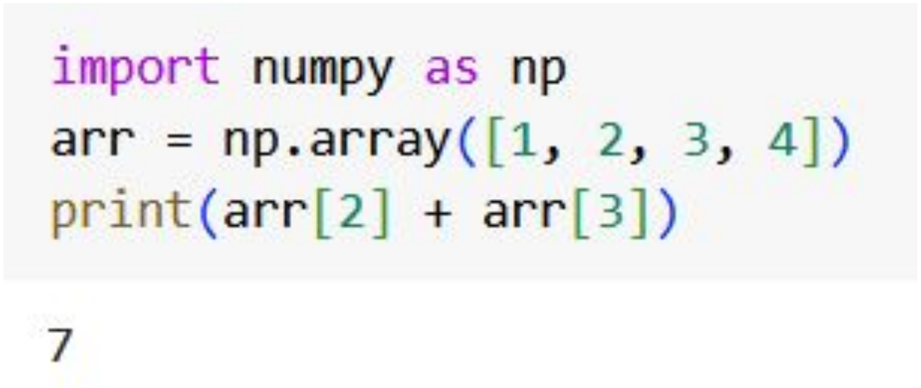
Access Array Elements



```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

```
1
```

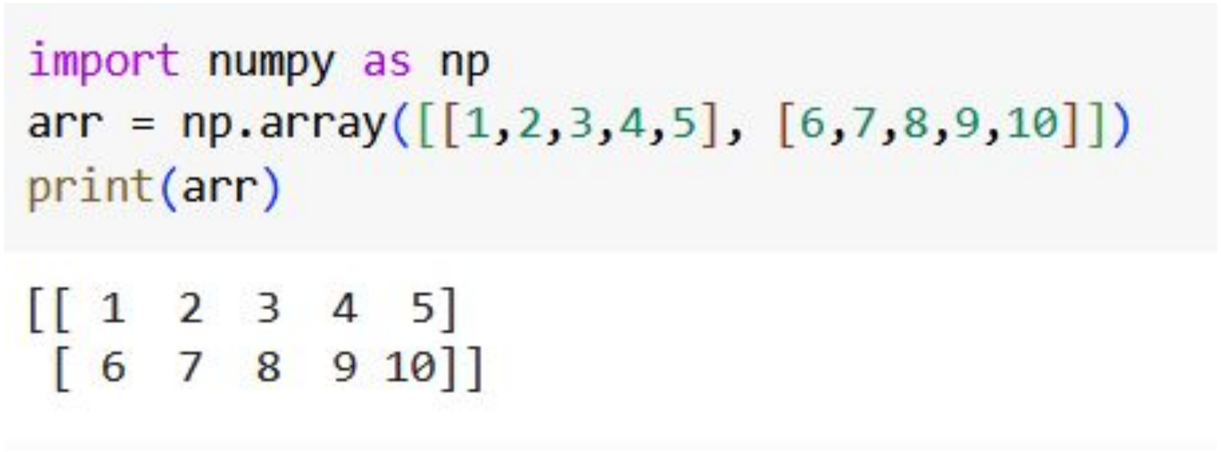
Access Array Elements



```python
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])
```

```
7
```

Access 2D Array Elements



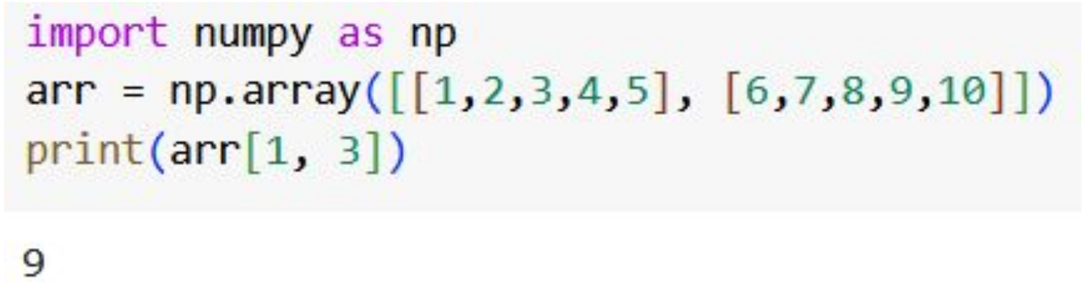
```python
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
```

How to access 5th element of 2nd row?

Access 2D Array Elements

How to access 4th element of 2nd row?



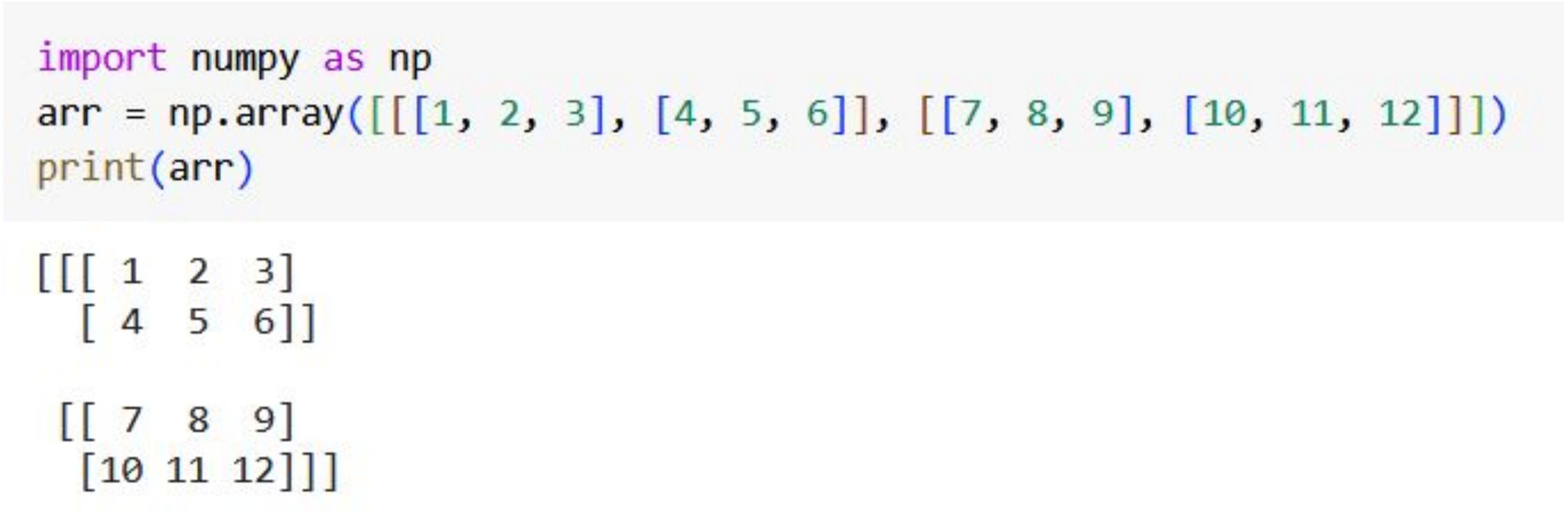
```python
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr[1, 3])
```

```
9
```

Try some example:

- Access 2nd element of 1st row.
- Access 5th element of 2nd row.

Access 3D Array Elements

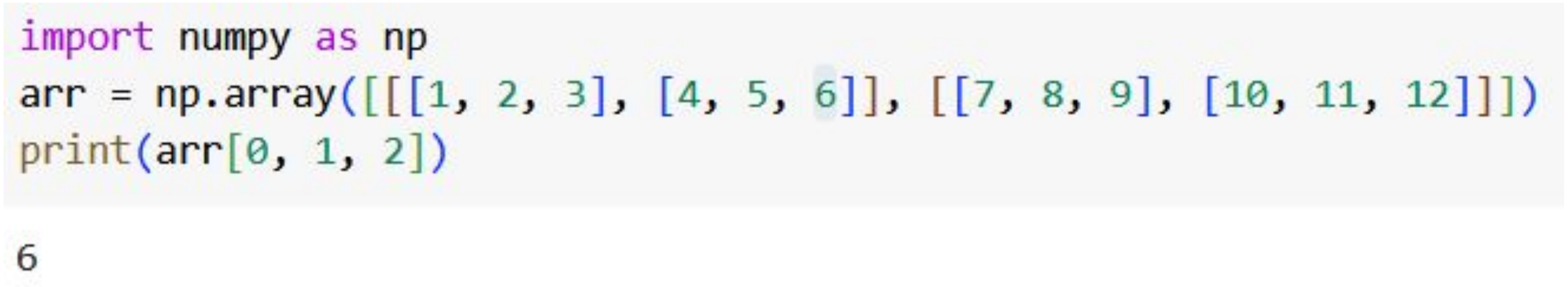


```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr)
```

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

 [[ 7  8  9]
  [10 11 12]]]
```

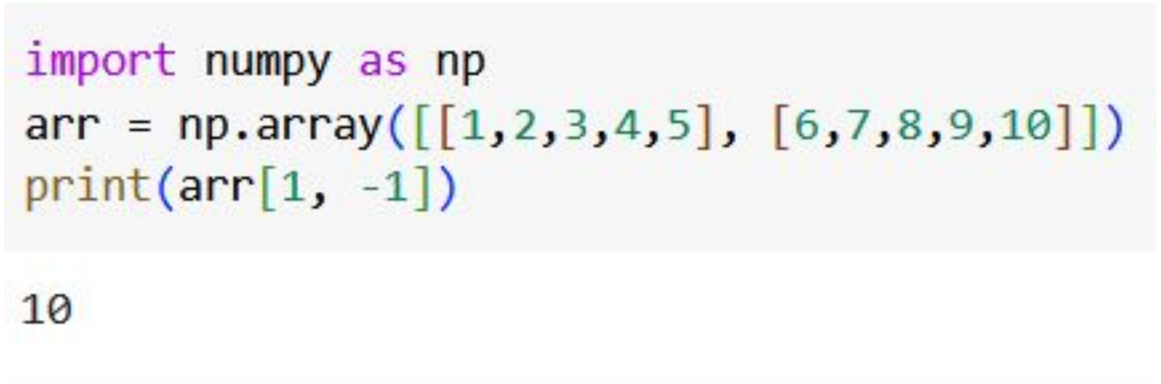
Access 3D Array Elements



```python
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

```
6
```

Negative Indexing

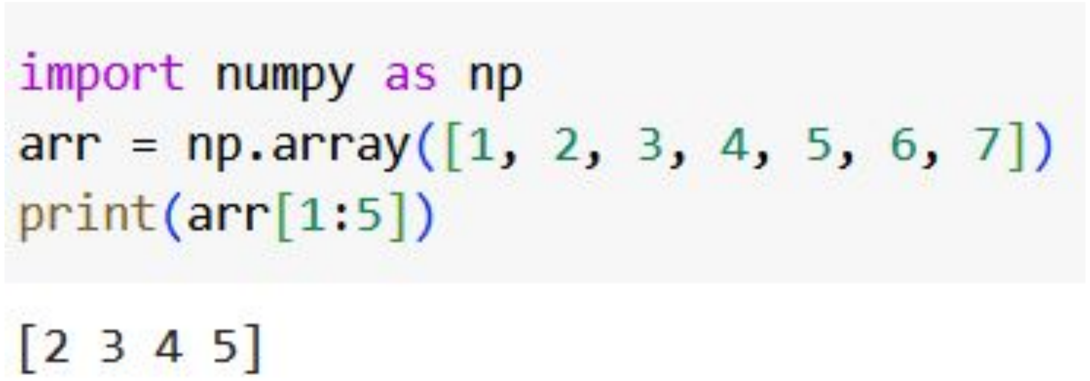


```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr[1, -1])
```

```
10
```

NumPy Array Slicing

Slice elements from index 1 to index 5 from the following array:

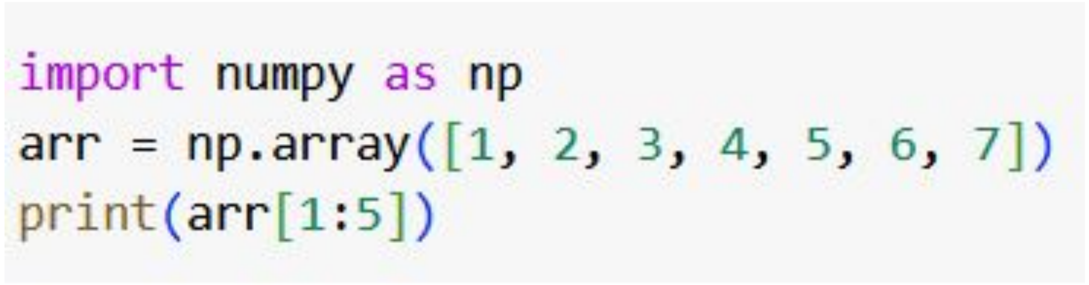


```python
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

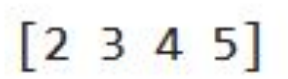
```
[2 3 4 5]
```

NumPy Array Slicing

Slice elements from index 4 to the end of the array:



```python
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```



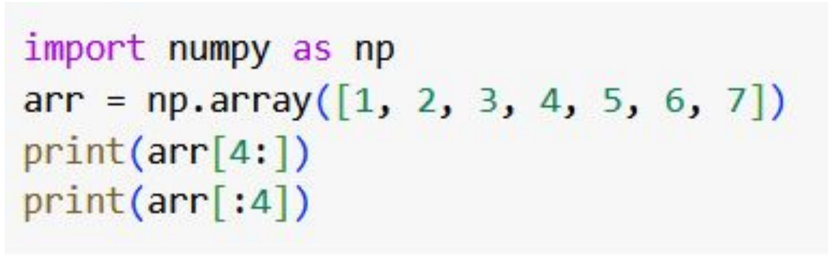
```
[2 3 4 5]
```

We pass slice instead of index like this: [start:end].

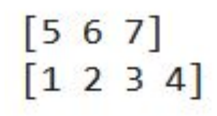
We can also define the step, like this: [start:end:step]

NumPy Array Slicing

Slice elements from index 4 to the end of the array and Slice elements from the beginning to index 4 (not included):



```python
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
print(arr[:4])
```



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

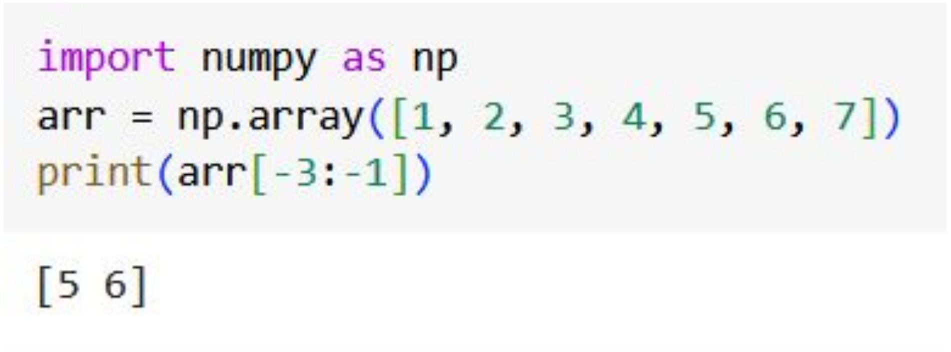
If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

NumPy Array Negative Slicing

Slice from the index 3 from the end to index 1 from the end:

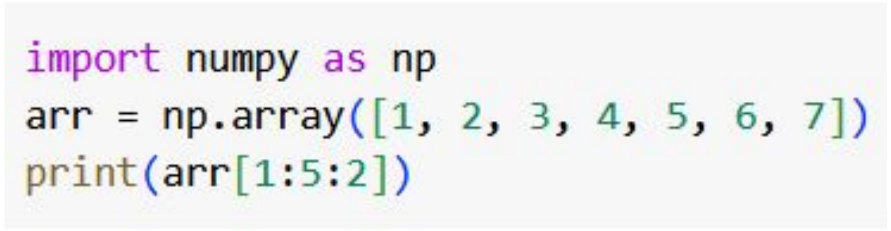


```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

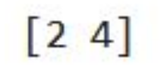
```
[5 6]
```

NumPy Array Slicing with Step

Return every other element from index 1 to index 5:



```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```



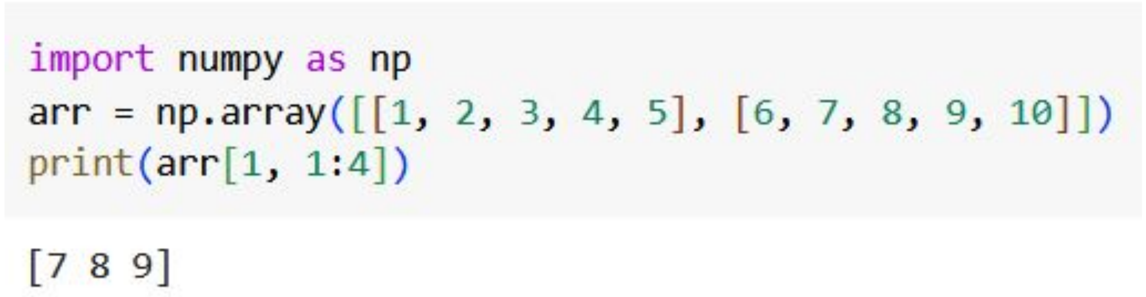
```
[2 4]
```

Try some example:

- Return every other element from the entire array:

NumPy 2D Array Slicing with Step

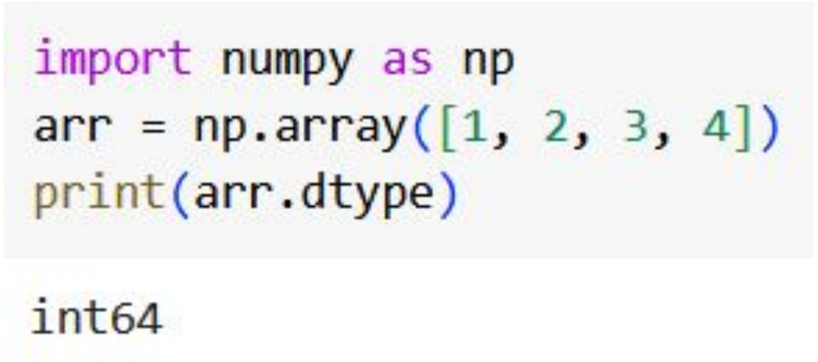
Return every other element from index 1 to index 5:



```python
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

```
[7 8 9]
```

Data Type of an Numpy Array



```python
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

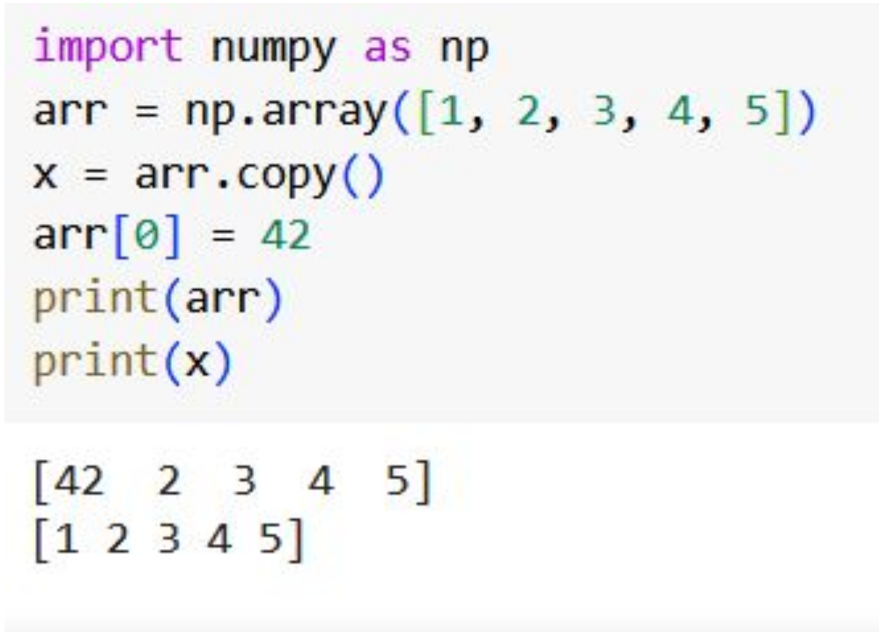
```
int64
```

Data Type of an Numpy Array

Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

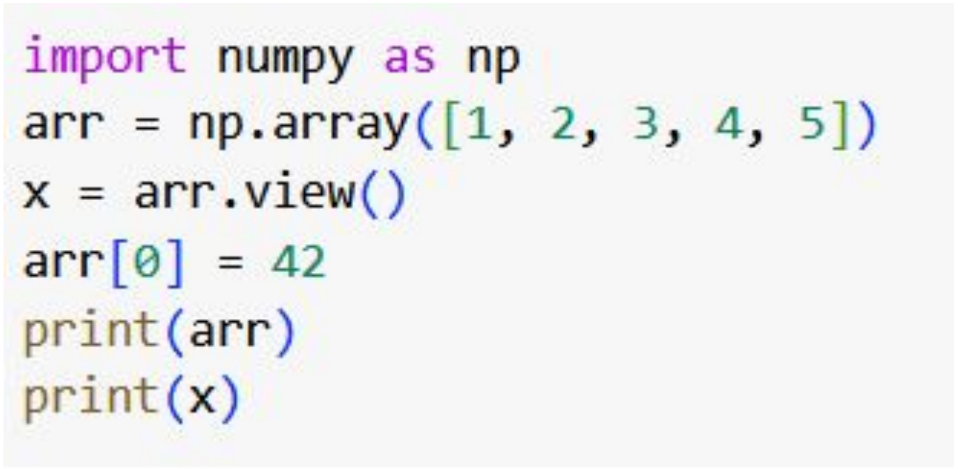
Copy Numpy Array



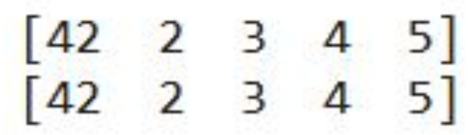
```python
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

View Numpy Array

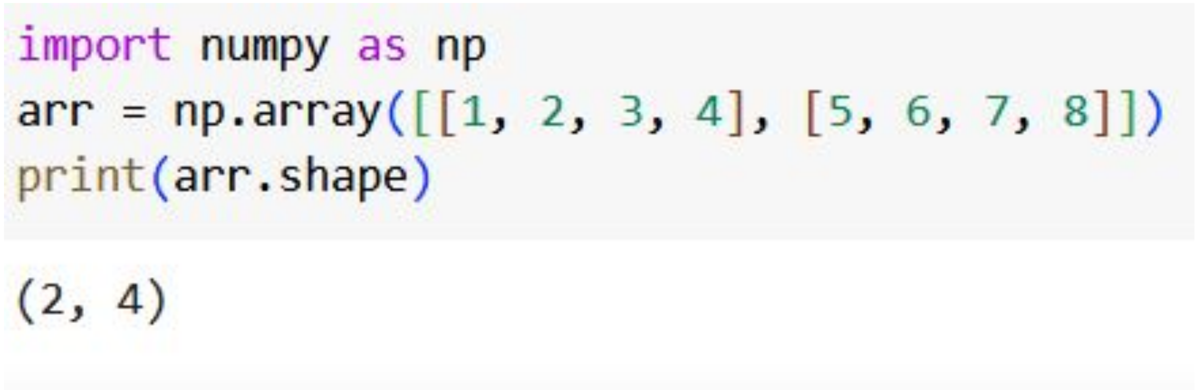


```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
print(arr)
print(x)
```



```
[42  2  3  4  5]
[42  2  3  4  5]
```

Shape Numpy Array

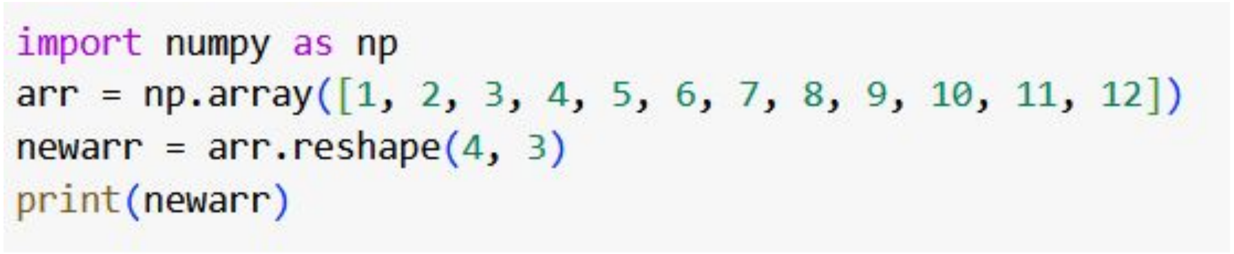


```python
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

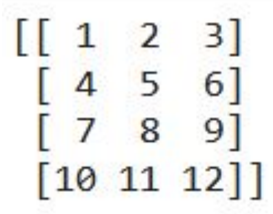
```
(2, 4)
```

Reshape Numpy Array

Convert 1-D array with 12 elements into a 2-D array.



```python
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)
```



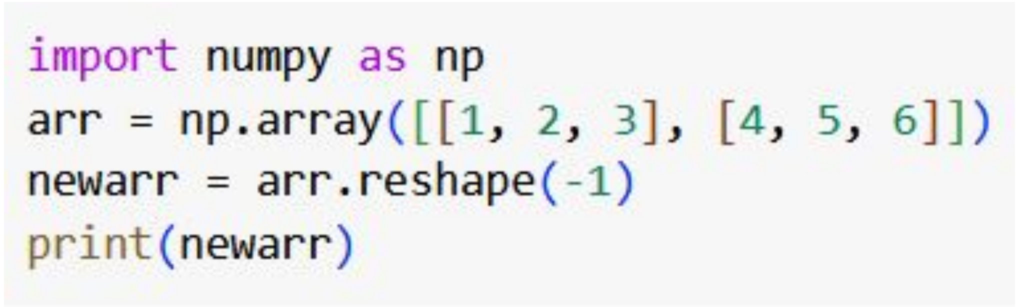
```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Try some example:

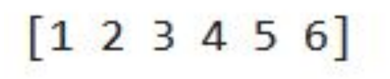
- Convert 1-D array with 12 elements into a 3-D array.

Flattening the arrays

Flattening array means converting a multidimensional array into a 1D array.

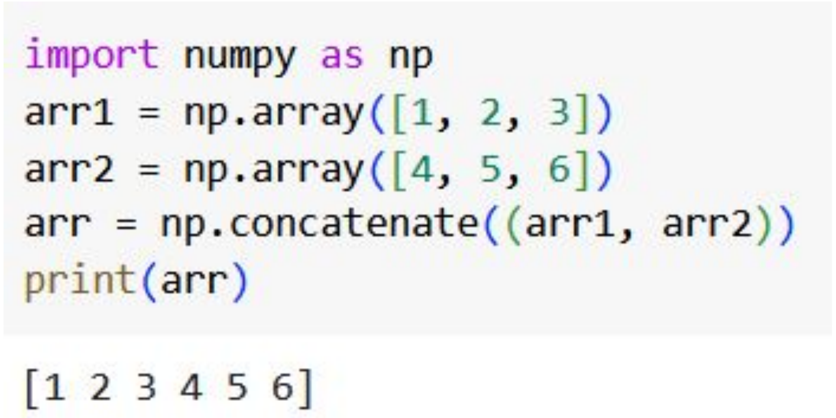


```python
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = arr.reshape(-1)
print(newarr)
```



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

Joining NumPy Arrays



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

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