NUMPY

WHAT IS NUMPY?

NumPy is the fundamental package for scientific computing in Python that provides a multidimensional array object.

NumPy stands for Numerical Python.

Lists can serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than lists

NumPy includes functions in domain of linear algebra, Fourier transform, and matrices.

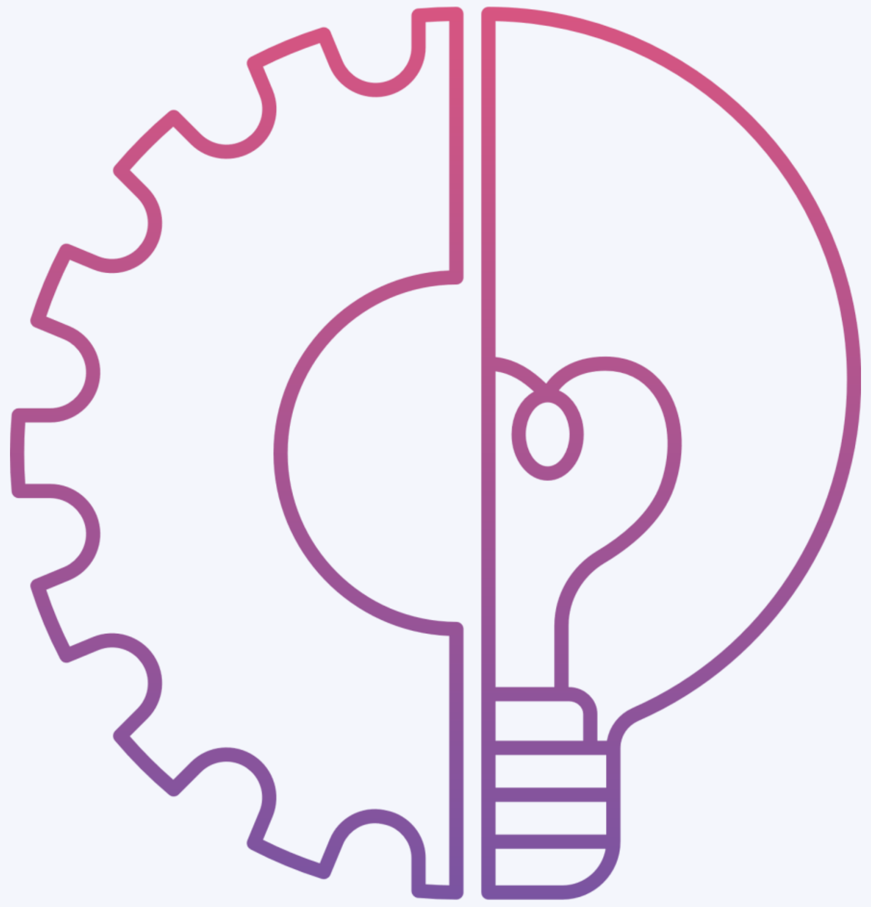
NumPy Features

01 Fast and versatile

02 NumPy supports a wide range of hardware and computing platforms

03 NumPy offers comprehensive mathematical functions, random number generators

04 Easy to use

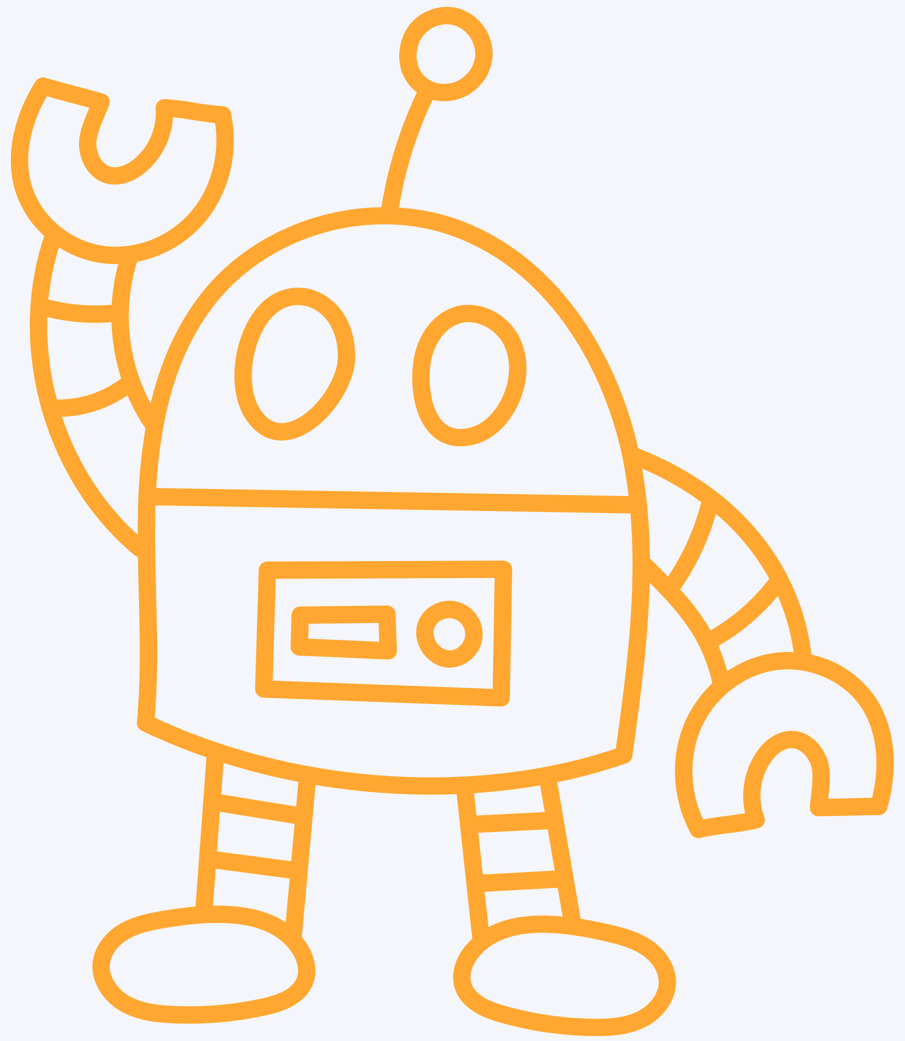


INSTALLATION OF NUMPY

- pip install numpy

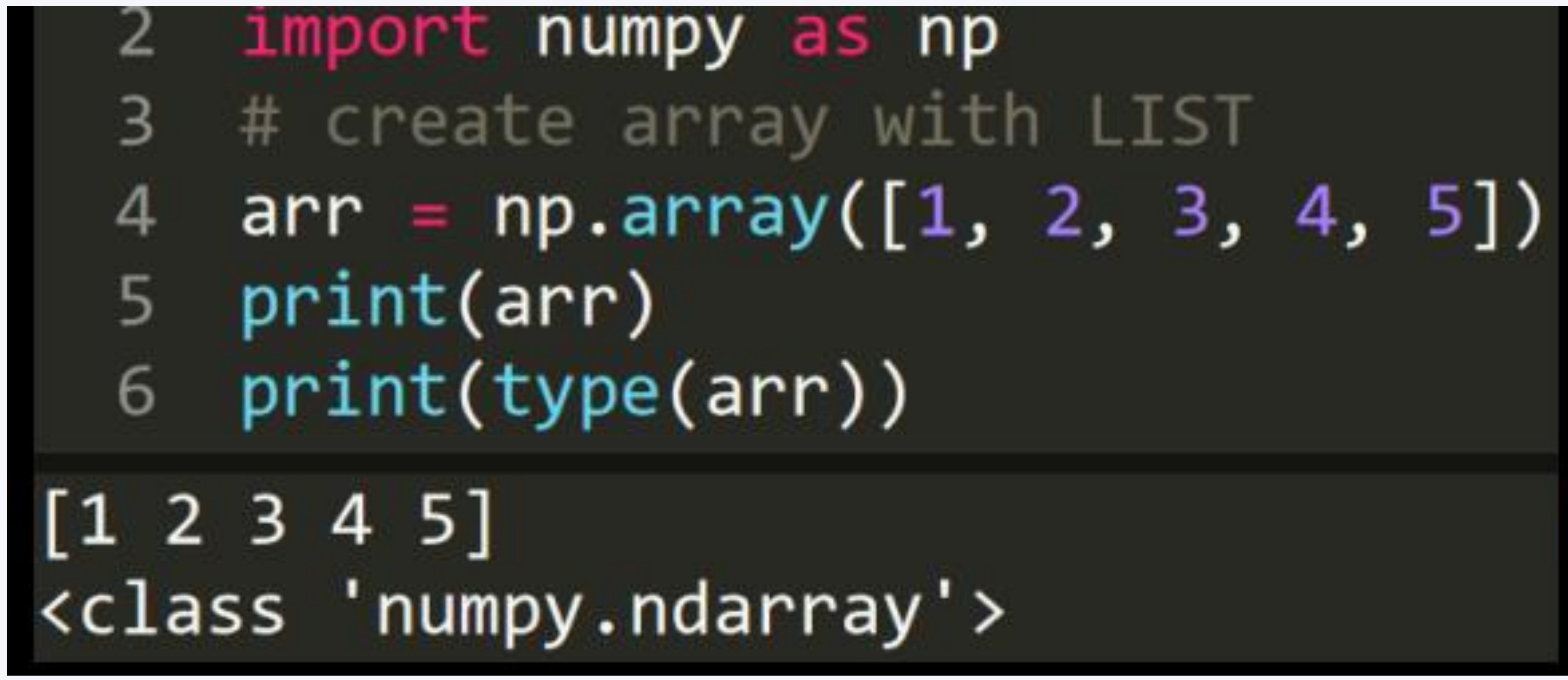
IMPORT NUMPY

- import numpy as np



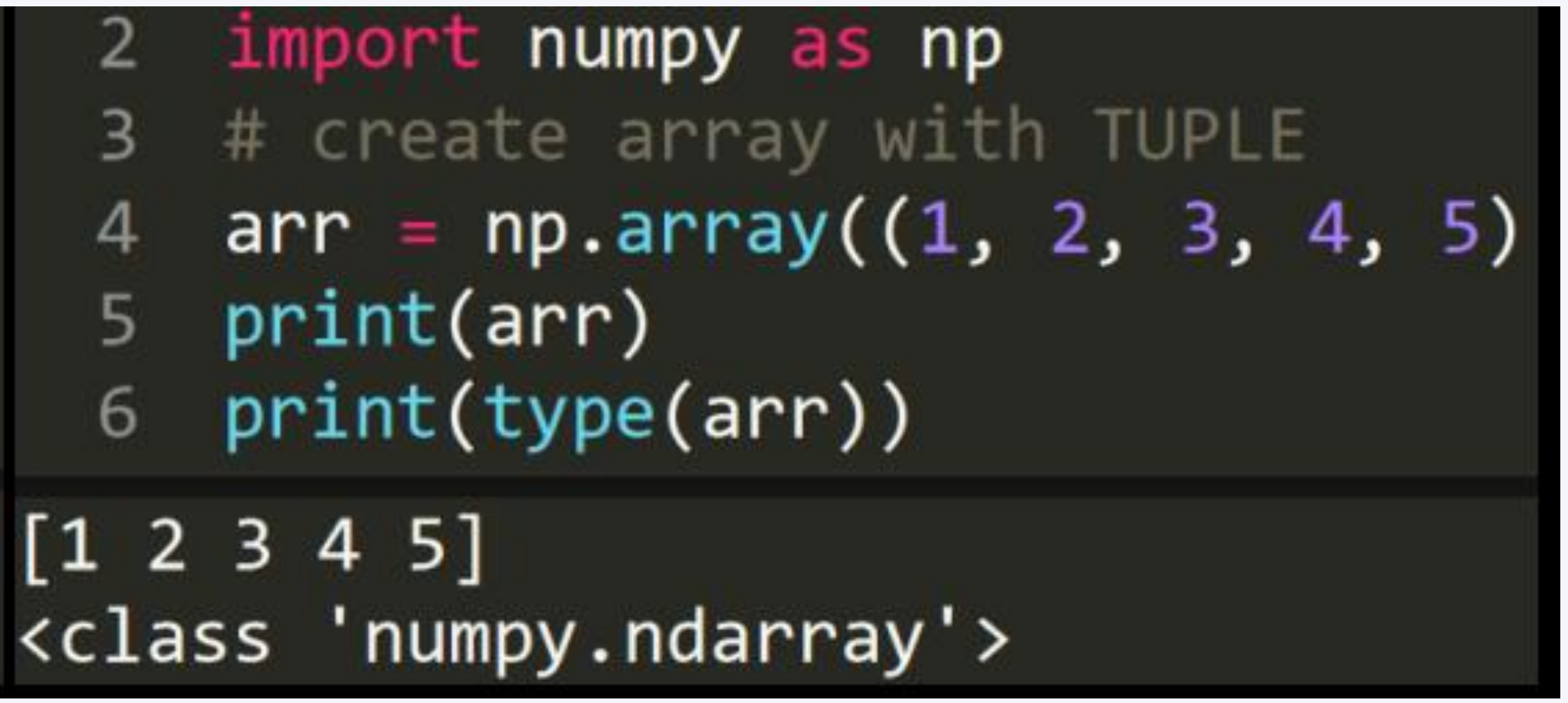
NUMPY CREATING ARRAYS

- The array object in NumPy is called ndarray.
- We can create a NumPy ndarray object by using the array() function.



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

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

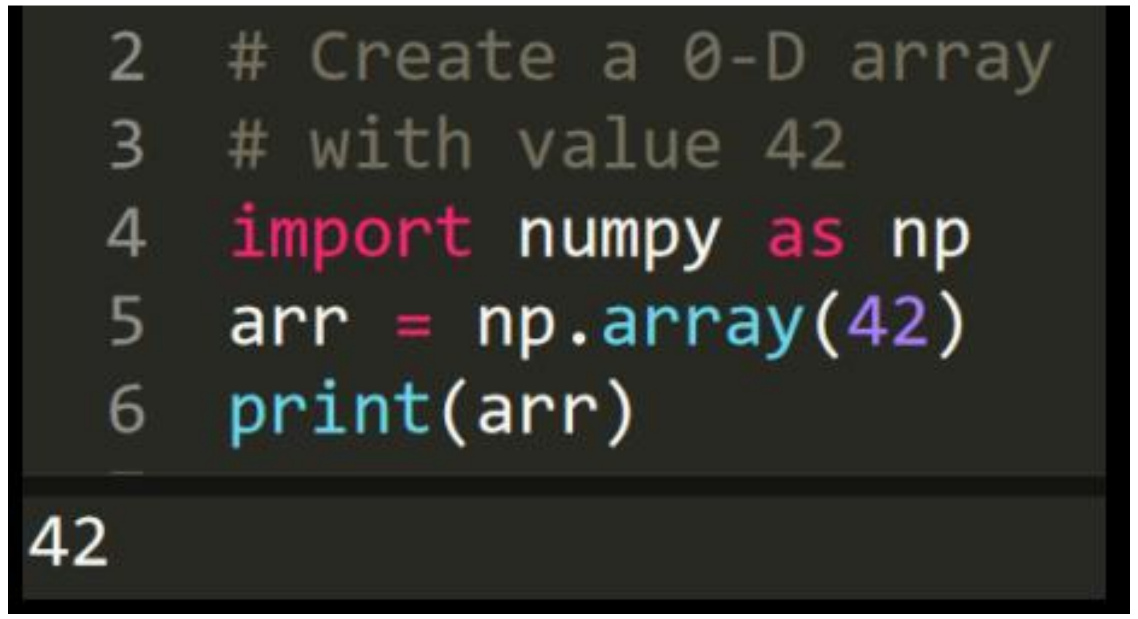


```
import numpy as np
# create array with TUPLE
arr = np.array((1, 2, 3, 4, 5)
print(arr)
print(type(arr))
```

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

NUMPY CREATING ARRAYS

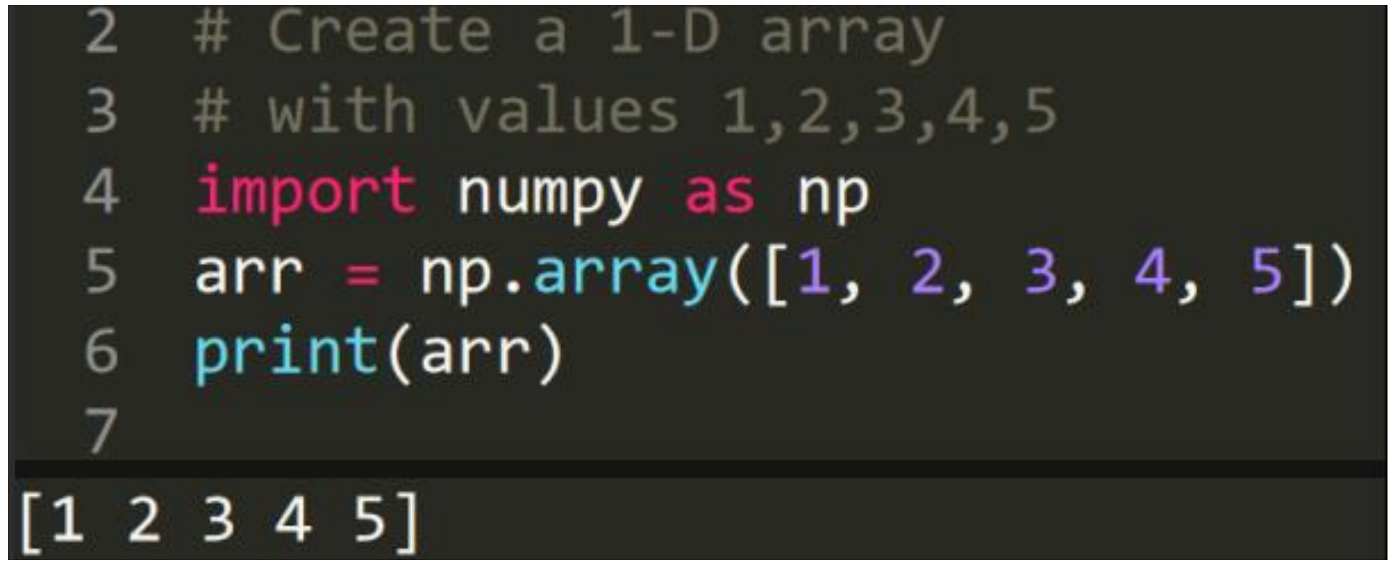
0-D ARRAYS



```
# Create a 0-D array
# with value 42
import numpy as np
arr = np.array(42)
print(arr)
```

```
42
```

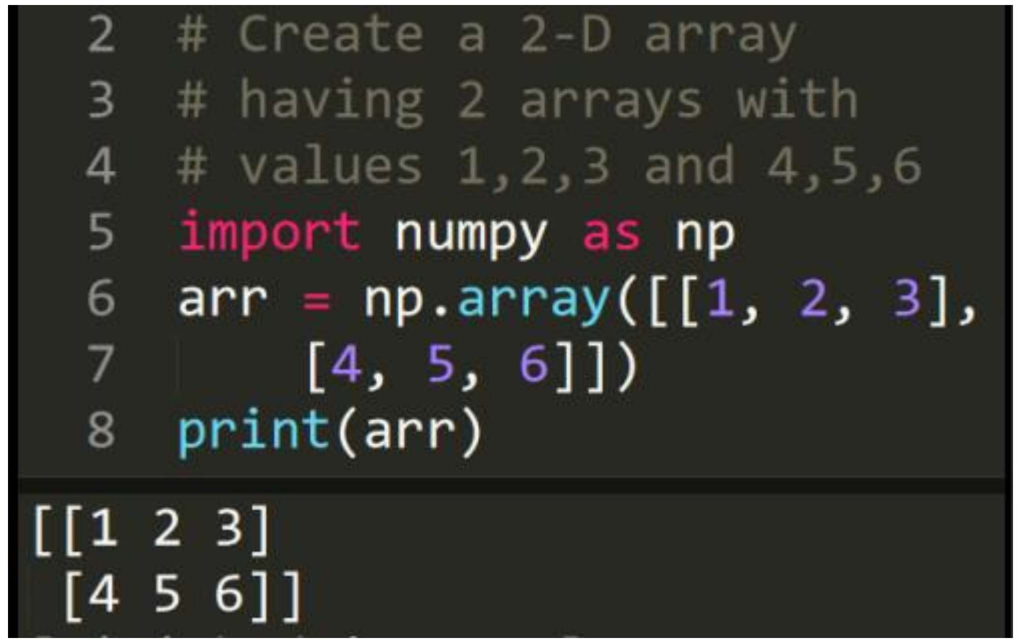
1-D ARRAYS



```
# Create a 1-D array
# with values 1,2,3,4,5
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)

```

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

2-D ARRAYS

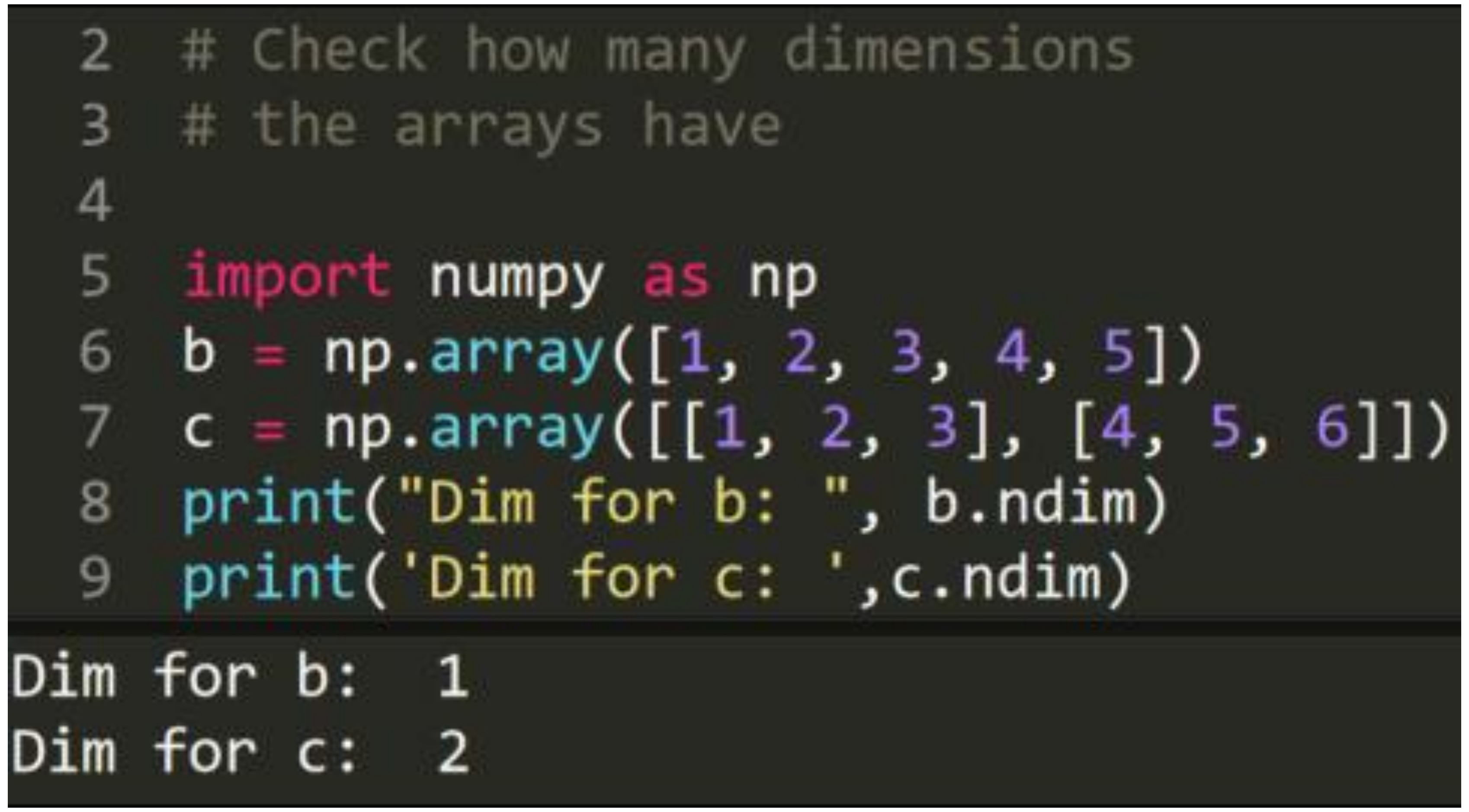


```
# Create a 2-D array
# having 2 arrays with
# values 1,2,3 and 4,5,6
import numpy as np
arr = np.array([[1, 2, 3],
    [4, 5, 6]])
print(arr)
```

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

CHECK NUMBER OF DIMENSIONS?

NumPy Arrays provides the **ndim** attribute that returns an integer that tells us how many dimensions the array have.

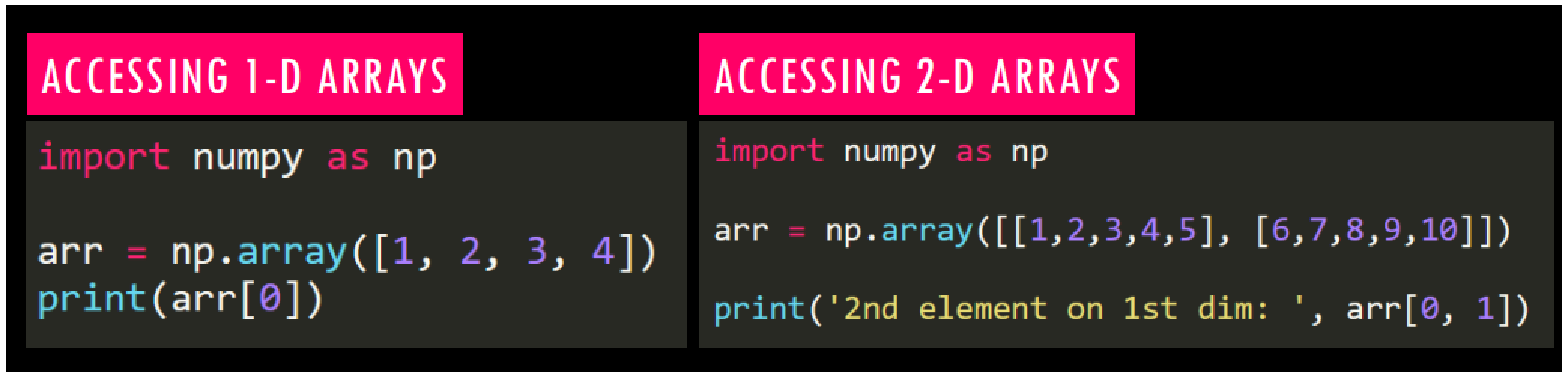


```python
# Check how many dimensions
# the arrays have

import numpy as np
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
print("Dim for b: ", b.ndim)
print('Dim for c: ',c.ndim)
```

```
Dim for b:  1
Dim for c:  2
```

ACCESSING NUMPY ARRAY VIA INDEXING

Array indexing is the same as LIST in Python, it have positive indexing and also negative indexing. • You can access an array element by referring to its index number (either positive or negative

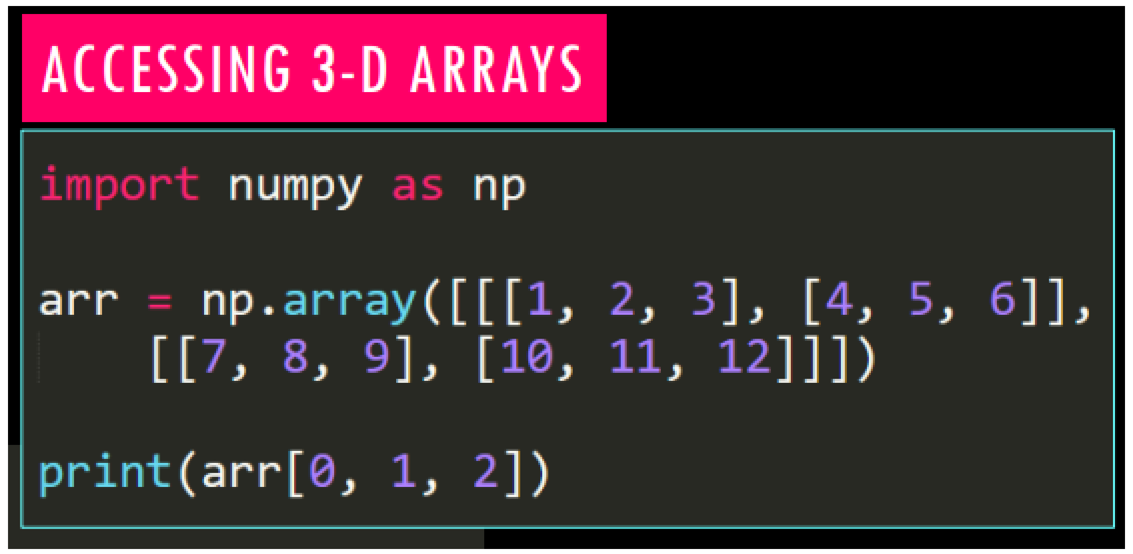


ACCESSING 1-D ARRAYS

```python
import numpy as np

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

ACCESSING 2-D ARRAYS

```python
import numpy as np

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

print('2nd element on 1st dim: ', arr[0, 1])
```

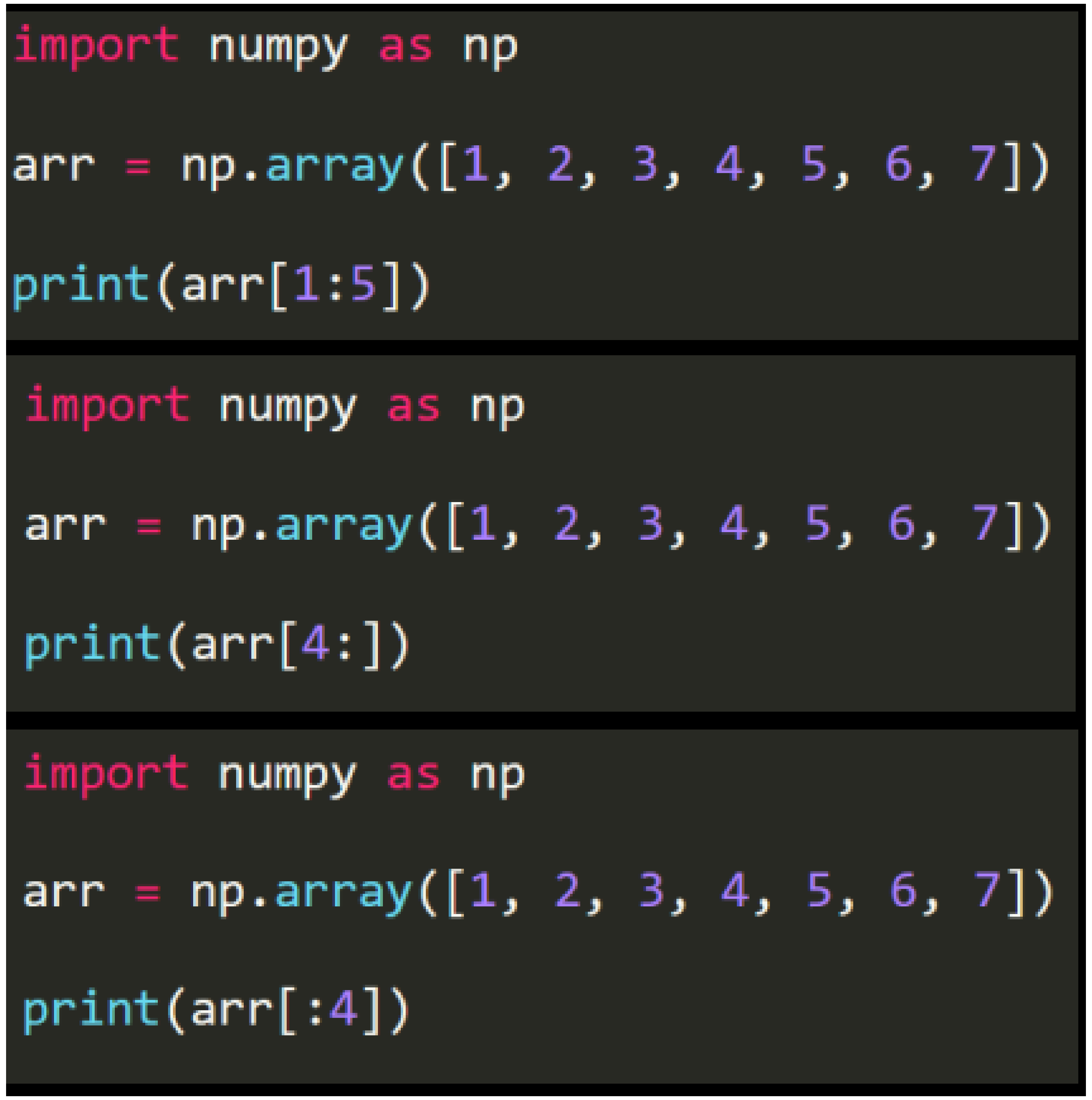


ACCESSING 3-D ARRAYS

```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])
```

NUMPY ARRAY SLICING

1-D ARRAYS



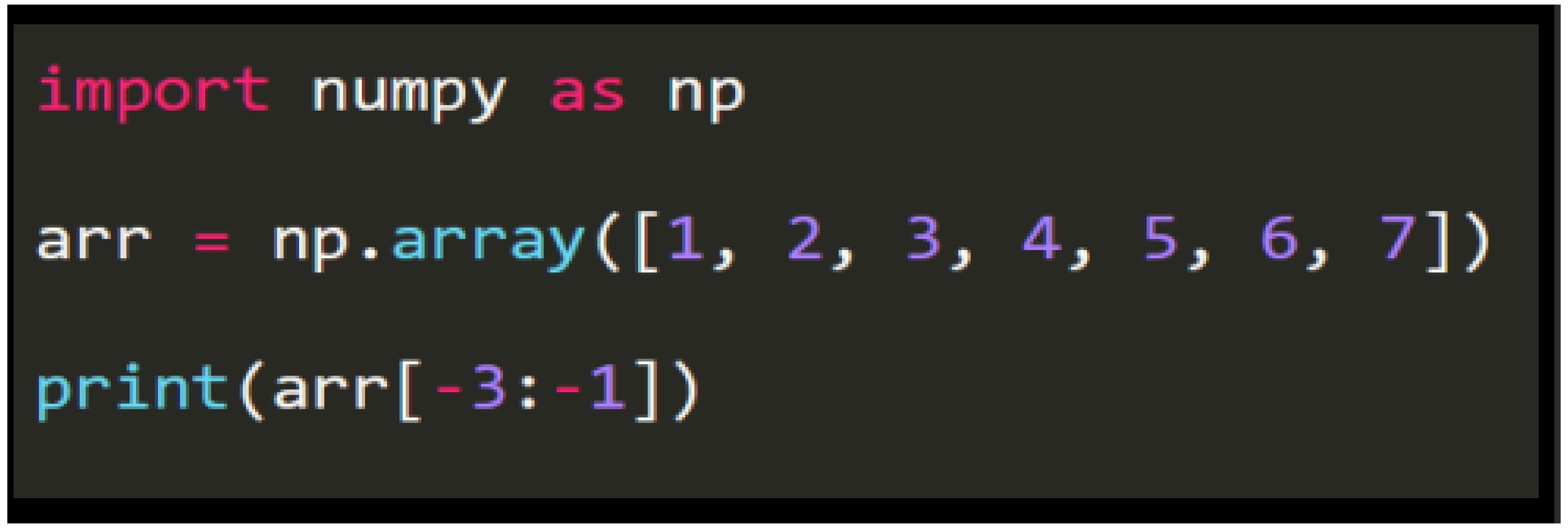
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])
```

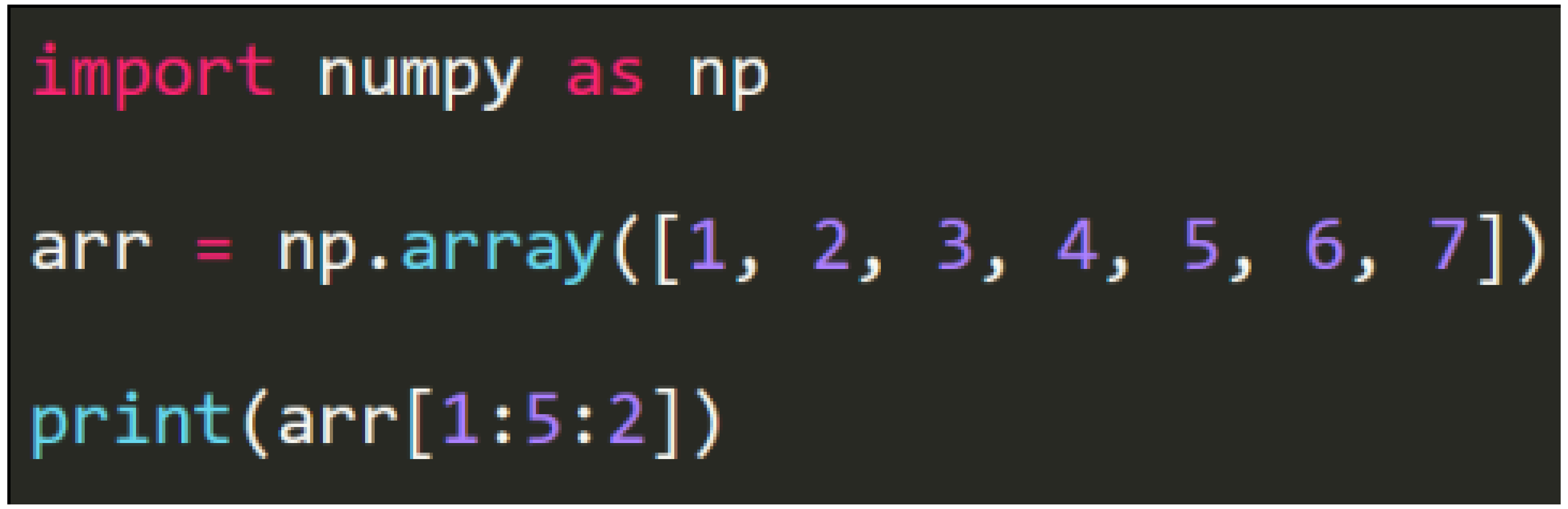
NUMPY ARRAY SLICING

NEGATIVE SLICING



```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```

STEP IN SLICING



```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5:2])
```

NUMPY ARRAY COPY VS VIEW

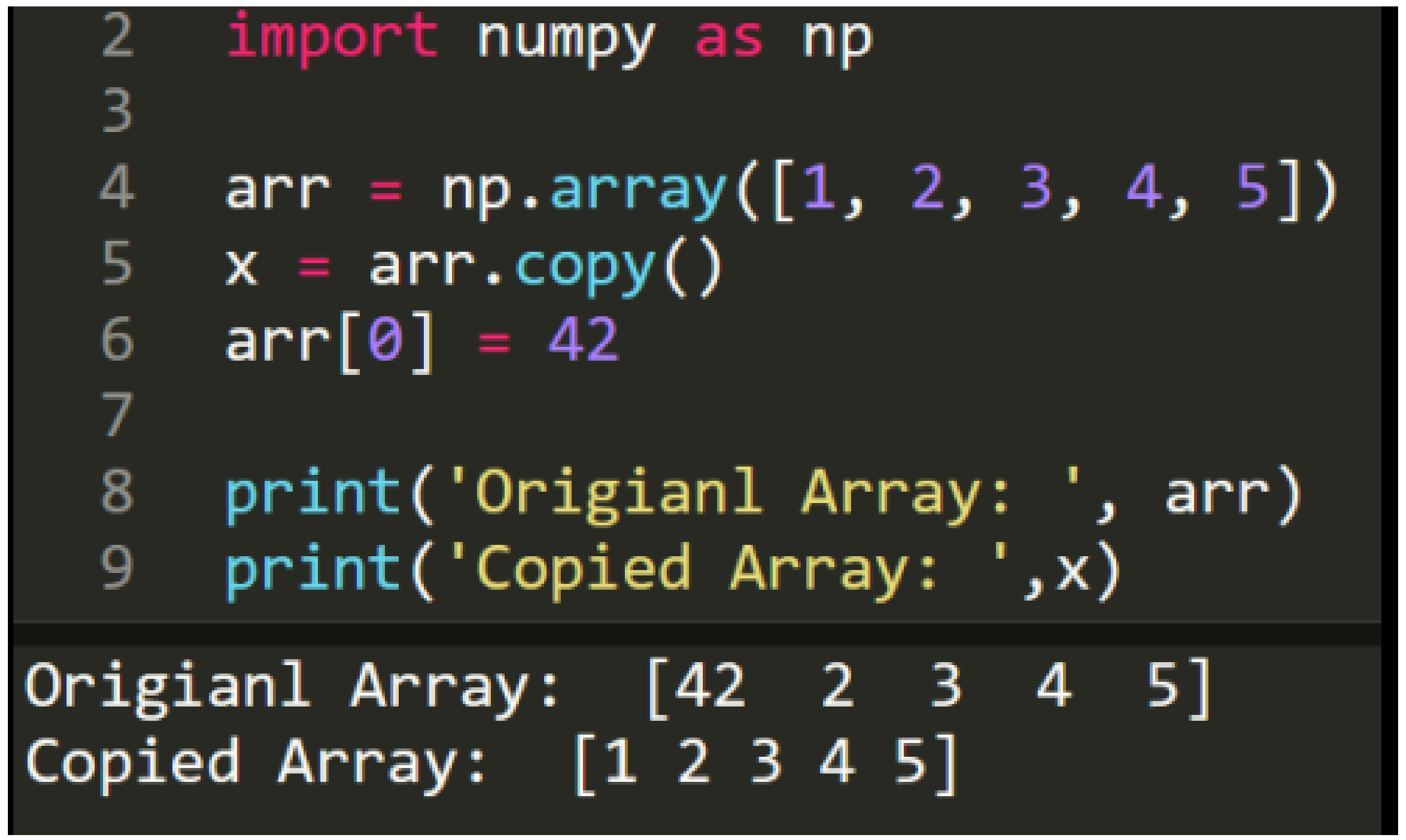
THE DIFFERENCE BETWEEN COPY AND VIEW

- **Copy** create new array from existing
- Any changes made to the copy will not affect original array, and
- Any changes made to the original array will not affect the copy

- **View** is another pointer to array
- Any changes made to the view will affect the original array, and
- Any changes made to the original array will affect the vie

COPY

- Make a copy, change the original array, and display both arrays
- The copy SHOULD NOT be affected by the changes made to the original array

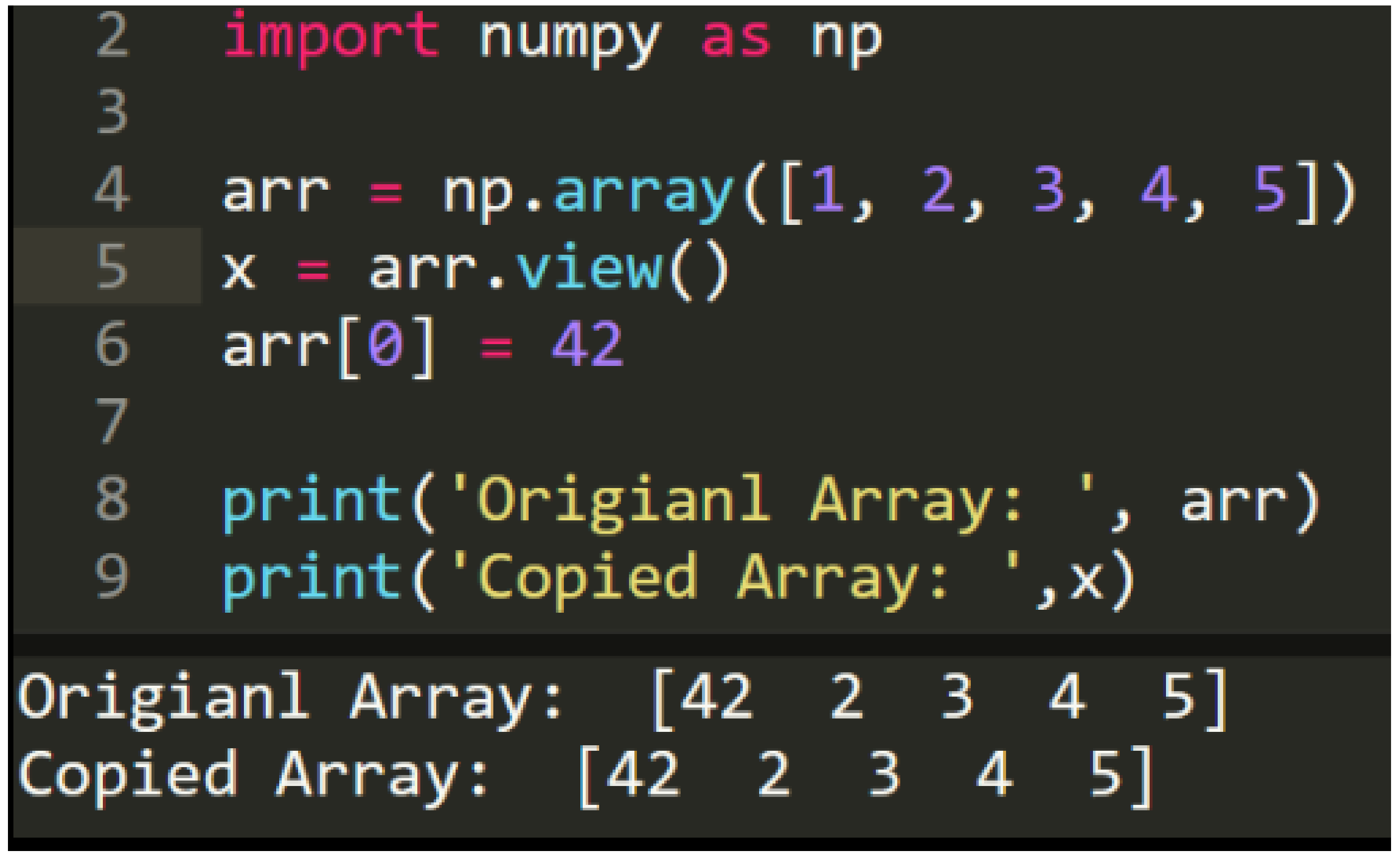


```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print('Origianl Array: ', arr)
print('Copied Array: ',x)
```

```
Origianl Array:  [42  2  3  4  5]
Copied Array:  [1 2 3 4 5]
```

VIEW

- Make a view, change the original array, and display both arrays
- The view SHOULD be affected by the changes made to the original array

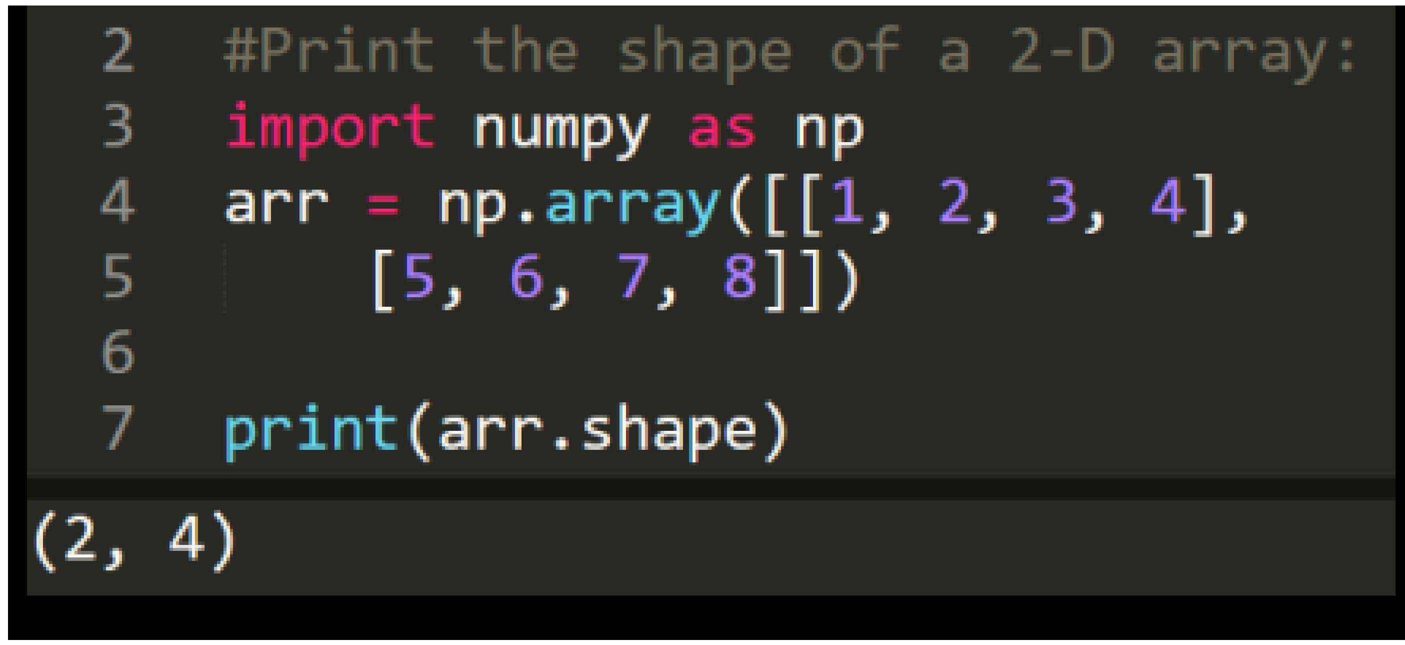


```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42

print('Origianl Array: ', arr)
print('Copied Array: ',x)
```

```
Origianl Array:  [42  2  3  4  5]
Copied Array:  [42  2  3  4  5]
```

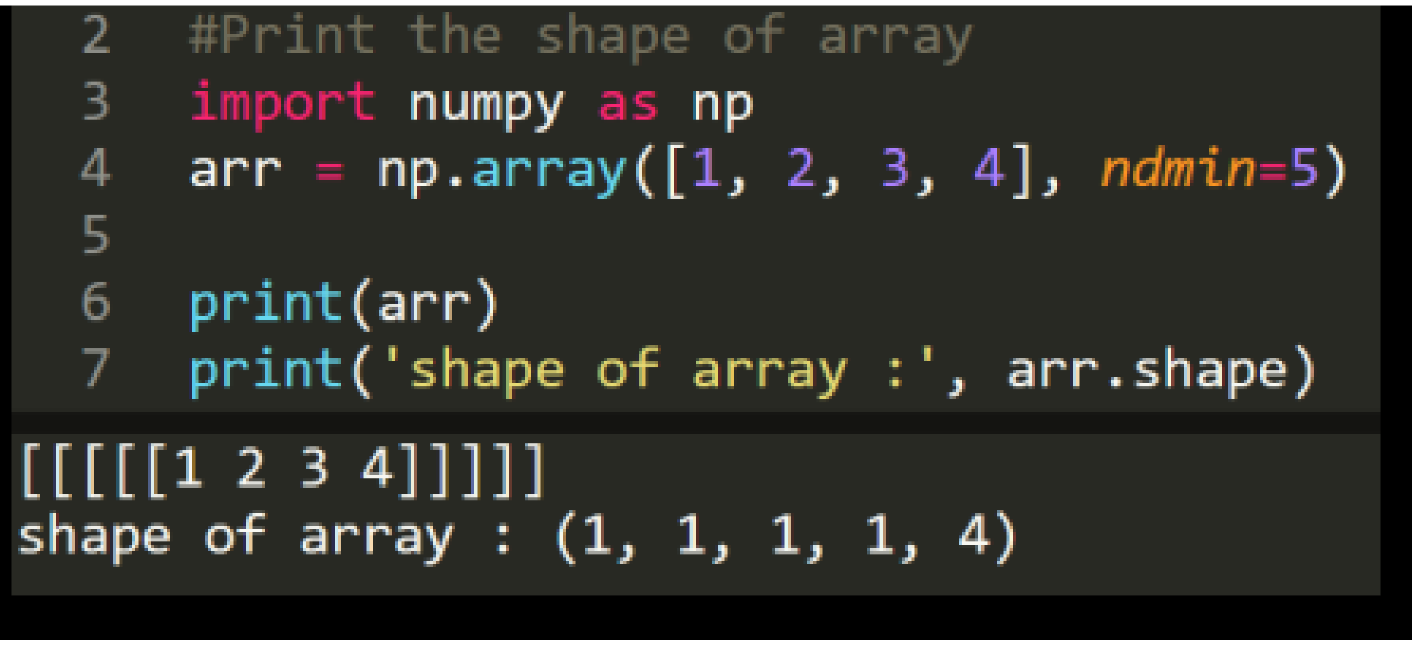
NUMPY ARRAY **SHAPE**

Shape that returns a tuple with each index having the number of corresponding elements, (return shape of the array



```
#Print the shape of a 2-D array:
import numpy as np
arr = np.array([[1, 2, 3, 4],
      [5, 6, 7, 8]])

print(arr.shape)
```

```
(2, 4)
```



```
#Print the shape of array
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
print('shape of array :', arr.shape)
```

```
[[[[[1 2 3 4]]]]]
shape of array : (1, 1, 1, 1, 4)
```

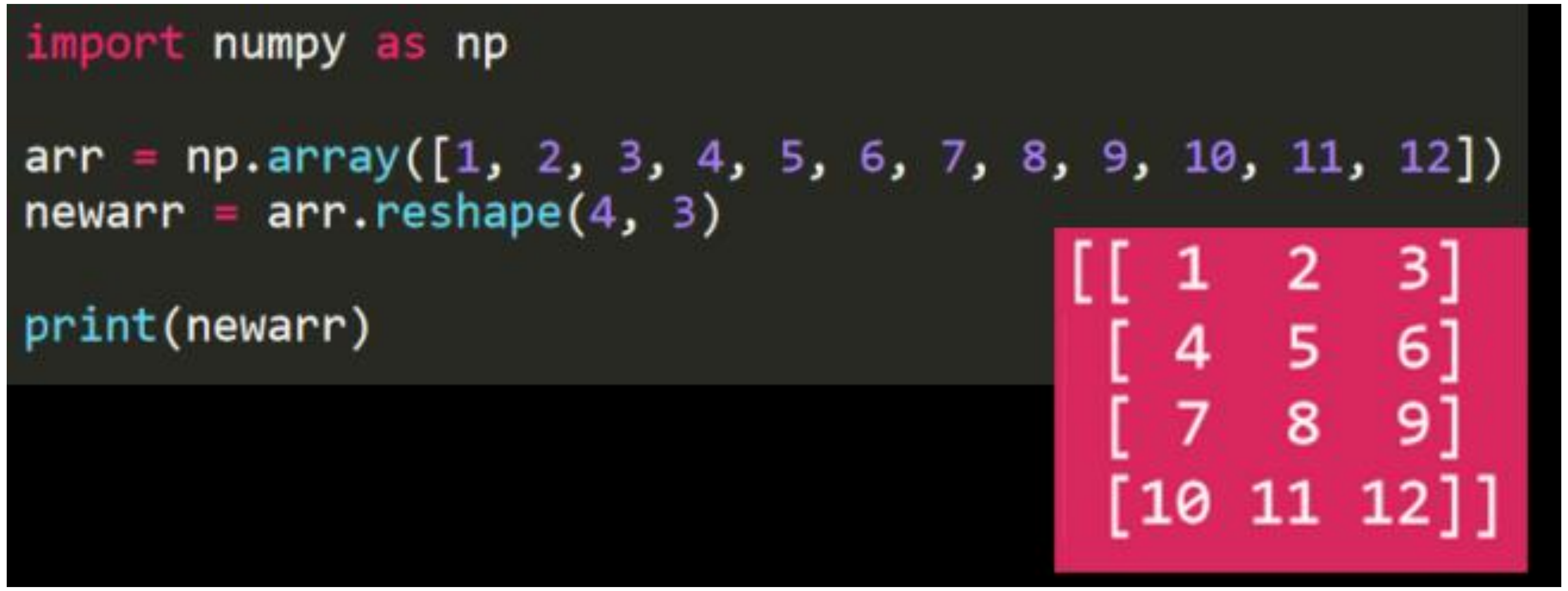
NUMPY ARRAY **RESHAPE**

- ❑ Reshaping means changing the shape of an array.
- ❑ The shape of an array is the number of elements in each dimension.
- ❑ By reshaping we can add or remove dimensions or change number of elements in each dimension

RESHAPE FROM 1-D TO 2-D EXAMPLE

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

  The outermost dimension will have 4 arrays, each with 3 elements:



```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)
```

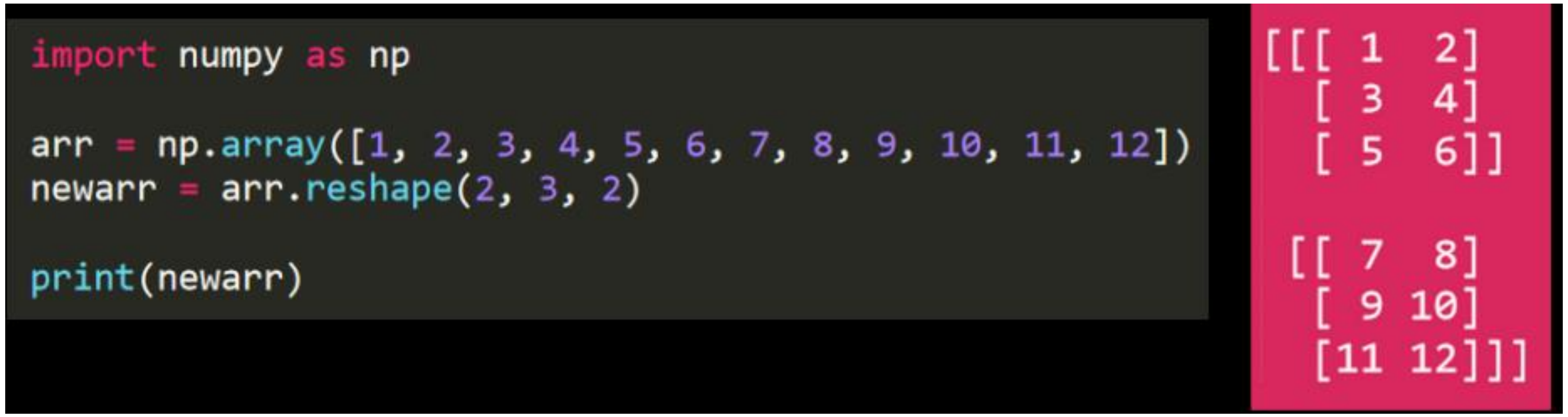
```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

RESHAPE FROM 1-D TO 3-D EXAMPLE

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

  The outermost dimension will have 2 arrays that contains 3 arrays,

  each with 2 elements:

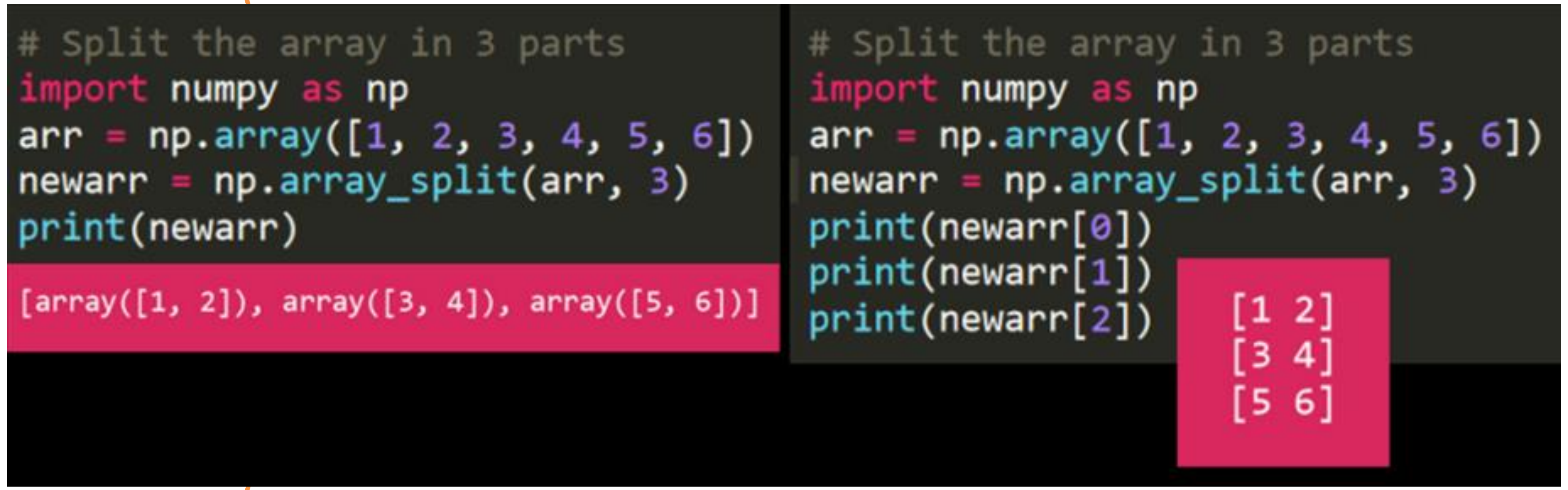


```python
import numpy as np

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

print(newarr)
```

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

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

NUMPY **SPLITTING** ARRAY

- ❑ Splitting is reverse operation of Joining, i.e. Joining merges multiple arrays into one and Splitting breaks one array into multiple.
- ❑ We use array_split() for splitting arrays
- ❑ we pass it the array we want to split and the number of splits.



```
# Split the array in 3 parts
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
```

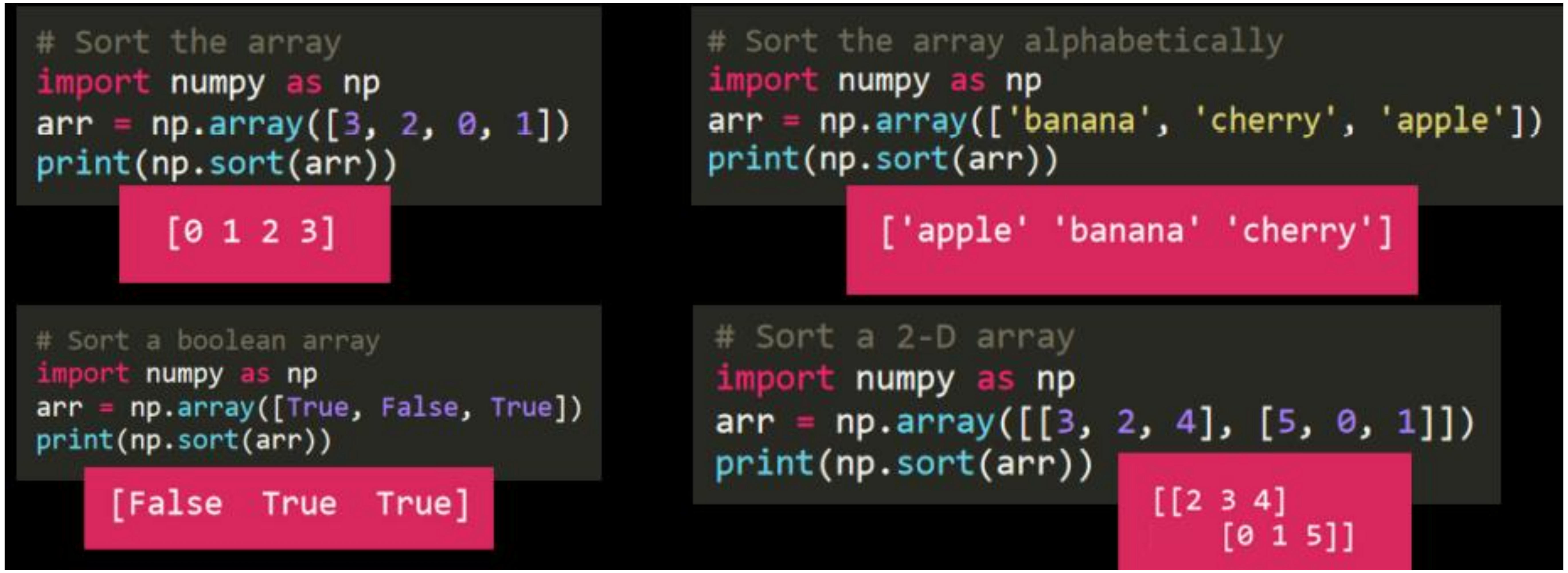
```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

```
# Split the array in 3 parts
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr[0])
print(newarr[1])
print(newarr[2])
```

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

NUMPY **SORTING** ARRAY

The NumPy ndarray object has a function called sort(), that will sort a specified array



```
# Sort the array
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

```
[0 1 2 3]
```

```
# Sort the array alphabetically
import numpy as np
arr = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr))
```

```
['apple' 'banana' 'cherry']
```

```
# Sort a boolean array
import numpy as np
arr = np.array([True, False, True])
print(np.sort(arr))
```

```
[False  True  True]
```

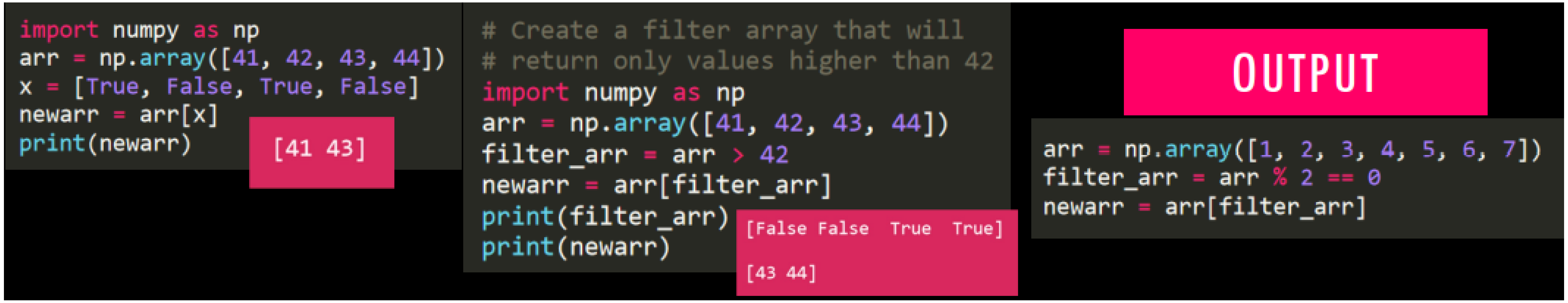
```
# Sort a 2-D array
import numpy as np
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
```

```
[[2 3 4]
 [0 1 5]]
```

NUMPY **FILTER** ARRAY

- ❑ Getting some elements out of an existing array and creating a new array out of them is called filtering.
- ❑ In NumPy, you filter an array using a boolean index list.
- ❑ A boolean index list is a list of booleans corresponding to indexes in the array.
- ❑ If index is True that element is contained in the filtered array
- ❑ If index is False that element is excluded from the filtered array.

NUMPY **FILTER** ARRAY - Example



```
import numpy as np
arr = np.array([41, 42, 43, 44])
x = [True, False, True, False]
newarr = arr[x]
print(newarr)
```

```
[41 43]
```

```
# Create a filter array that will
# return only values higher than 42
import numpy as np
arr = np.array([41, 42, 43, 44])
filter_arr = arr > 42
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

```
[False False  True  True]

[43 44]
```

OUTPUT

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
filter_arr = arr % 2 == 0
newarr = arr[filter_arr]
```

Thank You