Chapter 2: NumPy Arrays

Prepared by: Hanan Hardan

Arrays in Python

- Python does not have built-in support for arrays, but Python Lists can be used instead.
- To work with arrays in Python you will have to import a library, NumPy
- NumPy is short for "Numerical Python".

Why use NumPy?

- In Python we have lists that 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 traditional Python lists.
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.

* W3Schools.com

Installation of NumPy

- If NumPy is not already used, you have to install it using pip command that will be discussed later.
- If you are using editors such as anaconda, spyder, …etc, they already have NumPy installed in them.
- To use library NumPy and its functions, you have to import it in your program.

```
import numpy
```

Use of NumPy to Define a simple one-dimensional array

- Now, after importing NumPy library you can use it in the program.
- To define a simple array using NumPy:

```
import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```

- array here is a function that is included in NumPy library and cannot be used without importing the library first. Note how brackets () are used after it, indicating that it is a function. Inside the function, an array of elements using the ordinary array brackets [] is defined. Now, arr is the array that we can use in our program.

Use of NumPy to Define a simple one-dimensional array

- Also, note that we cannot call array function without starting with the name of the library numpy.

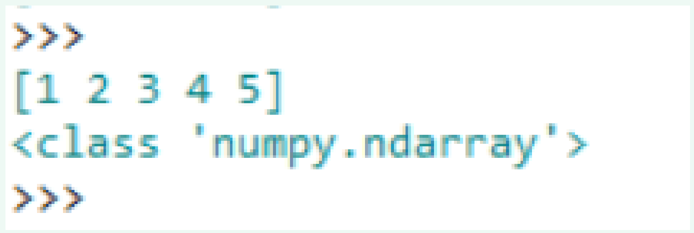
- Users of Python usually give alias (a special name) for the NumPy library when importing it, and thus they use this alias instead of the original library name.

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

Creating Arrays with NumPy

- NumPy is used to work with arrays. The array object in NumPy is called ndarray.
- Check the following example:

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



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

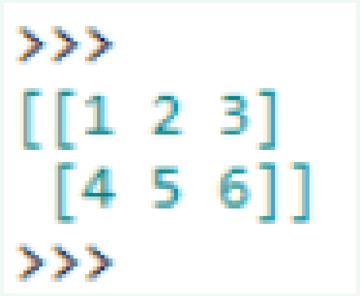
- The first print command will print the array directly, since it has a special data type as one unit.
- The second print command will display the special data type of the NumPy array which is ndarray.

Two-dimensional Arrays

- In Python, when you need to create a two-dimensional array, it is called Nested arrays, which are arrays that include arrays as their elements. But, of course you can deal with it as the ordinary two-dimensional arrays in C++, yet easier functions can be used in Python.

- To create two-dimensional array in Python use the following code:

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



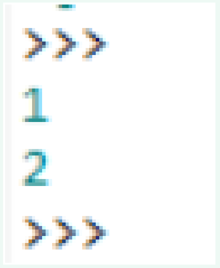
```
>>>
[[1 2 3]
 [4 5 6]]
>>>
```

Multi-dimensional Arrays

- In Python, you can define Multi-dimensional Arrays, more than 2D. It is not required in this course, but if you need to check how many dimensions your array have, you can use the funciton ndim.

```
import numpy as np
a = np.array([1, 2, 3, 4, 5])
b = np.array([[1, 2, 3], [4, 5, 6]])

print(a.ndim)
print(b.ndim)
```

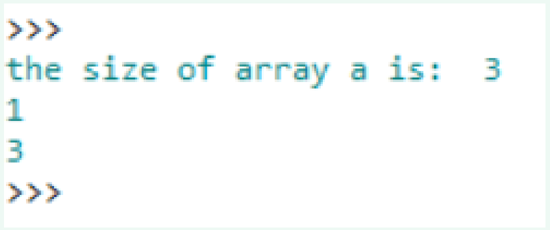


```
>>>
1
2
>>>
```

Array Indexing (Accessing)

- To access an array element, you need to use an index to specify its position. Array indices start from 0 to array size-1

```
import numpy as np
a = np.array([1,2,3])
print ('the size of array a is: ', a.size)
print (a[0])
print (a[a.size-1])
```



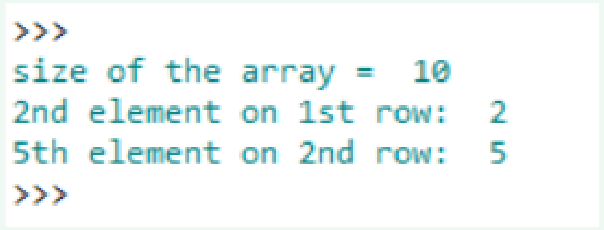
```
>>>
the size of array a is:  3
1
3
>>>
```

Array Indexing (Accessing)

- To access a two-dimensional array, you need to specify an index of the row and an index of the column

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

print ('size of the array = ',arr.size)
print('2nd element on 1st row: ', arr[0, 1])
print('5th element on 1st row: ', arr[0, 4])
```



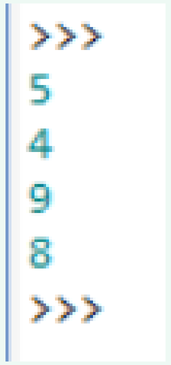
```
>>>
size of the array =  10
2nd element on 1st row:   2
5th element on 2nd row:   5
>>>
```

Negative Indexing

- In Python, an array accepts a negative value as an index, indicating the position from the end of the array

```
import numpy as np
arr1 = np.array ([1,2,3,4,5])
arr2 = np.array([[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15]])

print(arr1[-1])
print(arr1[-2])
print (arr2[1,-2])
print (arr2[-2,-3])
```



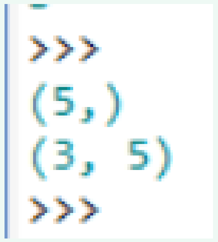
```
>>>
5
4
9
8
>>>
```

Get the shape of the array

- You can retrieve the dimensions of certain array using an attribute named shape.

```
import numpy as np
arr1 = np.array ([1,2,3,4,5])
arr2 = np.array([[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15]])

print(arr1.shape)
print (arr2.shape)
```

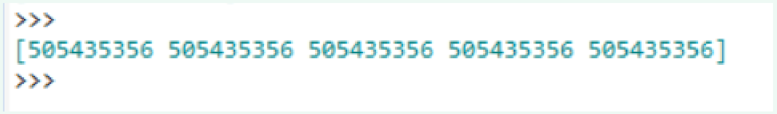


```
>>>
(5,)
(3, 5)
>>>
```

Define an Empty Array

- Sometimes you need to fill an array with user input, so you can define an empty array, with a certain shape and elements data type.

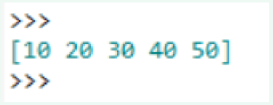
```
import numpy as np
arr = np.empty(5, dtype=int)
print (arr)
```



```
>>>
[505435356 505435356 505435356 505435356 505435356]
>>>
```

```
import numpy as np
arr = np.empty(5, dtype=int)
for i in range(5):
    arr[i] = int(input())

print (arr)
```

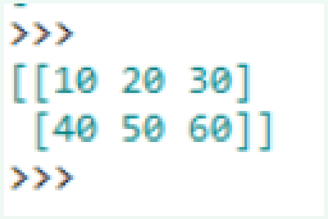


```
>>>
[10 20 30 40 50]
>>>
```

Define an Empty Array

- To create an empty two-dimensional array:

```
import numpy as np
arr = np.empty([2,3], dtype=int)
for row in range(2):
    for col in range (3):
        arr[row, col] = int(input ())
print(arr)
```

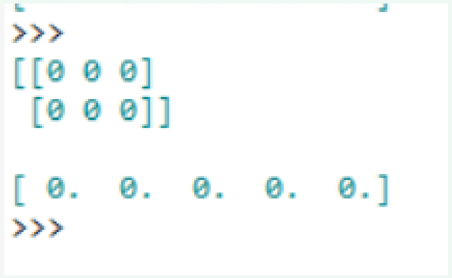


```
>>>
[[10 20 30]
 [40 50 60]]
>>>
```

Define an array filled with zeros or ones

- You can create an array and initialize it all with zeros or ones, in case of numeric array.

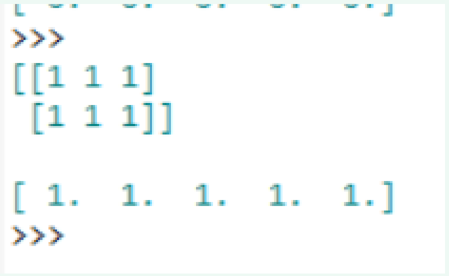
```
import numpy as np
arr = np.zeros([2,3], dtype=int)
print(arr)
print ()
arr2 = np.zeros (5, dtype = float)
print (arr2)
```



```
>>>
[[0 0 0]
 [0 0 0]]

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

```
import numpy as np
arr = np.ones([2,3], dtype=int)
print(arr)
print ()
arr2 = np.ones (5, dtype = float)
print (arr2)
```



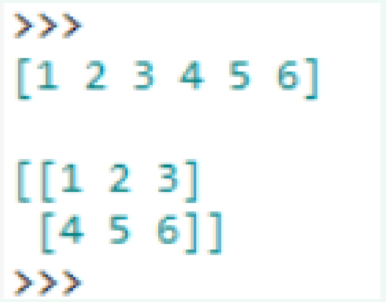
```
>>>
[[1 1 1]
 [1 1 1]]

[ 1.  1.  1.  1.  1.]
>>>
```

Array Reshaping

- Reshaping of the array means changing the dimensions of a certain array.

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



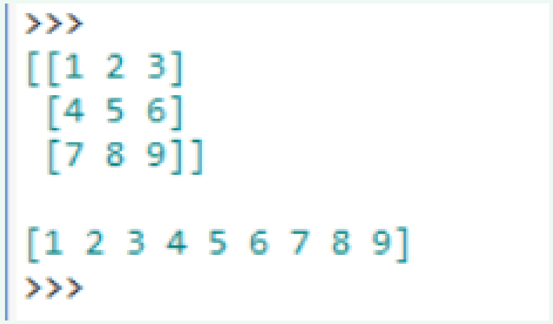
```
>>>
[1 2 3 4 5 6]

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

- Hint: make sure to change to an acceptable shape, for example you cannot reshape the array in the previous example to (3,4), since you cannot change the total size of an array from 6 to 12.

Array Reshaping – another example

```
import numpy as np
arr = np.array([[1,2,3],[4,5,6], [7,8,9]])
print(arr)
print ()
newarr = arr.reshape(9)  #note that 9 is the same number of elements
print (newarr)
```



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

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

Loop Through Array Elements

Example: Write a Python program that creates an empty array with 5 elements, read them from the user. And then asks the user to enter an element to search for, and returns back if it is found or not.

```
import numpy as np
arr = np.empty (5, dtype = int)
for i in range(5):
    arr[i] = int (input('Enter next element'))

x = int(input('Enter the element you need to seach for'))
found = False
for j in range(5):
    if arr[j] == x:
        found = True
        break;
if found:
    print (x, 'is found in the array')
else:
    print (x, 'is not found in the array')
```

Loop Through Array Elements

Another way to solve the previous example, is by looping through the array as a range of elements not indeces

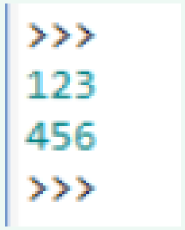
```
import numpy as np
arr = np.empty (5, dtype = int)
for i in range(5):
    arr[i] = int (input('Enter next element'))

x = int(input('Enter the element you need to seach for'))

found = False
for e in arr:
    if e == x:   #Note here e is the elements stored in the array not its index
        found = True
        break;
if found:
    print (x, 'is found in the array')
else:
    print (x, 'is not found in the array')
```

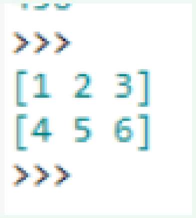
Loop Through Array Elements Two-dimensional array

```
import numpy as np
arr = np.array ([[1,2,3], [4,5,6]])
for i in range(2):
    for j in range(3):
        print (arr[i,j], end = '')
    print ()
```



```
>>>
123
456
>>>
```

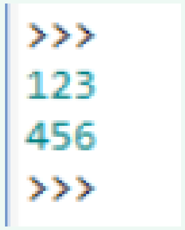
```
import numpy as np
arr = np.array ([[1,2,3], [4,5,6]])
for i in arr:
    print (i)
```



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

- Note in the previous code, that the output prints each row as one unit, so the loop actually iterates only twice. If you want to print each element alone, use the first code, or the next one.

```
import numpy as np
arr = np.array ([[1,2,3], [4,5,6]])
for i in arr:
    for j in i:
        print (j, end = '')
    print()
```



```
>>>
123
456
>>>
```

NumPy Joining Array

- Joining means putting contents of two or more arrays in a single array.
- Concatinate () function is used for this purpose.

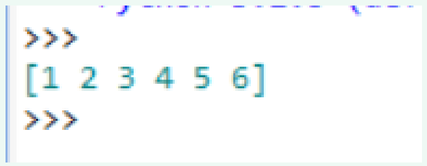
```
import numpy as np

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

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

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

print(arr)
```



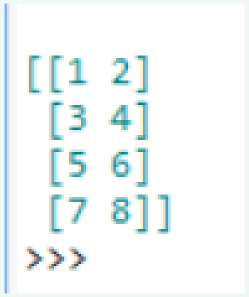
```
>>>
[1 2 3 4 5 6]
>>>
```

- To join two-dimensional arrays:

NumPy Joining Array

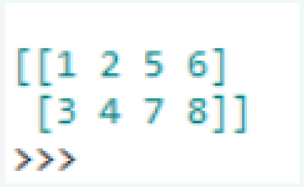
- To join two-dimensional arrays, setting axis parameter to 0:

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=0)
print(arr)
```



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

```
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)
```

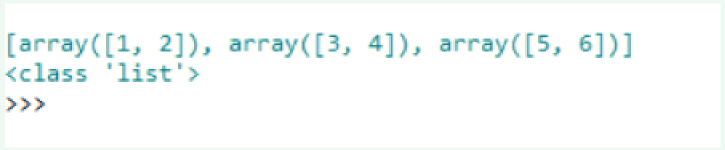


```
[[1 2 5 6]
 [3 4 7 8]]
>>>
```

NumPy Splitting Array

- Splitting is reverse operation of Joining.
- 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.

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

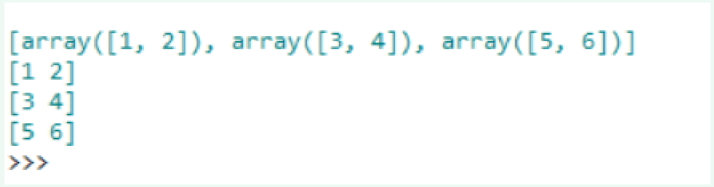


```
[array([1, 2]), array([3, 4]), array([5, 6])]
<class 'list'>
>>>
```

- You may ask, how can I store each array in a different variable?
- Note in the output that the data type of the newarr is list. List, as was mentioned before, is a data structure in Python that is sort of treated as an array. So, we can assign each element in this array to a separate variable. Check the next code.
- Hint: Lists will be discussed in details in a special chapter.

NumPy Splitting Array

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
a = newarr[0]
b = newarr[1]
c = newarr[2]
print (a)
print (b)
print (c)
```



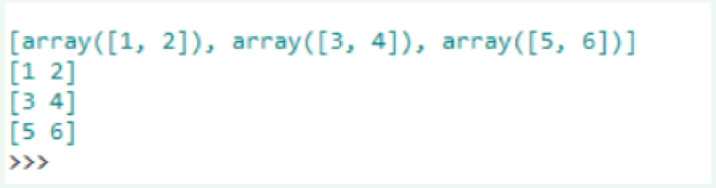
```
[array([1, 2]), array([3, 4]), array([5, 6])]
[1 2]
[3 4]
[5 6]
>>>
```

- Note here that a, b and c are printed as arrays, and you can deal with them now as one-dimensional arrays.
-

2022

NumPy Splitting Array

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
a = newarr[0]
b = newarr[1]
c = newarr[2]
print (a)
print (b)
print (c)
```

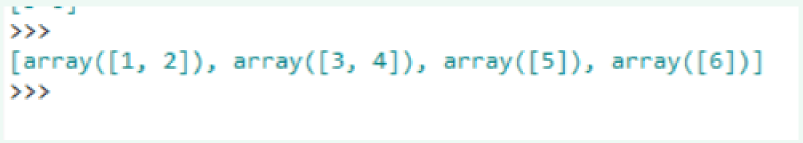


```
[array([1, 2]), array([3, 4]), array([5, 6])]
[1 2]
[3 4]
[5 6]
>>>
```

- Note here that a, b and c are printed as arrays, and you can deal with them now as one-dimensional arrays.
-

NumPy Splitting Array

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



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

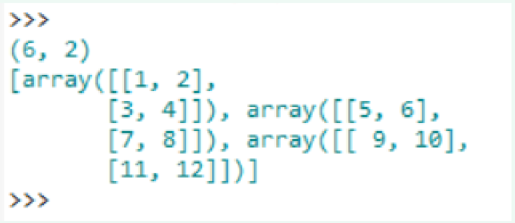
- In the previous example, note that the array has 6 elements, and you want to split it into 4 arrays, which naturally cannot be split equally. In this case, array_split() function, will split it in 4 arrays but with different sizes in a proper way.

- There is a split() method in NumPy that works almost the same way, but it won't word in such a case where the division of the array won't result arrays of the same size. (Check it)

NumPy Splitting Array

- For splitting a 2-dimensional array, check the following example:

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



```
>>>
(6, 2)
[array([[1, 2],
       [3, 4]]), array([[5, 6],
       [7, 8]]), array([[ 9, 10],
       [11, 12]])]
>>>
```

* Note that array arr has 6 rows and two columns, the code will split it into 3 two-dimensional arrays, each with 2 rows and two columns. Hence, it divides the original array according to the rows.

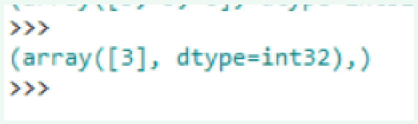
- array_split () function also has a parameter axis, with default = 1. That's why the array has been splitted by rows. Try to set the axis parameter to 0, and check the results.

```
newarr = np.array_split(arr, 3, axis = 1)
```

NumPy Searching Array

- You can search an array for a certain value, and return the indexes that get a match.
- To search an array, use the where() function.

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

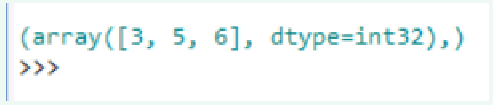


```
>>>
(array([3], dtype=int32),)
>>>
```

* This code will return the position where 4 was found in the array, and its data type.

* What if 4 is found several times in the array?

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



```
(array([3, 5, 6], dtype=int32),)
>>>
```

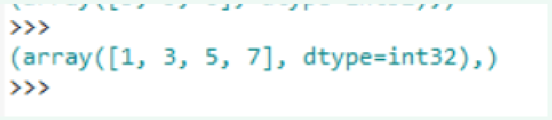
* Note that this time number 4 has been found 3 times in the array, and thus 3 indecies were retrieved in the result.

NumPy Searching Array

- Another interesting aspect of Python NumPy, is that you can search an array according to a certain condition.

- Ex: Find the indexes where the values are even:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
```



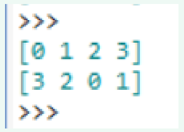
```
>>>
(array([1, 3, 5, 7], dtype=int32),)
>>>
```

- Check the type of the variable x, you will find that it is a Tuple. Tuples is a data structure in Python that will be discussed later. Then, you can find a way to go through the returned indecies to print the corresponding values.

NumPy Sorting Array

- Sorting means putting elements in an *ordered sequence*.
- You can sort numbered or string arrays ascending or descending
- Function sort() is used for this purpose.

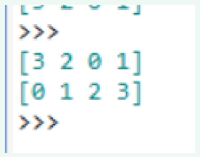
```
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
print (arr)
```



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

- Note here that sort() function doesn’t change the order of the original array, so you can call the function into a variable to create an ordered array

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

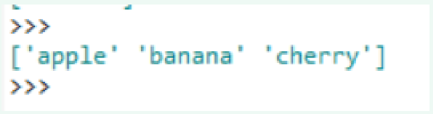


```
>>>
[3 2 0 1]
[0 1 2 3]
>>>
```

NumPy Sorting Array

- Sorting array of strings, will order the elements from alphabetically (from A to Z), according to ASCII code of characters.

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



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

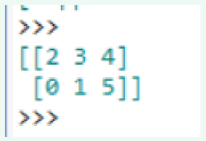
- Boolean arrays can also be ordered, putting all False values at the beginning then the True values. Check the following code:

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

NumPy Sorting Array

- Sorting a 2-dimensional array, will sort each row in the array separately.

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



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

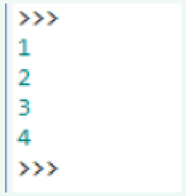
Exercise: Search the Internet of the way to order the array in descending order.

Pass an Array into a Function

- An array can be passed directly to a function, and the corresponding parameter is declared as a normal parameter, with no indication that it is an array

```
import numpy as np
def print_arr (a):
    for i in a:
        print (i)

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



```
>>>
1
2
3
4
>>>
```

2022