# Basic Programming in Python

## 6. Session: NumPy

Nohayr Muhammad Abdelmoneim

Summer Term 2023

June 5th, 2023

# Overview

- Introduction to NumPy library
- Why NumPy?
- NumPy arrays
- NumPy attributes
- NumPy methods

# NumPy library

- NumPy stands for Numerical Python.
- It’s an open-source library.
- NumPy provides arrays and multidimensional arrays of the same type.
- NumPy is used to store and manipulate various types of data.

# Why Numpy?

- NumPy provides powerful tools to manipulate and perform various operations on arrays.
- NumPy is one of the core libraries that is used in almost every application specially data science, machine learning, etc.
- NumPy is much faster than lists or other Python collections, which is required when dealing with large data.

# Numpy arrays

- A NumPy array is a collection of data **of the same type** stored in one or more dimensions.
- An array can be thought of as a list.
- Arrays in NumPy are called ndarrays as it can have n dimensions,  i.e., array of arrays.
- An array can be created in many ways, one of them is np.array() method.
- Using np.array() the required elements are passed as an argument.

# How to use Numpy?

- You need to install NumPy library if it's not already installed.
- You need to import it in your program
- You can use the library either by its name or by an alias.

- Using library name

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

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

- Using the conventional alias "np"

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

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

# List vs. Array

```
#Creating a list with different types
x=[3,5, "Hello", True, 7.2]
print(x)
print(type(x[0]))
print(type(x[2]))
print(type(x[3]))
print(type(x[4]))
print(x[0]+x[1])
```

```
[3, 5, 'Hello', True, 7.2]
<class 'int'>
<class 'str'>
<class 'bool'>
<class 'float'>
8
```

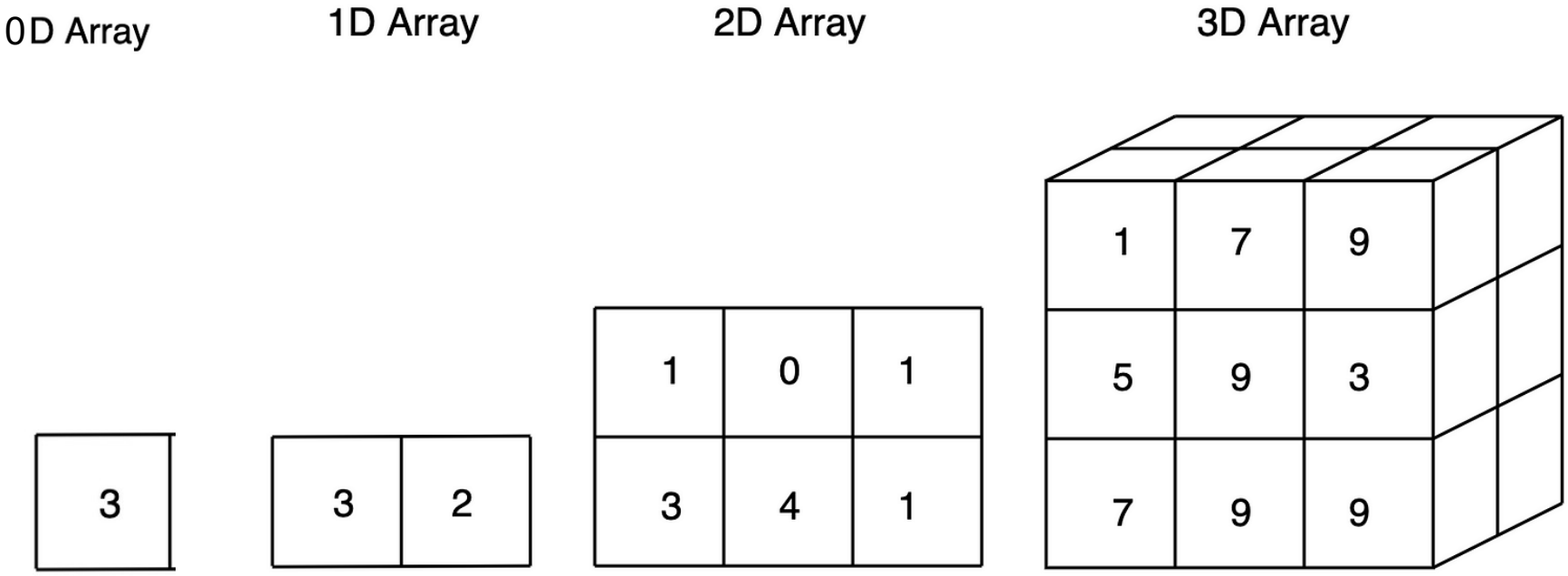
```
#Creating numpy array with different types
import numpy as np
y=np.array([3,5, "Hello", True, 7.2])
print(y)
print(type(y[0]))
print(type(y[2]))
print(type(y[3]))
print(type(y[4]))
print(y[0]+y[1])
```

```
['3' '5' 'Hello' 'True' '7.2']
<class 'numpy.str_'>
<class 'numpy.str_'>
<class 'numpy.str_'>
<class 'numpy.str_'>
35
```

# Arrays dimensions

- A dimension in arrays is one level of array depth.
- 0-D array is a single value (a scalar).
- 1-D array is like a vector, a sequence of values stored in a single dimension.
- 2-D arrays are usually called matrices. It's an array whose elements are 1-D arrays.
- 3-D arrays are like a cube. An array whose elements are 2-D arrays.
- n-D arrays are arrays whose elements are (n-1)-D arrays.

# Arrays dimensions



# Arrays dimensions

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

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

```
0
1
2
3
```

# Indexing

- You can access an array element by referring to its index number.
- The indexes in NumPy arrays start with 0 and ends at len-1.

```
import numpy as np

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

print(arr[0])
print(arr[2])
```

```
1
3
```

# Indexing in 2-D arrays

- To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

```
import numpy as np

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

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

```
2nd element on 1st row:  2
```

# Indexing in 3 and higher dimensional arrays

- What would that code output?

```
import numpy as np

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

print(arr[0, 1, 2])
```

**Answer is: 6**

And this is why:

- We have a 3-D array
- The first number represents the first dimension, which contains two arrays:
  [[1, 2, 3], [4, 5, 6]] and [[7, 8, 9], [10, 11, 12]]
  Since we selected 0, we are left with the first array:
  [[1, 2, 3], [4, 5, 6]]
- The second number represents the second dimension, which also contains two arrays:
  [1, 2, 3] and [4, 5, 6]
  Since we selected 1, we are left with the second array:
  [4, 5, 6]
- The third number represents the third dimension, which contains three values:
  4 5 6
  Since we selected 2, we end up with the third value:
  6

# Negative indexing

- Negative indexing allows accessing the elements of an array from the end.
- The last element is indexed -1, then -2, etc.

```
import numpy as np

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

print(arr[-1])
print(arr[-2])
print(arr[-3])
print(arr[-4])
print(arr[-5])
```

```
4
3
2
1

---------------------------------------------------------------------------
IndexError                                Traceback (most rec
Cell In[32], line 9
      7 print(arr[-3])
      8 print(arr[-4])
----> 9 print(arr[-5])

IndexError: index -5 is out of bounds for axis 0 with size 4
```

# Slicing

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: [*start*:*end*].
- We can also define the step, like this: [*start*:*end*:*step*].
- If we don't pass start it’s considered 0
- If we don't pass end it’s considered length of array in that dimension
- If we don't pass step it’s considered 1
- Item at index end is always excluded from result.

# Slicing Examples

```python
import numpy as np

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

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

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

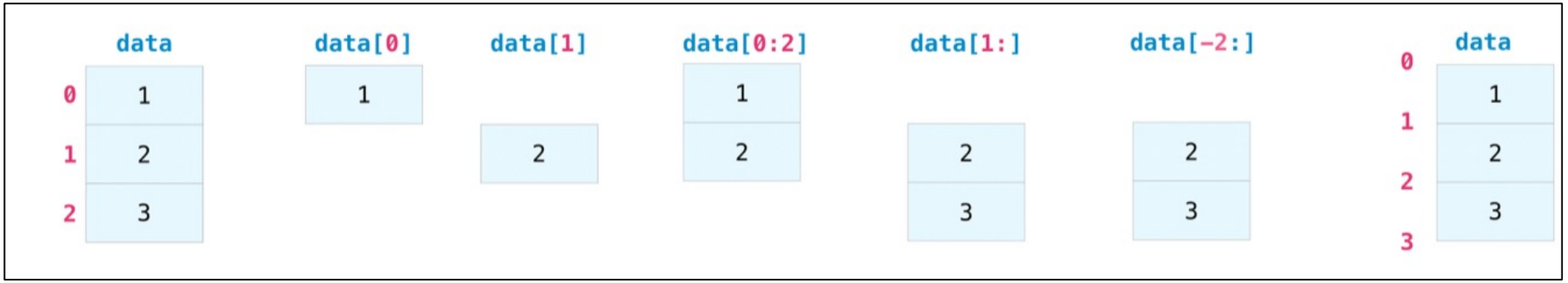
```python
import numpy as np

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

print(arr[4:])
```

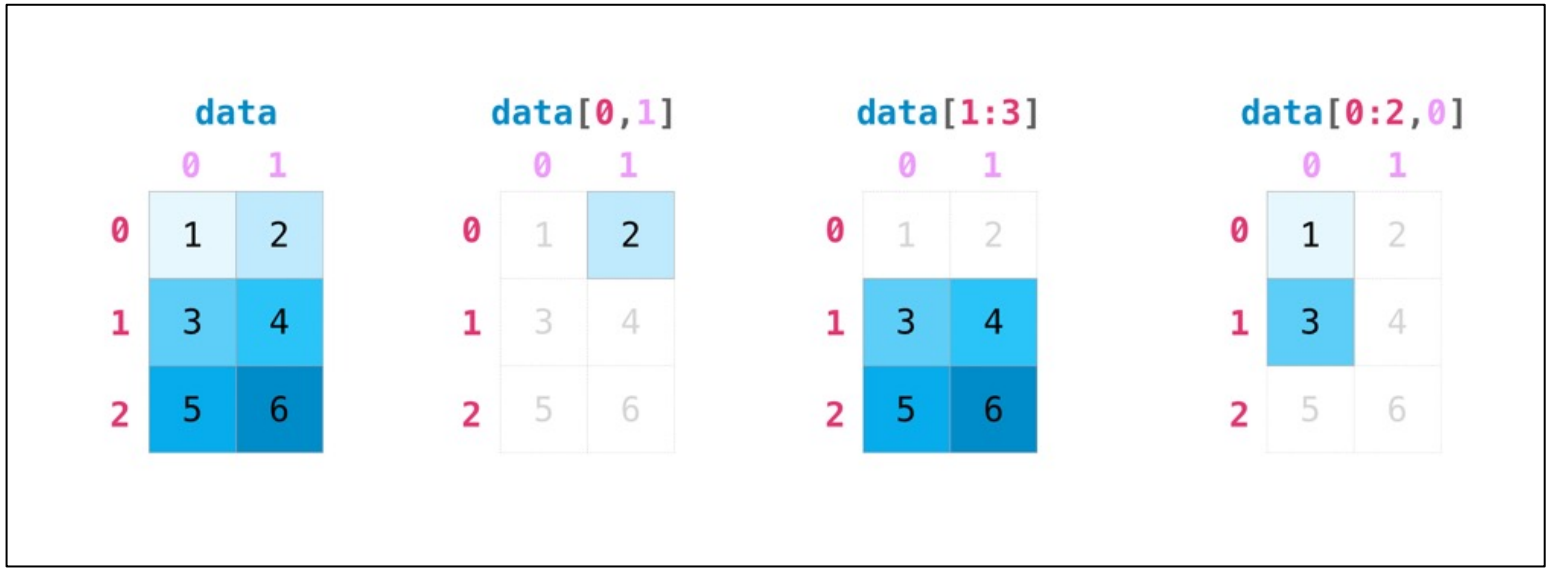
```
[5 6 7]
```

```python
import numpy as np

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

print(arr[:4])
```

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

# Slicing Examples



# Some NumPy arrays attributes

| | |
|---|---|
| `ndarray.shape` | Tuple of array dimensions. |
| `ndarray.strides` | Tuple of bytes to step in each dimension when traversing an array. |
| `ndarray.ndim` | Number of array dimensions. |
| `ndarray.data` | Python buffer object pointing to the start of the array's data. |
| `ndarray.size` | Number of elements in the array. |
| `ndarray.T` | View of the transposed array. |

# Some NumPy arrays attributes

```
import numpy as np

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

print("Dimension: ", arr.ndim)
print("Shape: ", arr.shape)
print("Size: ", arr.size)
print("T: ", arr.T)
```

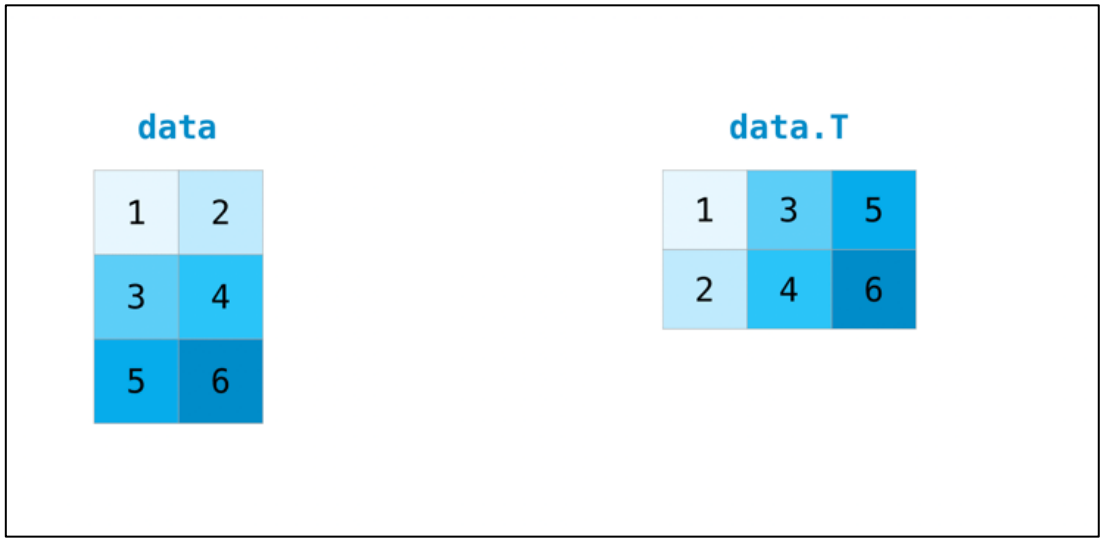
```
Dimension:  3
Shape:  (2, 2, 3)
Size:  12
T:  [[[ 1  7]
  [ 4 10]]

 [[ 2  8]
  [ 5 11]]

 [[ 3  9]
  [ 6 12]]]
```

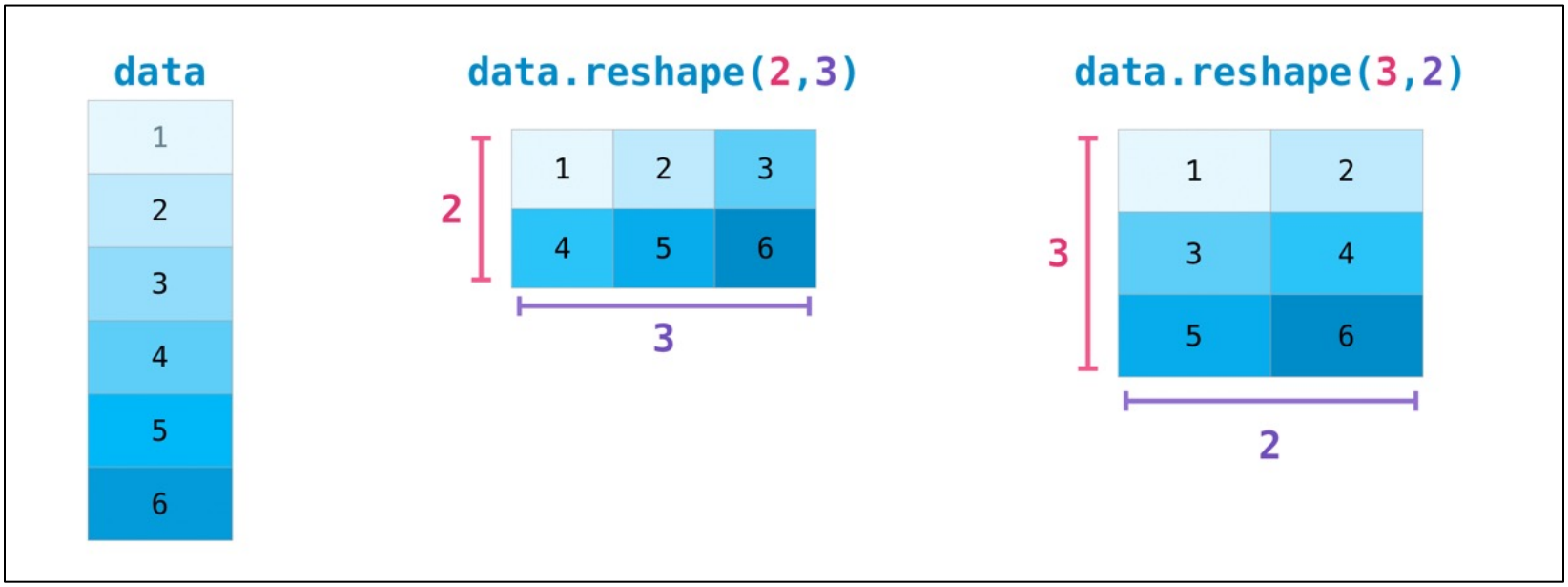
# Shape manipulation

For reshape, resize, and transpose, the single tuple argument may be replaced with `n` integers which will be interpreted as an n-tuple.

| | |
|---|---|
| `ndarray.reshape`(shape[, order]) | Returns an array containing the same data with a new shape. |
| `ndarray.resize`(new_shape[, refcheck]) | Change shape and size of array in-place. |
| `ndarray.transpose`(*axes) | Returns a view of the array with axes transposed. |
| `ndarray.swapaxes`(axis1, axis2) | Return a view of the array with *axis1* and *axis2* interchanged. |
| `ndarray.flatten`([order]) | Return a copy of the array collapsed into one dimension. |
| `ndarray.ravel`([order]) | Return a flattened array. |
| `ndarray.squeeze`([axis]) | Remove axes of length one from *a*. |

# Shape manipulation



# Automatic array creation

- np.zeros(shape): Creates an array full of zeros of the given shape.
- np.ones(shape): Creates an array full of ones of the given shape.
- np.empty(shape): Creates an array of initially random values of the given shape.
- np.arange(end)/np.arange(start,end)/np.arrange(start,end,step).
- np.linspace(start, end, number of elements).

# Automatic array creation

```
import numpy as np
arr1=np.zeros(5)
arr2=np.ones((2,3)) #shape is given as a tuple
arr3=np.empty((2,2))
arr4=np.arange(1,9,2)
arr5=np.linspace(1,4,6)
print("Zeros:", arr1)
print("\n Ones:", arr2)
print("\n Empty:", arr3)
print("\n arange:", arr4)
print("\n linspace", arr5)

arr6=np.zeros(2,3)
```

```
Zeros: [0. 0. 0. 0. 0.]

 Ones: [[1. 1. 1.]
 [1. 1. 1.]]

 Empty: [[4.9e-324 1.5e-323]
 [2.5e-323 3.5e-323]]

 arange: [1 3 5 7]

 linspace [1.  1.6 2.2 2.8 3.4 4. ]

---------------------------------------------------------
TypeError                                 Traceb
Cell In[60], line 13
     10 print("\n arange:", arr4)
     11 print("\n linspace", arr5)
---> 13 arr6=np.zeros(2,3)

TypeError: Cannot interpret '3' as a data type
```

# Some arithmetic methods

```python
import numpy as np

arr = np.array([[[1, 4, 3], [2, 5, 1]], [[0, 8, 9], [10, 11, 12]]])

print("max:", np.max(arr))
print("min:", np.min(arr))
print("argmax:", np.argmax(arr))
print("sum:", np.sum(arr))
print("mean:", np.mean(arr))
```

```
max: 12
min: 0
argmax: 11
sum: 66
mean: 5.5
```

# Simple search

```
import numpy as np

arr = np.array([[[1, 4, 3], [2, 5, 1]], [[0, 8, 9], [10, 11, 12]]])

print(arr>3)
print("\n Elements greater than 3: ", arr[arr>3])
```

```
[[[False  True False]
  [False  True False]]

 [[False  True  True]
  [ True  True  True]]]

 Elements greater than 3:  [ 4  5  8  9 10 11 12]
```

# Lecture questions

- This and the following slides are to address some of the questions raised in the lecture.

➢ What would happen if we remove the main square brackets when creating a multidimensional array?

```
import numpy as np

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

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

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[80], line 3
      1 import numpy as np
----> 3 arr = np.array([1,2,3,4,5], [6,7,8,9,10])
      5 print('2nd element on 1st row: ', arr[0, 1])

TypeError: Field elements must be 2- or 3-tuples, got '6'
```

# Lecture questions

- More slicing examples suggested in the lecture. Retrieving various elements of the same level

```
import numpy as np

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

print(arr[0:2, 1, 2])
```

```
[ 6 12]
```

```
import numpy as np

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

print(arr[0:2, 0:2, 2])
```

```
[[ 3  6]
 [ 9 12]]
```

# Lecture questions

- More slicing examples suggested in the lecture.
  Giving two dimenions for 3-D array

```
import numpy as np

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

print(arr[0:2, 1])
```

```
[[ 4  5  6]
 [10 11 12]]
```

# References

- https://www.w3schools.com/python/numpy/default.asp
- https://numpy.org/devdocs/user/absolute_beginners.html#

# QUESTIONS?