

SESSION 1

1. Introduction to Arrays
2. Arrays vs Lists
3. Creating Arrays using NumPy
4. Numpy Array Methods
5. Dimensions in Arrays
6. Indexing
 - 6.1 Negative Indexing
7. Slicing
 - 7.1. Negative Slicing

1. Introduction to Arrays

- In Python, array is a data structure which is used to store a collection of elements of the same data type.
- Unlike lists, which can contain elements of different data types, arrays are homogeneous, which means that all elements in an array must be of the same data type.

Advantages of Array over a List :

- Arrays are generally faster than lists, especially when dealing with large amounts of data.
- Arrays use less memory than lists, since they are stored as a contiguous block of memory.

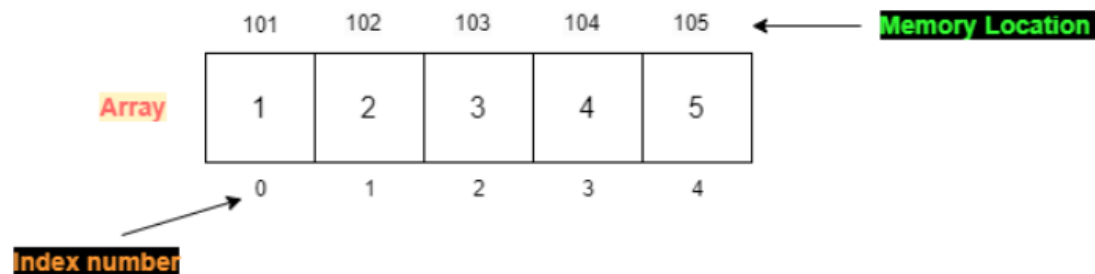
2. Arrays vs Lists

Difference between Array and List

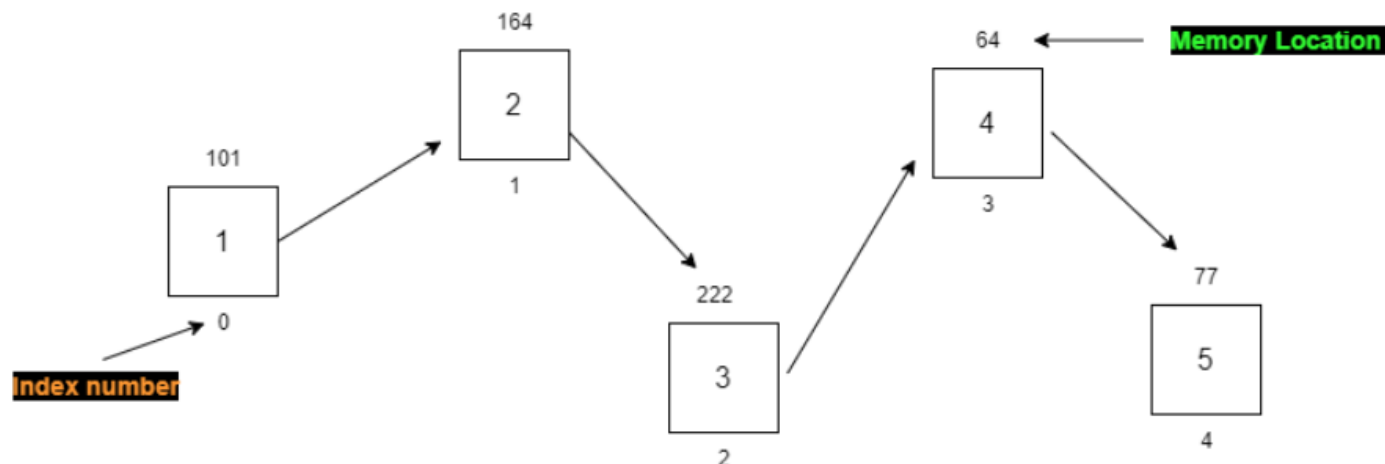
Array	List
1. Arrays are stored in contiguous memory location.	1. Lists are not in contiguous memory location.
2. Fixed in size	2. Dynamic in size.
3. Arrays are faster in execution	3. Lists are comparatively slower.

Memory Allocation of Lists & Arrays :

Contiguous Memory is allocated to arrays elements



Random memory is allocated to each list elements

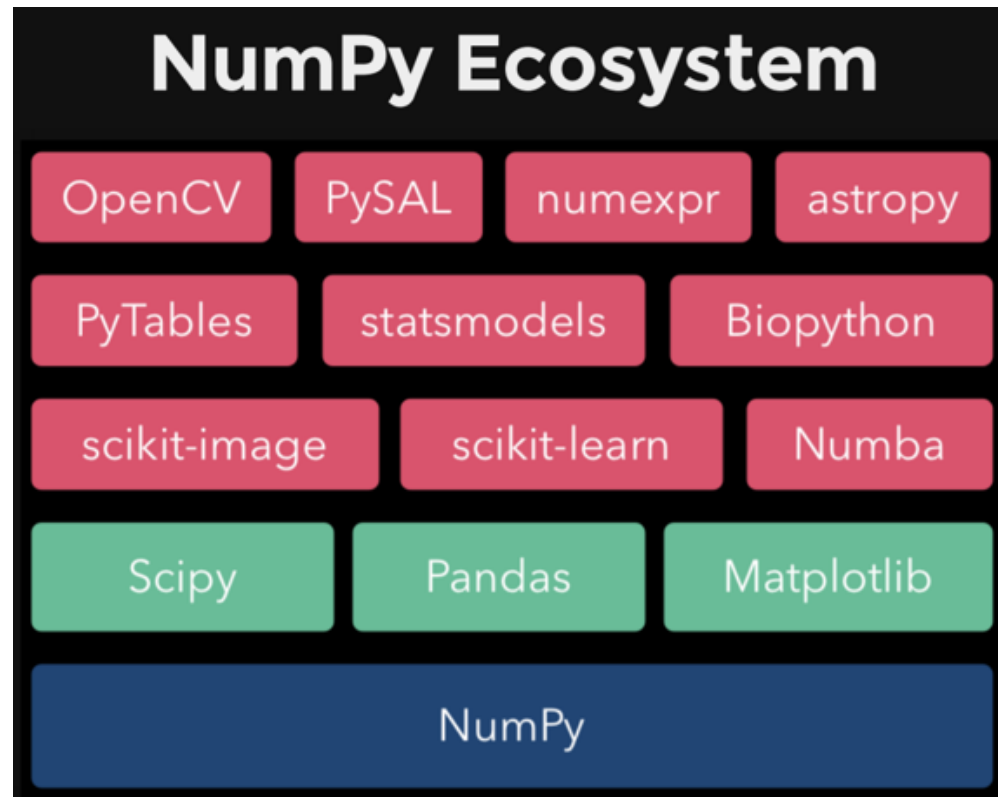


3. Creating Arrays using NumPy

What is NumPy?

- NumPy is a Python library.
- It stands for 'Numerical Python'.
- It is a library consisting of array objects and an extensive set of functions for fast operations on arrays.

Numpy is the foundation of the python scientific stack:



How to install NumPy?

```
pip install numpy
```

How to import numpy?

```
In [ ]: import numpy as np
# In Python, the as keyword is used to assign a new name to a module / library during import
```

- NumPy provides a powerful array computing functionality.
- It is widely used in scientific computing, data analysis, machine learning, and many other fields.
- A NumPy array is a collection of elements that are all of the same data type.

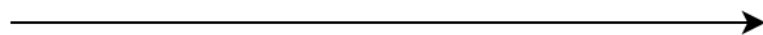
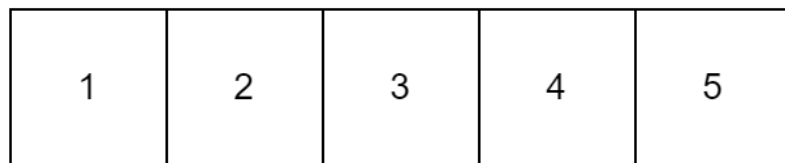
- NumPy arrays can be created using the `numpy.array()` function. This function takes a sequence of elements and converts them into a NumPy array. The array object in NumPy is called as ndarray.

In []:

```
# Creating a NumPy array:  
  
import numpy  
  
arr = numpy.array([1, 2, 3, 4, 5]) # The passed list gets converted into an array  
  
print(arr)
```

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

1 Dimension



axis 0

In []:

```
# Finding the type of the created array:  
  
import numpy as np  
  
arr = np.array([11, 12, 13, 14, 15])  
  
print(arr)  
  
print(type(arr))
```

```
[11 12 13 14 15]  
<class 'numpy.ndarray'>
```

- To create an ndarray, we can pass a list, tuple or any array-like object into the `array()` method, and it will be converted into an ndarray

```
In [ ]: # Passing a tuple to the array() function:

import numpy as np

arr = np.array((15, 25, 35, 45, 55))

print(arr)
```

```
[15 25 35 45 55]
```

4. Numpy Array Methods

`numpy.zeros()` method returns a new array of given shape and type, with zeros.

```
In [ ]: import numpy as np

b = np.zeros(2, dtype = int)
print(b)
print()
a = np.zeros([2, 2], dtype = int)
print(a)
print()
c = np.zeros([3, 3])
print(c)
```

```
[0 0]
```

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

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

`numpy.ones()` function returns a new array of given shape and type, with ones.

```
In [ ]: import numpy as np

b = np.ones(2, dtype = int)
```

```

print(b)
print()
a = np.ones([2, 2], dtype = int)
print(a)
print()
c = np.ones([3, 3])
print(c)

```

```
[1 1]
```

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

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

numpy.ndarray.fill() method is used to fill the numpy array with a scalar value (scalar value: numerical quantity or measurement that represents a single quantity).

If we have to initialize a numpy array with an identical value then we use numpy.ndarray.fill(). Suppose we have to create a NumPy array a of length n, each element of which is v. Then we use this function as a.fill(v). We need not use loops to initialize an array if we are using this fill() function.

```

In [ ]: # Python program explaining
        # numpy.ndarray.fill() function
        import numpy as np

        a = np.empty([3, 3])

        # Initializing each element of the array
        # with 42 by using nested loops
        for i in range(3):
            for j in range(3):
                a[i][j] = 42

        print(a)

        # now we are initializing each element
        # of the array with 42 using fill() function.
        a.fill(42)

        print(a)

```

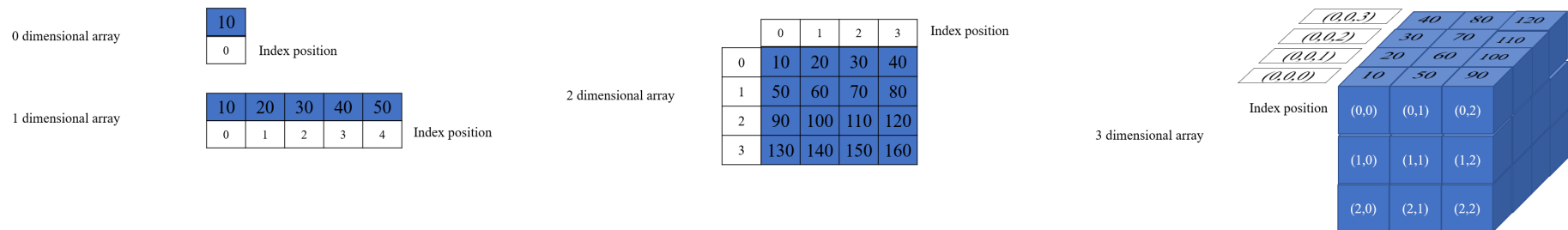
5. Dimensions in Arrays

- The dimension of an array can simply be defined as the number of subscripts or indices required to specify a particular element of the array.

Let's understand the dimensionality of an array by an analogy of a library.

In a Library, Let's consider books as individual elements. Books are kept on the shelves of the racks in the library where each rack and shelf are indexed.

- A single shelf can be viewed as a 1-D (1-Dimensional) array of books.
- A single rack with several shelves can be considered to be a 2-D (2-Dimensional) array.
- The complete library with several racks can be viewed as a 3-D (3-Dimensional) array.
- We require the rack number, shelf number, and the position of the book on the shelf to get a particular book from the library.
- Similarly, an institution can have several libraries on its campus and thus the institution can be viewed as a 4-D (4-Dimensional) array with individual libraries as its elements.



0-D Arrays

- 0-D arrays, or Scalars, are the elements in an array.
- Each value in an array is a 0-D array.

```
In [ ]: # Example of 0-D array:

import numpy as np

arr = np.array(72)
```



```
print(arr)
print(arr.ndim) # ndim attribute will print the number of dimensions of an array
```

```
72
0
```

1-D Arrays

- An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

In []:

```
# Example of 1-D array:
import numpy as np

arr = np.array([21, 32, 73, 54, 45])

print(arr)
print(arr.ndim)
```

```
[21 32 73 54 45]
```

2-D Arrays

- An array that has 1-D arrays as its elements is called a 2-D array.
- These are often used to represent a matrix.

In []:

```
# Create a 2-D array containing two arrays with the 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]]
```

3-D arrays

- An array that has 2-D arrays (matrices) as its elements is called as 3-D array.

In []:

```
# Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:
```

```
import numpy as np

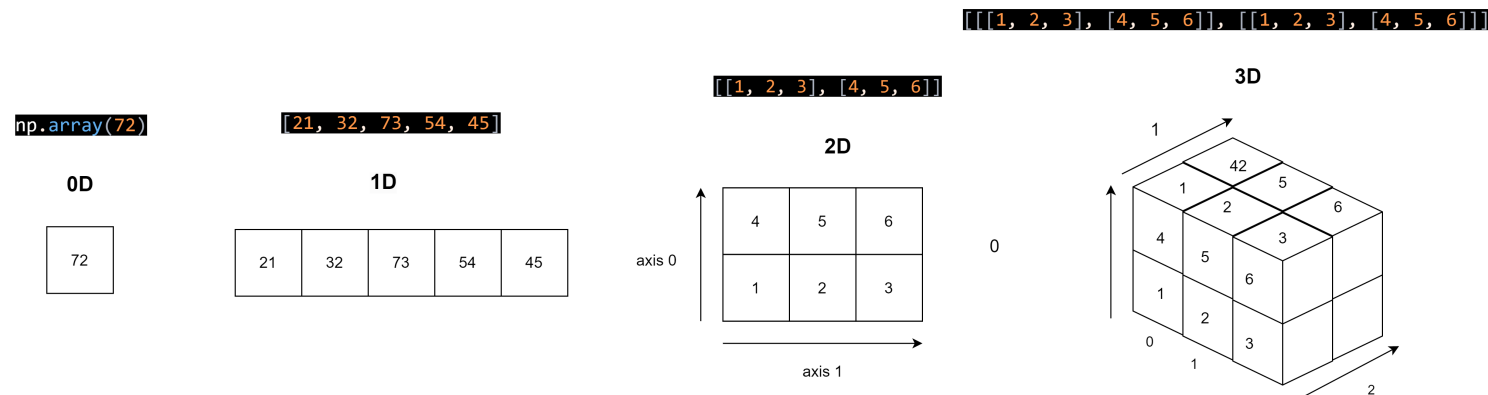
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
print("3-D Array Indexing:")
# This would print the element which is at index 1 in the 2nd list that is inside the outer list
print(arr[0,1,1])
```

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

  [[1 2 3]
  [4 5 6]]]
3-D Array Indexing:
5
```

Numpy Array Dimensions



Check Number of Dimensions

- NumPy Arrays provide the `ndim` attribute that returns an integer that tells us how many dimensions the array has.

```
In [ ]: # Check how many dimensions these below given arrays have:
```

```
import numpy as np

a = np.array(42) # 0-D
```

```

b = np.array([17, 28, 30, 42, 51]) # 1-D
c = np.array([[71, 22, 73], [14, 85, 61]]) # 2-D
d = np.array([[[10, 28, 31], [43, 85, 26]], [[41, 21, 32], [42, 52, 16]]]) # 3-D

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

```

0

1

2

3

Higher Dimensional Arrays

- An array can have any number of dimensions.
- When the array is created, you can define the number of dimensions by using the `ndmin` argument.

In []:

```

# Create an array with 5 dimensions and verify that it has 5 dimensions:

import numpy as np

arr = np.array([12, 32, 43, 94], ndmin=5) # The provided list would be converted into a 5-D array

print(arr)
print('number of dimensions :', arr.ndim)

```

```

[[[[[12 32 43 94]]]]]
number of dimensions : 5

```

6. Indexing

- Indexing in an array refers to accessing a specific element in the array by specifying its position or index.
- In most programming languages, arrays are zero-indexed, which means that the first element in the array is at index 0, the second element is at index 1, and so on.

In []:

```

# Indexing in 1-D array:
import numpy as np

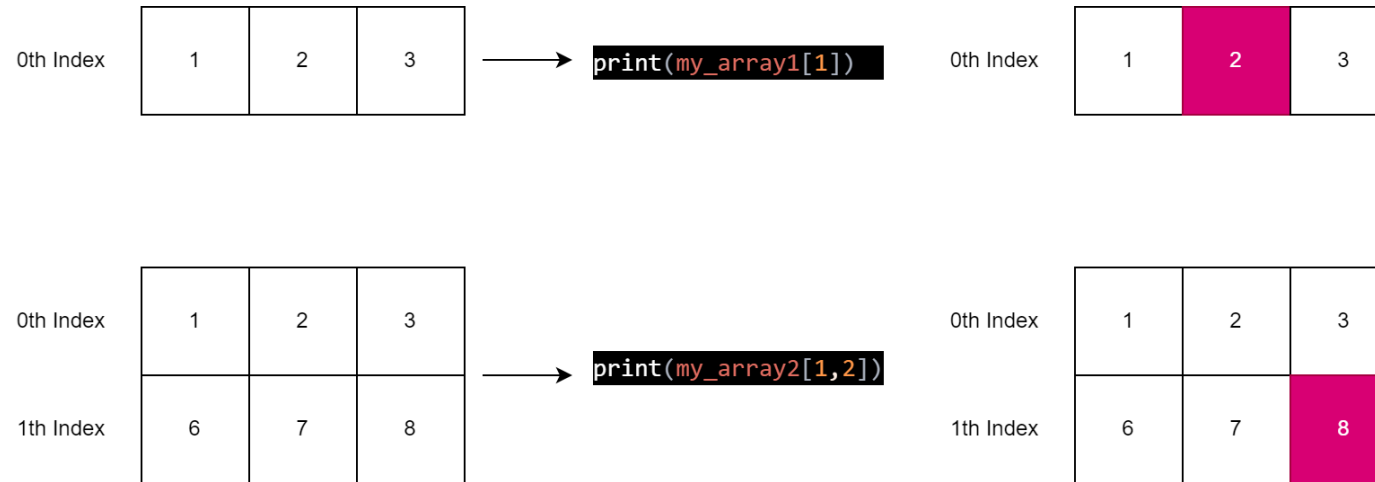
```

```
my_array1 = np.array([1, 2, 3])
print(my_array1[1])    # Output: 2

# Indexing in 2-D array:

my_array2 = np.array([[1, 2, 3], [4, 5, 6]])
print(my_array2[1,2])  # Output: 6
```

2
6



In []:

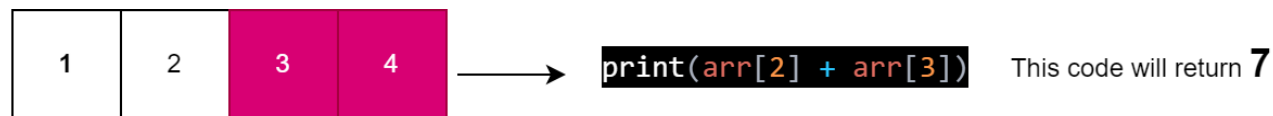
```
# Get the third and fourth element from the following array and return their sum.

import numpy as np

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

print(arr[2] + arr[3])
```

7



Getting the third and the fourth element

Access 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.

```
In [ ]: # Accessing 2-D Array elements

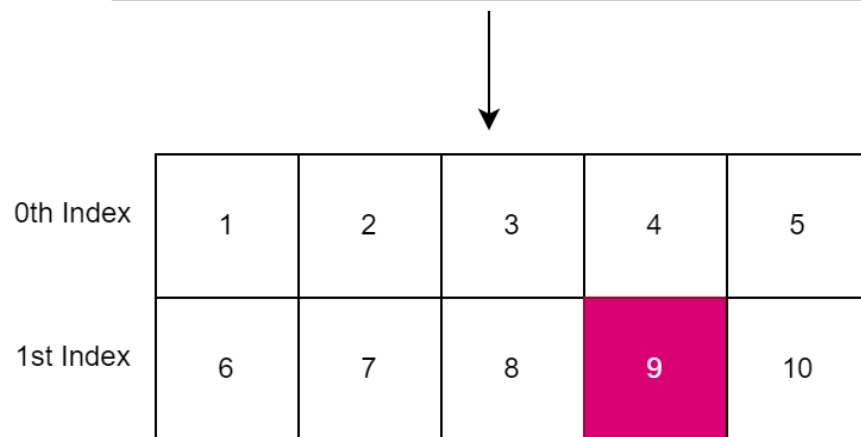
import numpy as np

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

print('2nd element on 1st row: ', arr[1, 3])
```

2nd element on 1st row: 9

```
print('2nd element on 1st row: ', arr[1, 3])
```



0th Index	1	2	3	4	5
1st Index	6	7	8	9	10

4th Element in second row

```
In [ ]: # Access the 3rd element on 2nd row :

import numpy as np

arr = np.array([[10,29,43,34,45], [16,47,38,29,10]])

print('3rd element on 2nd row: ', arr[1, 2])
```

3rd element on 2nd row: 38

Indexing on 2D numpy array

0th Index	10	29	43	34	45
1st Index	16	47	38	29	10

3rd Element in second row

Access 3-D Arrays

- To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

In []:

```
# Accessing elements from 3-D Arrays
# Access the third element of the second array of the first array:

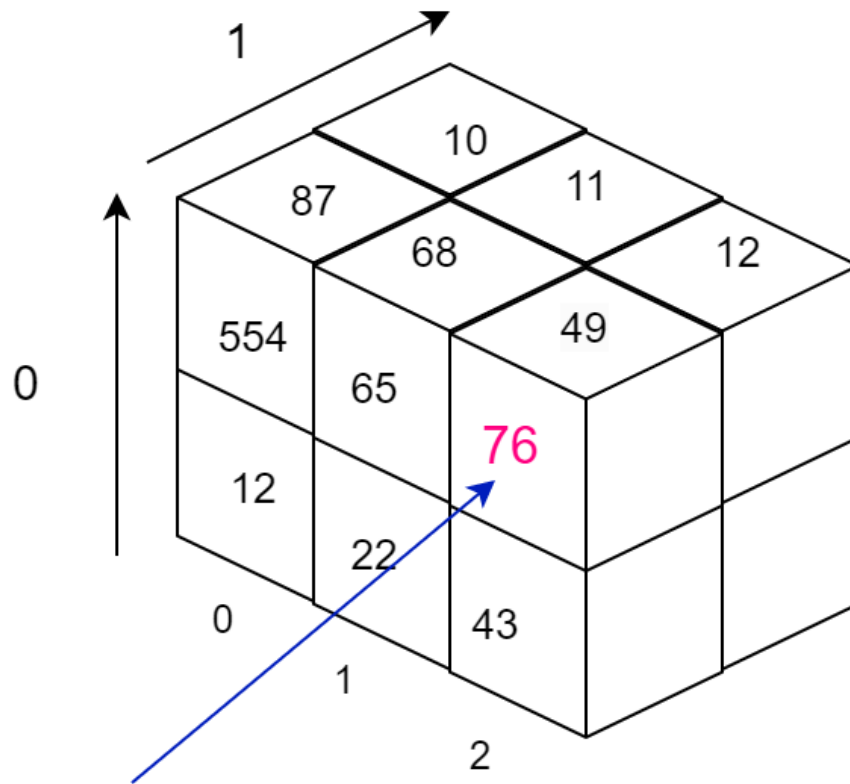
import numpy as np

arr = np.array([[[12, 22, 43], [554, 65, 76]], [[87, 68, 49], [10, 11, 12]]])

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

76

Indexing on 3D Array



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

6.1 Negative Indexing

Negative Indexing starts from -1 and here -1 refers to the last element in the array.

```
In [ ]: # Accessing elements from 2-D Arrays
# Print the Last element from the 2nd dim:

import numpy as np

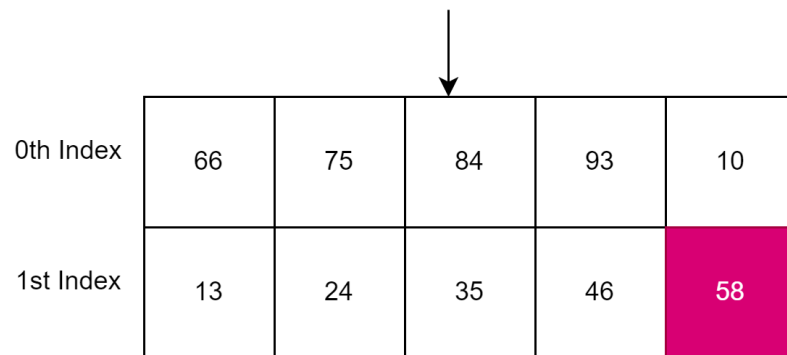
arr = np.array([[13,24,35,46,58], [66,75,84,93,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

Last element from 2nd dim: 10

Negative Indexing on 2D numpy array

```
print('Last element from 2nd dim: ', arr[1, -1])
```



0th Index	66	75	84	93	10
1st Index	13	24	35	46	58

Last element in 2nd row

7. Slicing

- Slicing refers to the technique of accessing a range of elements from a sequence (e.g. list, string, tuple) using the slicing operator ':'. It is a way to extract a subset of the sequence, starting from a specified index and ending at another specified index, while also specifying the step size.
- We pass a slice instead of an index like this: [start_index:stop_index + 1].

- We can also define the step_size, like this: [start_index:stop_index:step_size].
- If we don't pass the stop_index, the length of the array is considered as the stop_index.
- If we don't pass the step_size, it is considered as 1.

```
In [ ]: # Slice elements from index 1 to index 5 from the following array:

import numpy as np

arr = np.array([21, 32, 43, 64, 57, 68, 37])

print(arr[1:5]) # element at index 1 is included & element at index 5 is excluded
```

[32 43 64 57]

21	32	43	64	57	68	37
----	----	----	----	----	----	----

The highlighted section is the result we get after slicing

```
In [ ]: # Slice elements from the beginning to index 3 (not including the element at index 3):

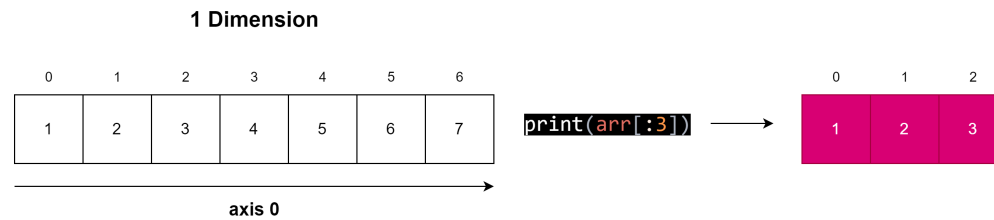
import numpy as np

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

print(arr[:3])
```

[1 2 3]

Slicing on 1D numpy array



7.1. Negative Slicing

```
In [ ]: # Negative slicing (1-D array):
# Slice a section from the given array that contains all the elements from index 3 to index 5 (including).

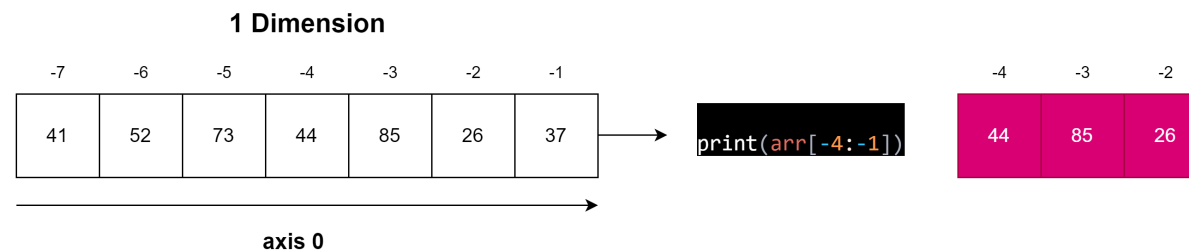
import numpy as np

arr = np.array([41, 52, 73, 44, 85, 26, 37])

print(arr[-4:-1])
```

[44 85 26]

Negative Slicing on 1D numpy array



```
In [ ]: # Slicing (2-D Array):
# From the second element, slice elements from index 1 to index 4 (not included):

import numpy as np

arr = np.array([[18, 72, 63, 54, 35], [46, 57, 68, 79, 10]])
```

```
print(arr[1, 1:4])
```

```
[57 68 79]
```

1st index	46	57	68	79	10
0th index	18	72	63	54	35

```
In [ ]:
```

```
# Slicing multiple elements of a 2-D array:

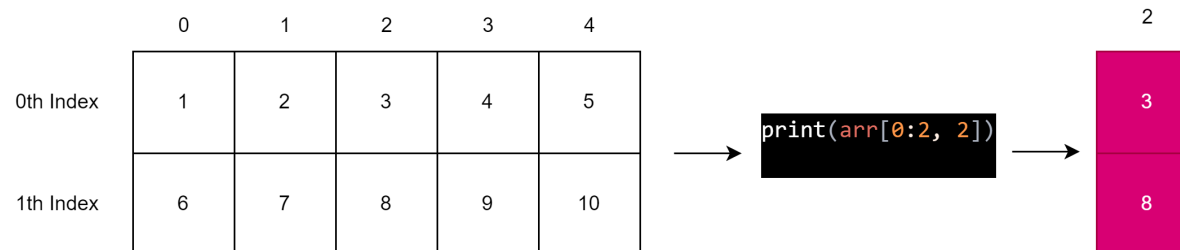
import numpy as np

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

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

```
[3 8]
```

Slicing on 2d numpy array



Homework Questions

1) Create a 0D array

- 2) Create a 1D array
- 3) Create a 2D array
- 4) Print the last element from the 2nd dimension
- 5) Access the third element of the second array of the first array
- 6) Get third and fourth elements from the following array and add them.

1. Create a numpy array of shape (3, 5) with random integers between 1 and 10. Slice the array to obtain the first row.

1. Create a numpy array of shape (5, 5) with integers from 1 to 20. Slice the array to obtain the second column, but only the first three elements.

For solutions of Homework questions, please refer to the `HomeworkSolution.ipynb` file.