SESSION 2

1. NumPy Data Types

- i - integer
- b - boolean
- u - unsigned integer
- uint8 - 8-bit unsigned integer (range: 0 through 255 decimal)
- uint64 - 64-bit unsigned integer (range: 0 through HEXA 0xFFFFFFFFFFFFFFFF)
- O - object
- U - unicode string

Checking the Data Type of an Array

- The NumPy array object has a property called `dtype` that returns the data type of the array.

In [ ]:
```
# Get the data type of an array containing integers:

import numpy as np

arr = np.array([11, 21, 31, 14])

print(arr.dtype) # Print Data Type of arr
```

```
int32
```

In [ ]:
```
# Get the data type of an array containing strings:

import numpy as np

arr = np.array(['c', 'python', 'java'])

print(arr.dtype) # Print Data Type of arr
```

```
<U6
```

Creating Arrays With a Defined Data Type

- We use the `array()` function to create arrays, this function can take an optional argument: `dtype` that allows us to define the expected data type of the array elements

In [ ]:
```
# Create an array with data type string:

import numpy as np

arr = np.array([14, 12,63, 74], dtype='S')

print(arr)
print(arr.dtype)
# [b'14' b'12' b'63' b'74'] , Where 'b' stands for bytes. In Python, a byte string is just a sequence of bytes.
```

```
[b'14' b'12' b'63' b'74']
|S2
```

In [ ]:
```
# Create an array with data type 4 bytes integer:

import numpy as np

arr = np.array([1, 2, 3, 4], dtype='i4')

print(arr)
print(arr.dtype)
```

```
[1 2 3 4]
int32
```

2.Converting datatype of an array

The `astype()` function creates a copy of the array, and allows you to specify the data type as a parameter.

The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

In [ ]:
```
# Change data type from integer to boolean:

import numpy as np

arr = np.array([1, 0, 3])

newarr = arr.astype(bool)

print(newarr)
print(newarr.dtype)
```

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

3. Array Reshaping

- Reshaping an Array 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.

Reshaping an Array From 1-D to 2-D

In [ ]:
```
# Convert the following 1-D array with 12 elements into a 2-D array.
# The outermost dimension will have 3 arrays, each with 4 elements

import numpy as np

arr = np.array([11, 22, 33, 34, 45, 56, 67, 78, 79, 10, 11, 12])
```

```
newarr = arr.reshape(3, 4)

print(newarr)
```

```
[[11 22 33 34]
 [45 56 67 78]
 [79 10 11 12]]
```

1 D array

11	22	33	34	45	56	67	78	79	10	11	12

3 D array

0th index	11	22	33	34
1 st index	45	56	67	78
2nd index	79	10	11	12

Shape(3x4)

Reshaping an Array From 1-D to 3-D

In [ ]:
```
# 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:
```

```
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]]]
```

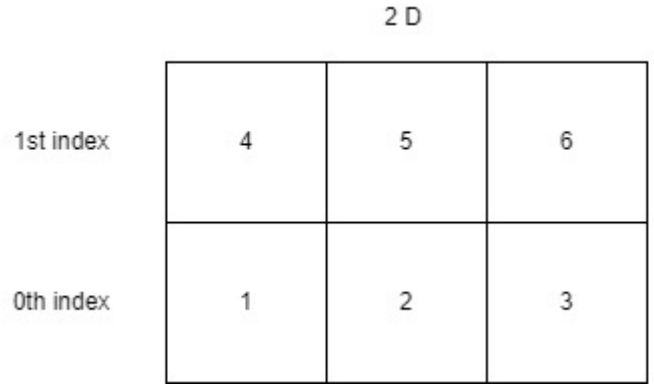
4. Joining Arrays

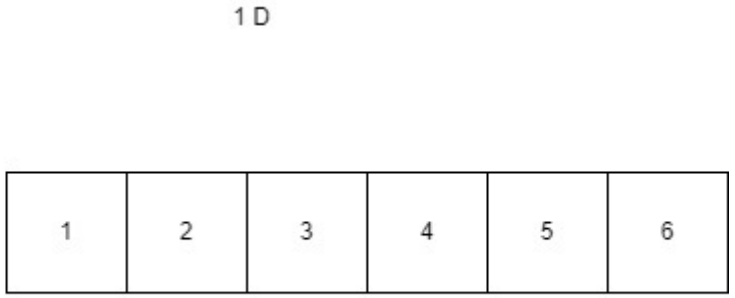
- Joining means putting contents of two or more arrays into a single array.
- We pass a sequence of arrays that we want to join to the `concatenate()` function, along with the axis. If axis is not explicitly passed, it is taken as 0.

In [ ]:

```
# Join two arrays

import numpy as np

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

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

arr = np.concatenate((arr1, arr2)) # By Default Axis=0

print(arr)
```

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





In [ ]:
```
# Join two arrays with Axis 0 & 1

import numpy as np

arr1 = np.array([[4, 5], [6, 8]])

arr2 = np.array([[3, 5], [7, 9]])

arr_horizontal = np.concatenate((arr1, arr2), axis = 0) # Axis Horizontal , Axis=0
arr_vertical = np.concatenate((arr1, arr2), axis = 1) # Axis Vertical , Axis=1
print("Array_horizontal")
print(arr_horizontal)
print("Array_vertical")
print(arr_vertical)
```

```
Array_horizontal
[[4 5]
 [6 8]
 [3 5]
 [7 9]]
Array_vertical
[[4 5 3 5]
 [6 8 7 9]]
```

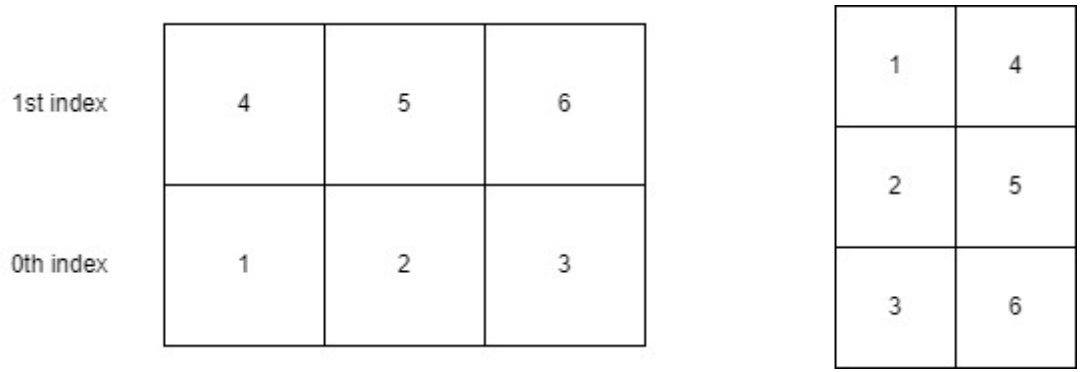
5. Array Stacking

- Stacking is same as concatenation, the only difference is that stacking is done along a new axis.

- We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, ie. stacking.
- We pass a sequence of arrays that we want to join to the stack() method along with the axis. If axis is not explicitly passed it is taken as 0 by Default.

In [ ]:
```
#Array Stacking with Axis 1
import numpy as np

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

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

arr = np.stack((arr1, arr2), axis=1)

print(arr)
```

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



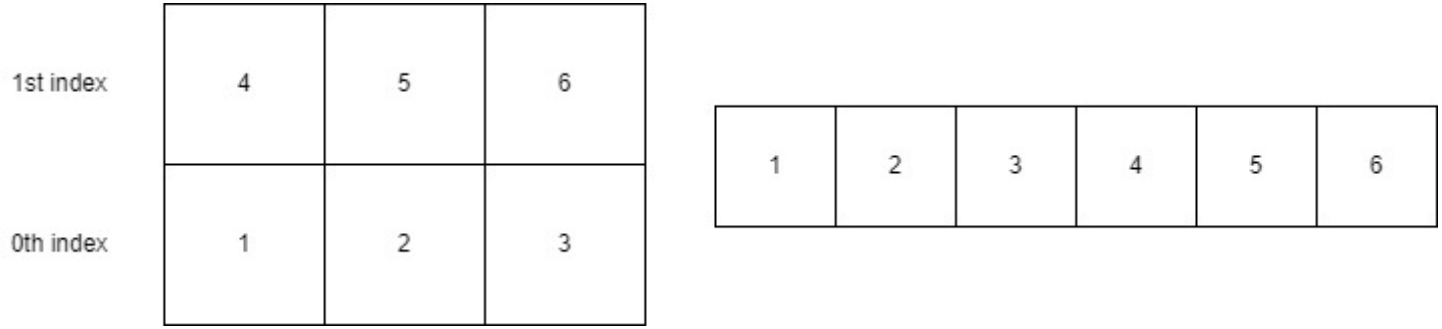
5.1 Horizontal Stack - Stacking Along Rows

- NumPy provides a helper function: `hstack()` to stack along rows.

In [ ]:
```
# Horizontal Stacking
# By Default the Axis value of hstack() is 0
import numpy as np

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

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

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

print(arr)
```

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



5.2 **Vertical Stack - Stacking Along Columns**

- NumPy provides a helper function: `vstack()` to stack along columns.

In [ ]:

```
# Vertical Stacking
# By Default the Axis value of vstack() is 1
import numpy as np

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

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

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

print(arr)
```

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

6. Filtering NumPy Array

- Numpy Array Filtering refers to the process of extracting a subset of elements from a NumPy array based on a certain condition or set of conditions.
- In NumPy, filtering is often accomplished using Boolean indexing, where a boolean array of the same shape as the original array is used to select elements that meet a specified condition. For example, if we have an array a and want to select all the elements that are greater than 5, we can create a Boolean array b using the expression `b = a > 5` and then use this Boolean array to index into a as follows: `a[b]` .
- We can also use logical operators such as `& (and), | (or), and ~ (not)` to combine multiple conditions. For example, to select all the elements that are greater than 5 and less than 10, we can create a Boolean array b using the expression `b = (a > 5) & (a < 10)` and then use this Boolean array to index into a as follows: `a[b]` .
- Overall, NumPy Array Filtering is a powerful technique that allows us to extract specific subsets of data from a larger array based on a variety of criteria, making it a valuable tool for data analysis and manipulation.

Array Filtering Example:
In this example, we create an array 'a' with some integer values. We then create a Boolean array 'b' using the expression `b = (a > 5) & (a < 10)` , which evaluates to True for all elements in a that are greater than 5 and less than 10.

Finally, we use boolean indexing to filter the array a based on the Boolean array b. The resulting filtered array only contains the elements of a that satisfy the condition specified in b.

In [ ]:
```
# Steps to filter a NumPy array "a" based on a Boolean array "b" created using the expression b = (a > 5) & (a < 10):
import numpy as np
```

```
# create an array 'a'
a = np.array([1, 2, 6, 7, 9, 12])

# create a boolean array 'b'
b = (a > 5) & (a < 10)

# filter the array 'a' using boolean indexing
filtered_array = a[b]

print(filtered_array)  # output: [6 7 9]
```

```
[6 7 9]
```

Homework Questions

1) Checking the data type of array

2) Create two Arrays & Stack them vertically

3) Create two Arrays & Stack them horizontally

4) dtype of the given array object is ‘int32’. Now change this to ‘float64’ type.

5) convert int into strings of arr [1, 2, 3, 8, 7, 5]

6) Stack the given arrays along columns, arr1=[1, 2, 3] & arr2=[12,13,14]

7) Stack the given arrays along Rows, arr1=[1, 2, 3] & arr2=[12,13,14]

8) Join the given arrays arr1=[15,16,17] & arr2=[78,89,45]

9) Create a 1D array & then Convert it into a 2D array of shape 4x2 & 2x4

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