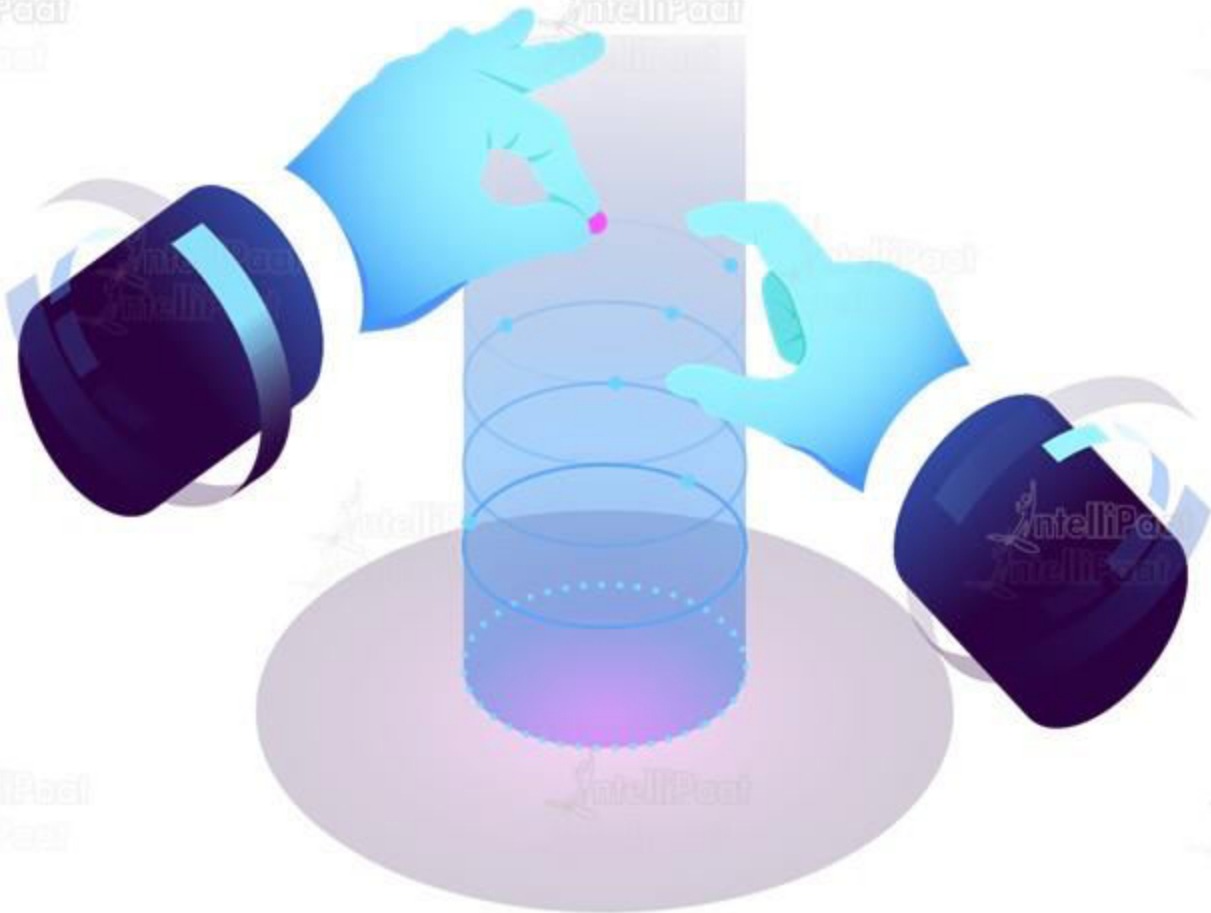
# Data Science with Python

## Data Manipulation



# Agenda

01 What is Data Manipulation?

02 Why Data Manipulation?

03 What is NumPy?

04 Why NumPy over Lists?

05 What is a NumPy Array?

06 Applications of NumPy

07 Create a NumPy array

08 Array initialization

09 Array inspection

10 Array mathematics

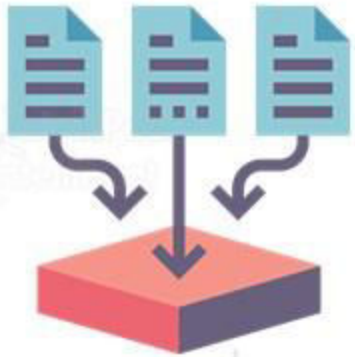
11 Indexing, Slicing and Iterating in NumPy

12 Array Manipulation

# What is Data Manipulation?

# What is Data Manipulation?

Data Manipulation is the process converting data into a format that is easy to process and is more organized



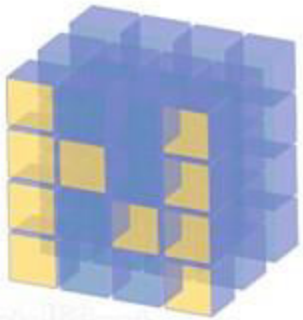
Data extraction from multiple sources

Manipulating data using Python

Organized and readable information

# What is Data Manipulation?

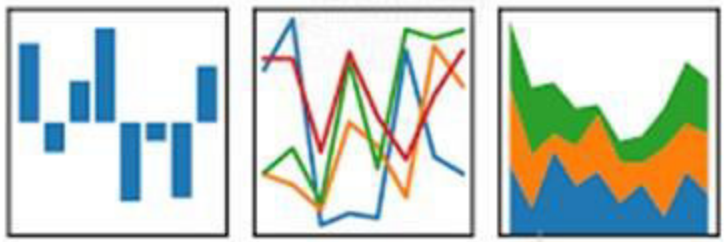
These are the popular Python libraries for data manipulation



NumPy

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



SciPy

matplotlib

# Why Data Manipulation?

# Why Data Manipulation?

As a Data Scientist, we gather large volumes of data from multiple sources. Many times data from these sources has problems

# Why Data Manipulation?

There could be multiple problems with the data some of those are:

- Missing Values
- Incorrect Format
- Different Units
- Unnecessary Data

Since the data is accumulated from multiple sources some values in a row might be missing. This could be due to multiple reasons such as equipment not being available, user not willing to share data etc.

# Why Data Manipulation?

There could be multiple problems with the data some of those are:

- Missing Values
- Incorrect Format
- Different Units
- Unnecessary Data

Sometimes data from different sources might be formatted differently such as having different date formats or formats of currency etc.

# Why Data Manipulation?

There could be multiple problems with the data some of those are:

| Missing Values | Incorrect Format | Different Units | Unnecessary Data |
|---|---|---|---|

Different sources might also have different Units of measurements such as temperature being measured in Celsius, Fahrenheit and Kelvin or distance being measured in Miles and Kilometers etc.

# Why Data Manipulation?

There could be multiple problems with the data some of those are:

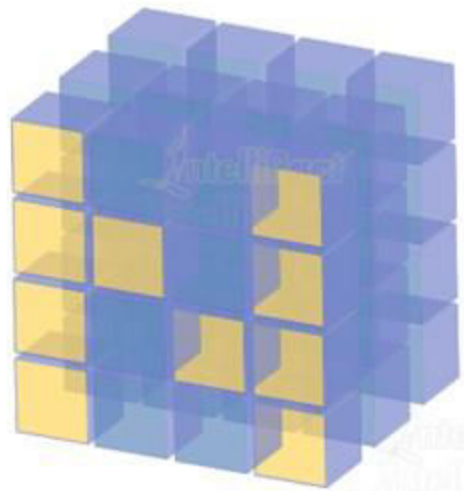
- Missing Values
- Incorrect Format
- Different Units
- Unnecessary Data

Sometimes we have a large dataset with columns that contain values that are not relevant to the tasks that you are performing. For example Some datasets have unique id columns which are not important.

# What is NumPy?

# What is NumPy?

The NumPy library is a very popular Python library and the abbreviation is "Numerical Python". The purpose of NumPy library is to do scientific computation and apply them to python applications.



NumPy

# What is NumPy?

**How will NumPy help in Data Science?**

1. First of all, It is a open source Python library
2. It is fast because it is written in C and Python
3. In Python, there is no in-built array capabilities
4. You can use List as an alternative for arrays, but NumPy is better. But how is it better? We will discuss that now.

# What is NumPy?

**What features does NumPy provide?**

1. A durable N-Dimensional array
2. Tools to Manipulate and work with the array
3. Perform mathematical and logical operations on the array
4. Powerful pre-defined functions

# What is NumPy?

**Operations of the NumPy Library**

1. Fourier Transform and Shape Manipulation
2. Linear Algebra and Random Number Generation
3. Tools for integrating C/C++
4. Easily integrate with databases

# Why NumPy over Lists?

# Why NumPy over Lists?

01 NumPy arrays consume lesser memory due to contiguous memory allocation

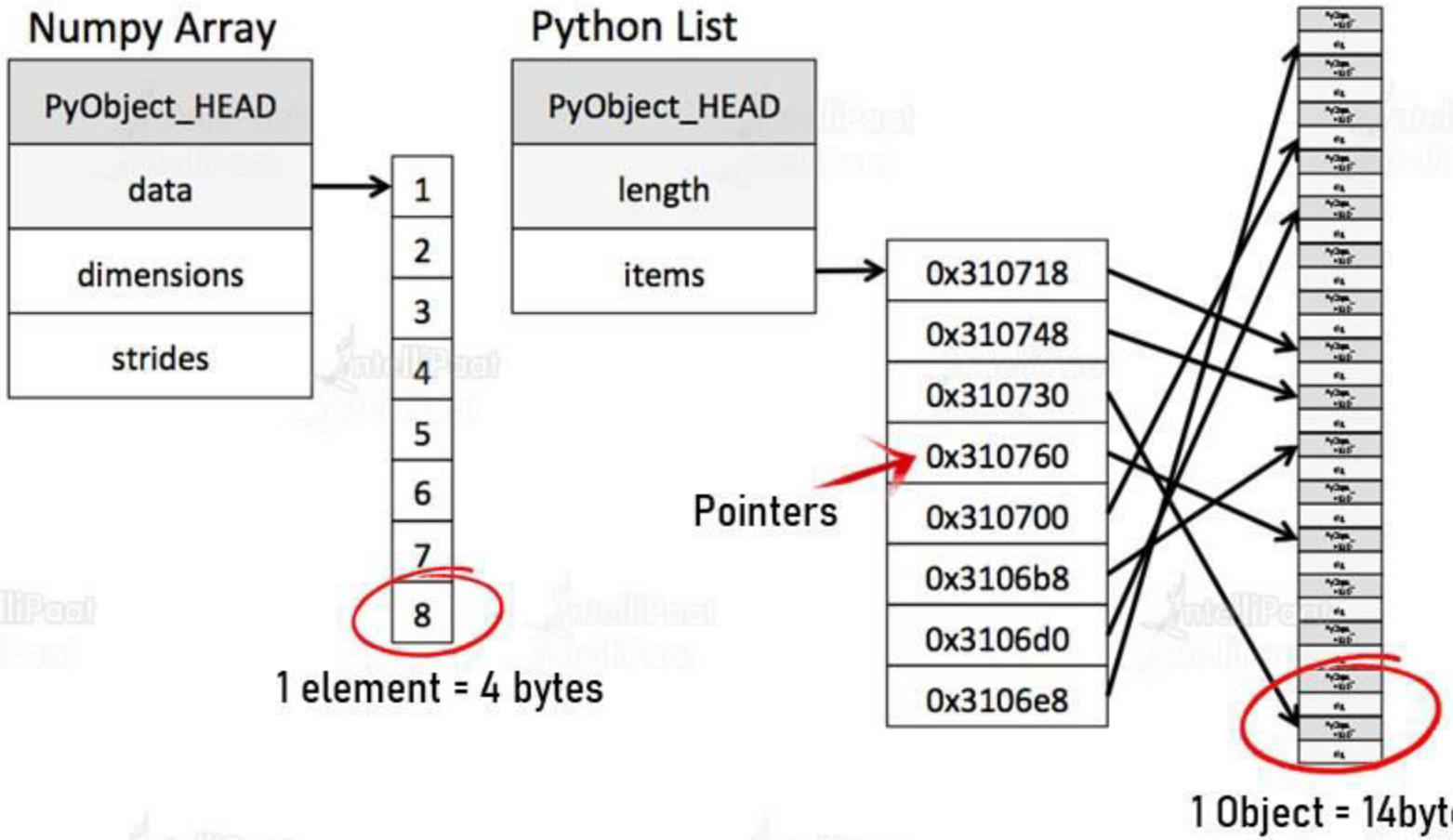
02 Operations complete quicker in NumPy because of its better runtime behavior

03 Pre-defined functions for linear algebra operations are available

04 NumPy arrays have fixed types, so there is no type checking in execution which saves time

# Why NumPy over Lists?

A simple example of why NumPy arrays are better than Python Lists

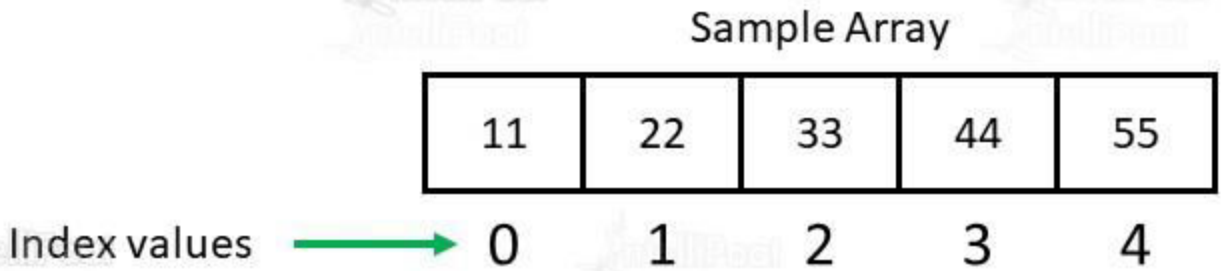


# What is a NumPy Array?

# What is a NumPy Array?

Array is the data structure which the NumPy library revolves around. To be more precise, it is a matrix/grid of values which are all of the same data type. It is also called a **ndarray** because it is a N-Dimensional array.

The are indexed using non-negative integers starting from 0. NumPy array act similar to a Python list, but is totally different in execution and is relatively a lot quicker.



# Applications of NumPy

# Applications of NumPy



01 Can be used of Mathematical operations and also can be an MATLAB alternative

02 For any Backend code, NumPy is important because Pandas & NumPy work well together

03 Helps in plotting while using the Matplotlib Python library

04 A important part of Machine Learning and Data Science algorithms

# Create a NumPy array

# Create a NumPy array

First of all, to create an NumPy array you should have it installed

Before installing NumPy, make sure you have Python installed. But as we installed Anaconda, NumPy comes with it. If you are not using Anaconda, then use the below commands

**Using PIP**

```
python -m pip install --user numpy
```

**Ubuntu/Debian**

```
sudo apt-get install python-numpy
```

**Fedora/CentOS**

```
sudo dnf install numpy
```

**Mac**

```
brew install numpy
```

# Create a NumPy array

The first step to create a NumPy array is to import the NumPy package

```
import numpy as np
```

**import** – This command is to get access to another module/library to get access to their code

**numpy** – This is the module which we should get access to do any NumPy related operations

# Create a NumPy array

Different ways to create a NumPy array

01 Converting other Python structures to arrays (Lists, tuples)

02 Using NumPy array creation objects or NumPy pre-defined functions (ones, range, zeros, etc)

03 Use of special Library functions (random)

# Create a NumPy array

Different ways to create a NumPy array

01

```
In [2]: np.array([1, 2, 3])
Out[2]: array([1, 2, 3])
```

02

```
In [3]: import numpy as np
        np.zeros((3,4))
Out[3]: array([[0., 0., 0., 0.],
               [0., 0., 0., 0.],
               [0., 0., 0., 0.]])
```

03

```
In [11]: import numpy as np
         np.random.random((2,2))
Out[11]: array([[0.49123681, 0.92284889],
                [0.40263909, 0.19302602]])
```

# Hands-on: Create NumPy Array

# Array Initialization

# Array Initialization

The difference between creating an array and initializing the array is that while you create, the array can be empty as well. But when you initialize, you are actually entering values in the array.

You can initialize a NumPy array in all the ways you create an array. While you use those pre-defined functions, they initialize the array after creating it

# Array Initialization

Few examples of Array Initialization

Filling the same number throughout the array

```
In [9]: import numpy as np
        np.full((2,2),5)

Out[9]: array([[5, 5],
               [5, 5]])
```

Arranging numbers between **X (10)** & **Y (25)**, with an interval of **Z (5)**

```
In [4]: import numpy as np
        np.arange(10,25,5)

Out[4]: array([10, 15, 20])
```

Arranging **Z (6 & 5)** numbers b/w **X (5)** and **Y (10)**

```
In [6]: import numpy as np
        np.linspace(5,10,6)

Out[6]: array([ 5.,  6.,  7.,  8.,  9., 10.])
```

```
In [7]: import numpy as np
        np.linspace(5,10,5)

Out[7]: array([ 5.  ,  6.25,  7.5 ,  8.75, 10.  ])
```

# Array Inspection

# Array Inspection

**Why do you need Array inspection?**

Any real world problem involving data will have millions of rows and thousands of columns. So, if you are using or creating a DS algorithm to manipulate that data, then it will be very helpful if you are able to inspect the structure of your arrays

# Array Inspection

| Inspect Functions | Description |
|---|---|
| **ndarray.shape** | Gives a tuple with the array dimensions. You can also use this function to resize the array |
| **ndarray.size** | Returns the count of number of elements in the given array |
| **ndarray.ndim** | Provides the dimension of the given array |
| **ndarray.dtype** | Returns the datatype used by the array (eg. Int32, float64) |

# Array Inspection

**ndarray.shape**

Tuple with the array dimensions

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

        (2, 3)
```

Changing the shape of an array

```
In [7]: # this resizes the ndarray
        import numpy as np

        a = np.array([[1,2,3],[4,5,6]])
        a.shape = (3,2)
        print (a)

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

# Array Inspection

**ndarray.size**

```
In [13]: import numpy as np
         a = np.arange(24)
         print(a.size)

         24
```

**ndarray.ndim**

```
In [12]: import numpy as np
         a = np.arange(24)
         print(a.ndim)
         b = a.reshape(2,4,3)
         print(b.ndim)

         1
         3
```

# Array Inspection

**ndarray.dtype**

```
In [14]: import numpy as np
         a = np.arange(24,dtype = float)
         print(a.size)
         print(a.dtype)
         b =a.reshape(3,4,2)
         b

         24
         float64

Out[14]: array([[[ 0.,  1.],
                 [ 2.,  3.],
                 [ 4.,  5.],
                 [ 6.,  7.]],

                [[ 8.,  9.],
                 [10., 11.],
                 [12., 13.],
                 [14., 15.]],

                [[16., 17.],
                 [18., 19.],
                 [20., 21.],
                 [22., 23.]]])
```

# Demo: Array Inspection

# Array Mathematics

# Array Mathematics

1. `np.sum(a,b) #a+b` → Addition of a and b
2. `np.subtract(a,b) #a-b` → Difference of a and b
3. `np.divide(a,b) #a/b` → Dividing a with b
4. `np.multiply(a,b) #a*b` → Multiplying a and b
5. `np.exp(a) #e^a` → e^a where e is the Euler's number (2.718)
6. `np.sqrt(a)` → Square root of a
7. `np.sin(a)` → Sine value of angle in degrees
8. `np.cos(a)` → Cosine value of angle in degrees
9. `np.log(a)` → Calculates natural logarithm

# Array Mathematics

You can do all these mathematical operations with the use of an existing function which makes using NumPy much simpler

**Addition**

```
In [3]: import numpy as np
        np.sum([10, 20])
Out[3]: 30
```

```
In [2]: a,b=10,20
        np.sum([a,b])
Out[2]: 30
```

```
In [5]: np.sum([[0, 1], [0, 5]], axis=0)
Out[5]: array([0, 6])
```

```
In [6]: np.sum([[0, 1], [0, 5]], axis=1)
Out[6]: array([1, 5])
```

# Array Mathematics

**np.equal** → Checks if elements or an whole array is equal or not by comparing

**Element-wise comparison**

```
In [7]: import numpy as np
        a = [1,2,4]
        b = [2,4,4]
        c = [1,2,4]
        np.equal(a,b)

Out[7]: array([False, False,  True])
```

```
In [8]: import numpy as np
        a = [1,2,4]
        b = [2,4,4]
        c = [1,2,4]
        np.equal(a,c)

Out[8]: array([ True,  True,  True])
```

**Array-wise comparison**

```
In [9]: import numpy as np
        a = [1,2,4]
        b = [2,4,4]
        c = [1,2,4]
        np.array_equal(a,b)

Out[9]: False
```

# Array Mathematics

Aggregate functions for Mean, Median, Standard deviation are also readily available

```
In [10]: import numpy as np
         a = [1,2,4]
         b = [2,4,4]
         c = [1,2,4]
         print(np.sum(a)) #Array wise sum
         print(np.min(a)) #Min of an array
         print(np.mean(a)) #Mean of the array
         print(np.median(a)) #median of the array
         print(np.corrcoef(a)) # correlation coefficient of array
         print(np.std(a)) #Standard Deviation of array

         7
         1
         2.3333333333333335
         2.0
         1.0
         1.247219128924647
```

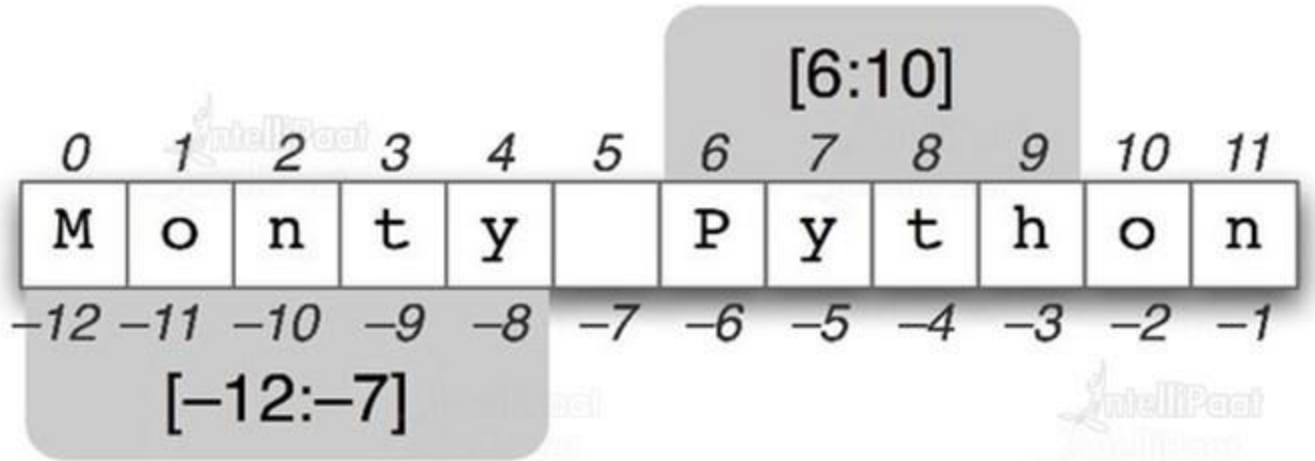
# Demo: Array Mathematics

# Indexing and Slicing in NumPy

# Indexing and Slicing in NumPy

**Indexing**

Index refers to the position of an element in the array



# Indexing and Slicing in NumPy

**Slicing**

Using slicing you can build new arrays out of an existing array. You can also slice across multiple dimensions in NumPy arrays. The data will be fetched from the same location, but in a different order

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 |
| 2 | 7 | 8 | 9 |

**Let's learn to extract/slice the array**

# Indexing and Slicing in NumPy

**Slicing**

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 |
| 2 | 7 | 8 | 9 |

**How to extract the selected element?**

My selection is in 1st row = 0th index

```
A[0] ------#includes all the elements from the first row
A[:1] ------ #Extract first row from the array.
```

# Indexing and Slicing in NumPy

**Slicing**

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 |
| 2 | 7 | 8 | 9 |

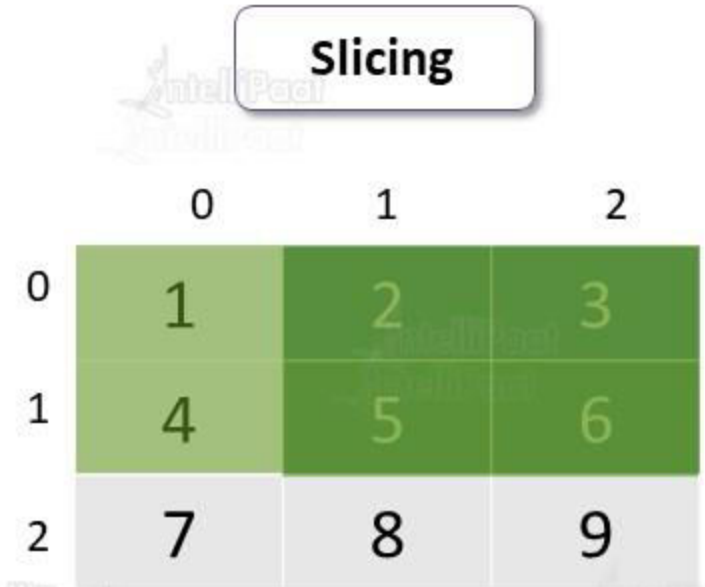
**How to extract the selected element?**

My selection is in 1st row = 0th index

```
A[:1] ------#Extracting till row = 0 (that is 0th row)
A[:1,1:]------#Extracting till row = 0 then select
```

the col index starting from 1 till last

# Indexing and Slicing in NumPy




**How to extract the selected element?**

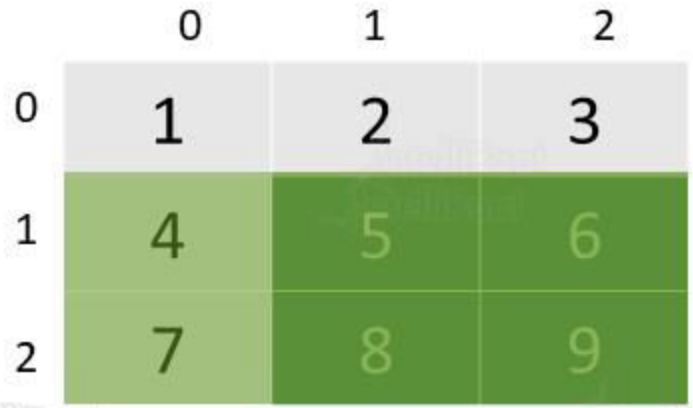
My selection is in 1st two rows = 0,1 index

```
A[:2] ------#Extracting till row = 1 (that is 0,1)
A[:2,1:] ------ #Extracting till row = 1 (that is 0,1) then
```

select the col index starting from 1 till last

# Indexing and Slicing in NumPy

**Slicing**



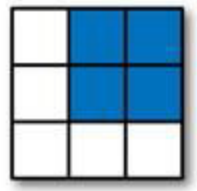
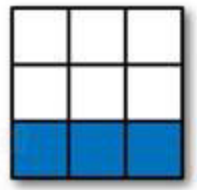
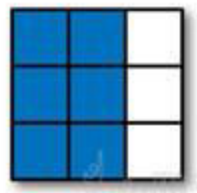
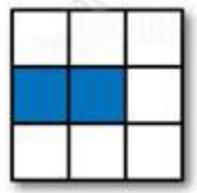

**How to extract the selected element?**

My selection is in 1st two rows = 0,1 index

```
A[1:,] ------ #Extracting starts from row = 1 till end
A[1:,1:] ------ #Extracting starts from row = 1 till end
then select col index = 1 till end
```

# Indexing and Slicing in NumPy

**Slicing**

| | Expression | Shape |
|---|---|---|
|  | `arr[:2, 1:]` | (2, 2) |
|  | `arr[2]` | (3,) |
| | `arr[2, :]` | (3,) |
| | `arr[2:, :]` | (1, 3) |
|  | `arr[:, :2]` | (3, 2) |
|  | `arr[1, :2]` | (2,) |
| | `arr[1:2, :2]` | (1, 2) |

Slicing the first 2 elements of the array

```
In [2]: arr = np.array([1, 2, 3, 4, 5])
        print(arr[:2])

        [1 2]
```

# Demo: Array Indexing and Slicing

# Array Manipulation

# Array Manipulation

| Manipulation Functions | Description |
|---|---|
| **numpy.concatenate(a,b)** | Concatenates two strings together and returns the result |
| **numpy.vstack(a,b)** | Stacks arrays provided in a Row-wise order (vertically) |
| **numpy.hstack(a.b)** | Stacks arrays provide in a Column-wise order (horizontally) |
| **numpy.column_stack()** | Stacks the elements of multiples arrays vertically as multiple arrays |
| **numpy.hsplit()** | Splits an array into multiple sub-arrays in a column-wise fashion |

# Array Manipulation

## Concatenation

```
In [21]: np.concatenate((a,b), axis = 0)
Out[21]: array([1, 2, 4, 2, 4, 4])
```

## Vstack

```
In [22]: np.vstack((a,b))
Out[22]: array([[1, 2, 4],
                [2, 4, 4]])
```

## Hstack

```
In [23]: np.hstack((a,b))
Out[23]: array([1, 2, 4, 2, 4, 4])
```

## Column wise

```
In [24]: np.column_stack((a,b))
Out[24]: array([[1, 2],
                [2, 4],
                [4, 4]])
```

# Array Manipulation

Hsplit example

```
In [31]: x = np.arange(16.0).reshape(4, 4)
         print(x,"\n\n")
         print(np.hsplit(x, 2),"\n\n")
         print(np.hsplit(x, np.array([3, 6])))
```

```
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
 [12. 13. 14. 15.]]


[array([[ 0.,  1.],
        [ 4.,  5.],
        [ 8.,  9.],
        [12., 13.]]), array([[ 2.,  3.],
        [ 6.,  7.],
        [10., 11.],
        [14., 15.]])]


[array([[ 0.,  1.,  2.],
        [ 4.,  5.,  6.],
        [ 8.,  9., 10.],
        [12., 13., 14.]]), array([[ 3.],
        [ 7.],
        [11.],
        [15.]]), array([], shape=(4, 0), dtype=float64)]
```

# Hands-on: Array Manipulation

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24/7 Chat with Our Course Advisor