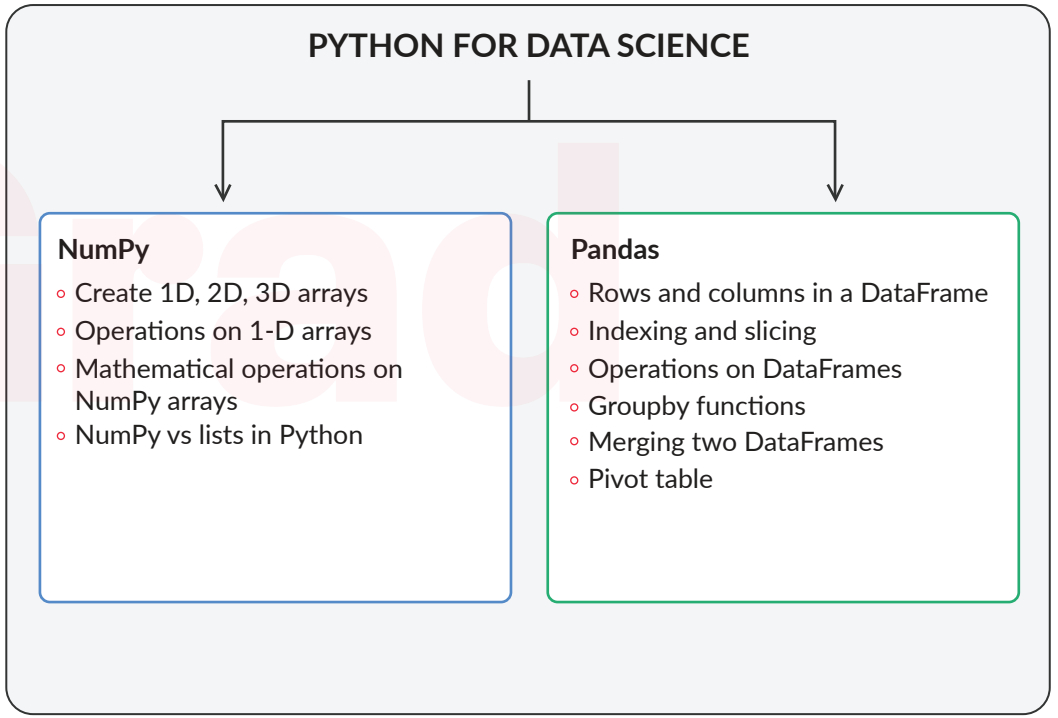
PYTHON FOR DATA SCIENCE

You learnt about Python's two most essential and popular libraries, NumPy and Pandas.

- You studied NumPy arrays in different dimensions and performed various mathematical operations on NumPy. NumPy offers an enormous library of high-level mathematical functions that efficiently operate on arrays and matrices
- Then, you learnt about Pandas which is built on top of NumPy. Pandas allow you to slice, index, and execute other DataFrame operations that are useful for cleaning and analysing data

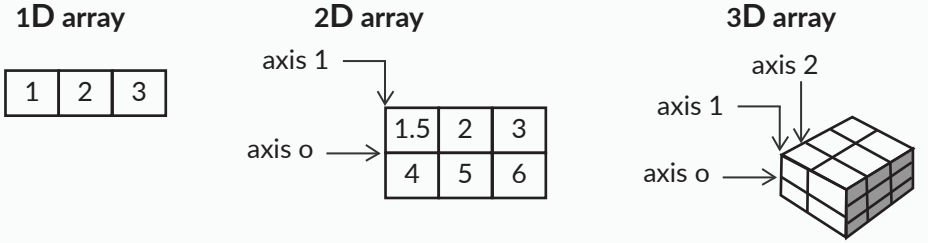
Common Interview Questions:

1. What is NumPy?
2. How is vstack() different from hstack() in NumPy?
3. List the advantages NumPy Arrays have over (nested) Python lists.
4. How do you convert a Pandas DataFrame to a NumPy array?
5. What are the different types of data structures in Pandas?
6. What are the most important features of The Pandas library?
7. How do you get the frequency count of the unique items in a series?
8. What are the different ways of creating DataFrame in Pandas? Explain with examples.
9. How are loc and iloc different in Pandas?
10. How does the groupby() method works in Pandas?



NUMPY(import numpy as np)

NumPy array




```
a = np.array([20,24,28,32, 36, 40]) #1D array
b = np.array([(1.5,2,3),(4,5,6)],dtype = float)#2D array
c = np.array([[(1.5,2,3),(4,5,6)],[(3,2,1),(4,5,6)]], dtype = float)
#3D array
```

Slicning

1D array

```
a[:]   #[20, 24, 28, 32, 36, 40] #selects everything
a[2:5] #[28,32,36] #Selects the 2nd through the 4th rows (does not include
the 5th row)
```

2D array

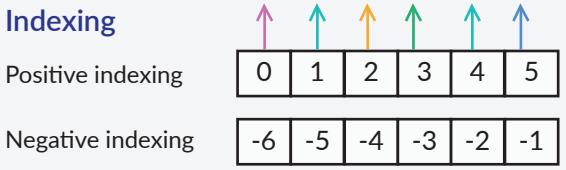
```
b[:,:] #[[1.5, 2., 3.),[4., 5., 6. ]] #Selects all rows and all columns
b[:,0] [1.5, 4. ) #Selects all rows, and the zeroth column
b[0,:] #[1.5, 2., 3. ) #Selects the zeroth row, and all columns in that row
b[0:2,:] #[[1.5,2.,3.1,[4.,5.,6.]]#Selects the zeroth and first row,
but NOT the second
b[0:2,0:2] #[1.5,2.],[4.,5.]]#Selects the zeroth and first row, and the
zeroth and first column
```

Note: For three-or more-dimensional arrays, the slicing method remains similar

Indexing

```
a = np.array([20,24,28,32,36,40])
```

Indexing



Syntax: array[index]

Mathematical operations:

```
a.sum()  #180; Sum of all elements
a.min()  #20; To find minimum
a.max()  #40; To find maximum
a.mean() #30; Average of all numbers
a/4      #[5., 6., 7., 8., 9., 10.) #Rowwise operation
```

Array Manipulation

```
a1 = np.array([20, 21, 22, 23, 24, 25])
a2 = ntp.arange(6) #[0, 1, 2, 3, 4, 5]
a1.reshape (2,3) #Reshaping arrays without changing data
                 #[[20, 21, 22],
                 #[23, 24, 25]]
np.concatenate((a1, a2)) #Concatenate arrays
np.hstack((a1, a2)) #Stack arrays horizontally
                  #[20, 21, 22, 23, 24, 25, 0, 1, 2, 3, 4, 5]
np.vstack((a1, a2)) #Stack arrays vertically
                    #[[20, 21, 22, 23, 24, 25),
                    #[ 0, 1, 2, 3, 4, 5]]
```

PANDAS(import pandas as pd)

NumPy array

- Pandas Series
- Pandas DataFrame

Pandas Series:

```
s = pd.Series([3,-5,7,4], index=[1,2,3,4])
```

Output

```
1    3
2   -5
3    7
4    4
dtype: int64
```

Create a DataFrame from a dictionary:

Syntax: pd.DataFrame(dictionary_name)

Read an external CSV file:

Syntax: pd.read_csv(filepath, sep = ', ', header = ' infer')

- separator (by default ',')
- header (takes the top row by default, if not specified)
- names (list of column name)

```
df = pd.read_csv('country.csv',index_col=0)
```

Country_name	Capital	Population_in_millions
India	New Delhi	1393.40
Brazil	Brasília	201.00
Canada	ottawa	38.23

Basic information about DataFrame

```
df.info()  #Information about DataFrames
df.describe() #To get statistical information like mean,
median, mode, percentile
df.head()  #To identify the first five rows in a DataFrame
df.sort.index() #Hierarchical indexing
df[start_index:end_index] #Subset the rows according to the
start and end indices
```

#Conditional operator

```
df[df['Population_in_millions']>100]
```

	Country_name	Capital	Population_in_millions
0	India	New Delhi	1393.4
1	Brazil	Brasília	201.0

loc vs iloc in Pandas DataFrame

#loc selects rows and columns with specific labels

```
df.loc[[0,1], ['Country_name']]
```

	Country_name
0	India
1	Brazil

#iloc selects rows and columns at specific integer positions

```
df.loc[[1,2] #Element in first row and secound column
```

214.0

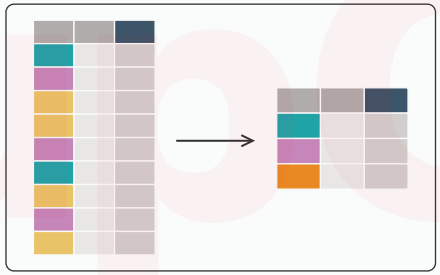
PANDAS(import pandas as pd)

Statistical summary in Pandas

```
df.sum() #Sum values of each object
df.cumsum() #Cummulative sum values of each object
df.min()/df.max() #Min/max value of each object
df.idxmin()/df.idxmax() #Min/Max index value of each object
df.mean() #Mean of each object
df.median() #Median of each object
df.median() #Standard of each object
```

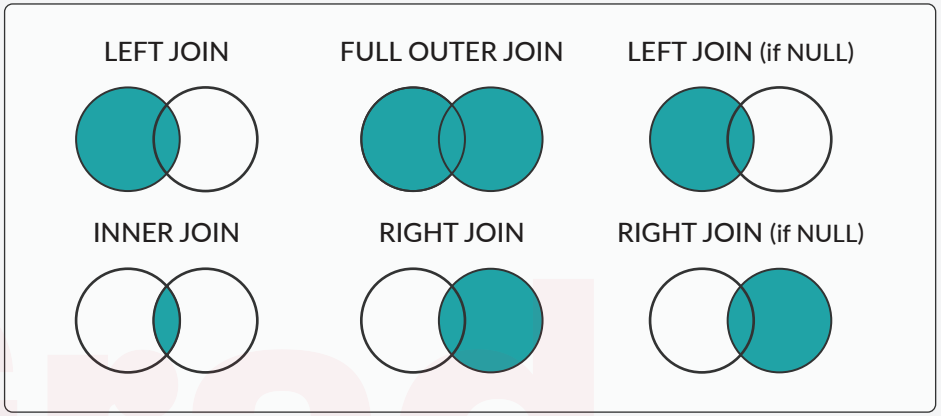
GroupBy function:

```
DataFrame.groupby(by['col_name'])
df.groupby(by="col") #Return a GroupBy object, grouped by values in column named "col".
df.groupby(level="ind") Return a GroupBy object, grouped by values in index level named "ind".
```



Merging two DataFrames in Pandas

df_1.merge(df_2, on = ['column_1', 'column_2'], how = '____')
The attribute 'how' specifies the type of merge that is to be performed.
Merges are of several types as shown below:



- left: Selecting the entries only in the first DataFrame.
- right: Considering the entries only in the second DataFrame
- outer: Union of all the entries in the DataFrames
- inner: Intersection of the keys from both DataFrames

Pivot Table: #Summarise a DataFrame. Pivot table works like groupby function but it represents a data in a structured and simplified manner

```
df.pivot(columns='grouping_variable_col',
values='value_to_aggregate', index='grouping_variable_row')

df.pivot_table(values, index, aggfunc=
{'value_1': np.mean,'value_2': [min, max, np.mean]})
```