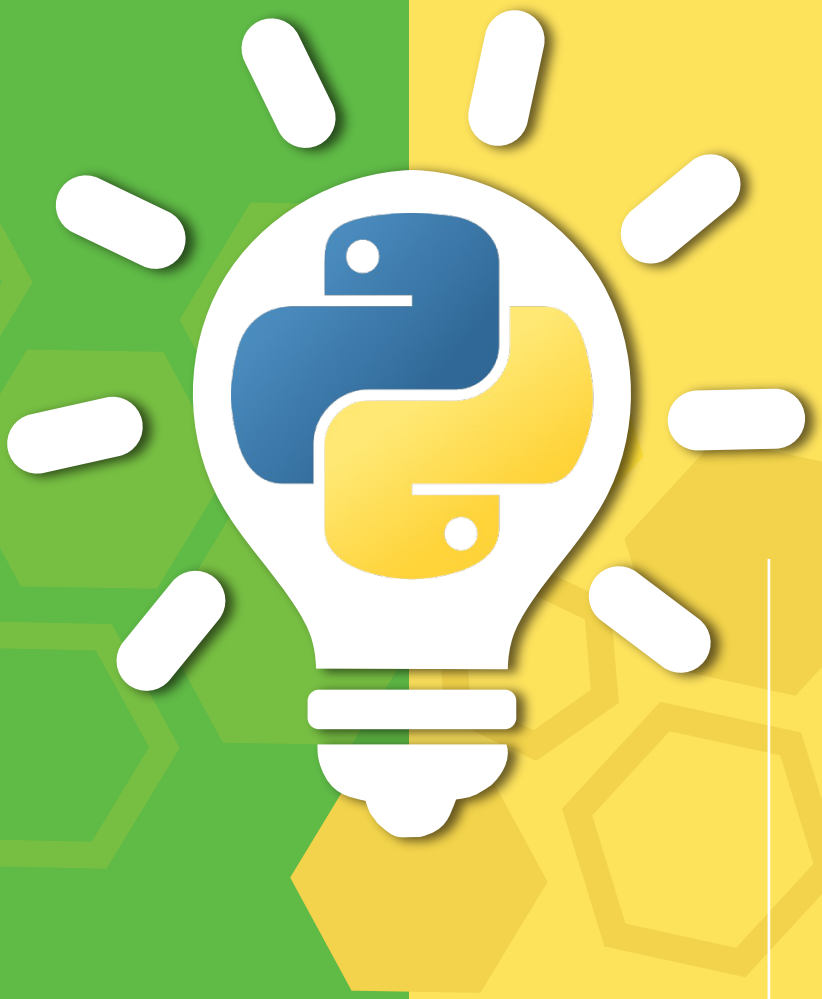
**Follows TIY**
**( Teach it Yourself )**
**Approach**

# PYTHON PROJECTS, LIBRARIES, MODULES & MORE

SECONDARY SCHOOL & COLLEGE



**In Line with CBSE & in Tune with Child's Aspirations & their Technological Future**

# LIST OF CONTENT

CHAPTER

# 01 Migration from IDLE to IDE

Migration from IDLE to IDE is the process of taking you from the status of a python **STUDENT** to a python **PROFESSIONAL.** Till now, all our learning has been **theoretical** based on python IDLE. As we shift from theory to **practical**, we will **shift from** python IDLE to python IDE. The good news is that whatever we have learnt using an **IDLE** is applicable to working with IDE.

## 1 Need for Migration from IDLE to IDE

Say we are **making an interactive app.** We ask a person to give us two random nums one by one. We then add them & give the answer. This would require two lines of code.
Using an IDLE the first line of code would be:

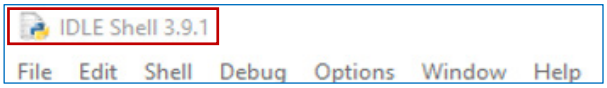
```
>>> num1 = float(input('Enter first number: '))
```

However, moment we press enter, it gets executed (see blue) & we cannot enter 2nd num.

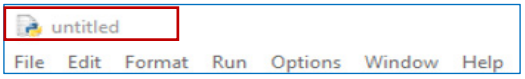
```
>>> num1 = float(input('Enter first number: '))
Enter first number:
```

IDLE has a **solution.** However, we did not use it earlier, because that was not the right time for its use. We wanted to **focus on fundamentals.** This solution is a **bridge** between IDLE & an IDE. If you understand this difference, you will easily understand IDE.

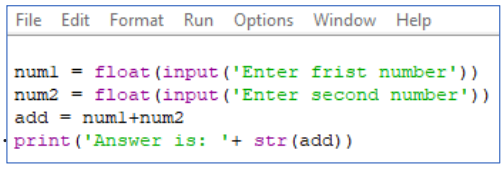
**To understand the problem:**
Click on file in **IDLE shell.**




In dropdown, select new file.
A new shell named **untitled** will open.



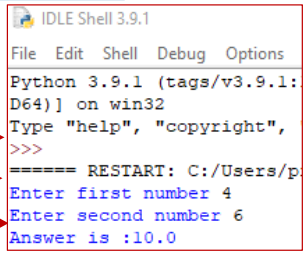

This shell in reality is **akin to an IDE shell.**
Now, write the same lines in this untitled shell.



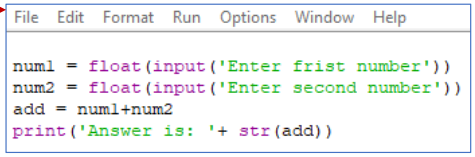

Having written them, we need to run them.
This is where a small **procedural problem** comes in.

To run this code, in untitled shell , go to **run.**


In dropdown select **run module.** Python asks to save the project. Click ok. In new window mention the name & place to save (**Demo.py.C:/user/.....**).
Moment, you select save, this window opens & Python asks you to enter the 1st num. Then the 2nd.
Moment, it is entered, output is displayed below the nums.




**Point to note** is **input is entered in one shell**
**Output appears in another (Red shell above).**



From **beginners, perspective** the code being split between two shells is undesirable.

Now take your mind to the **Learning** so far. It was based on codes executed to show results at the end. This made **it fun.**

**If learning was based on above** procedure, for each code, you would have found saving etc laborious & soon lost interest. **It was for this reason that we refrained from burdening you with too many projects earlier. We shall do it now using a Python IDE.**

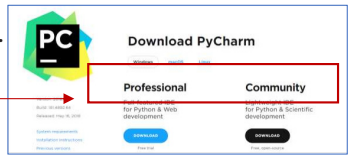
## 2 Choice of IDE's

Teaching will be based on **PyCharm.** What you learn using this IDE will apply to others IDE's as well. In vol 3 we shall introduce you to some other IDE;s.

When we start coding devices for real life projects, we shall shift to **mBlock 5.**

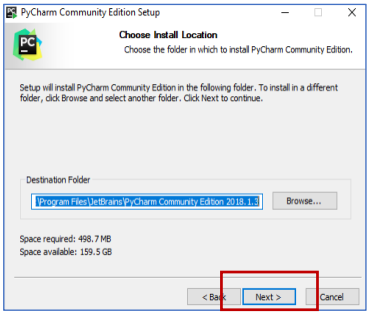

## 3 Downloading of PyCharm IDE

https://www.jetbrains.com/pycharm/download/#section=windows. Open google & enter this link. A window with 2 options opens:




- **Professional** gives free rial for few days.
- **Community** is free. Select community. Download starts:

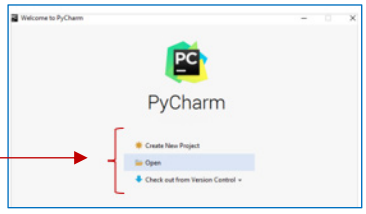
Now create a folder on your desktop. Go to browse & choose the created folder. IDE will get installed. In that:




- Click on next. A new window opens.
- Click on **install.** Installing process will start.
- A new window opens. Click on finish.

PyCharm IDE is now **installed** in the designed folder in your PC. **Procedure to download IDE's other than PyCharm is similar.**

**3.1. Understanding PyCharm IDE Screen.** When you launch PyCharm for the **first time,** you can see a welcome screen with entry points to IDE such as: Create new project. Open, View documents. Select **create new project** (1st option).



## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 02 File Handling

## 1 Lesson Overview

In has lesson we shall learn what are files, what is file handling, how does it work in Python, & the various file handing operations offered by python.

When we are working on a computer, the information or data or our work keeps getting stored in a volatile random access memory **(RAM)**. This is a **temporary storage facility.** You would have noticed if during this time the computer is switched off without saving, your work gets lost. Since we want this data for later use, we store it in a non volatile memory like a hard disk. This is like a **permanent storage facility.**

## 2 File Handling

**What are Files?** They are nothing but **named locations,** for storing collections of data, on a secondary memory like a **hard disk** of a computer. They are a collection of data that is stored on a computer, smartphone, or other electronic device. Files can contain many types of data, including Text, Images, Audio, Video, Executable code, & Instructions for the computer to perform tasks. They are essential for storing, organizing, and managing data on digital devices. They can be created, modified, read, and deleted by users or software applications.

Python's data structure provides an important feature for reading data from the file & also writing data into a file. This enables us to create, update, read, & delete files stored on the file system. It is an integral part of its OS.

## 3 Working on Files

To read from or write to a file, we need to open it. This is a temporary activity. When we are done, it needs to be closed, so that the resources tied with the file are freed.

### 3.1. File Handling By Python.

Python undertakes its file handling operations in three steps:

- Open a file from its storage.
- Read or write or perform operation in it.
- Close the file & place it back in the same storage loc or a different one.

Learning of these has been divided in to two parts:

- File **Opening.**
- File **Operations.**

### 3.2. File Opening.

- **Opening a File.** Python has a built-in func **open()** to open a file. This func returns a file object called a handle. This is used to carry out operations on the file.
- **Handle.** This is a **temporary reference** (typically a number) automatically assigned by the operating system to a file that an application has asked it to open. The handle is used throughout the session to access the file. This temporary ref num, removes the trouble of remembering the name of a file that is currently opened in the system.

**3.3. File Name & Mode. open()** func takes two parameters – File name & file mode.

- **File Name.** As the name implies, this is the name users need to give to this temporary
- reference, to enable its storage & retrieval as a saved file.
- **File Mode.** Mode determines:
  - Where the file is to be **positioned** when opened.
  - What functions are **allowed.**
- After you close a file, you can reopen the file in the same mode or in a different mode, depending on what operation you want to do.

**3.4. Types of File Opening Modes.** Type of mode to be used depends on the intended operation. For ex, in order to **write data** into a file, we must open the file in **write mode.** We need to be very careful while writing data into the file as **it overwrites** the content present inside the file that you are writing, & the previous data will be erased.

There are four modes for opening of a file:

- **"r"** - Read - Default mode. Opens a file for reading, error if the file does not exist.
- **"a"** - Append - Opens a file for appending, creates the file if it does not exist.
- **"w"** - Write - Opens a file for writing, creates the file if it does not exist.
- **"x"** - Create - Creates the specified file, returns an error if the file exists.

In addition, we can specify if the file should be handled as text or in binary mode:

- **"t"** - Text - Default value. Text mode.
- **"b"** - Binary - Binary mode (ex images).

**3.5. Syntax to Open a Text File.** To open a text file for reading the syntax will be: `f = open('demofile.txt', 'rt')`
However, because **"r"** for read, & **"t"** for text are the default values, `f = open('demofile.txt')` you do not need to specify them. it is enough to specify the name of the file:
**Note:** Use any syntax. Will get an error if file does not exist.

## 4 File Handling Operations

**4.1. Saving a File & its Storage.** Before we start any project, we need to define a loc where this project will be saved. Once this location has been defined this loc will be a permeant loc for this project (project loc). Say the name of our project is 'file_opening'. In our case it is saved in **C < users < desktop < pycharm_projects < file_opening.**

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 03 Arrays & Lambda Functions

## 1 Arrays

Python arrays are a **data type like Lists.** They have a multi element structure like lists. Lists elements are heterogeneous, while **array elements are homogeneous.** They can be **iterated** & have several built-in methods to handle them. We can use these to retrieve or change data.

**1.1. Real Life Applications.** Arrays are used everywhere in data processing. Ex:

- **Save contact nums.** The contact module on our phone stores these nums in the form of an array. Adding or deleting a contact in our phone, is done by adding or deleting elements in its array.
- **On-line ordering.** As we add products to the cart, they get stored in the cart module in the form of an array.
- **Data Matrices.** These are nothing but 2D arrays widely used in marking surveys. They represent graphs & statics used in almost all walks of our life by all of us.

**1. 2. Similarities & Differences of Arrays with Lists**

- Both are used for storing data.
- Both are mutable.
- Both can be indexed & sliced.
- Both can be looped through & iterated upon.

**1.3. Differences between List & Array:** They lie in the **operations we can perform** on them:

| Parameter | List | Array |
|---|---|---|
| Structure & declaration | Built-in. No need to declare | Needs to import array module or NumPy licbrary in order to declare |
| Data types | Supports mixed data types | Must be of same type |
| Size | Can be resized | Have a constant unalterable size |
| Data size | Preferred for small data | Must for large data size |
| Display | Elements can be displayed without a loop | Requires a loop to display elements |
| Mathematical operations | Difficult to use directly | Designed for advance operations |

## 2 Methods of Handling Arrays

There are two methods:

- **Without Array Modules.** This is convenient when the num of elements are small.
- **With Array Modules.** Designed for handling large data bases.

## ❸ Handling of Arrays (without the use of Array Module)

**3.1 Declaring an Array.** The procedure is same as for lists. Syntax to declare an array is as shown in line 1. Here fruits is the name of the array & its elements are in square brackets.

```
>>> fruits = ['Apples', 'Oranges', 'Mangoes']
>>> print(fruits)
['Apples', 'Oranges', 'Mangoes']
```

When we print it, the output contains the elements as declared.

On assignment it is saved with a **unique ID** & called using the name.

```
>>> type(fruits_array)
<class 'list'>
>>> id(fruits_array)
2703356937408
```

**Note:** Python does not have a separate support for arrays. Howeverpython lists can be used as the variable values when defining them. Thereafter, **all that applies to lists will apply to arrays.**

**3.2. Structure of an Array.** Syntax for making arrays is:
**array(name) = [ array elements]**
Ex - **students = [ 'Ram', 'Rohan', 'Rita', 'Amar'].**
In a small **list** of say 50 students, if we need to loop through the students to find a specific student the task is simple. However, say if the list is of 5000 students, we would face problems. In such situations, the solution is **arrays.**

**3.3. Array Operations.** We can carry out following basic operations:
Say our array is:

```
>>> students = ['Ram', 'Rohan', 'Rita', 'Amar']
```

**1 Calling an Array. I**ndividual elements of an array can be accessed by calling our list & specifying an index number.

```
>>> print(students[3])
Amar
```

We can also use negative indexing. This is of use in case we have a long list & counting backwards.

```
>>> print(students[-2])
Amar
```

**2 Slicing Elements** in an Array. We can slice our lists to get multiple elements out. This is done creating a range of index numbers, separated by a colon. Elements at index 1 & 2 have been pulled out. This has retrieved elements before index num 3.

```
>>> print(students[1:3])
['Rohan', 'Rita']
```

```
>>> print(students[:3])
['Ram', 'Rohan', 'Rita']
```

**3. Update/Insert Elements** in an Array. We canchange element values by referencing each item by its index value & assigning it a new value.

```
>>> students[3] = 'Madhu'
>>> print(students)
['Ram', 'Rohan', 'Rita', 'Madhu', 'Rose']
```

**4. Traversing an Array.** You can traverse by looping through the array elements.

```
>>> for x in fruits:
        print(x)
```

```
Apple
Banana
Mango
```

**Note:** Here we are showing how to use lists as arrays. We can also work with arrays by importing a library, like NumPy. We shall do so in the lesson on NumPy.

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 04 Python Package

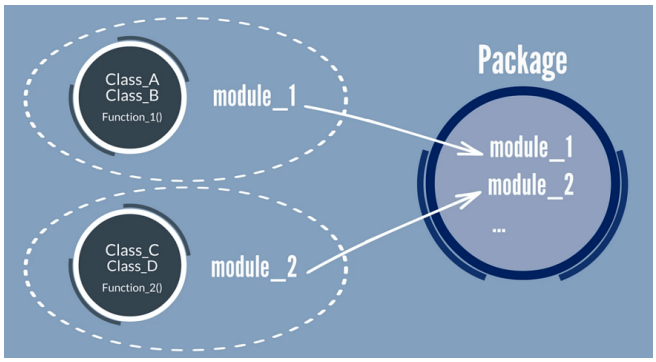
In the lesson on file, we found the need of organizing our files in different folders and subfolders based on some criteria, so that they can be managed easily and efficiently. The same analogy holds good for Python Packages. They are funcs filed as modules, packages & libraries.

### Understanding Python Modules, Packages, Libraries & Frameworks

Like functions, these three are a collection of pre-written codes that we can use in our projects without having to reinvent the wheel.

## 1 Python Modules

A Python module is a file that contains Python codes, such as funcs, classes, vars, or runnable code. File name is the module name, with the suffix ".py" appended to it. Modules help structure Python code logically by grouping related code into one file. This makes it easier to understand and use code.

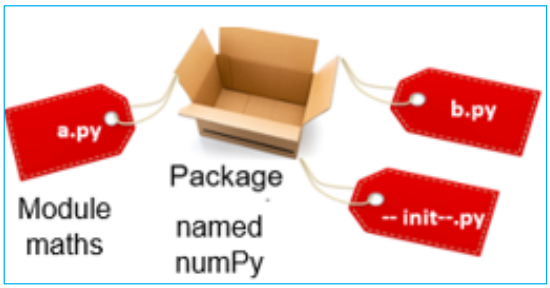



## 2 Python Package

In very simple words, it is a bigger folder that contains modules, in addition to functions, classes, and other resources that can be used to perform specific tasks. Package contains modules of reusable code, that can be imported into other programs or interactive mode as an init.py file; A file that tells Python that the directory is a package.

Package name, binds the modules together in a package, which are a fundamental building block in programming and are used for creating large programs. They:

- Allow us to reuse code that's already been written.
- Help us organize and document our code.




## 3 Python Libraries

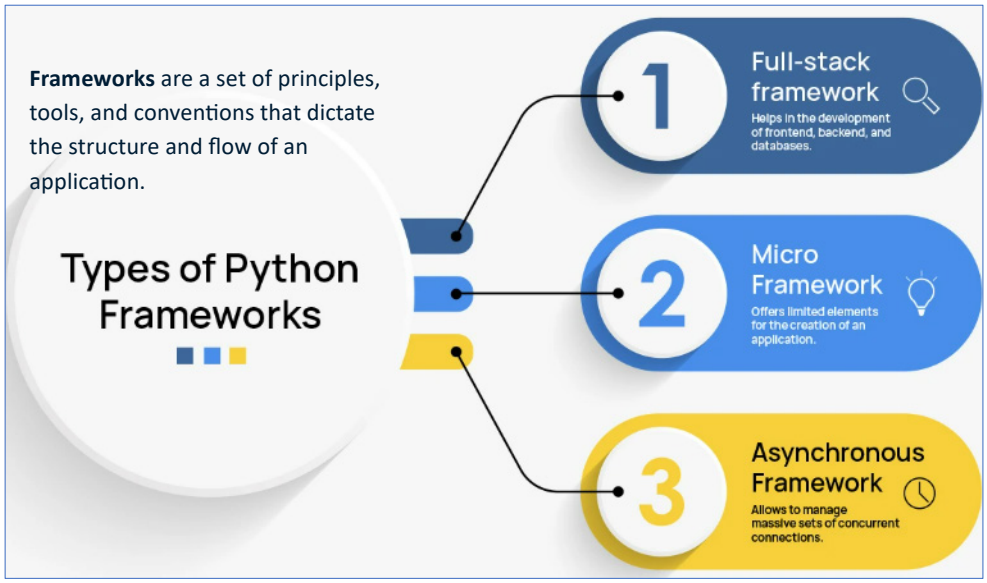
Like modules & packages python libraries are a bigger entity containing collections of pre-written code that provide reusable functions, classes, and constants to perform specific tasks. They can help programmers speed up development, avoid reinventing the wheel, build more robust and complex applications, and make everyday tasks more efficient. Some examples of Python libraries include:

- **Pandas:** A tool for tabular data analysis.
- **Matplotlib:** A tool for data visualization.
- **NumPy:** A tool for numerical computing.
- **Keras:** A tool for building  neural networks.
- **Requests:** A tool for making HTTP requests.
- **Scikit-learn:** A tool for machine learning.
- **TensorFlow:** A tool for deep learning.
- **PyTorch:** A machine learning library.


## 4 Differences between libraries, packages, and modules

- **Modules** are a single file that contains code, such as functions, classes, and variables. Modules are the fundamental building blocks for organizing code.
- **Packages** are a directory that contains multiple Python modules, often organized hierarchically. A directory becomes a package when it contains an init.py file.
- **Libraries** are a collection of packages or modules that provide specific functionality. Libraries are broader than modules and packages, and can include multiple packages, a single package, or a single module.

## 5 Python Frameworks




## 6 Understanding Pip

### 1) What is Pip

It is a package manager for Python that's used to install and manage software packages. It is the standard tool for installing Python packages and their dependencies. It's written in Python and is recommended by the Python Software Foundation. It is an acronym that can stand for either **"Pip Installs Packages"** or "Pip Installs Python". Pip also stands for **"preferred installer program".**

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 05 Pandas Basics & Pandas Series

## 1 What is Pandas

Name **"Pandas"** has reference to both **"Panel Data"** (multi-dimensional data involving measurements over time) & **"Data Analysis".**
Created by Wes McKinney in 2008, it is a data analysis cum manipulation library used for working with data sets & handling tabular data. It provides numerous funcs & methods that expedite data analysis & pre-processing. It takes data from a wide range of sources such as CSV files, Excel files, HTML tables on the web, & text files. **It scales very well with large datasets.**

It is built on top of **NumPy** package (that handles **numeric data),** meaning a lot of the structure of NumPy is used or replicated in Pandas. Data in pandas is often used to feed statistical analysis in **SciPy,** plotting funcs in **Matplotlib,** & machine learning algorithms in **Scikit-learn**


The source code for Pandas is located at https://github.com/pandas-dev/pandas

## 2 Installing & Importing Pandas

### 1) Installing Pandas.

We already have Python installed in your system. To start, we need to install pandas in our system. For this first install **pip** from **PyPi** (it is a repository of software. It helps you find and install software developed and shared by the Python community). Next learn about installing packages (package authors use PyPI to distribute their software).
This allows us to install individual packages to our system using the **pip install** command. It works with python installations greater than or equal to 2.7.9 & all the way to the latest.
Having installed **pip,** installation of **Pandas** is very easy. It is done using command:
**C:\Users\Your Name>pip install pandas.**

### 2) Importing Pandas.

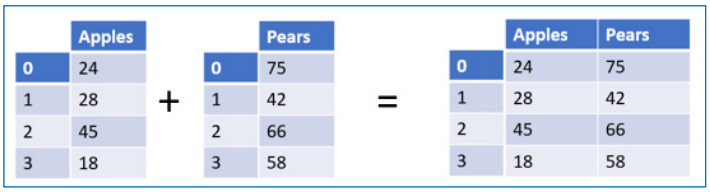
First line of every code starts with importing the library.
Syntax to import is **import pandas** or **import pandas as pd.** Once this is done, we are ready to code.

#### Basic Terms to Learn Pandas

Data sets sit at **core of Data Science,** where we study how to store, use & analyse data for deriving information from it. This in turn sits at the **core of Pandas.**

## 3) Python Series & Data Frames.

In the diagram att, Apples is one series & Pears is the other.
In Pandas, the **two series** put together make a **data Frame.**



| | Apples |
|---|---|
| 0 | 24 |
| 1 | 28 |
| 2 | 45 |
| 3 | 18 |

+

| | Pears |
|---|---|
| 0 | 75 |
| 1 | 42 |
| 2 | 66 |
| 3 | 58 |

=

| | Apples | Pears |
|---|---|---|
| 0 | 24 | 75 |
| 1 | 28 | 42 |
| 2 | 45 | 66 |
| 3 | 18 | 58 |

## 4) Series.

It is a one-dimensional array of data. It can hold data of any type: string, integer, float, Dictionaries, lists, Booleans, and more. The easiest way to conceptualize a Series is a single column in a table, which is why it's considered one-dimensional.

## 5) Panel.

It is a three-dimensional container of data. To create a panel, we can use ndarrays (multidimensional arrays) and a dictionary of DataFrames (one of the Pandas 2-D data structures that contain data in the tabular form of rows and columns).

## 6) Data Sets

Data sets are a collection of values or nums that **relate to a particular subject.** For example, record of a patient's parameters, over a 24-hour period in a hospital, for each patient, in a particular department is a data set. Student marks of different classes, in different tests, is a data set.

## 7) Data-Base Size

In real life, data bases we need to handle are huge. Say our school has a student data base of ten columns & 1250 rows. Pandas provides simple tools to manipulate & update this.. During learning we shall make us of small data bases. However, whatever applies to one column, or one row, or to the small table will apply equally to the larger tables.

## 8) Pandas Funcs

Pandas provides funcs for analysing, cleaning, exploring & manipulating data. These allow us to analyse big data & make conclusions based on statistical theories. These funcs help us clean messy data sets, & make them readable & relevant.

## 9) Labels.

In Pandas labels are containers containing values. `x = [24, 48, 45, 18]`
Here **x** is a label, representing apples, having a list, that contains their values.

These values can be called by their index num (**2**). `print(Apples[2])` `45`
Default values for index nums is - **0** for 1st label, **1** for 2nd & so on. `print(Apples[0])` `24`

Labels need not be unique but must be a hashable type. Lists & dict are best choice for labels.

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 06 Pandas Data Frames

## 1 Understanding Data Frames

In Pandas **Data Sets** are usually called **Data Frames.** A Pandas Data Frame is a **2-dimensional data structure, like a 2-dimensional array, or a table with rows & columns.** In simple words, if **series** is like a column in a table, a **Data Frame** is like the whole table.

### 1) What are Pandas Data Frames (DF)?

Look at a data frame like an excel sheet. These are data structures that contain:

- Data organized in rows & columns.
- Labels that correspond to these, rows & columns.
- Cells that get created at the intersection of the two.
- Data that needs to be entered in the cell. Cell can be empty but null used to avoid error.

In addition, there are some housekeeping possibilities, primarily to give them a **user appeal.** These will be studied separately under **data visualisation** (**matplotlib**).

### 2) Sample DF for Our Learning.

To understand,
let us take a sample based on data from Wikipedia.
It is a DF of four rows & four columns making a total of 16 (4x4) cells. We shall use this for our explanations.

| | Name | Ht (m) | Sumitted | Range |
|---|---|---|---|---|
| 0 | Everest | 8848 | 1953 | Himalaya |
| 1 | K2 | 8611 | 1954 | Baltoro |
| 2 | Kanchan | 8586 | 1955 | Mahalangur |
| 3 | Lhotse | 8516 | 1956 | Himalaya |

### 3) What is an Index in Data Frames (DF)?

Look at index like an address. This index is then used to access any **data point or cell across the DF or series.** Rows & columns both have indexes.
Each column has a name associated with it, also known as a label. These are of two types.

- Default labels are the left most column.
- In row 'Name', 'Ht (m)', 'Sumitted', & 'Range' we can set (**Name**) to be our **named index.** It will then be used as row labels for indexing.

| | Name | Ht (m) | Sumitted | Range |
|---|---|---|---|---|
| 0 | Everest | 8848 | 1953 | Himalaya |
| 1 | K2 | 8611 | 1954 | Baltoro |
| 2 | Kanchan | 8586 | 1955 | Mahalangur |
| 3 | Lhotse | 8516 | 1956 | Himalaya |

### 4) Methods of Making DF.

The three common ways to create DF are:

- Using the data frame constructor & providing the data, labels, & other information.
- Passing the data as a two-dimensional list, tuple or NumPy array.
- Passing the data as a dict or as an instance of panda's series.

## 5) Making a Simple DF.

It makes use of the constructor & is like making a series.

```
1  import pandas as pd
2  # Data Block
3  names = ["Everest", "K2"]
4  values = [[8848, 'Himalaya'],
5            [8611, 'Baltoro']]
6  items = ['Ht m', 'Range']
7  # Operational block
8  Origdf = pd.DataFrame(values, index=names, columns=items)
9  print(Origdf)
```

**Note:** 1. Lines 3 to 6 make the data block.
2. Lines 8 & 9 are the operations block.
In this **D** & **F** in word Data Frame
(line 8) are **case sensitive.**
3. Names are row labels, items are column labels, & rest are values.

## 6) A word about Names in Syntax.

We have reproduced the above code as shown here. Kindly note two changes:

- **Change 1.** In line 1, we have used **P** instead of **pd.** We can use any alphabets or name; with any capitalization.
- **Change 2.** In Line 6, Sample is a var that represents the data frame. This var can also be given any name ex **table.**

```
1  import pandas as P
2  names = ["Everest", "K2"]
3  values = [[8848, 'Himalaya'],
4            [8611, 'Baltoro']]
5  items = ['Ht m', 'Range']
6  Sample = P.DataFrame(values, index=names, columns=items)
7  print(Sample)
```

```
5  items = ['Ht m', 'Range']
6  table = P.DataFrame(values, index=names, columns=items)
7  print(table)
```

# ❷ Data Frame Operations

## 1) Real-Life Data Frame

Let us now make a real-life DF using sample data of top five mountains. Its code is similar & has the same four steps.
Let us understand some of the basic operations.
Its output is in red box.

```
import pandas as pd
data = {'Name': ['Everest', 'K2', 'Kanchan', 'Lhotse'],
        'Ht (m)': [8848, 8611, 8586, 8516],
        'Sumitted': [1953, 1954, 1955, 1956],
        'Range': ['Himalaya', 'Baltoro', 'Mahalangur', 'Himalaya']}
BaseData = pd.DataFrame(data)
print(BaseData)
```

| | Name | Ht (m) | Sumitted | Range |
|---|---|---|---|---|
| 0 | Everest | 8848 | 1953 | Himalaya |
| 1 | K2 | 8611 | 1954 | Baltoro |
| 2 | Kanchan | 8586 | 1955 | Mahalangur |
| 3 | Lhotse | 8516 | 1956 | Himalaya |

**Try Example 1.** Make a three row DF of students of your class, corresponding to the first three rows they sit in. Specify first name, height in kg, weight in cm, & sex of the students. Use null for any empty text, in any row.

## 2) Returning One Row of a Data Frame.

By convention, there are two methods:

- Use (**loc**) to take out row or column by label.
- Use (**iloc**) to take out row or columns by position (index num).

```
1  import pandas as pd
2  data = {'Name': ['Everest', 'K2', 'Kanchan', 'Lhotse'],
3          'Ht (m)': [8848, 8611, 8586, 8516],
4          'Sumitted': [1953, 1954, 1955, 1956],
5          'Range': ['Himalaya', 'Baltoro', 'Mahalangur', 'Himalaya']}
6  BaseData = pd.DataFrame(data)
7  print('value of row 1')
8  print(BaseData.iloc[1])
```

```
Name          K2
Ht (m)      8611
Sumitted    1954
Range     Baltoro
```

Note 1.The changes from previous code.
2. Use of **.loc** in line 8 will also work.
3. Output appears as a series containing details of the row.

**Try Exercise 2.** In the code made for **try 1,** take out first row & last row. In similar manner take out column related to range.

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 07 Matplotlib – Line Plots & Markers

Data visualization is the process of translating nums, text, or large data sets into various types of graphs such as histograms, maps, bar plots, pie charts, etc. For visualizations, we need some tools or technology.

If one is thinking of data analytics or data visualization, one such tool you need to think of is Matplotlib. **Matplotlib Library** is one of the most powerful libraries in python for data visualization. It is able to create different types of visualization reports like line plots, scatter plots, histograms, bar charts, pie charts, box plots & many more plots. The library also supports 3-dimensional plotting.

## 1 Introduction to Matplotlib

Matplotlib is an open source, low level graph plotting library that helps represent data visually. The source code for Matplotlib is located at this github repository: https://github.com/matplotlib/matplotlib

### 1) Installing Matplotlib

To install we have two methods:

- Using Python Interpreter.
- Using PIP.

To install using python interpreter, open PyCharm, select file, in dropdown select stetting's. In new window under Project select Python Interpreter. A new window opens.
click on + at bottom left of this window. In new window enter Matplotlib & click on install package at bottom left. Matplotlib will get installed.

To install using PIP go to terminal, write pip install name of module. Press enter & wait for few sec.

```
pip install matplotlib
```

Now Matplotlib package/lib is installed on our PC. Whenever we want to use it, we need to import it in our application using statement

```
import matplotlib
```

### 2) Checking Matplotlib version.

The version string is stored under **__version__** attribute. To check use: The output will appear in the red box.

```
import matplotlib
print(matplotlib.__version__)
```

## ❷ Working with Plots

Most used interfaces to work on plots are:

- **Pyplot.**
- **Pylab.**
- **NumPy.**

Other interfaces like **pandas** are also used. Thus, **pyplot is the preferred interface.** We shall focus on this in this lesson. When working on plots, we can also make use of the **Object-Oriented interface** of matplotlib. We shall touch upon it at the end.

## ❸ Working with Pyplot

It is a collection of funcs that make matplotlib work like MATLAB (a high-performance language for technical computing. It integrates computation, visualization, & programming in an easy-to-use environment where problems & solutions are expressed in familiar mathematical notation).

Each Pyplot func makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

Some of the common **plot categories** of Matplotlib are:

**Line plots, Sub Plots, Scatter plots, Bar Charts, Histograms, & Pie charts.**

### 1) Importing Pyplot

To work on Pyplot we start with Import of pyplot & NumPy modules.

```
import matplotlib.pyplot
import numpy
```

Or as →

```
import matplotlib.pyplot as plt
import numpy as np
```

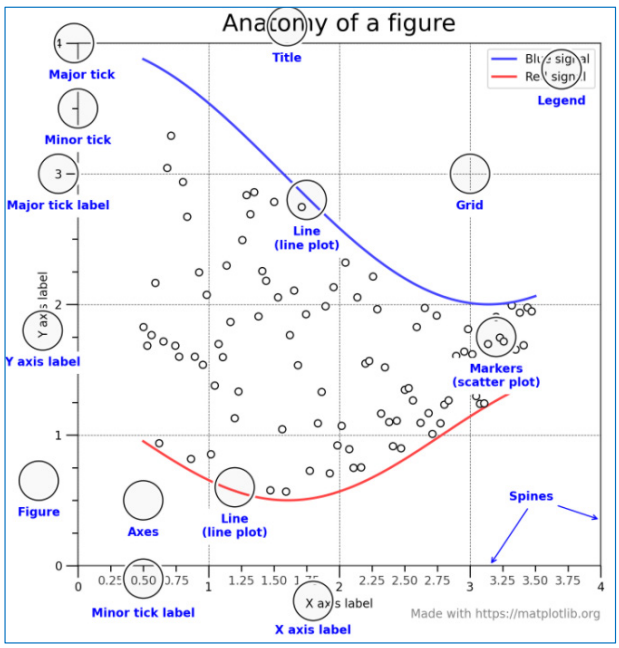
Or even using any other alphabet/alphabets. →

```
import matplotlib.pyplot as P
import numpy as N
```

Using abbreviated version, avoids use of full names when used inside the code later.

**Basic Anatomy of a Plot.** Before starting it is important to know the key terms associated with plots. These have been annotated on the att diagram. Kindly follow in following sequence:

- Line.
- Markers.
- Axis.
- X axis label & Y axis label.
- Title
- Legend.
- Major tick & major tick label.
- Minor tick & minor tick label.
- Spines.
- Grids.
- Figure.



## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 08 Matplotlib – Labels & Titles

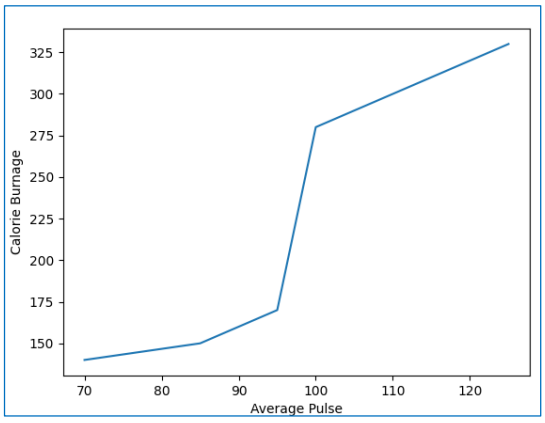
## 1 Labels & Titles in Plots

Graphs & plots are an excellent means of visual representation of data. However, without proper labelling, the graph won't make sense. For this we need to label the x-axis, y-axis & title our graphs so that they can be understood by people without having to ask what they represent. Labels & titles are the name given to elements of a plot (ex average pulse & calories Burnage).

### 1) Creating labels for a plot.

Use the **xlabel()** & **ylabel()** func to set a label for the x & y-axis. Here we have **Average pulse** on xAxis & **Calories burnt** on y axis.

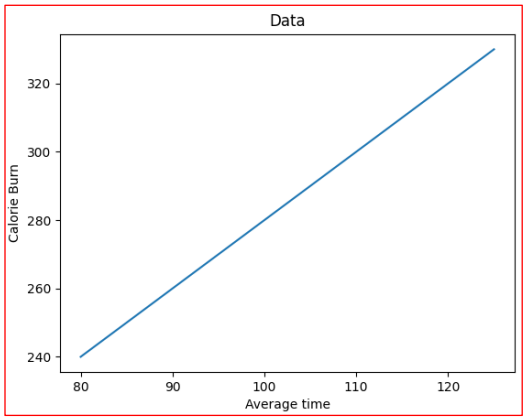
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([70, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([140, 150, 160, 170, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```




### 2) Creating Title for a plot

Use the **title()** function to set a title (say **Data**).

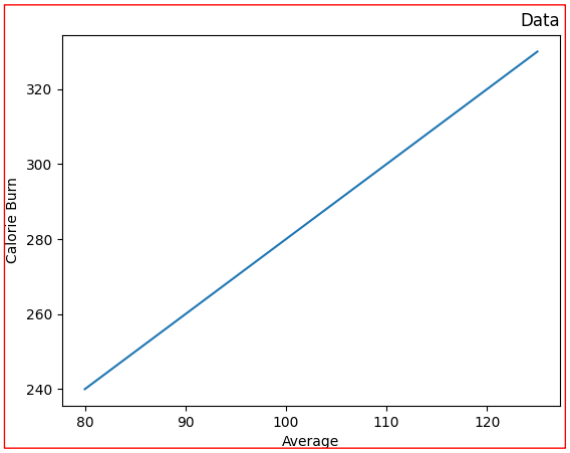
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.title("Data")
plt.xlabel("Average time")
plt.ylabel("Calorie Burn")

plt.show()
```



## 3) Positioning the Title.

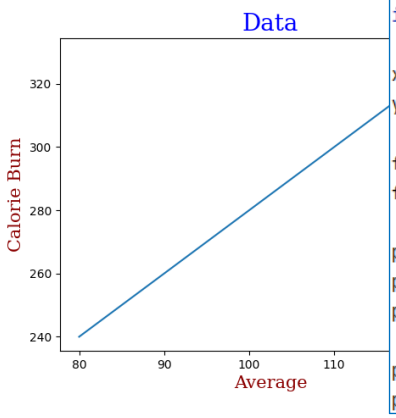
Use **loc** in **title()** to position title. Legal values are: 'left', 'right' & 'center'. Default is 'center'.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Data", loc = 'right')
plt.xlabel("Average")
plt.ylabel("Calorie Burn")

plt.plot(x, y)
plt.show()
```




## 4) Set Font Properties.

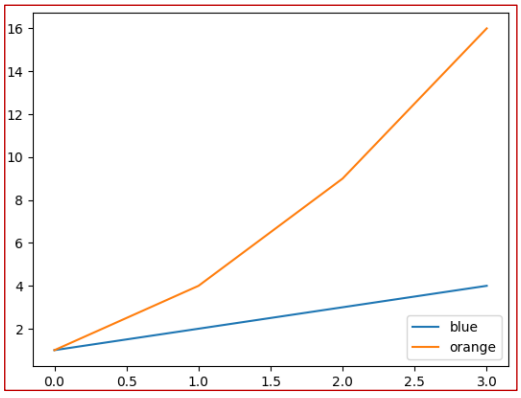
Use **fontdict** parameter in **xlabel()** **ylabel()** & **title()** to set font properties for the title & labels (4th & 5th lines).




```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Data", fontdict = font1)
plt.xlabel("Average", fontdict = font2)
plt.ylabel("Calorie Burn", fontdict = font2)

plt.plot(x, y)
plt.show()
```

## 5) Adding & Positioning Legend.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(4)
y1 = [1, 2, 3, 4]
y2 = [1, 4, 9, 16]

# Func to add lables to plot
plt.plot(x, y1, label ='Nums')
plt.plot(x, y2, label ='Square of nums')
# Func to add legend to plot
plt.legend(["blue", "orange"], loc ="lower right")
plt.show()
```



## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

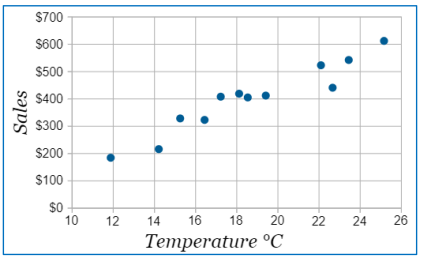
www.bdseducation.in

CHAPTER

# 09 Matplotlib – Scatter Plots & Bar Charts

## 1 Matplotlib Scatterplot

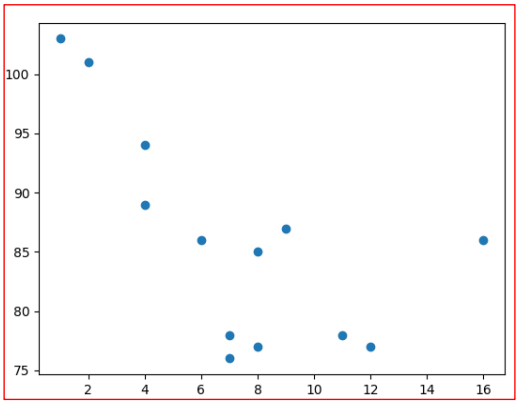
A scatterplot (scatter chart or scatter graph) is **a type of data display that shows the relationship between two numerical var.** Each member of the dataset gets plotted in x-y coordinates relating to the values of the two variables. This ex shows the relation between two var, temp & average sale of ice cream for 12 instances.




### 1) Creating Scatter Plots.

**scatter()** func helps to plot one dot for each observation. It needs two same length arrays. One each for values of x-axis, & y-axis.

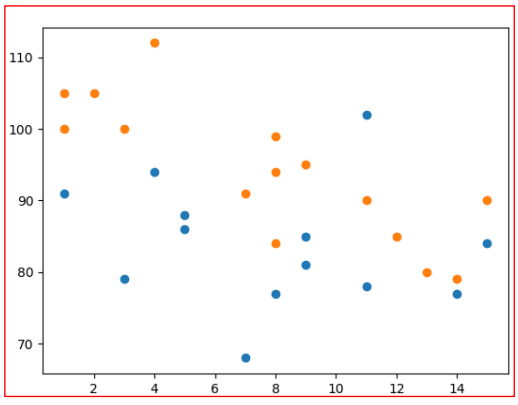
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([4,7,8,7,2,16,1,9,4,11,12,8,6])
y = np.array([89,76,77,78,101,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```




### 2) Comparing Plots.

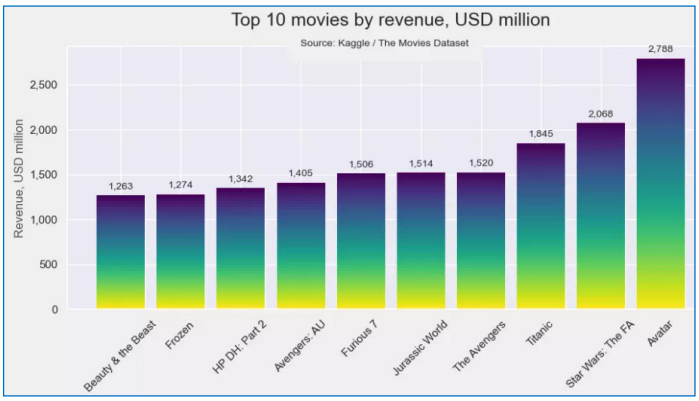
Say we have data arrays of age & speed of car of two days. Will scatter plot tell us something else? Orange is day 1. Try yourself.

```
#day one, the age and speed of 13 cars:
x = np.array([3,5,8,7,1,15,11,9,4,11,14,9,5])
y = np.array([79,86,77,68,91,84,102,81,94,78,77,85,88])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([1,1,8,2,15,8,11,9,8,3,14,4,7,13,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)
```



## ❷ Matplotlib Bars

A bar chart or bar graph presents categorical data as rectangular bars plotted vertically or horizontally & having heights or lengths proportional to the values they represent.
A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value. It helps compare values of different categories in the data.




### 1) Python Bar Chart Libraries.

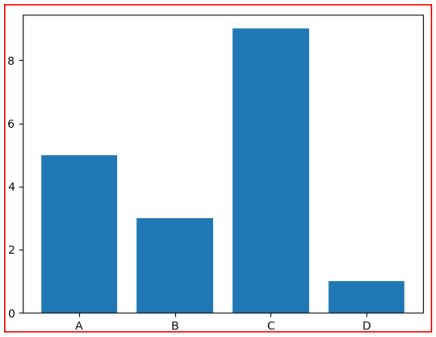
These include:

- Matplotlib.
- Seaborn.
- Plotly.

### 2) Creating Bars.

Use the **bar()** func to draw bar graphs. Code is simple & as shown.
Play around with values.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([5, 3, 9, 1])

plt.bar(x,y)
plt.show()
```




### 3) Horizontal Bars.

If you want the bars to be displayed horizontally instead of vertically, use the **barh()** func
Try this yourself using the hint:

```
plt.barh(x, y)
plt.show()
```

### 4) A Word about Colours.

Color is defined by its mix of **Red, Green** & **Blue**, each of which can be in the range: 0 to 255 in decimal or 00 to FF in Hexadecimal.
Because each of the three colors can have values from 0 to 255 (256 possible values), there are: 256 x 256 x 256 = 15.772.216 possible colour combinations. Therefore, you see claims of "16 Million Colors" on computer equipment.

### 5) Web format for Colours.

The format ("notation") used on web pages is **#RRGGBB,** where RR is how much Red (using two hexadecimal digits), GG is how much Green, and BB how much Blue.

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 10 Matplotlib – Histograms & Pie Charts

## 1 Matplotlib Histograms

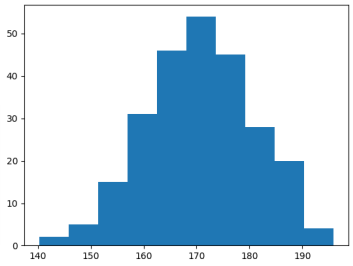
Histograms are the most common method for visualizing the distribution of a variable. A simple histogram can be very useful to get a first glance at the data.

In coding, a histogram is a graph showing frequency distributions.

It is a graph showing the number of observations within each given interval. Ex: Say you analyse the height of 250 people.

You might end up with a histogram like this: This reads:

| | |
|---|---|
| 2 people from 140 to 145cm | 53 people from 168 to 173cm |
| 5 people from 145 to 150cm | 45 people from 173 to 178cm |
| 15 people from 151 to 156cm | 28 people from 179 to 184cm |
| 31 people from 157 to 162cm | 21 people from 185 to 190cm |
| 46 people from 163 to 168cm | 4 people from 190 to 195cm |

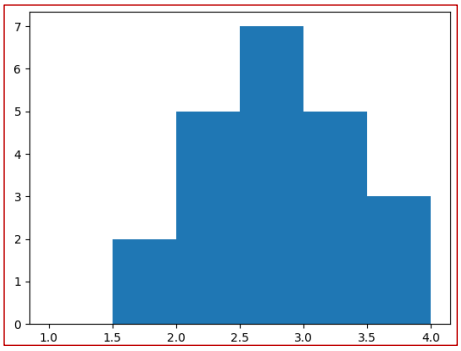



### 1) Creating Histogram.

We use the **hist()** func to create histograms. The **hist()** func will use an array of nums to create a histogram. The array is sent into the func as an arg.

The data in this has a range from 1 to 4. In addition, we have defined to see the output in 6 bins. The code is:

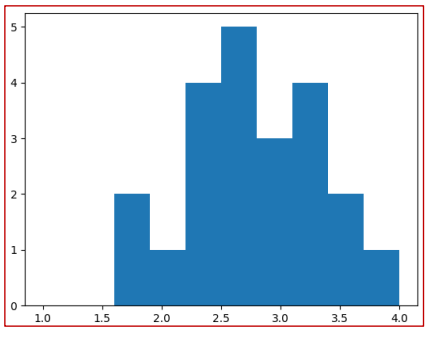
```
import matplotlib.pyplot as plt
data = [1.7,1.8,2.0,2.2,2.2,2.3,2.4,2.5,2.5,2.5,2.6,2.6,2.8,
        2.9,3.0,3.1,3.1,3.2,3.3,3.5,3.6,3.7,4.1,4.1,4.2,4.3]
#this histogram has a range from 1 to 4
#and 6 different bins
plt.hist(data, range=(1,4), bins=6)
plt.show()
```




### 2) Changing the Num of Bins.

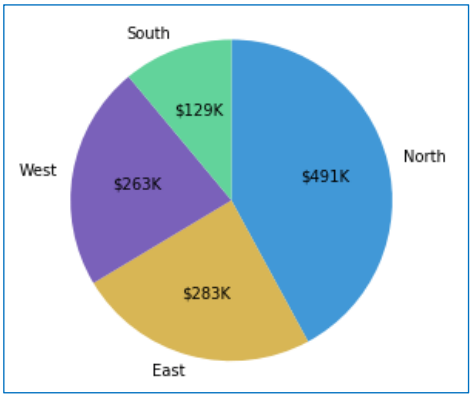
In this we have changed num of bins from 6 to 10. Data values distributed in smaller intervals.

```
import matplotlib.pyplot as plt
data = [1.7,1.8,2.0,2.2,2.2,2.3,2.4,2.5,2.5,2.5,2.6,2.6,2.8,
        2.9,3.0,3.1,3.1,3.2,3.3,3.5,3.6,3.7,4.1,4.1,4.2,4.3]
#this histogram has a range from 1 to 4
#and 10 different bins
plt.hist(data, range=(1,4), bins=10)
plt.show()
```



## ❷ Pie Carts

It is a type of graph in which a circle is divided into sectors. The sectors are proportional to the fraction of the whole in each category. In other words, **each slice of the pie is relative to the size of that category in the group.** In order to use a pie chart, you must have some kind of total amount that is divided into several distinct parts. Prime objective of a pie chart should be **to compare each group's contribution to the whole** as opposed to comparing groups to each other. Kindly see this ex.

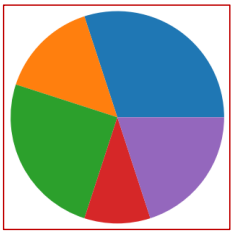



| REGION | TOTAL REVENUE |
|---|---|
| North | 491 064.51 |
| East | 283 445.43 |
| South | 128 753.87 |
| West | 263 391.13 |

### 1) Creating a Simple Pie Chart.

Use the **pie()** func to draw pie charts. Its code is: In line 3 def the **Wedge**. It is a sector or piece of circle. Try to keep num of wedges to below seven. **Default start angle of wedge** is the x axis (red arrow).

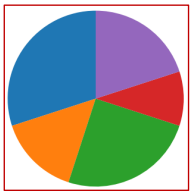
```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([30, 15, 25, 10, 20])
plt.pie(y)
plt.show()
```



### 2) Changing the Start Angle.

The start angle has been changed to 90 degree. Change details have been added in line 4.

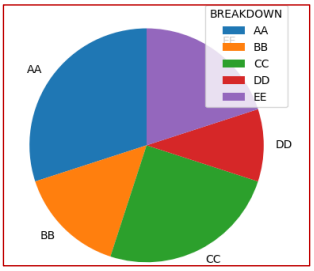
```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([30, 15, 25, 10, 20])
plt.pie(y, startangle = 90)
plt.show()
```



### 3) Adding Labels, Legend & Legend Title.

- Labels are added as strings in line 4.
- Legend is added in line 7. Legend title is added as a string in legend def in line 7.

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([30, 15, 25, 10, 20])
mylabels = ['AA', 'BB', 'CC', 'DD', 'EE']
plt.pie(y, startangle = 90, labels = mylabels)
plt.legend(title = 'BREAKDOWN')
plt.show()
```




### 4) Exploding Wedges.

We may want one or more of the wedges to stand out? This is done in line 5 of code below. Here **0.2 & 0.1** represent the amount of stand out. Wedges that do not require to be brought

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 11 Numpy Basics

NumPy is an open-source package designed for scientific computing in Python. It is a python library that stands for **Numerical Python.** It is used for working with nums & arrays. It also has functions for working in domain of linear algebra, Fourier transform, & matrices.

In Python we have lists that serve the purpose of arrays, but they are slow to process. At the core of its package, is the **ndarray** object, that is up to 50x faster than traditional Python lists.

Unlike lists, NumPy arrays are stored at one continuous place in memory. Thus, processes can access & manipulate them very efficiently. Parts of NumPy library that require fast computation are written in C or C++. Together, they make NumPy faster than lists.

## ❶ Installing & Importing NumPy

**Installing NumPy.** If your system has python & PIP installed, then install NumPy using the command - C:\Users\Your Name>pip install NumPy

**Importing NumPy.** This is like other libraries. Its code is:

```
import numpy
numarr = numpy.array([11, 22, 33, 44, 55])
print(numarr)
```

```
[11 22 33 44 55]
```

Var in line 2 can have any name. Output appears s a list:

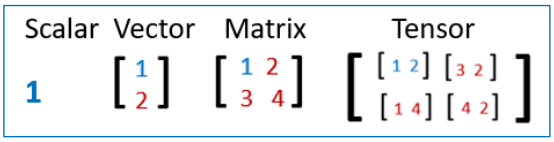
It can also be imported as:

```
import numpy as np
```

## ❷ Understanding NumPy Arrays

The most import data structure for scientific computing in Python is the NumPy array. NumPy arrays are used **to store lists of numerical data & to represent vectors, matrices, & even tensors**. NumPy arrays are designed to handle large data sets efficiently with minimum fuss.

### 1) What are Scalar, Vector, Matrix & Tensor.




### 2) Creating a NumPy Array Object.

NumPy is used to work with arrays. The array object thus created is called **nparray.** It is created using the **array()**. Its code is simple. 4th line is for confirming the type.

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55])
print(arr)
print(type(arr))
```

Array can be created by passing a list, tuple or any array like object into **array()** method. Here we have used a tuple

```
arr = np.array((11, 22, 33, 44, 55))
```

## 3) Dimensions of an Array.

We have four options. These are: 0-D Array, 1-D Array, 2-D Array, & 3-D Array.

## 4) Creating Multi Dimension Arrays.

```
import numpy as np
arr = np.array(55)
print(arr)
```

- **0-D** Array using num 55. It has only one element.
- **1-D** Array using values 11, 22, 33, 44 & 55.

An array that has 0-D arrays (55) as its elements is called uni-dimensional or 1-D array.

```
arr = np.array((11, 22, 33, 44, 55))
```

These are most common & basic arrays.

- **2-D** Array. An array that has 1-D arrays as its elements is called a 2-D array. These are used to represent matrix. In addition, **numpy.mat** is a sub module dedicated to matrix ops.

```
arr = np.array([[11, 22, 33], [44, 55, 66]])
```

- **3-D** Array. An array that has 2-D arrays (matrices) as its elements is called 3-D array. These are often used to represent a 3rd order tensor.

```
arr = np.array([[[11, 22, 33], [44, 55, 66]], [[12, 22, 32], [42, 52, 62]]])
```

```
[[[11 22 33]
  [44 55 66]]

 [[12 22 32]
  [42 52 62]]]
```

**Note:** In simple terms, vectors are one-dimensional data structures & matrices are two-dimensional. **Tensor is a dimensional data structure.** Second-rank tensors can be represented as matrices (2-D array).

## 5) Checking Num of Dimensions.

NumPy Arrays provide **ndim** attribute that returns an integer that tells us how many dimensions the array has. Output is 2.

```
import numpy as np
x = np.array([[11, 22, 33], [44, 55, 66]])
print(x.ndim)
```

## 6) Higher Dimension Arrays.

An array can have any num of dimensions (dim). When array is created, you can def num of dimensions by using the **ndmin** arg. In this array, innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dimhas 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

```
import numpy as np
arr = np.array([11, 22, 33, 44], ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
```

```
[[[[[11 22 33 44]]]]]
number of dimensions : 5
```

# ❷ Array Indexing

**Accessing Array Elements.** This is easy. All rules of indexing as for others apply.

```
import numpy as np
arr = np.array([11, 22, 33, 44])
print(arr[2])
```

```
33
```

**Accessing two or more Elements.**

```
print(arr[1] , arr[3])
```

```
22 44
```

**Accessing two Elements & Add Them.**

```
print(arr[1] + arr[3])
```

```
66
```

**Similarly do other Arithmetic Ops.**

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

```
363
```

```
print(arr[0] * arr[2] + arr[3])
```

```
407
```

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 12 NumPy Operations - 1

## ❶ Data Types in NumPy

Data types in NumPy are represented by the following characters:

- Integers – **i.**
- Boolean – **b.**
- Unsigned integer – **u.**
- Float – **f.**
- Complex float – **c.**
- Timedelta – **m.**

Datetime – **M.**
Object – **o.**
String – **s.**
Unicode string – **U.**
Void – **V.** These are **flexible data types.** There is no pre-defined type associated to these.

### 1) Checking the Data Type of an Array.

It is important to keep checking the type of array you are working with.
This is done as shown:

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55, 66])
print(arr.dtype)
```

```
int32
```

**Checking the Data Type of an Array of Strings.**

It is similar. Check it yourself using the hint & its output.

```
arr = np.array(['Apple', 'Orange', 'Mango'])
```

```
<U6
```

## ❷ Copy & View

These are two frequently used funcs. Difference between the two is that copy returns a new array, while view is just a view of the original array.

**Copy** owns the data. Changes made to the copy will not affect original array, & any changes made to the original array will not affect the copy.

**View** does not own the data. Changes made to view will affect the original array, & any changes made to the original array will affect the view.

### 1) Copy.

Use copy() to make a copy.
2nd line is our original array. 3rd line is var x for its copy.4th line makes a change in original array. 6th line prints the original array with the change made. 5th line prints the copy. Note **changes made to original have not effected the copy.**

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55, 66, 77])
x = arr.copy()
arr[0] = 999
print(x)
print(arr)
```

```
[11 22 33 44 55 66 77]
[999  22  33  44  55  66  77]
```

If we now make **changes to copy** (2nd line att code) it will **not affect the original** (2nd line of output).

```
y = x.copy()
x[2] = 777
print(x)
print(y)
```

```
[ 11  22 777  44  55  66  77]
[11 22 33 44 55 66 77]
```

### 2) View.

Use **view()** to view an array. In similar manner as above, let us make a view, change the original array, & display both arrays. Whet do we see? Unlike copy, view has also been affected by the change.

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55,  66])
x = arr.view()
arr[0] = 999
print(arr)
print(x)
```

```
[999  22  33  44  55  66]
[999  22  33  44  55  66]
```

Now let us make a change in this view. What do we, see? Unlike copy, **original has also been affected** by the change.

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55,  66])
x = arr.view()
x[2] = 666
print(arr)
print(x)
```

```
[ 11  22 666  44  55  66]
[ 11  22 666  44  55  66]
```

### 3) Checking if Arrays Owns the Data.

As seen above, copies own the data, & views do not. Let us check this statements correctness. Every NumPy array has the attribute **base** that returns **None** if the array owns the data. Otherwise, the **base** attribute refers to the original object. The output of this Code shows that data is owned by a copy array, & not by view array.

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55,  66])
x = arr.copy()
y = arr.view()
print(x.base)
print(y.base)
```

```
None
[11 22 33 44 55 66]
```

## 3 Shape of NumPy Array

Shape is **a tuple that gives dimensions of the array.** It gives you an indication of the no. of dimensions in the array. It is usually used to get the **current shape of an array.** It may also be used to reshape the array in-place by assigning a tuple of array dimensions to it.
It is **a tuple of integers**, that denote the lengths of the corresponding array dimension.

### 1) Getting the Shape of an Array

Let us take a 2D array & get its shape. What applies to this 2 D array, will apply to others as well. The attribute for this is **shape.**
See the code. Its output is a tuple of two integers. 1st is 2 representing that it the array has two dimensions. 2nd is 4 representing that the first dimension has 2 elements (1st or outer square bracket set) & the 2nd element has four elements( the inner square bracket set). Try for other arrays.

```
import numpy as np
arr = np.array([[11, 22, 33, 44], [55, 66, 77, 88]])
print(arr.shape)
```

```
(2, 4)
```

### 2) Creating a Multi-dimensional Array

let us create an array with 5 dimensions using **ndmin** & a vector with values 11,22,33,44. In shape output, integers at every index tell us about, number of elements the corresponding dimension has. In this example, at index - 4 we have value 4. So, we can say that 5th (4 + 1 th) dimension (indexing starts at 0) has 4 elements. Rest have one.

```
import numpy as np
arr = np.array([11, 22, 33, 44], ndmin=5)
print(arr)
print('shape of array :', arr.shape)
```

```
[[[[[11 22 33 44]]]]]
shape of array : (1, 1, 1, 1, 4)
```

**Note:** A five-dimensional array is **an array of array, of  arrays**

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 13 NumPy Operations - 2

## 1 Joining Arrays

### 1) NumPy's concatenate() function.

It concatenates two arrays either row-wise or column-wise. In SQL we join tables based on a key, whereas in NumPy we join arrays by axes.Concatenate function can take two or more arrays of sameshape & pass a sequence of arrays that we want to join to **concatenate()** function, along with the axis. If axis is not explicitly passed, it is taken as 0.

```
import numpy as np
a1 = np.array([1, 2, 3])
a2 = np.array([4, 5, 6])
a3 = np.array([7,8,9])
arr = np.concatenate((a1, a2, a3))
print(arr)
```

```
[1 2 3 4 5 6 7 8 9]
```

### 2) Joining 2D Arrays.

In this example, we have taken a 2D array & joined it along axis = 1. Writing axis=2 will give an error – Out of bound for 2D.

```
import numpy as np
a1 = np.array([[1, 2], [3, 4]])
a2 = np.array([[5, 6], [7, 8]])
a3 = np.array([[4,6],[2,8]])
arr = np.concatenate((a1, a2, a3), axis=1)
print(arr)
```

```
[[1 2 5 6 4 6]
 [3 4 7 8 2 8]]
```

### 3) Joining Arrays Using Stack Func.

We can concatenate two or more 1-D arrays along the second axis which would result in putting them one over the other. 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.

```
import numpy as np
a1 = np.array([[1,2,3], [3,4,5]])
a2 = np.array([[5,6,7], [7,8,9]])
a3 = np.array([[4,6,8],[2,8,6]])
arr = np.stack((a1, a2, a3), axis=1)
print(arr)
```

```
[[[1 2 3]
  [5 6 7]
  [4 6 8]]

 [[3 4 5]
  [7 8 9]
  [2 8 6]]]
```

## 2 Stacking Arrays

Stacking is the concept of joining arrays in NumPy. Arrays having the same dimensions can be stacked. The stacking is done along a new axis. Stacking leads to increased customization of arrays. We can combine the stack function with other functions to further increase its capabilities. NumPy supports stacking along three dimensions:

- **vstack().** It performs vertical stacking along the rows.
- **hstack().** It performs horizontal stacking along with the columns.
- **dstack().** It performs in-depth stacking along a new third axis.

### 1) Stacking along Rows.

NumPy provides a helper func **hstack()** to stack along rows.

```
import numpy as np
a1 = np.array([[1,2,3], [3,4,5]])
a2 = np.array([[5,6,7], [7,8,9]])
a3 = np.array([[4,6,8],[2,8,6]])
arr = np.hstack((a1, a2, a3))
print(arr)
```

```
[[1 2 3 5 6 7 4 6 8]
 [3 4 5 7 8 9 2 8 6]]
```

## 2) Stacking along Columns.

NumPy provides a helper function - **vstack() to stack along columns.**
Try it yourself. Hint: `arr = np.vstack((a1, a2, a3))` The six lists will get stacked vertically.

## 3) Stacking along Height (Depth).

NumPy provides
a helper function - **dstack()**
to stack along height
which is the same as depth.

```
import numpy as np
a1 = np.array([[1,2,3], [3,4,5]])
a2 = np.array([[5,6,7], [7,8,9]])
a3 = np.array([[4,6,8],[2,8,6]])
arr = np.dstack((a1, a2, a3))
print(arr)
```

```
[[[1 5 4]
  [2 6 6]
  [3 7 8]]

 [[3 7 2]
  [4 8 8]
  [5 9 6]]]
```

# 3 Splitting Arrays

## 1) Splitting.

It breaks one array into multiple.
We have two methods. In both we access the entire array as a split array.
**Method 1.** Using **split()** in line 3, we indicate the number of splits (4).

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55, 66, 77, 88])
newarr = np.split(arr, 4)
print(newarr)
```

```
[array([11, 22]), array([33, 44]), array([55, 66]), array([77, 88])]
```

**Method 2.** This is similar but syntax is **Array_split().**

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55, 66, 77, 88, 99])
newarr = np.array_split(arr, 3)
print(newarr)
```

```
[array([11, 22, 33]), array([44, 55, 66]), array([77, 88, 99])]
```

If the array has less elements than what you have called (5) it will adjust from the end accordingly.
Note: Method **split** does not adjust if elements in the source are less then called. Try.

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55, 66, 77, 88])
newarr = np.array_split(arr, 5)
print(newarr)
```

```
[array([11, 22]), array([33, 44]), array([55, 66]), array([77]), array([88])]
```

## 2) Accessing Def Element of Split Array.

The above methods split a big array into smaller arrays. This method will help you access a desired element of the split array.

```
import numpy as np
arr = np.array([11, 22, 33, 44, 55, 66, 77, 88, 99])
newarr = np.array_split(arr, 3)
print(newarr[1])
```

```
[44 55 66]
```

## 3) Splitting a 2 D Array into Multiple

**2 D Arrays.** Use the **array_split()** method, pass in the array you want to split & the num of splits you want.

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9],
                [10, 11, 12], [13, 14, 15], [16, 17, 18]])
newarr = np.array_split(arr, 3)
print(newarr)
```

```
[array([[1, 2, 3],
        [4, 5, 6]]), array([[ 7,  8,  9],
        [10, 11, 12]]), array([[13, 14, 15],
        [16, 17, 18]])]
```

Output is 3 x 2D arrays of 3 elements each.

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 14 OOP Basics

## 1 Programming Styles

Python offers four coding styles to programmers. These are:

- **Procedural.** This is focused on step-by-step operations, where common tasks are placed in Funs that are called as needed. This coding style favours iteration, sequencing, selection, & modularization. Python excels in implementing this particular paradigm.
- **Object-oriented.** This relies on data fields that are treated as objects & manipulated through prescribed methods. This favours code reuse. Python support this paradigm to a great extent.
- **Imperative.** This style is useful when manipulating data structures & produces, using elegant yet simple codes. Focus is on how a program operates. Programs change state information as needed in order to achieve a goal
- **Functional.** In this every statement is treated as a mathematical equation & any forms of stateor mutable data are avoided. It lends itself well to parallel processing as there is no state to consider.

While learning basics, we have used **procedural programming.** It is typically a list of instructions that execute one after the other starting from the top. This was best for learning the basics. As we advance, we need to shift to **Object-Oriented Programming (OOP).** Due to the overwhelming popularity of the above two, we shall skip the remaining two. The good part is that there are no rules that say you can't combine styles. Python doesn't stop in the middle of interpreting your application to display a style error when you mix styles. Having finished learning basics, **use the style most comfortable to you** for the given task.

## 2 Procedural & Object-Oriented Programming

- The objective of procedural programming is to break down a program into a collection of var & data structures focused on records, modules & call procedures.
- The main aim of object-oriented programming is to break down a programming task into objects focused on methods, message & classes.
- In simple words, procedural programming uses procedures to operate on data structures, while object-oriented uses objects for this purpose.

## Comparison between Procedural & Object-Oriented Programming.

| Procedural Programming | Object-Oriented Programming |
|---|---|
| Focused on the processes & funcs | Focused on data & classes |
| The different parts of the program are connected via parameter passing | The different functions of objects are connected via message passing |
| Does not support advance concept like inheritance | It has four main concepts – Abstraction, Encapsulation, Inheritance, and Polymorphism |
| Data hiding is not possible. | Data hiding is possible, hence more secure than procedural. |
| Requires more development time, is difficult to maintain & is slower | Requires lesser development time, is easy to maintain & is operationally faster |
| Limited operator overloading | Operator overloading is a key attribute |
| Designs cannot be reused & recycled throughout the program. | Can be used throughout |
| While solving issues, they need to be addressed individually. | Objects and classes can be referenced throughout the program. |

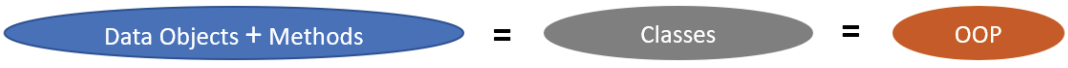
# 3 Introduction to Object Oriented Programming (OOP)

## 1) What is OOP.

Coding done by us so far is also referred to as Procedural Programming. In this we have seen that programming involves the use of data, along with methods & funcs, that work on this data as per specified procedures, to give a desired output when called. Diagrammatically it is represented as:


While this approach is the easiest to learn basics, it is more difficult to work with, having limited flexibility, capabilities & code reuse.

OOP is a **programming concept** that adds an element of logic into the data objects. As against working on data & methods/funcs as separate entities, OOP binds data (objects) with methods & funcs that work on them, being converted into a single entity. Diagrammatically it can be represented as below;




It is focused on creating reusable codes.

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

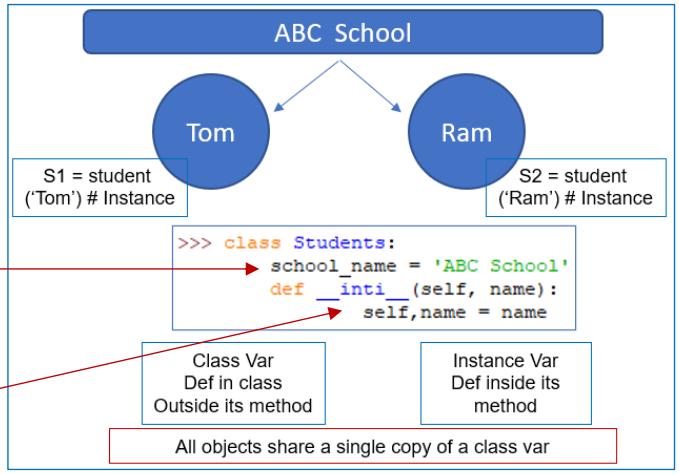
# 15 OOP – Working with Classes

## 1 Class Variables & Instance Variables

### 1) Data Members/Variables in a Class.

These are var (ex self.name), that def attributes that hold data values, associated with a class & its objects. There are two types of data members or data var:

- **Class Variables:** A class var is a var that is declared inside a class, but outside its methods.
- **Instance variables.** If the value of a var varies from object to object, then such var are called instance var.




### 2) Wrong Use of a Class Var.

While coding, we should take particular care in the use of class var. This is because all objects share the same copy. Thus, if one of the objects modifies the value of a class var, then all objects start referring to the fresh copy. This is undesirable.

### 3) Difference between Class & Instance Var.

- If the **value of a var is not varied from object to object,** such a var is called **class var** or **static var.** Values of an instance var, vary from object to object. Ex, in Student class, we can have different instance var such as name & roll num because each student's name & roll number will be different.
- Only one copy of the class variable will be created & must be **shared by all instances of a class.** Ex, if we want to include school's name with the student class, we must use the class var instead of an instance var. This is because school's name is the same for all students. So instead of maintaining a separate copy in each object, we can create a single class var holding school's name, sharable by all students (objects).
- We can add any number of class var or instance var in a class.

### 4) Comparison between Instance & Class Var.

| Instance Var | Class Var |
|---|---|
| Not shared by other objects. Every object needs its own copy | Shared by all instances |
| Declared inside the constructor or Instance method of making the class | Declared inside the class definition but outside any of the instance methods or constructor |
| Gets created when an instance of the class is created | It is created when the program begins to execute |
| Changes made to these variables through one object will not reflect in another object | Changes made in the class var will reflect in all objects |

## ❷ Working with Class Variables

### 1) Accessing Class Variables.

This is the first operation we conduct on any code. Say our code is:
Once made it is Stored in the memory.
Now when we want to access it, we need to create our first object named p2 (lines 1 & 2 of blue box below). On call, output is displayed:

```
class Phone():
    def __init__(self, cost,color, battery, ram): #using constructor and
                        # passing 4 parameters(cost, color, battery, ram)
        self.cost = cost #initialising cost, similar for other parameters
        self.color = color
        self.battery = battery
        self.ram = ram
    def show_specs(self): #making a func for print statements
        print('Cost of Phone -', self.cost)
        print('Color is', self.color)
        print('Battery in mAh -', self.battery)
        print('Ram in GB -', self.ram)
```

```
p2 = Phone(30000, 'Black', 6000, 6) # making first obj and passing all parameters value
p2.show_specs() #invoking class
```

### 2) Making another Object to Re-use the Methods of above Class.

Say we want to use it for another phone costing 5000, colour blue, battery in mAh 7000 & 8 GB ram. These values now need to be added as args in the new object p3 shown below.

```
p3 = Phone(50000, 'Blue', 7000, 8) # making second obj and passing all parameters value
p3.show_specs() #invoking class
```

```
Cost of Phone - 50000
Color is Blue
Battery in mAh - 7000
Ram in GB - 8
```

In similar way we can add more objects and access their parameters.

### 3) Modifying Class Variables.

All changes to the attributes of the class like amend, add or delete, will have to be done in line 2. Here we have added **duration.**

```
1  class Employee:
2      def __init__(self, name, project, duration):
3          self.name = name
4          self.project = project
5          self.duration = duration
```

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

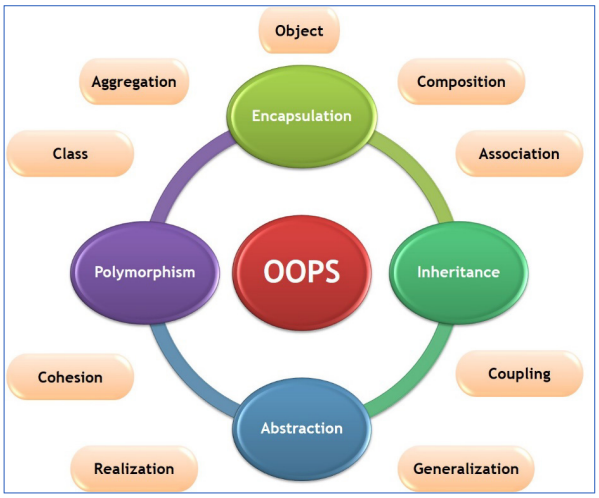
CHAPTER

# 16 OOP – Working with Classes

So far, we have learnt about role of objects& classes in OOPS.
To harness its full potential, we now need to learn the remaining methods:
The four most important are:

- Inheritance.
- Polymorphisms.
- Encapsulation
- Abstraction.

Let us learn them one by one.

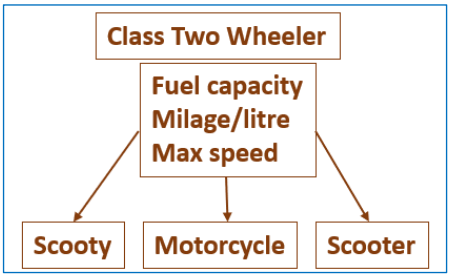



## 1 Inheritance

### 1) What is Inheritance.

It is one of the most important attribute of OOP. Inheritance implies **transfer of properties of one class, to another classes,** that is derived, or can be derived from it. This improves code re-usability.

Consider this diagram. We need to create three classes named scooty, scooter & motorcycle. The attributes fuel capacity, milage/litre & max speed will be same for all three. In normal way, we will have to repeat the code for all three classes. Using inheritance, we can write the funcs once, & the rest can inherit them from the base class. This is done by creating a class two-wheeler & writing the attributes in it. Thereafter, for others will inherit the attributes from the two-wheeler class, thus avoiding duplication.




### 2) Classes of Inheritance.

Inheritance has two classes:

- **Super Class:** The class whose properties are inherited by sub class is called Base Class or class A or Super class or **parent class.**
- **Sub Class:** The class that inherits properties from another class is called Sub class or class B or Derived Class or **child class.**

In above ex, two-wheeler is the s

In above ex, two-wheeler is the super class & scooty, scooter & motorcycle are sub class. The interesting thing is, **along with the inherited properties & methods, a child or sub class can have its own properties & methods.**

## 3) Creating a Parent Class.

This is done in two steps:

**Step 1.** Lines 1 to 5 as (named Parent1) with patient name, age & disease as attributes.

**Step 2.** Line 6 & 7 make a func to print the attributes. With this the parent class is made.

```
class Parent1:
    def __init__(self, patient_name, age, diesease):
        self.patient_name = patient_name
        self.age = age
        self.diesease = diesease

    def func(self):
        print(self.patient_name, self.age, self.diesease)
```

**Verification.** To verify if the code is correct we make an object & def a call method. This step is not mandatory here, but a good practice.

```
patient1  = Parent1('John', '25 : ', 'Dengue')
patient1.func()
```

## 4) Creating a Child Class.

This has two steps:

**Step 1.** Create a child class (named Child1), this is followed by parent1 in round brackets & : to enable it to inherit parent class functionalities. To do so we add a **pass** statement because we are not adding any other attribute to this child class. This will enable it to inherit them from the parent.

```
class Child1(Parent1):
    pass
```

**Step 2.** Now we need to create an object for child class, as we did for parent class (step3). Here this step is Mandatory. Output is in red box.

```
patient2 = Child1('Thomas', '28 : ', 'Covid')
patient2.func()
```

```
John 25 :  Dengue
Thomas 28 :  Covid
```

## 5) Making more Childs of same Parent.

To make more child of same parent (parent1) with same parameters, follow steps 1 & 2 of above code. They all will inherit, all the attributes of the parent class.

## 6) Adding Parameters to a Child in addition to those of Parent.

In case we have to add parameters, other than those of the parent, in any of the child, then we have to initialise the child class the way we did the parent class in step 1. Taking the above ex, we will create a new child class named child2 & **add a parameter 'Condition'.** In 1st line we create child2. In 2nd line using--init-- we are passing attribute incl new parameter (condition). In 3rd line we are using **super(),** which is reserve word of python. This makes the child inherit all methods & properties from parent.

```
class Child2(Parent1):
    def __init__(self, patient_name, age, diesease, condition):
        super().__init__(patient_name, age, diesease)
        self.condition = condition

    def func2(self):
        print(self.patient_name, self.age, self.diesease, self.condition)

patient2 = Child2('Nick', '28 : ', 'Covid | ', 'Normal')
patient2.func2()
```

```
John 25 :  Dengue
Thomas 28 :  Covid
Nick 28 :  Covid |  Normal
```

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER 17

# OOP – Encapsulation & Abstraction

## 1 Encapsulation

In collections we wrap **data** in brackets to create data types like list, tuple etc. The methods of working on them are kept separate. Encapsulation **describes the idea of wrapping data & the methods that work on them within a single unit (Method + Data).** Besides other advantages, this puts restrictions on accessing variables and methods directly & can prevent the accidental modification of data. A class is an example of encapsulation as it binds all data members (instance var) & methods into a single unit.

### 1) Encapsulation & Data Security.

When working with classes & dealing with sensitive data, providing global access to all the var in the program is not a good idea. Encapsulation offers a way to access the required var without providing full-fledged access to all var. Ex giving access to different users & administrators in a web site.

Updating, modifying, or deleting data from variables can be done using methods that are defined specifically for the purpose. The benefit of using this approach to programming is improved control over the input data & better security.

### 2) Levels of Access & Data Security Provided.

Encapsulation restricts privileges & access to the program. It provides three levels of access:

- **Public access.** This is provided to any func in any class. This provides low level data protection.
- **Protected access.** This is provided to classes where declared, or to classes which inherit them. This provides med level of security.
- **Private access.** This is provided only by class where declared. This provides high level of data security.

### 3) Methods of Access Control in Python.

Concept of encapsulation is same for all object-oriented languages. Language like Java uses **access modifiers** (public or private) to provide access to its var & methods. Since python does not have access modifiers, it uses a **few different methods** to control the access to variables within its programmes.

### 4) Option 1 – Using Single Underscore.

A common Python convention to identify a private var is by prefixing it with an underscore (see age in Line 4, 8 & 15).

The output is in red box.
It shows that the var access directly from outside (lines 14 & 15) is unchanged to that of access using class method (line 12).
The question is, this is really not private, can we do anything to make it really private?
For this see option 2.

```
Ram
22
Ram
22
```

```
1   class Students:
2       def __init__(self, name, age=0):
3           self.name = name
4           self._age = age
5
6       def display(self):
7           print(self.name)
8           print(self._age)
9
10  selectedStudent = Students('Ram', 22)
11  # accessing using class method
12  selectedStudent.display()
13  # accessing directly from outside
14  print(selectedStudent.name)
15  print(selectedStudent._age)
```

## 5) Option 2 – Using Double Underscore to Access Private Var.

The code is similar to option1, except that

- It uses double underscore before age in lines 4, 8 & 15.

```
4           self.__age = age
```

- It adds a new line 16.

```
16  print('Trying to access var from outside the class')
```

Now when we run the code, we notice two things:

- The output for accessing using class method Line 12 is as shown in this red block.

```
Ram
22
```

It has access to both methods (name and age).

- When we access directly from outside, method name is printed

but when it comes to accessing age, it is denied with an error.

```
    print(selectedStudent.__age)
AttributeError: 'Students' object has no attribute '__age'
```

## 6) Option 3 – Using Getter & Starter Methods to Access Private Var.

The primary purpose of using getters & setters is to ensure data encapsulation. Use getter method to **access** (get) data members & the setter methods to **modify** (set) data members.

# 2 Abstraction

Coding is all about giving a **good experience to the end user** who will ultimately use the code. Classes, methods & funcs are tools a programmer uses to make codes that will deliver this experience. Use of normal classes may require a user to give inputs related to the backend processing. As against using normal classes, if a programmer **makes the code using abstract classes,** In may apps, these will **hide all the backend requirements** from the end user, making his experience better.

Take the example of power point or excel or photo shop or any other. It has hundreds of utilities which we keep using without bothering about how the background processing is done. We simply keep entering the most basic inputs. When we go to a coffee machine it gives us many choices but asks for no further details. **This capability only imparts efficiency.** Much the same way in object-oriented programming, abstraction is able to hide a lot of processing complexities, to give us a front end using which we focus only on the end result, as against the route to the end result.

## To view more, please contact:


Diwaker :- +91 93122 64502

Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

CHAPTER

# 18 Your Journey Ahead

Scope of programming with Python is huge. There are various aspects of Python that makes it perfect for software & app development of any kind. Apart from being used as the primary programming language in projects, software developers also use Python as a support programming language for project management, build control, and testing. In order to take Python as a professional partner, we would recommend:

- **Achieve** full clarity on the basics using its IDLE.
- **Consolidate** that experience through simple projects using any IDE.
- **Advance** to get working understanding of some of its key packages, modules & libraries like Pandas, Matplotlib, NumPy, OpenCV, Datetime module, QR module etc. These will give you the required confidence besides clarity in what to do.
- **Select** advance modules & libraries mentioned in the applications below. This will help you land a good job in a short time.

## 1 Data Science and Data Visualization

Data plays a decisive role in the modern world. Why? Its because it is key to understanding the people and their taste in things around them by gathering and analyzing crucial insights about them. This is what the entire domain of Data Science revolves around. Data Science involves identifying the problem, data collection, data processing, data exploration, data analysis, and data visualization. Python ecosystem offers several libraries that can help you tackle your Data Science problems head-on. These include **Pandas, NumPy, Matplotlib, Plotly, Ggplot, SciPy, & TensorFlow.** In addition are libraries that specialize in creating and fine-tuning Deep Learning and Machine Learning models, performing intensive data crunching and data manipulation.

## 2 Artificial Intelligence and Machine Learning

These are undoubtedly among the hottest topics of this decade & it works on data. These are the **brains behind the smart tech** that we so rely on today to help us make optimized decisions. Python, along with a handful of other programming languages, has seen a steep increase in their use for developing AI & ML powered solutions.

**Python's stability and security** make it a perfect programming language for handling the intensive computations that keep the Artificial Intelligence and Machine Learning systems running. More so, Python's vast collection of libraries facilitate the development of models and algorithms that run modern AI and ML systems. Some of the popular libraries for the job are:

- **Pandas** for data analysis and manipulation.
- **SciPy** for scientific and technical computing.
- **Keras** for Artificial Neural Networks.
- **TensorFlow** for Machine Learning tasks, especially Deep Neural Networks NumPy for complex mathematical functions and computation.
- **Scikit-Learn** for working with various Machine Learning models.

## 3 Image Processing

Due to the ever-increasing use of Machine Learning, Deep Learning, and Neural Networks, the role of image (pre)processing tools has also skyrocketed. To fulfill this demand, Python offers a host of libraries that simplify much of the initial preparatory tasks of a Data Scientist. Some of the popular image processing Python libraries include **Sckit-Image & OpenCV.** Common image processing applications that use Python are **GIMP, Blender, Houdini, & Corel PaintShop.**

## 4 Text Processing

Text Processing is among the most common uses of Python. For the uninitiated, Text Processing is very closely related to Natural Language Processing. Text Processing allows you to handle enormous volumes of text while giving you the flexibility to structure it as you wish. If you're thinking about sorting lines, extracting text, reformatting paragraphs, and such, you're correct. With Python's text processing capabilities, you can do a lot more than that.

## 5 Embedded Applications

By far one of the most fascinating applications of Python is the ability to run on embedded hardware. For those new to this, embedded hardware is a tiny computer that's created to perform limited actions. An embedded application is what drives the hardware, aka the firmware. Popular examples of these applications include **MicroPython, Zerynth, PyMite, and EmbeddedPython.** As of today, we have an exhaustive list of embedded devices because they're almost everywhere. For example, Digital Cameras, Smartphones, Raspberry Pis, and Industrial Robots are just some of the many devices that can be controlled with Python. FYI, not a lot of people know this, but Python can also be used as an abstraction layer in a device firmware while C/C++ handles the system level side of things.

## 6 CAD Applications

Mostly used for product design by industries such as automotive, aerospace, architectural, and many more, **Computer-Aided Design(CAD) applications** allow product designers and engineers to design products with precisions right down to the millimetres. Python has also conquered the domain of CAD with its highly popular and efficient offerings, such as **FreeCAD, Fandango, PythonCAD, Blender, and Vintech RCAM.** These applications provide industry-standard features like macro recording, workbenches, robot simulation, sketcher, support for multi-format file import/export, technical drawing modules, and much more.

## To view more, please contact:


Diwaker :- +91 93122 64502

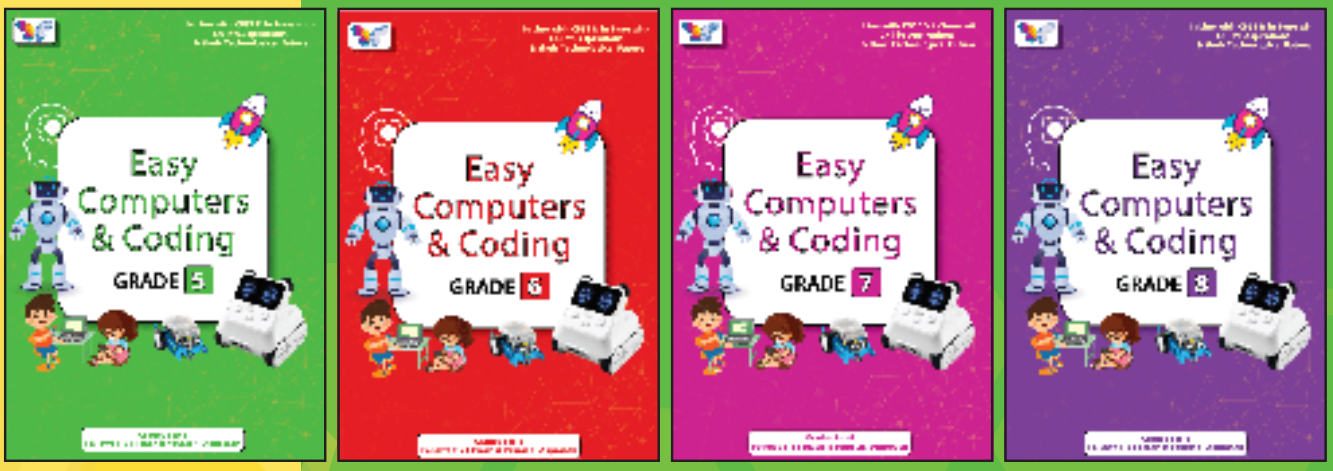
Pankaj Kabir Panthi :- +91 88514 60895

www.bdseducation.in

# OUR BOOK TITLES

## Level-1


## Level-2




## Level-3


BDS EDUCATION