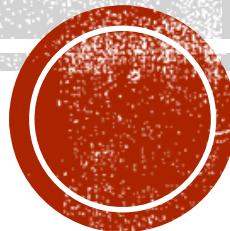


# **INTERNSHIP ON DATA SCIENCE WITH PYTHON**



# SIDE A

- Basics of Python
- Lists
- Tuples
- Dictionaries
- Numpy
- Pandas
- Matplotlib
- Webscraping
- Case Study on Numpy
- Project on Pandas (Predicting Covid 19 cases in India)



# UNIQUE IN THIS PROGRAM

## Python for Data Science

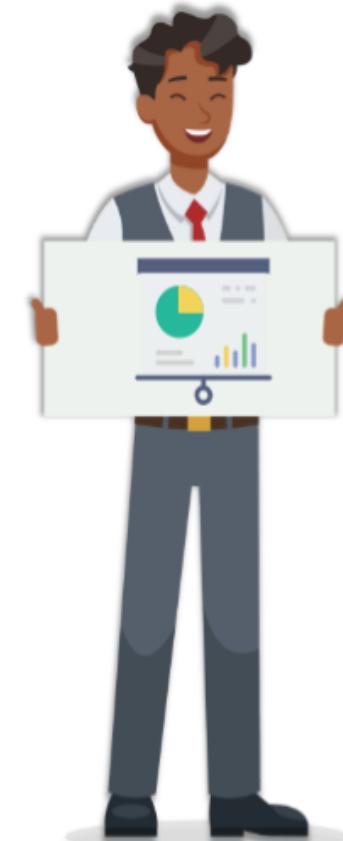


- ☞ Python is a widely used programming language which is highly preferred for Data Science.
- ☞ It offers many libraries for analysis purpose that we are going to cover in our program:
  - Pandas
  - NumPy
  - Matplotlib
  - Seaborn
  - Scikit-learn



# Probability and Statistics

- ⌚ Knowledge of mathematics is required to become a Data Scientist, especially probability and statistics.
- ⌚ This program will cover **Statistical Analysis** that allows you to summarize your data and draw useful insights from it.
- ⌚ This will involve performing:
  - Exploratory Data Analysis (EDA)
  - Descriptive Statistics
  - Inferential Statistics



# Machine Learning



- Machine learning helps a system to learn automatically and improve from experience without being explicitly programmed.
- You will learn how to train a machine learning model and predict the results.

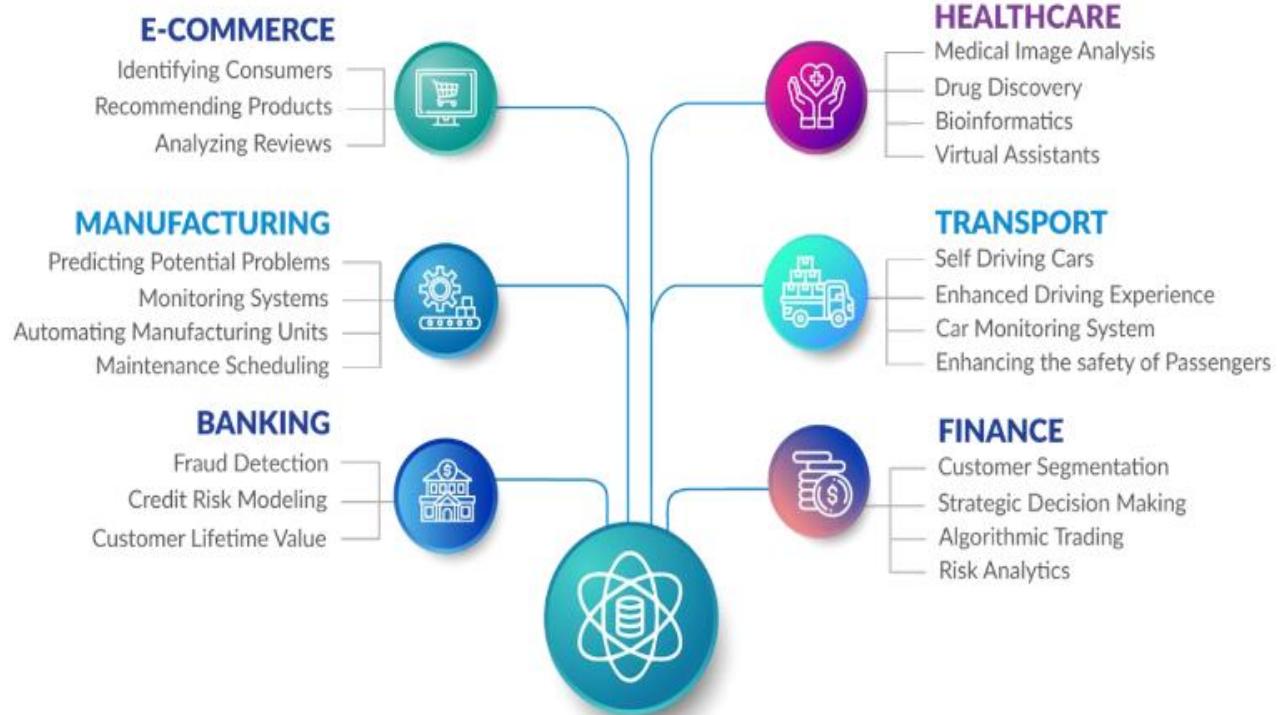


# What is Data Science?

Data Science is basically the use of data to make business decisions by:

- ⌚ Analyzing data for insights
- ⌚ Using statistics to establish relationships
- ⌚ Creating machine learning models for prediction

## Data Science Applications



# Who is a Data Scientist?

---

A data scientist is one who uses data to help businesses in decision making by analyzing, processing, modeling, and visualizing data to interpret the results.



# Data Science FAQs

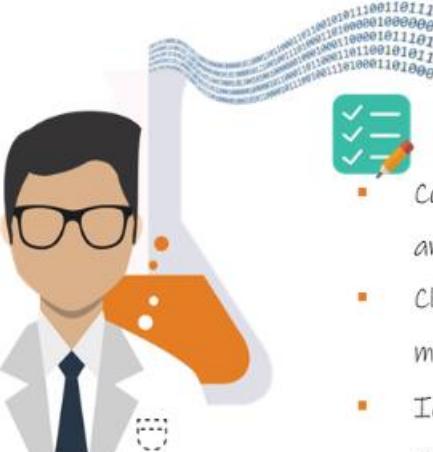
## Who Am I?

I am part analyst and a part artist. I use my technical and analytical skills to extract insights out of the data

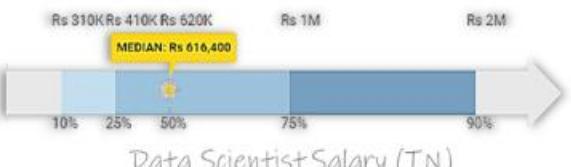


## What did I learn?

- Python/R
- Statistical Analytics
- Predictive Modelling
- Machine Learning
- Natural Language Processing
- Deep Learning



## How much do I earn?



## What do I do?

- Collect data and analyse it from various angles
- Clean existing raw data and build predictive models out of it
- Identify correct business problems and give solution with visualizations

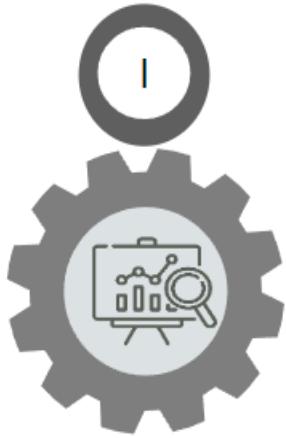
## How do I help my organization?



- Cost Optimization
- Develop Strategies
- Improve Operational Efficiency
- Risk Optimization
- Build Recommender System
- Increase data accuracy



# Evolution of Data Science



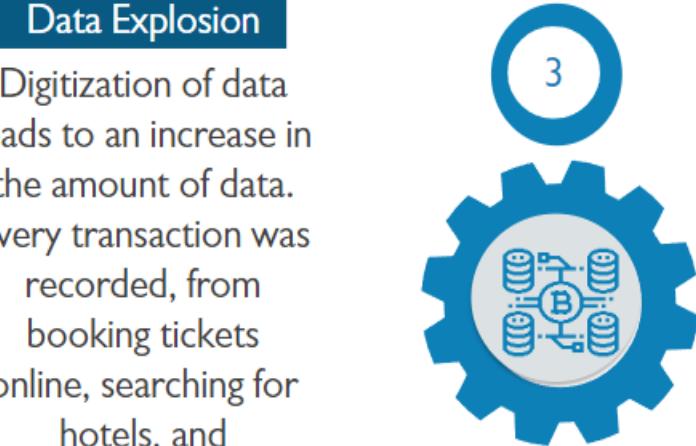
## Statistical Analysis

Performing simple algorithms on small amount of data available in formats such as Company Registers, Application Forms, Feedback Forms, Invoices etc.



## Data Explosion

Digitization of data leads to an increase in the amount of data. Every transaction was recorded, from booking tickets online, searching for hotels, and purchasing items on e-commerce websites.



## Big Data

A large amount of data along with the high speed at which it was getting generated, led to the discovery of tools that can manage such data – Big Data!

## Improved Tools

Big Data, along with technological advances such as Cloud Computing resulted in availability of tools that can handle, process, and analyse large amount of data – Data Science!

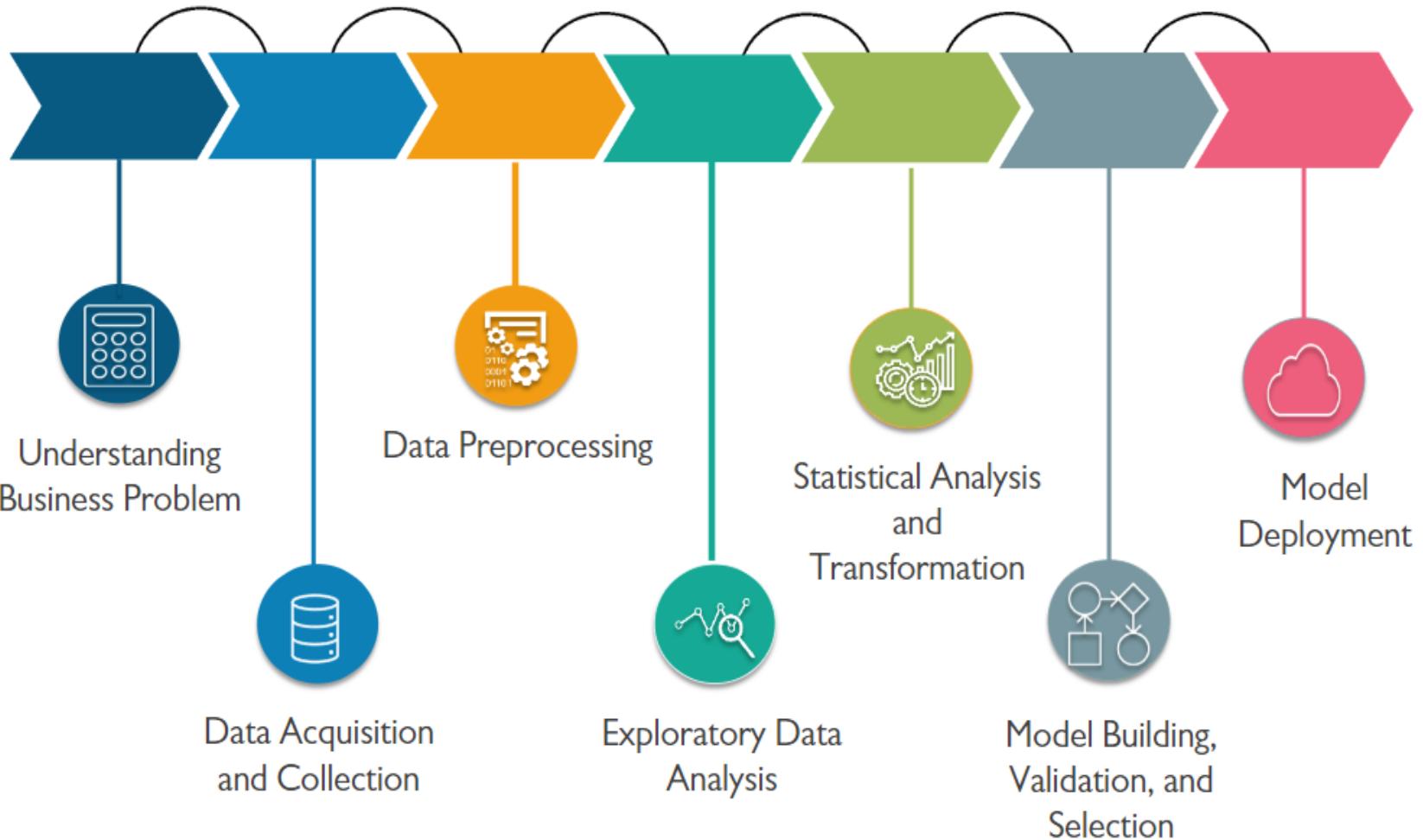


## Data Science

Current generation of data science includes algorithms such as Statistics, Regression and Classification, Clustering, Natural Language Processing, Artificial Intelligence, and Big Data Analytics

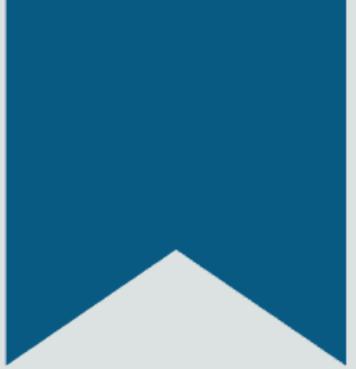


# Data Science Development Methodology



# Introduction to Python





# How are Tech Giants using Python?

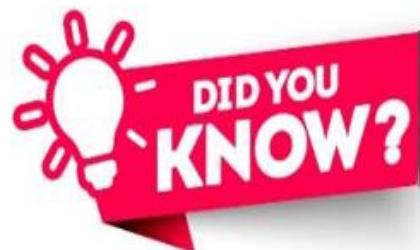


# How are Tech Giants using Python – Google

---

- Python is an official language at Google
- Python and C++ are used to write the core search algorithms at Google
- YouTube: Previously written in PHP (Hypertext Preprocessor) but eventually got replaced by Python
- Google uses Python to implement Machine Learning, AI as well as robotics projects
- Google uses SciPy, Scikit, NumPy, and various kinds of notebook UIs for performing data analysis

Google Logo



Until 2012, the designer and creator of Python, Guido van Rossum was employed by Google, when he left to work for Dropbox

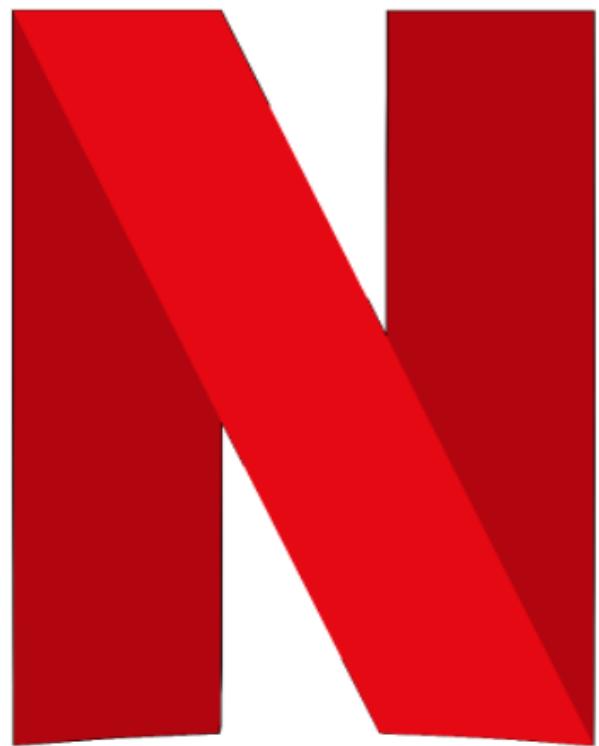


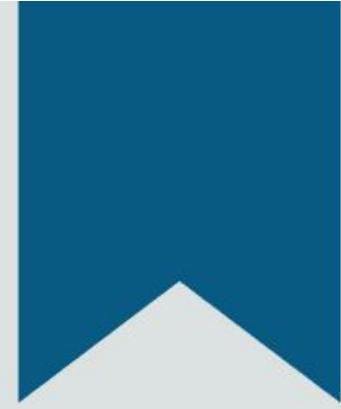
# How Tech Giants are using Python – Netflix

---

- Netflix uses Python for enhancing machine learning capabilities that analyze movies, optimize streaming, and pull-out images to display thumbnails
- Operations: Netflix uses Python libraries like NumPy and SciPy to perform numerical analysis
- Security: Python for security automation, risk classification, remediation, and vulnerability identification

Netflix Logo





# Fundamentals of Python



# Fundamentals of Python – Syntax Rules

01

Python is case sensitive. **Hello** is not equal to **hello**

02

Python does not have a **command terminator**

03

Each line can have one statement at most. For multiple statements, you should use **semicolon ;**

04

Comments in Python start with '# or '''. For 1-line comments, we use '#' and for multiline use ''' (triple quotes either single or double)

05

Python supports **multiline statements**, i.e., **Line Continuation**. To it, we put backslash '\ at the end of each line

Calculator





# Python Token



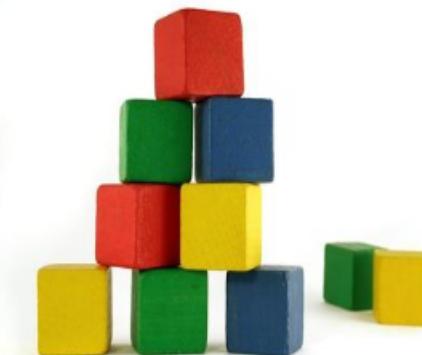
# Tokens in Python

Python breaks every logical line into a set of elementary lexical elements known as **Token**.

## Code Example

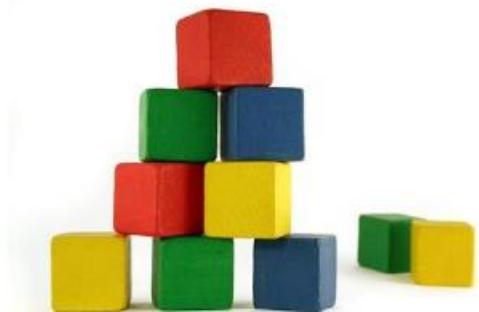
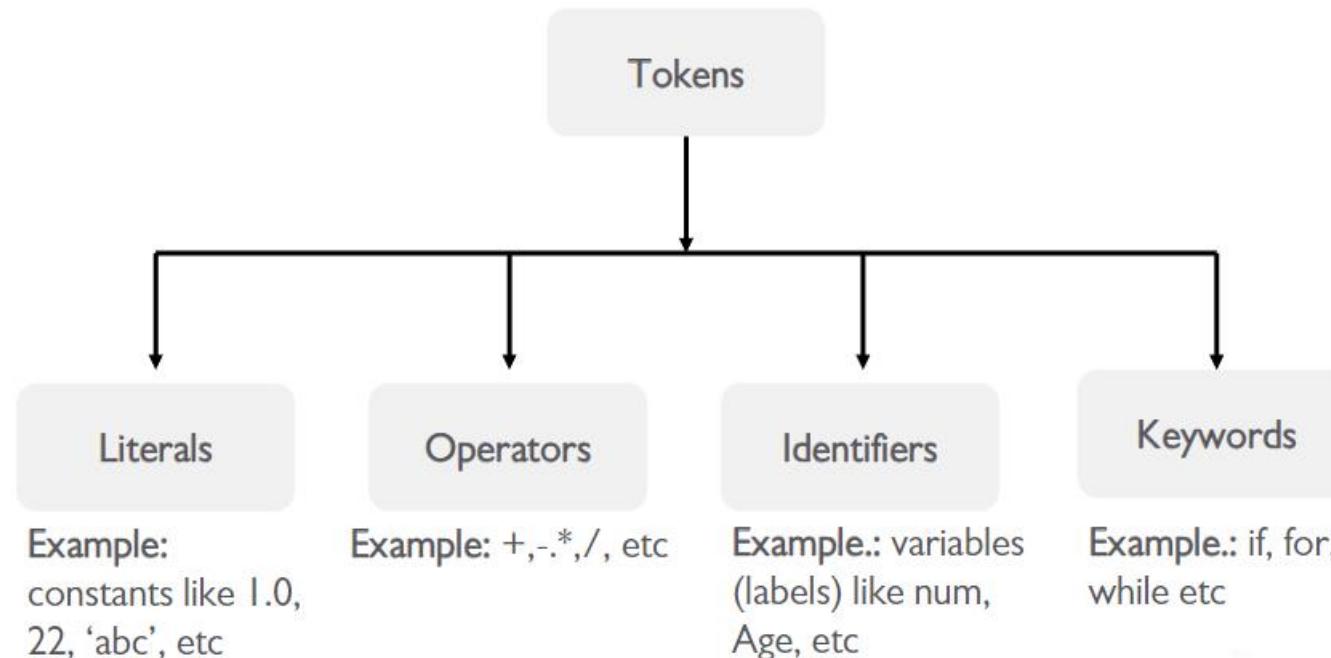
```
# Sum of two numbers in Python
a = 10
b = 20
print(a+b)

# Here 10,20 are literals, stored in
variable(identifier) a, b.
print is a keyword and + is an
operator
```

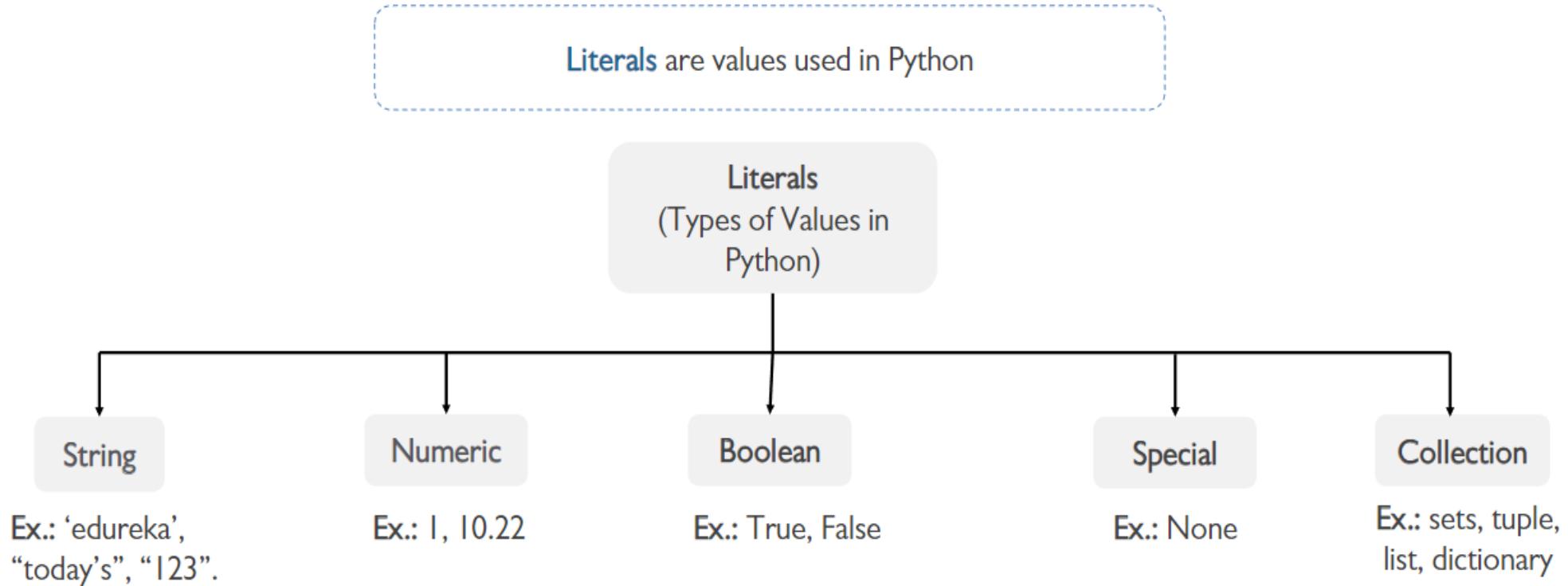


# Tokens in Python: Types

---



# Literals in Python



In other words, literals are data assigned to a variable or a constant



# Keywords in Python

---

Keywords are set of reserved words with some specific functionality

Terminal command for keywords

```
help> keywords

Here is a list of the Python keywords. Enter any keyword to get more help.

False          class           from            or
None           continue        global           pass
True            def             if              raise
and             del             import          return
as              elif            in              try
assert         else            is              while
async           except          lambda          with
await           finally         nonlocal       yield
break          for
```



# Identifier in Python – Just like Labels

An **identifier** is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another

Assigning values *10* and *edureka!* to  
variables *var* and *Var* respectively



## Code Example

```
var = 10  
Var = 'edureka!'  
print(var,Var)
```

OUTPUT

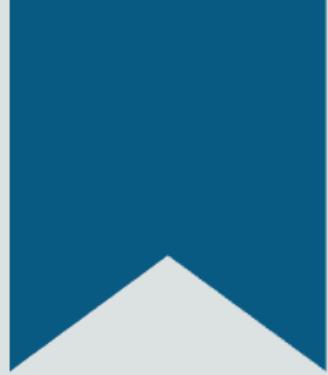


10 edureka!



# Operators and Variables in Python





# Use Case 1 – Build a Basic Calculator



# Use Case I – Building a Basic Calculator

Problem Statement: Write a Python code to build a basic calculator



NOTE: Input must be taken from the user to perform various arithmetic operations



# Operators in Python

Operators are used to perform operations on the operands

For example,  $2 + 3 = 5$ , here **2** and **3** are **operands** and **+** is called **operator**

`+ , -, /, *, %, //, **`

`<, >, !=`

`&, |, >>, <<, ~, ^`

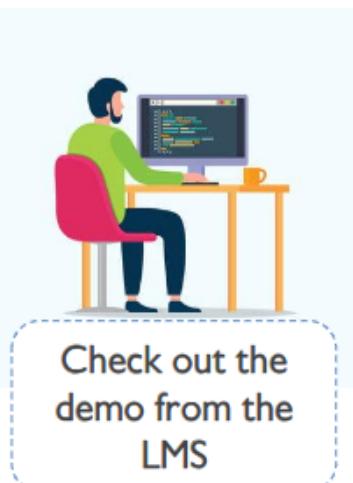
`in, not in`

Arithmetic

Comparison

Bitwise

Membership



Assignment

`= += -= *= /=`

Logical

and, or

Identity

is, is not



# Variables

---

Variables are reserved memory locations to store values

Assigning values 10 and edureka! to variables A and B respectively

Code example

```
1 A=10  
2 B='EDUREKA'  
3 print(A,B)
```



Variable cannot use any special character except underscore



Keyword cannot be a variable name



Variable cannot start with a number

Code output

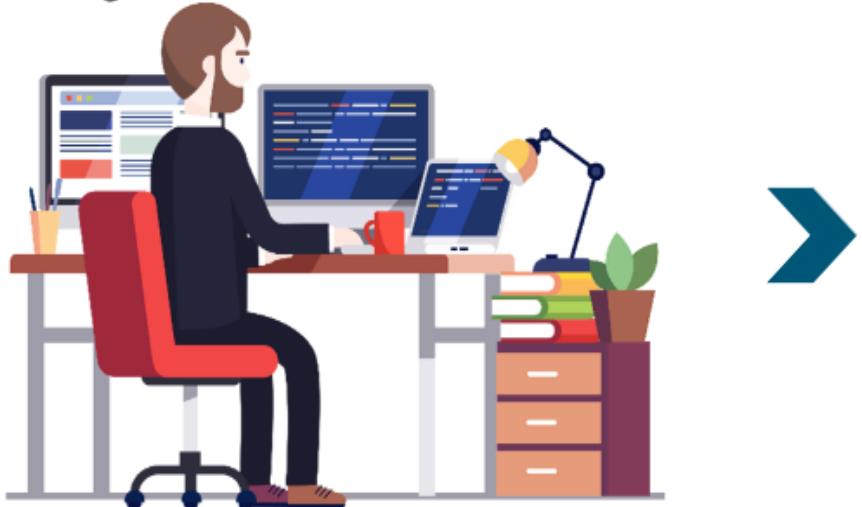
```
10 EDUREKA
```



# Use Case 2 – Multiplication Table

Problem Statement: Write a python code to print the multiplication table of user's choice

Coding



Expected code output

```
1 n = int(input("Enter a number "))
2 for i in range(1, 11):
3     print(n,"X",i,"=",n*i)
```

```
Enter a number 7
7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70
```

NOTE:

- ☞ `input()` is taken from the user of type integer
- ☞ `for` is looping statement, will be covered in subsequent modules



# Input from the User – Illustration 2

- Imagine you are writing a program for cricket game where you give liberty to the user to choose his players
- User enters all three values together, using `split()` function value is assigned to individual variable

Code example

```
All_rounders, Batsmen, Bowlers = input("Enter three values: ").split()  
print("Number of All-rounders : ", All_rounders)  
print("Number of Batsmen are : ", Batsmen)  
print("Number of Bowlers are : ", Bowlers)  
print()
```



Code output

```
Enter a three value: 3 4 4  
Number of All-rounders : 3  
Number of Batsmen are : 4  
Number of Bowlers are : 4
```

Output – Notice multiple  
input entered



# Functions in Python I



# Functions in Python

- Functions in Python are a group of related statements that performs a specific task and can be called repeatedly
- They make extensive programs more organized and manageable

Syntax:

```
def  
function_name(parameters):  
    """docstring"""  
    statement(s)
```

Add docstring to define what a function does

```
def keyword      name      parameter  
def celsius_to_fahr(temp):  
    return 9/5 * temp + 32  
return statement  ↗  
return value     ↙
```

NOTE: Click [here](#) to learn about built-in python function



# Function Definition and Function Call

---

## Function Definition

The set of statements are written once and executed multiple times

## Function Call

Once we define the function, we can call it multiple times

### Code example

```
def print_name(str):  
    print("Welcome to Python, ",str)  
    return()
```

```
str = input("Enter your name :")  
print_name(str)
```

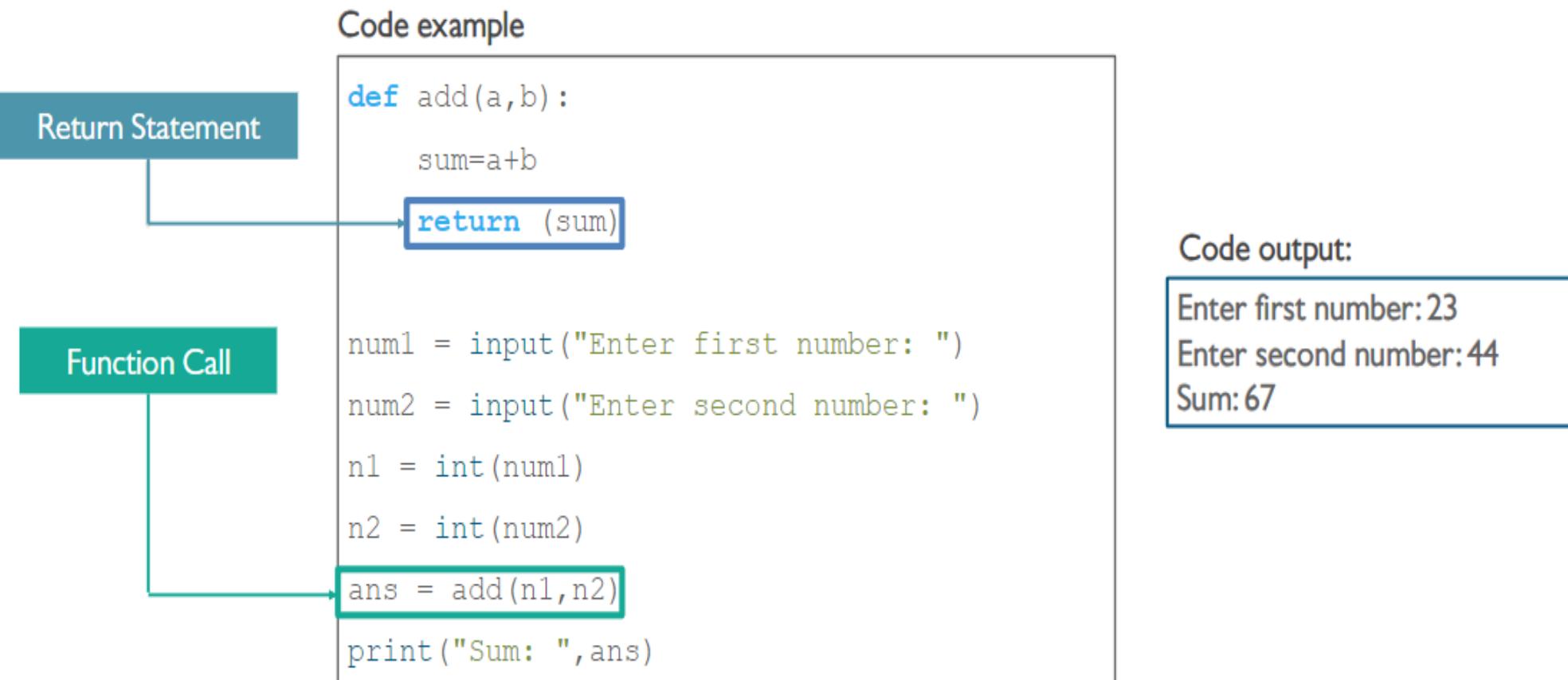
### Code output:

```
Enter your name :Jack  
Welcome to Python,Jack
```



# The Return Statement

The Return statement terminates the execution of a function and returns control to the calling Function



# Why Use Functions in Python - Clarity

Replace multiple lines of code with a single function call, thereby making it easier to understand and maintain

## Without Function

```
print('Hello World')
```

## With Loop

```
count = 10
while count>0:
    print("Hello World")
    count-=1
```

## With Loop & Function

```
def print_msg(count, msg):
    while count>0:
        print(msg)
        count-=1

#Calling the defined function
print_msg(10, 'Hello World')
```



# Why Use Functions in Python - Reuse

Functions which are tried and tested can be reused within/outside the program

Code example

```
print_msg(10, 'Hello World'  
)  
print_msg(15, 'Hola')
```



Prints Hello World 10 times  
Prints Hola 15 times

**NOTE:** You can call the 'print\_msg'  $n$  number of times



# Why Use Functions in Python – Division of Labor

Workload in a large project can be divided with each person working on different functions of the program

**Coder 1**

```
def add(n1,n2):  
    return n1+n2
```

**Coder 2**

```
def sub(n1,n2):  
    return n1-n2
```

**Coder 3**

```
def mul(n1,n2):  
    return n1*n2
```

**Coder 4**

```
def div(n1,n2):  
    return n1/n2
```

**NOTE:** This is just an example symbolizing that all the functions (sub-tasks) can be worked upon in-parallel in a team and can be later combined in the main function



# Range of Function

Python range( ) function generates a sequence of numbers in the form of a list

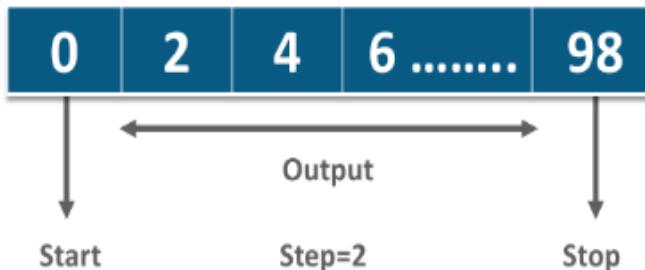
Syntax:

```
range(start, stop, step)
```

Example:

```
for i in  
range(0,98,2):  
    print(i,end='')
```

Parameter	Description
start	Integer specifying the position to start, default is 0 (Optional)
stop	Optional Integer specifying the position to end
step	Optional Integer specifying incrementation, Default is 1



# Scope of a Variable - Function

Global Variables



SCOPE

Local Variables

Declared outside the function and can be used anywhere in the program

Declared within any function and can be used within that function only

Code example

```
a = 50
def number():
    b = 30
    print(b)

print(a)
number()
```

Code example

```
a = 50
def number():
    b = 30
    print(b)

print(a)
number()
```



# Scope of a Variable – Function (contd.)

## Code example

```
a = 30
def sum(b):
    c=30
    sum = b+c
    print('Addition is', sum)
sum(3)
print('Value of c', c)
```

‘a’ is a global variable and can be accessed anywhere in the program

## Result

```
Addition is 33
-----
NameError                                 Traceback (most recent call last)
<ipython-input-53-a10954081147> in <module>()
      5     print('Addition is', sum)
      6 sum(3)
----> 7 print('Value of c', c)

NameError: name 'c' is not defined
```

‘c’ is a local variable and so it cannot be accessed outside the function



# Packages and Modules in Python

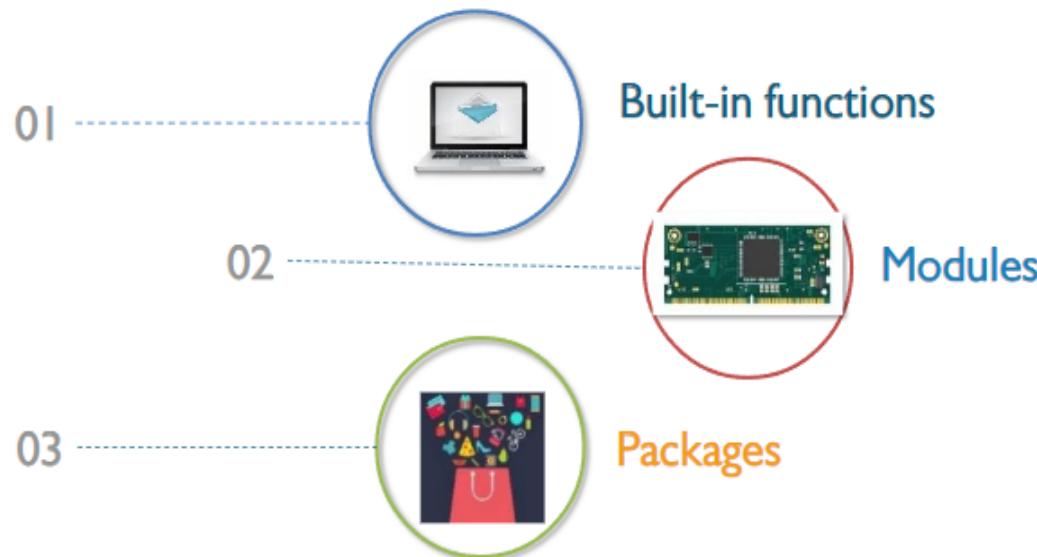


# Understanding Packages and Modules

- Standard library is a collection of tools that come with Python
- A module is a file containing Python definitions and statements
- Python modules are nothing but Python files with .py extension
- A package is a collection of modules example, SciPy is a collection of modules for statistical operations



Standard  
Libraries



# Import Statement

---

- ☞ Use `import` to load a module in Python or `import modulename as desiredname`
- ☞ The module name is an identifier when it is imported
- ☞ When the interpreter encounters an import statement, it imports the module, if the module is present in the search path

## Code example



```
import math  
import random as r
```



# Dir Function

A “dir” is a built-in function which returns the list of strings containing the name defined by the module

## Code example

```
import math  
print(dir(math))
```



## Functions inside math library

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',  
'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',  
'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10',  
'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```



# Import Statement

---

The `from import` statement does not import the entire module, it just imports required files in the module

`from math import sqrt`

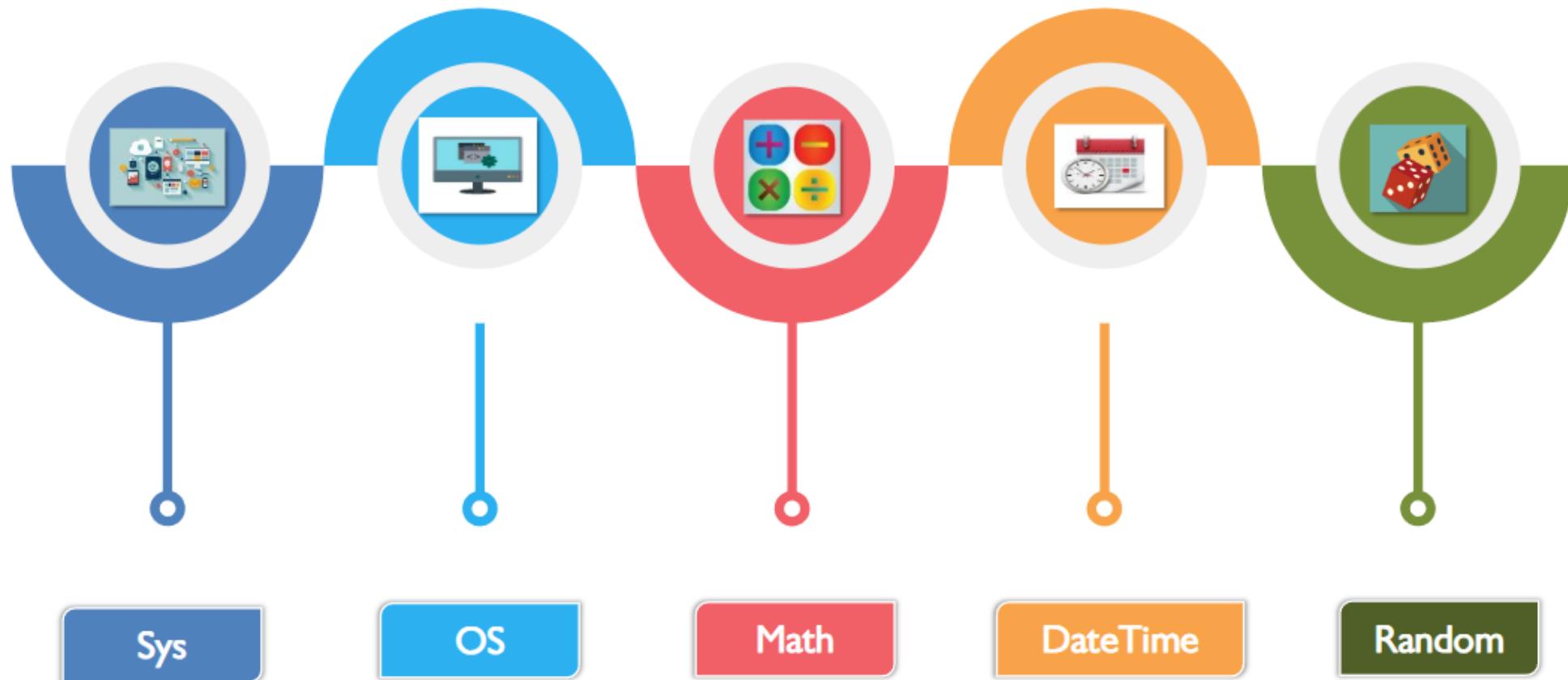
The `from import *` allows you to import all the attributes from the required module  
This provides an easy way to import all the items from a module into the current namespace

`from math import *`

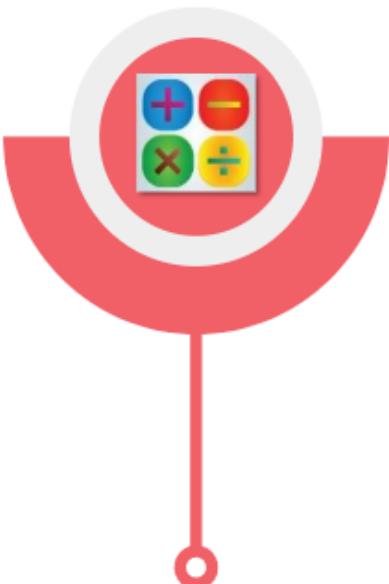


# Important Modules in Python

---



# Math Module



Math

Math module provides access to the mathematical functions

## Code example

```
import math  
  
print(math.ceil(10.098))  
  
print(math.copysign(10, -1))  
  
print(math.fabs(-19.7))
```

Return the ceiling of x as an integer

Return x with the sign of y. On a platform that supports signed zeros, copysign (1.0, -0.0) returns -1.0

## Code output

```
11  
-10.0  
19.7
```

Return positive integer of entered negative value

# Random Module



Random

This module implements pseudo-random number generators for various distributions.

## Code example

```
import random
```

```
num = random.randrange(100)  
print(num)
```

```
ran=random.randrange(0,100,20)  
print(ran)
```

```
inte=random.randint(0,30)  
print(inte)
```

Code output

```
39  
20  
30
```

Generates a random integer within the given range

Return a randomly selected element from range(start, stop, step)

Return a random integer N such that a <= N <= b

# File Handling in Python



# Opening and Closing Files

- Before reading and writing any data into a file, it is important to learn how to open and close a file
- Unless you open a file, you cannot write anything in a file or read anything from it
- And once you are done with reading or writing, you should close the file
- Here are the `open()` and `close()` functions



Opening Files

Code Example

```
file_Object=open(file_name, [access_mode])
```

Name of the file that  
you want to access

Mode in which the  
File has to be  
opened



Closing Files

Code Example

```
file.close()
```



# Open Function – Some Access Modes

---

Modes	Description
r	This is the default mode: Opens a file for reading only
r+	Opens a file for both reading and writing
a	Opens a file for appending
a+	Opens a file for both appending and reading



# Writing and Reading Files

---



```
fileObject.write(string)
```

The **write()** method writes content in an open file.  
Python strings can have binary data and not just text



```
fileObject.read([count])
```

The **read()** method reads a string from an open file  
“count” - counts number of lines in a file



# Renaming and Removing Files

---

Renaming files

```
os.rename(current_file_name, new_file_name)
```

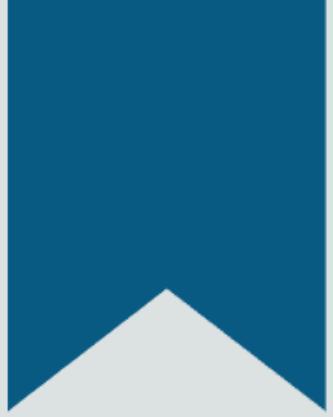
The **rename ()** method takes two arguments, the current filename and the new filename

Removing files

```
os.remove(file_name)
```

The **remove ()** method removes the file





# Classes and Objects



# Creating a Class and Object

---

```
class_name(object):statement(s)
```

## Code Example

```
#Creating a Class
class number():
    pass
```

```
#Creating Instance of
Class
x=number()
print(x)
```

## Output

```
<__main__.number object at 0x7efcf038c860>
```



# Introduction to NumPy Arrays





# Example Scenario: Social Media Opinion Tracker



# Example Scenario: Problem

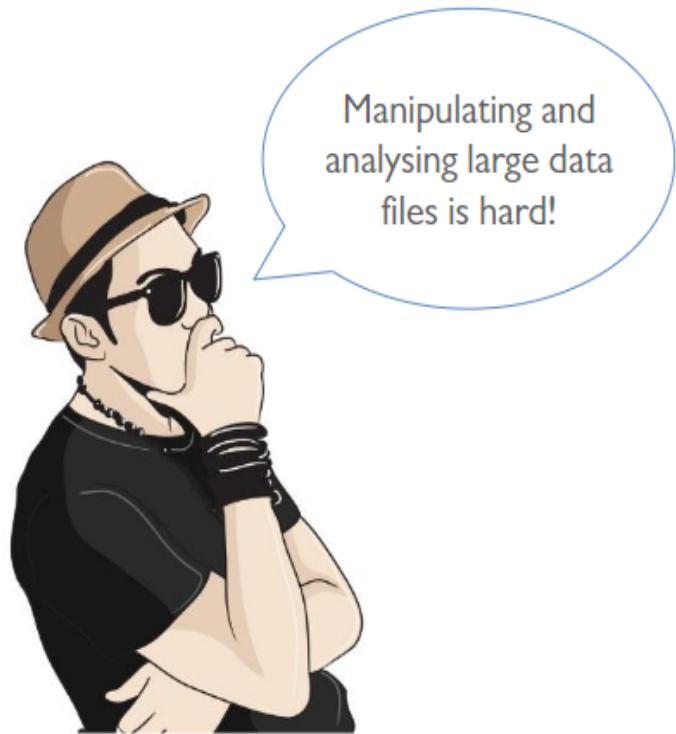
---

- Bob wishes to analyze social media search trends to track user opinion
- He collects all the raw information obtained from internet in the form of various file formats and wants to extract useful information from the collected data



## Example Scenario: Problem (contd.)

---

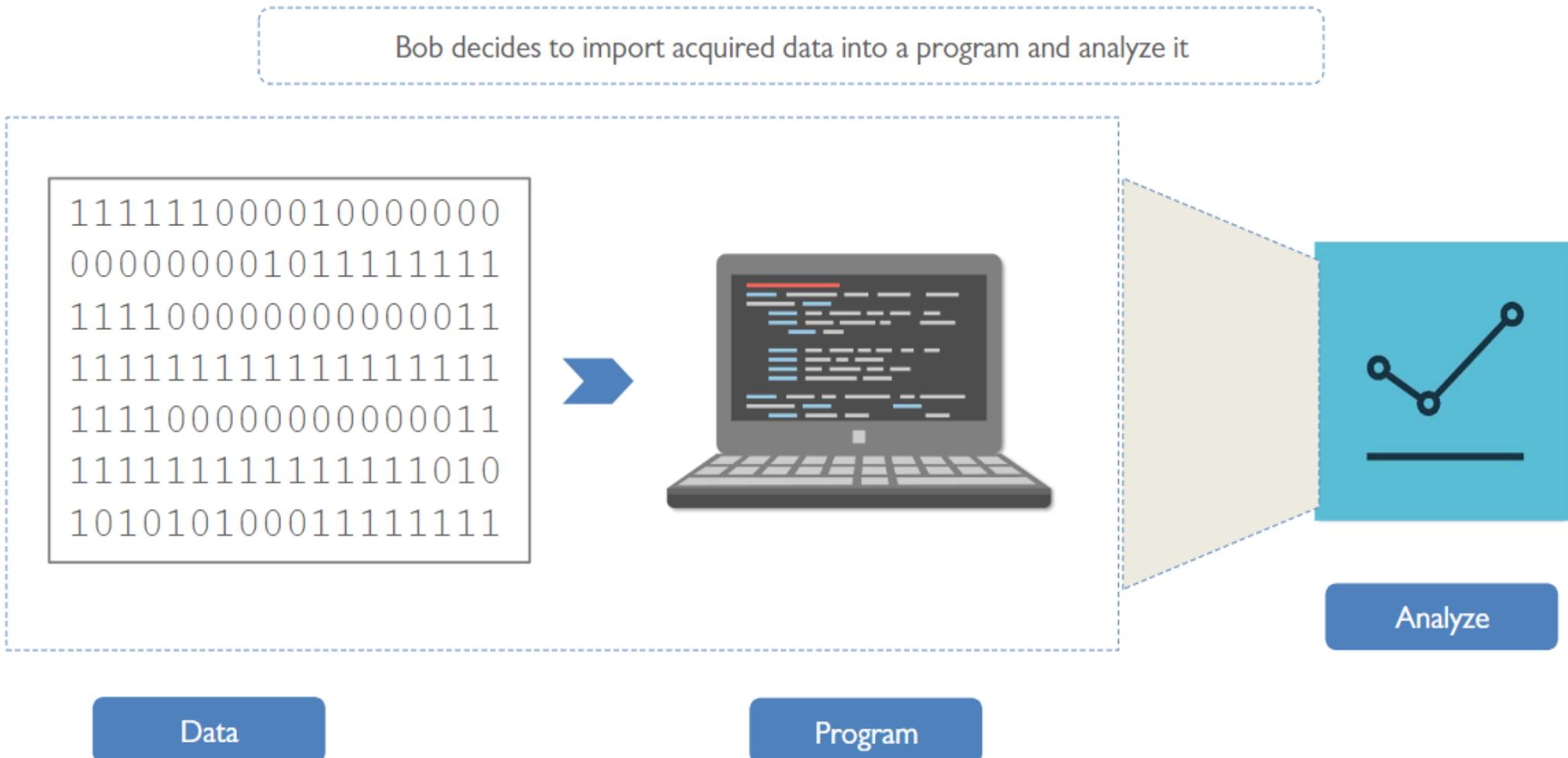


A	B	C	D	E	F	G	H	I	J	K	L	M

He was working with the data files directly!



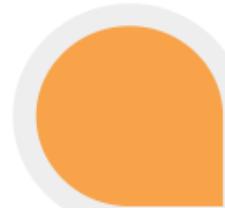
# Example Scenario: Solution



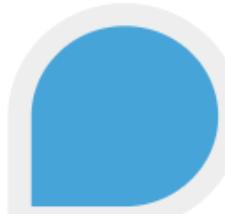
# Example Scenario: Solution using NumPy

There are some features of NumPy which can help Bob to solve his problem

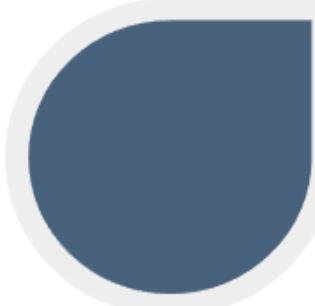
Element by element  
operation



Multidimensional array



Method for processing array



Mathematical operation on  
large dataset which is  
executed efficiently with less  
code





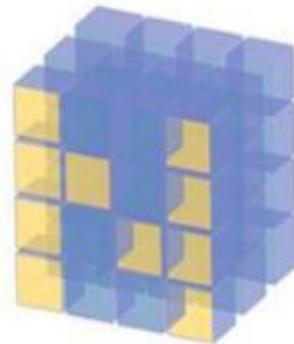
# Introduction to NumPy



# What is NumPy?

---

- ➊ NumPy is the foundation library for scientific computation in Python
- ➋ It contains among other things:
  - ➌ Array creation routines
    - A powerful n-dimensional array object
    - Sophisticated functions
    - Tools for integrating with other languages



NumPy



# Installing NumPy

Open the terminal and install NumPy using pip command as shown below:

Run the command to  
install NumPy in  
command prompt

1

```
(edureka) C:\Users\██████████>pip install numpy
Collecting numpy
  Downloading https://files.pythonhosted.org/packages/bd/51/7df1a3858ff0465f760b48
  2514f1292836f8be08d84aba411b48dda72fa9(numpy-1.17.2-cp37-cp37m-win_amd64.whl (12.8
  MB)
    ████████████████████████████████████████████████████████████████████████████████████████ | 12.8MB 384kB/s
Installing collected packages: numpy
Successfully installed numpy-1.17.2
```

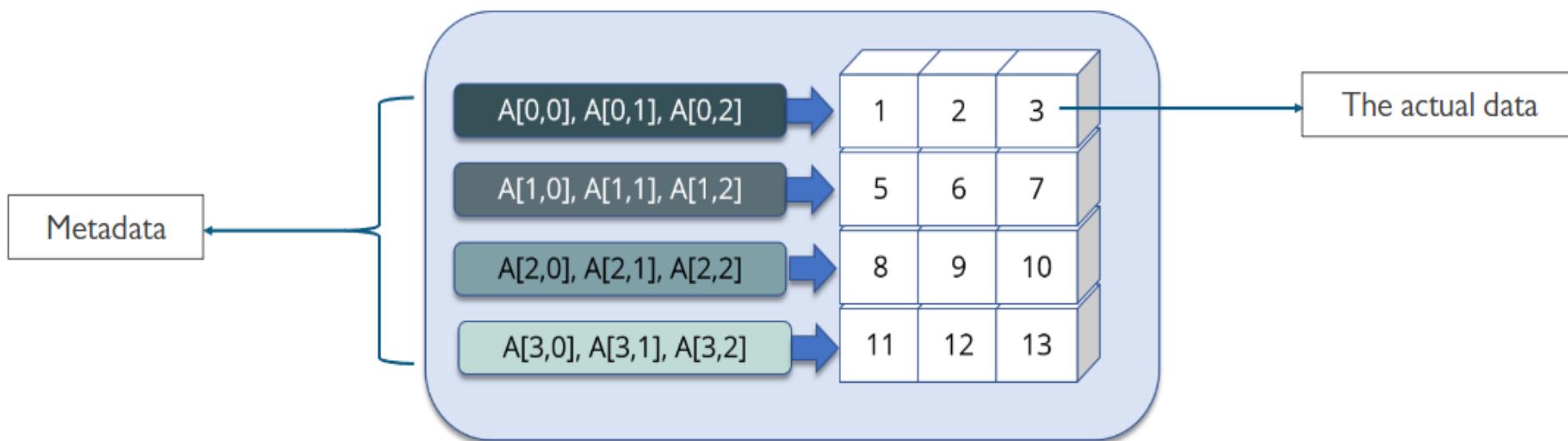
2  
Ensure successful  
installation of the  
included packages



# ndarray (N-Dimensional Array)

- ➊ The ndarray is a multi-dimensional array object consisting of two parts:
  - The actual data
  - Metadata which describes the stored data
- ➋ They are indexed just like sequences in Python, starting from 0

Multi-dimensional array object



# Attributes of ndarray

- NumPy's array class is called ndarray and it is also known by the alias array
- In NumPy, dimensions are called axes
- The important attributes of ndarray objects are:

Attributes	Description
ndarray.ndim	The number of axes of the array
ndarray.shape	This is a tuple of integers indicating the size of the array in each dimension
ndarray.size	The total number of elements of the array
ndarray.dtype	The type of the elements in the array
ndarray.itemsize	The size in bytes of each element of the array



# ndarray – Example

---

Creating a ndarray

```
x = [[0,1,2], [3,4,5], [6,7,8]]  
arr = np.array(x)  
arr
```

```
array([[0, 1, 2],  
       [3, 4, 5],  
       [6, 7, 8]])
```

Performing different attributes on ndarray

Input arr.shape

Output (3, 3)

Input arr.ndim

Output 2

Input arr.size

Output 9

Input arr.dtype

Output dtype('int32')

Input arr.itemsize

Output 4





# Array Creation Routines



# Array Creation Routines



# ones () and zeros ()

---

The zeros () function, creates a full array of zeros with dimensions defined by the shape argument

Code example and output

```
import numpy as np  
arr = np.zeros((5,5))  
print(arr)
```

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

While the ones () function, creates an array full of ones in a very similar way

Code example and output

```
import numpy as np  
arr = np.ones((5,5))  
print(arr)
```

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



# From Existing Data – array()

Creation of a single-dimension NumPy array:

Code example and output

```
import numpy as np  
  
a = np.array([1, 2, 3])  
  
print(a)
```

Import the NumPy module

Calling the array function

[1 2 3]

# From Existing Data – copy ()

- In copy () function the contents are physically stored in different location
- To generate a complete and distinct array, use the copy () function

Code example and output

```
import numpy as np  
a = np.array([1,2,3,4])  
c = a.copy()  
print(c)  
a[0]=0  
print(c)
```

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

Copy of “a” is created

Note: Even if the value of the items of a is changed, c remains unchanged



# Using Numerical Ranges – arange ()

arange () returns evenly spaced values within a given interval

Code example and output

```
import numpy as np  
arr = np.arange(0, 20)  
print(arr)
```

Array of numbers from 0 up to 19 and excluding 20

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```



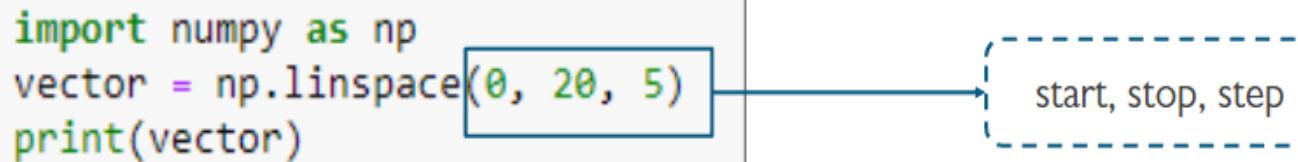
# Using Numerical Ranges – `linspace()`

---

`linspace()` returns a NumPy array of evenly spaced numbers over a specified interval

Code example and output

```
import numpy as np  
vector = np.linspace(0, 20, 5)  
print(vector)
```

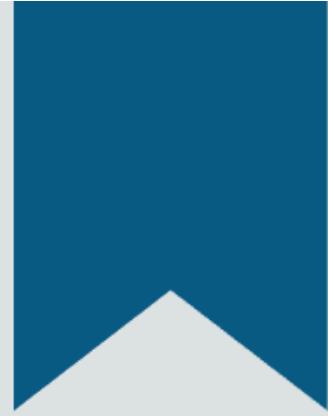


```
[ 0.  5.  10.  15.  20.]
```



# Basic NumPy Operations



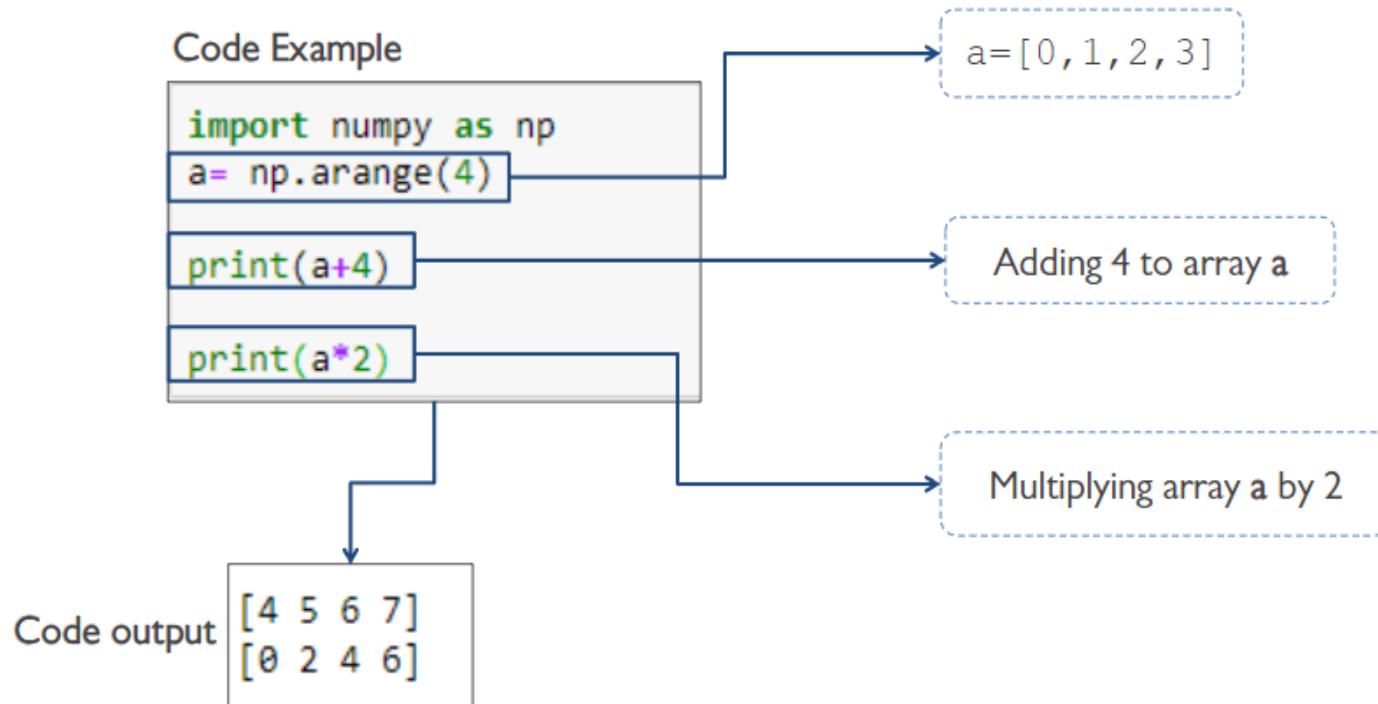


# Arithmetic Operators



# Arithmetic Operators

- Arithmetic operators are used for performing arithmetic operations such as addition, multiplication, and so on
- Let's look at an example:



# Arithmetic Operators (contd.)

---

- Operators can also be used between two arrays
- In NumPy, these operations are element-wise, i.e., Operators are applied only between corresponding elements

Array				
a	0	1	2	3
	+	+	+	+
b	4	5	6	7
	↓	↓	↓	↓
a+b	4	6	8	10



# Arithmetic Operation on Single Dimensional Arrays

Code Example

```
import numpy as np  
a= np.arange(4)  
  
b= np.arange(4,8)  
print(a+b)  
print(a-b)  
print(a*b)
```

a = [0,1,2,3]

b = [4,5,6,7]

Arithmetic operations are performed  
between array a and b

Code output

```
[ 4  6  8 10]  
[-4 -4 -4 -4]  
[ 0  5 12 21]
```

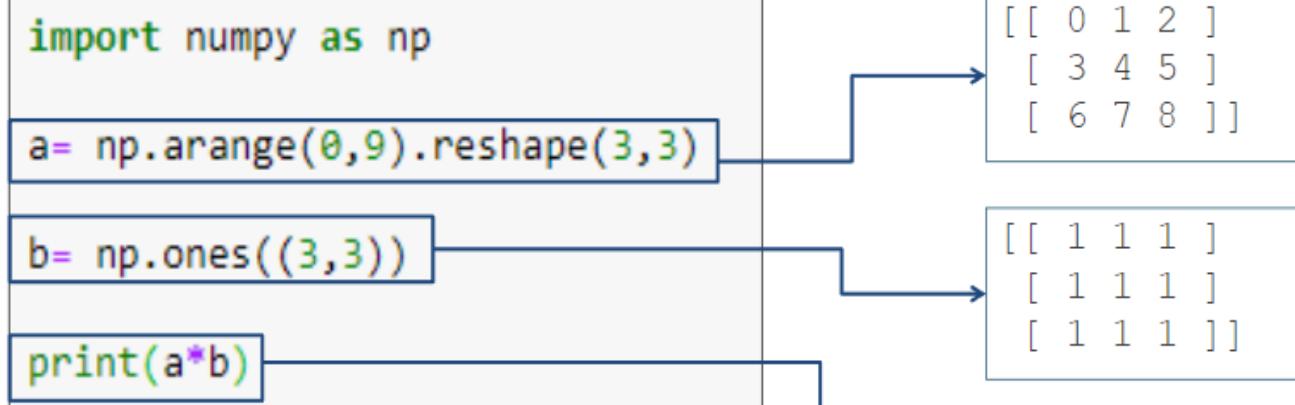


# Arithmetic Operations on Multi-Dimensional Arrays

- Even on the multidimensional array, the arithmetic operators continue to operate element-wise
- Let's have a look at an example:

Code Example

```
import numpy as np  
  
a= np.arange(0,9).reshape(3,3)  
  
b= np.ones((3,3))  
  
print(a*b)
```



The diagram illustrates the execution flow of the provided Python code. It shows the code being executed line by line, with arrows indicating the flow of data from the code to the resulting arrays.

- The first line, `import numpy as np`, is shown without an arrow.
- The second line, `a= np.arange(0,9).reshape(3,3)`, has an arrow pointing to the resulting 3x3 array:  
$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$
- The third line, `b= np.ones((3,3))`, has an arrow pointing to the resulting 3x3 array:  
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$
- The fourth line, `print(a*b)`, has an arrow pointing to the final code output:  
$$\begin{bmatrix} 0. & 1. & 2. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \end{bmatrix}$$

Code output

```
[[0. 1. 2.]  
 [3. 4. 5.]  
 [6. 7. 8.]]
```

Performing multiplication  
of two 2-dimensional  
arrays



# Matrix Product

---

- Matrix product is not an element-wise operation
- Here, we have performed matrix product operation on matrices a and b
- For the highlighted element of the  $a*b$  matrix , we must use the first row of a and first column of b
- In NumPy, this can be performed using dot()

Matrix Depiction

$b$	1	1	1
	1	1	1
	1	1	1

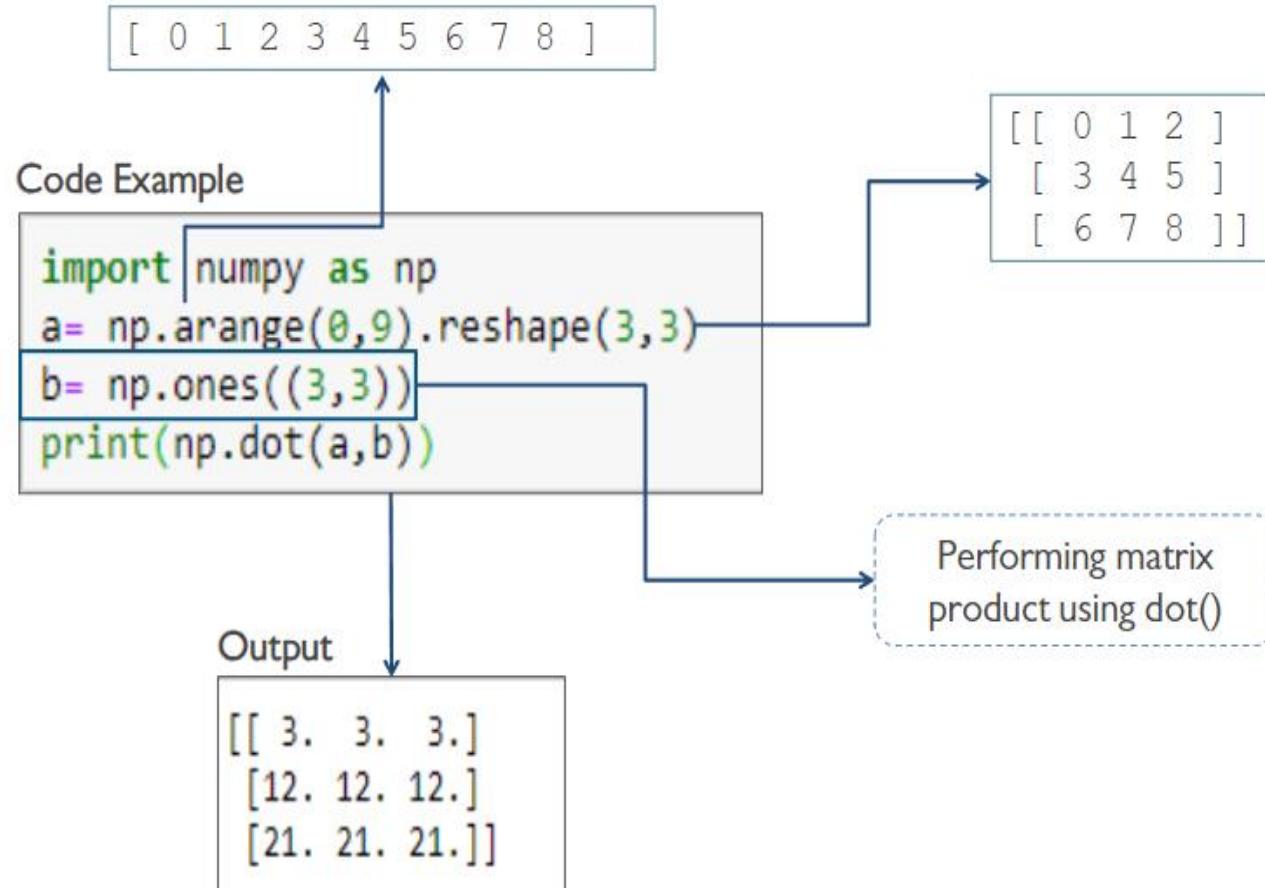
$a$	0	1	2	→	3	3	3
	3	4	5		12	12	12
	6	7	8		21	21	21

$a*b$



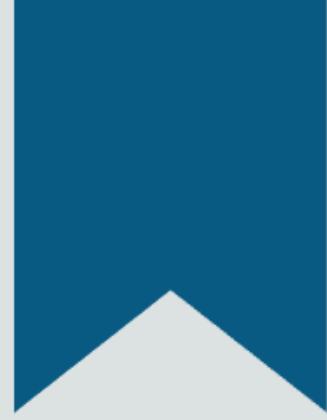
# Matrix Product – Example

---



# NumPy Functions



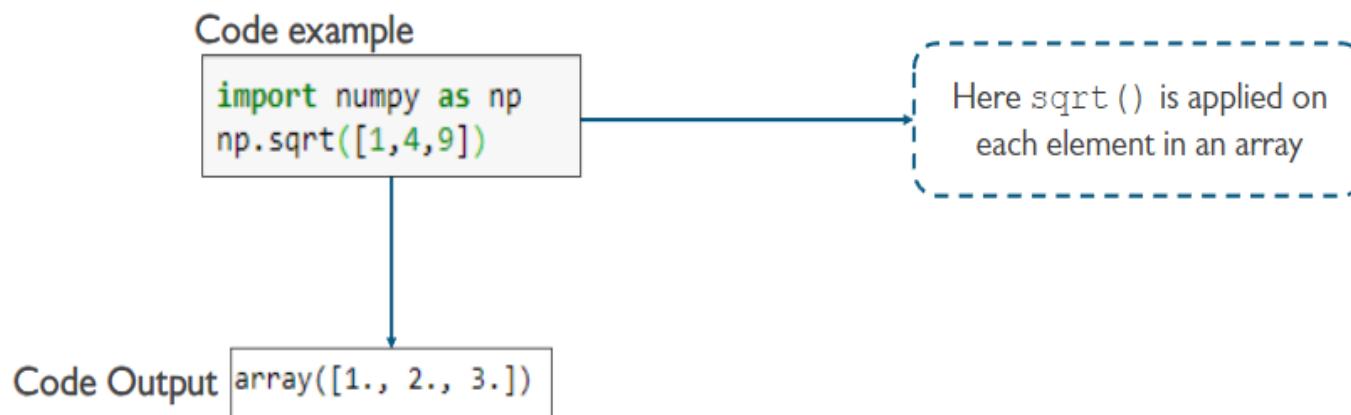


# NumPy Functions: Universal Functions

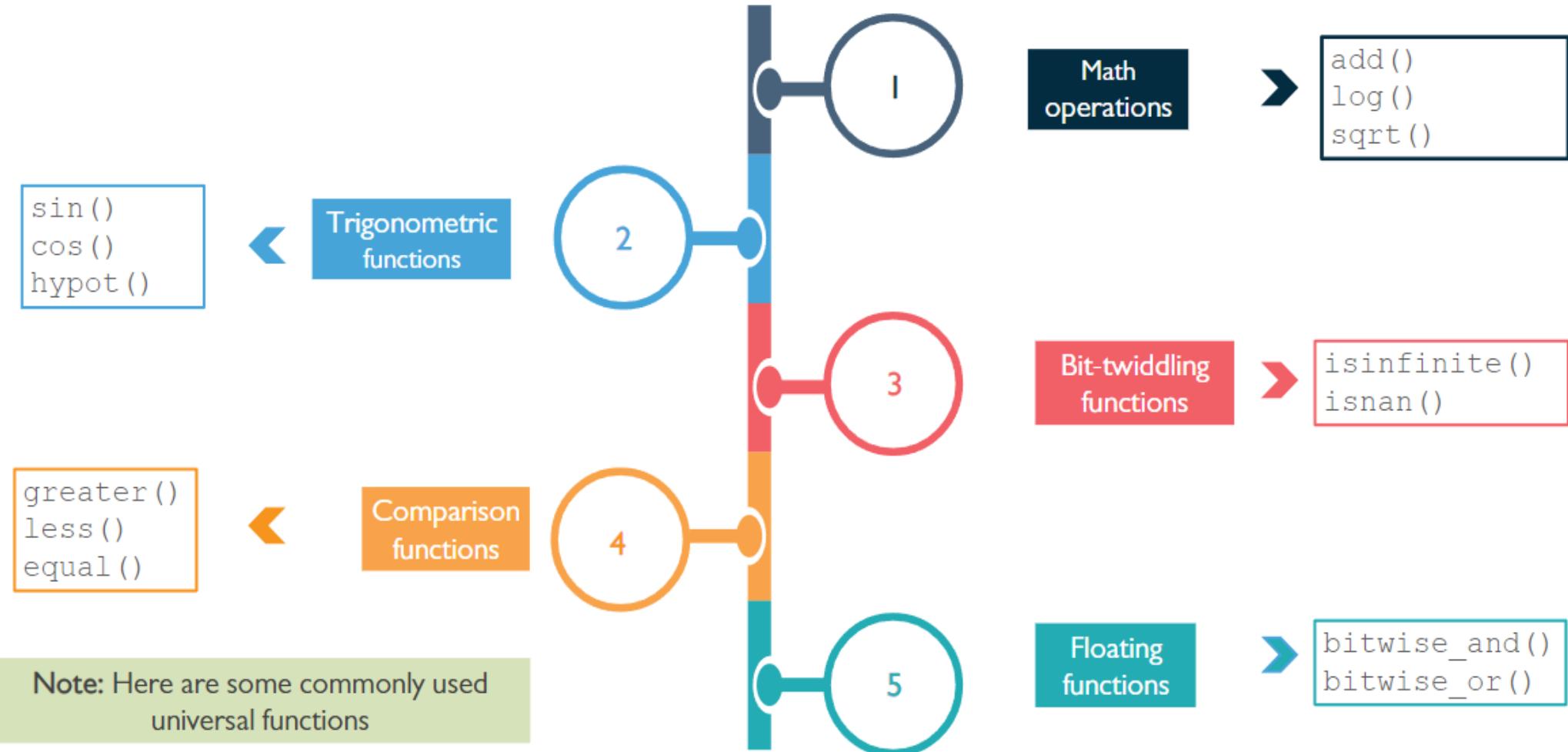


# Universal Functions

- A universal function is a function that operates on ndarrays
- It performs element-wise array operations
- Let's look at an example:



# Types of Universal Functions



# Commonly used Universal Functions

---

Code example  
`mod()`

```
import numpy as np
x = np.mod(np.arange(7), 5)
print(x)
```

Code output  
[0 1 2 3 4 0 1]

Code example  
`log()`

```
import numpy as np
x = np.log([1,np.e,0])
print(x)
```

Code output  
[ 0. 1. -inf]

Code example  
`bitwise_and()`

```
import numpy as np
x = np.bitwise_and(9, 10)
print(x)
```

Code output  
8



# Commonly used Universal Functions (contd.)

`isnan()` tests element-wise for NaN (not a number) and returns result as a boolean array

Code example

```
import numpy as np  
Base=6  
Height=4  
Hypotenuse=np.hypot(Base,Height)  
  
print(Hypotenuse)
```

Code output `7.211102550927978`

Code example

```
import numpy as np  
x = ([np.log(-1),1.,np.log(0)])  
y = np.isnan(x)  
print(y)
```

Code output `[ True False False]`





# NumPy Functions: Aggregate Functions



# Aggregate Functions

---

- Aggregate functions perform an operation on a set of values and produce a single result
- Therefore, the sum of all the elements in an array is an aggregate function
- Consider `a = np.array([1, 2, 3, 4, 5])`, here are some mostly used aggregate functions performed on array a

Function	Description	Syntax	Output
sum	Calculate the sum of the given numbers	<code>a.sum()</code>	15
min	Finds out the minimum number from the given	<code>a.min()</code>	1
max	Finds out the maximum number from the given	<code>a.max()</code>	5
mean	Calculate the mean of the given numbers	<code>a.mean()</code>	3
std	Calculate the standard deviation of the given numbers	<code>a.std()</code>	1.414





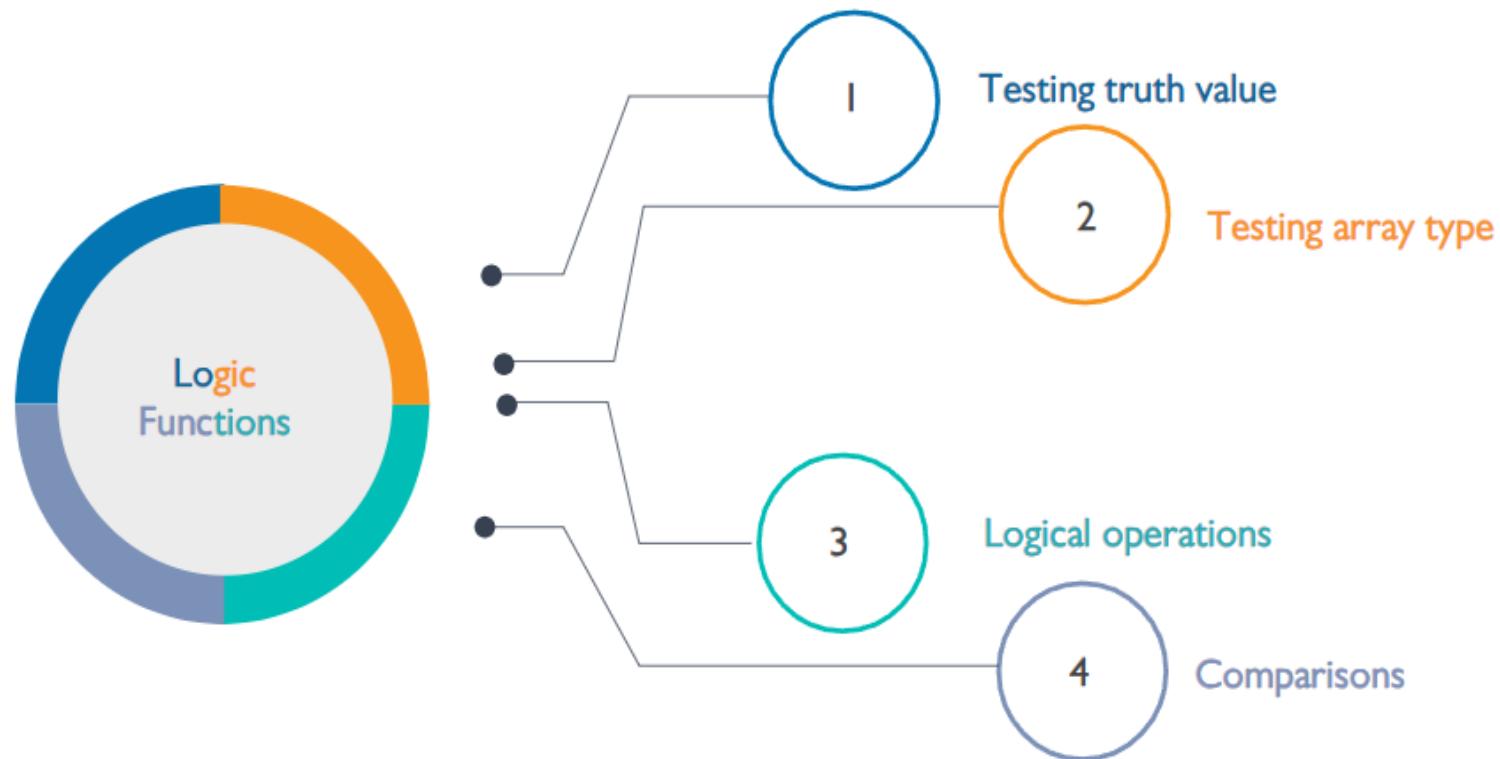
# NumPy Functions: Logic Functions



# Logic Functions

---

Here are some commonly used logic functions:



# Logic Functions – Testing Truth Value – all ()

numpy.all() tests whether all array element along a given axis evaluates to True

Code example  
For boolean values

Code example  
For boolean values, axis = 0

Code example  
For numbers

```
np.all([[True, False], [True, True]])
```

```
np.all([[True, False], [True, True]], axis=0)
```

```
np.all([-1, 4, 5])
```

Code output `False`

Code output `array([ True, False])`

Code output `True`

**Note:** Not a number (NaN), positive infinity and negative infinity evaluate to true because these are not equal to zero



# Logic Functions – Testing Truth Value – any ()

numpy.any () tests whether any array element along a given axis evaluates to True

Code example  
For boolean values

```
np.any([[True, False], [True, True]])
```

Code output

True

Code example  
For boolean values, axis = 0

```
np.any([[True, False], [False, False]], axis=0)
```

Code output

array([ True, False])

Code example  
For numbers

```
np.any([-1, 4, 5])
```

Code output

True

Note: Not a Number (NaN), positive infinity and negative infinity evaluate to true because these are not equal to zero



# Logic Functions – Testing Array Type

---

Syntax	Returns a bool array	Example	Output
<code>np.iscomplex</code>	True if input element is complex	<code>np.iscomplex([2+1j, 3+0j, 3, 4.5, 7+5j])</code>	<code>array([ True, False, False, False, True])</code>
<code>np.isreal</code>	True if input element is real	<code>np.isreal([2+1j, 3+0j, 3, 4.5, 10 ])</code>	<code>array([False, True, True, True, True])</code>



# Logic Functions – Element-Wise Logical Operations

---

Consider  $x = [0, 1, 2, 3, 4]$

Syntax	Example	Output
<code>np.logical_and()</code>	<code>np.logical_and(x&gt;1, x&lt;4)</code>	<code>array([False, False, True, True, False])</code>
<code>np.logical_or()</code>	<code>np.logical_or(x&gt;1, x&lt;4)</code>	<code>array([ True, True, True, True, True])</code>
<code>np.logical_not()</code>	<code>np.logical_not(x&gt;1, x&lt;4)</code>	<code>array([False, False, False, False, True])</code>
<code>np.logical_xor()</code>	<code>np.logical_xor(x&gt;1, x&lt;4)</code>	<code>array([ True, True, False, False, True])</code>



# Logic Functions – Element-Wise Comparisons

---

Syntax	True when	Example	Output
<code>np.greater()</code>	$x1 > x2$	<code>np.greater([3,5,2,7,9],[1,7,4,5,10])</code>	<code>array([ True, False, False, True, False])</code>
<code>np.greater_equal()</code>	$x1 \geq x2$	<code>np.greater_equal([3,5,2,7,9],[1,7,4,5,10])</code>	<code>array([ True, False, True, True, False])</code>
<code>np.less()</code>	$x1 < x2$	<code>np.less([3,5,2,7,9],[1,7,4,5,10])</code>	<code>array([False, True, False, False, True])</code>
<code>np.less_equal()</code>	$x1 \leq x2$	<code>np.less_equal([3,5,2,7,9],[1,7,4,5,10])</code>	<code>array([False, True, True, False, True])</code>
<code>np.equal()</code>	$x1 == x2$	<code>np.equal([3,5,2,7,9],[1,7,4,5,10])</code>	<code>array([False, False, True, False, False])</code>
<code>np.not_equal()</code>	$x1 \neq x2$	<code>np.not_equal([3,5,2,7,9],[1,7,4,5,10])</code>	<code>array([ True, True, False, True, True])</code>



# Indexing and Slicing of NumPy Arrays



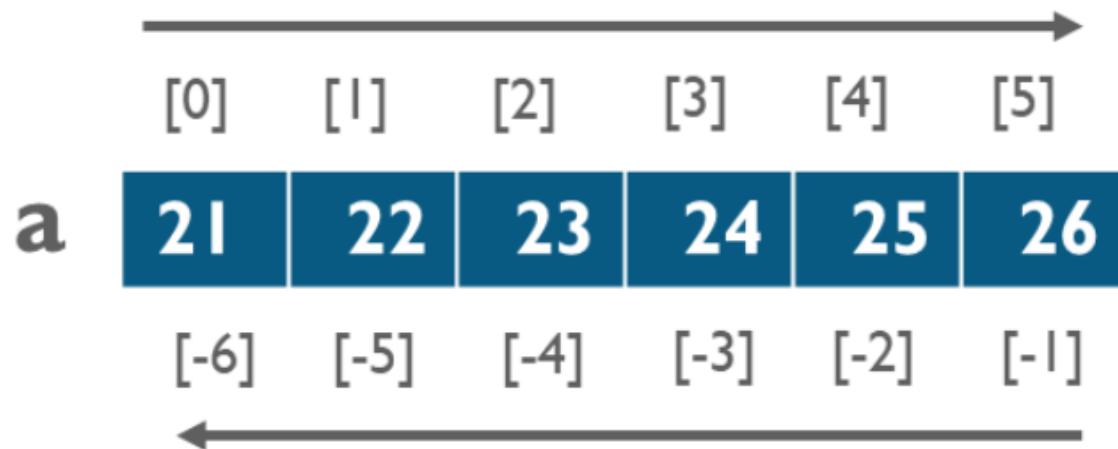


# Indexing of NumPy Arrays



# Indexing

- Array indexing uses square bracket “[ ]” to index the elements of the array so that the elements can then be referred individually for various uses such as extracting a value, selecting items, or even assigning a new value
- When you create a new array, an appropriate scale index is also automatically created



Here is the indexing of a mono-dimensional ndarray



# Indexing – Example

---

Code example  
For positive values

```
import numpy as np
arr=np.arange(21,26)
element=arr[3]
print(element)
```

Code output

24

Code example  
For negative values

```
import numpy as np
arr=np.arange(21,26)
element=arr[-5]
print(element)
```

Code output

21



# Indexing – Two-Dimensional Array

- Indexing in two-dimensional array is represented by a pair of values - the first is the index of the row and the second is the index of the column

A	[,0]	[,1]	[,2]
[0,]	21	22	23
[1,]	24	25	26
[2,]	27	28	29

Here is the indexing of a bidimensional ndarray

Code example

```
import numpy as np  
arr=np.arange(21,30).reshape((3,3))  
element=arr[1,2]  
print(element)
```

Code output 26



# Fancy Indexing

---

- ⌚ Fancy indexing means passing an array of indices to access multiple array elements at once
- ⌚ This allows us to very quickly access and modify complicated subsets of an array's values

Code example

```
import numpy as np
a = np.array([3,78,23,12,90,10,45,56])
ind = np.array([[3,5],[2,4]])
print(a[ind])
```

`[[12 10]  
[23 90]]`

Output

By using fancy indexing, the shape of the result reflects the shape of the index array rather than the shape of the array being indexed





# Slicing of NumPy Arrays



# Slicing

---

- 💡 Slicing allows you to extract portion of an array to generate a new array
- 💡 The slice object is constructed by using start, stop and step parameters in slice() function

Code example

```
import numpy as np  
arr=np.arange(20)  
arr_slice=slice(1,10,2)  
print(arr[arr_slice])
```

```
[0 1 2 3 4 5 6 7 8  
9 10 11 12 13 14 15 16  
17 18 19]
```

Start, stop, step

Code output

```
[1 3 5 7 9]
```



# Slicing (contd.)

---

## Slicing items beginning with a specified index

Code example

```
import numpy as np  
arr=np.arange(20)  
print(arr[2:])
```

Code output

```
[ 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

All elements starting from the index 2

## Slicing items until a specified index

Code example

```
import numpy as np  
arr=np.arange(20)  
print(arr[:15])
```

Code output

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

All elements with index lesser than 15



# Slicing Multi-Dimensional arrays

Extracting specific rows and columns using slicing

Code example

```
import numpy as np  
a=np.array([[1,2,3],[3,4,5],[4,5,6]])  
print(a)  
print(a[0:2,0:2])
```

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

Code output

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

Slice the first two rows  
and the first two columns



# Iterating in a NumPy array

- NumPy package contains an iterator object `numpy.nditer`
- It is an efficient multidimensional iterator object using which it is possible to iterate over an array

Code example

```
import numpy as np
a = np.arange(9)
a = a.reshape(3,3)
print('Original array is:')
print(a)
print('Modified array is:')
for x in np.nditer(a):
    print(x)
```

Code output

```
Original array is:
[[0 1 2]
 [3 4 5]
 [6 7 8]]
Modified array is:
0
1
2
3
4
5
6
7
8
```

Note: Here we can access individual elements of the arrays



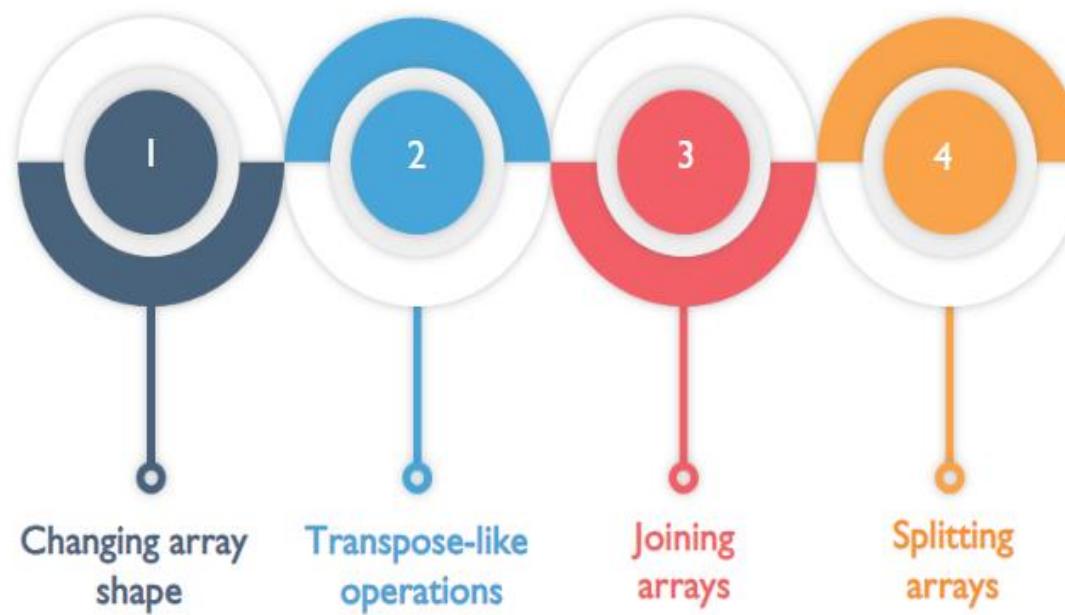
# Array Manipulation in Python - I



# Array Manipulation

---

- Often, we need to shape an array or create an array using already created arrays. We need to perform array manipulation on it
- Here are some array manipulation routines:



# Changing Array Shape – `reshape()`

- Often, we need to shape an array or create an array using already created arrays; we need to perform array manipulation
- Here are some array manipulation routines

`reshape()` gives a new shape to an array without changing its data

Syntax:

```
np.reshape(a, newshape)
```

Code example

```
x = [[0,1,2], [3,4,5]]  
arr = np.array(x)  
print(arr)  
arr1 = np.reshape(arr, (3, 2))  
print(arr1)
```

```
[[0 1 2]  
 [3 4 5]]
```

Code output

```
[[0 1]  
 [2 3]  
 [4 5]]
```



# Changing Array Shape – `ravel()`

`ravel()` function is used to convert a two-dimensional array into a one-dimensional array

Syntax:

```
np.ravel(a)
```

Code Example

```
import numpy as np
a= np.arange(0,9).reshape(3,3)
print('Original Array is:')
print(a)
print('After ravel is:')
b= a.ravel()
print(b)
```

Code output

```
Original Array is:
[[0 1 2]
 [3 4 5]
 [6 7 8]]
After ravel is:
[0 1 2 3 4 5 6 7 8]
```



# Transpose-like Operations – transpose ()

The transpose () function is used to invert columns with the rows

Code Example

```
import numpy as np  
a = np.arange(12).reshape(3,4)  
print('Original Array: ')  
print(a)  
b = np.transpose(a)  
print('After Transpose: ')  
print(b)
```

Code output

```
Original Array:  
[[ 0  1  2  3]  
 [ 4  5  6  7]  
 [ 8  9 10 11]]  
After Transpose:  
[[ 0  4  8]  
 [ 1  5  9]  
 [ 2  6 10]  
 [ 3  7 11]]
```

Note: .T gives the same output as transpose(). transpose () provides more flexibility

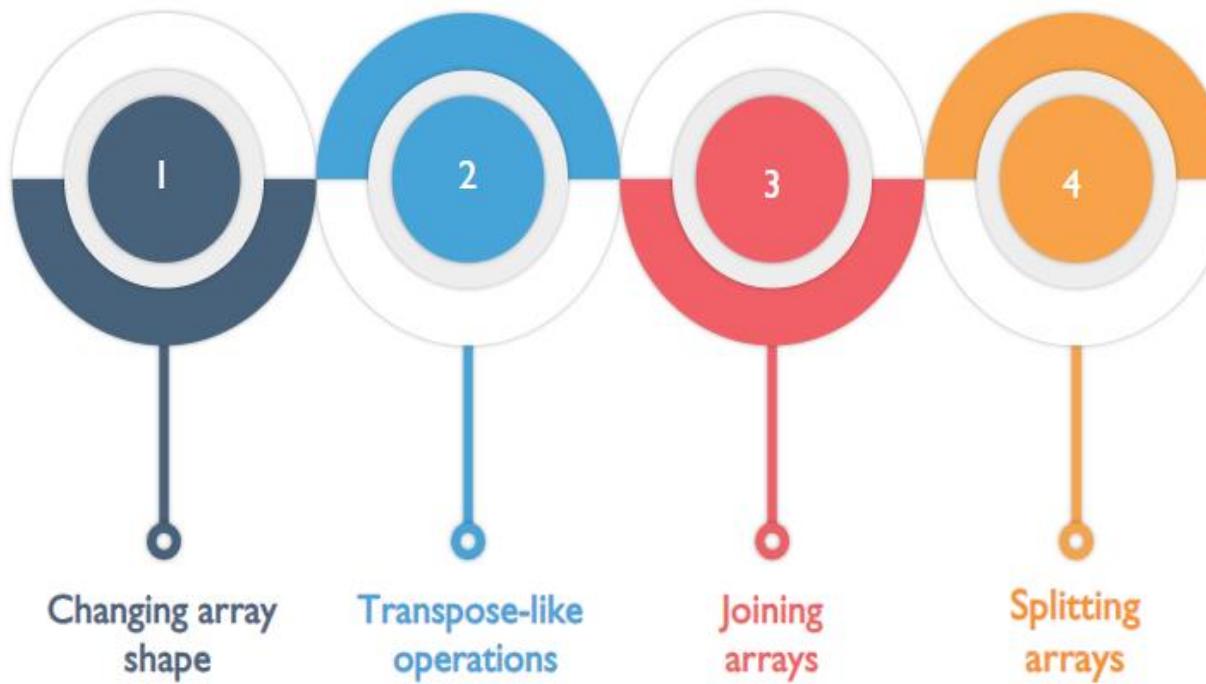


# Array Manipulation in Python - II



# Array Manipulation

---



# Joining Arrays – concatenate ()

- Multiple arrays are merged to form a new one that contains all the arrays
- Here are some functions for joining arrays :
  - concatenate () joins a sequence of arrays along an existing axis

Code Example    For axis = None

```
import numpy as np  
a = np.array([[1, 2], [3, 4]])  
b = np.array([[5, 6]])  
np.concatenate((a, b), axis=None)
```

Code output

```
array([1, 2, 3, 4, 5, 6])
```

Code Example    For axis = 1

```
import numpy as np  
a = np.array([[1, 2], [3, 4]])  
b = np.array([[5, 6]])  
np.concatenate((a, b.T), axis=1)
```

Code output

```
array([[1, 2, 5],  
       [3, 4, 6]])
```

Code Example    For axis = 0

```
import numpy as np  
a = np.array([[1, 2], [3, 4]])  
b = np.array([[5, 6]])  
np.concatenate((a, b), axis=0)
```

Code output

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

Note: The axis parameter specifies the index of the new axis in the dimensions of the results



# Joining Arrays – `stack()` and `column_stack()`

---

`stack()` joins a sequence of arrays along a new axis

`column_stack()` stacks 1-D arrays as columns into

Code Example

```
import numpy as np  
a = np.array([1, 2, 3])  
b = np.array([2, 3, 4])  
np.stack((a, b))
```

Code output

```
array([[1, 2, 3],  
       [2, 3, 4]])
```

Code Example

```
import numpy as np  
a = np.array([0,1,2])  
b = np.array([3,4,5])  
c = np.array([6,7,8])  
print(np.column_stack((a,b,c)))
```

Code output

```
[[0 3 6]  
 [1 4 7]  
 [2 5 8]]
```

Note: Size of the arrays must be same in `stack()`



# Joining Arrays – hstack() and vstack()

---

hstack() adds the second array to the column of the first array

vstack() combines the second array as new row in the first array

Code Example

```
a = np.array((1,2,3))  
b = np.array((2,3,4))  
np.hstack((a,b))
```

Code output

```
array([1, 2, 3, 2, 3, 4])
```

Code Example

```
a = np.array((1,2,3))  
b = np.array((2,3,4))  
np.vstack((a,b))
```

Code output

```
array([[1, 2, 3],  
       [2, 3, 4]])
```



# Splitting Arrays—split(), hsplit() and vsplit()

split() splits an array into multiple sub-arrays

Code Example

```
import numpy as np  
x = np.arange(9)  
np.split(x, 3)
```

Code output

```
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]
```

hsplit() splits the array horizontally

Code Example

```
import numpy as np  
a = np.arange(16).reshape((4,4))  
[b,c]= np.hsplit(a,2)  
print('b=',b)  
print('c=',c)
```

Code output

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

vsplit() splits the array vertically

Code Example

```
import numpy as np  
a = np.arange(16).reshape((4,4))  
[b,c]= np.vsplit(a,2)  
print('b=',b)  
print('c=',c)
```

Code output

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

# File Handling using NumPy



# Loading and saving Data In Binary File (.npy, .npz)

- NumPy provides a pair of functions called `save()` and `load()` that enables to save and retrieve data stored in binary format
- To recover data from binary files `load()` function is used and to save an array to binary file `save()` function is used

Code example

```
numpy.save(file.npy, arr)
```

[0]	[1]	[2]	[3]	[4]
73	98	86	61	96



101100  
010110  
100101

Code example

```
numpy.load(file.npy)
```

101100  
010110  
100101



[0]	[1]	[2]	[3]	[4]
73	98	86	61	96



# Loading and saving Data in Text Files (.txt, .csv)

- NumPy provides a set of functions called `savetxt()` and `genfromtxt()` that helps us to save and retrieve data in text format
- To save an array to text file `savetxt()` is used and to load data from text file `genfromtxt()` is used `loadtxt()` is also used to load data from a text file (it does not handle missing values)

Code example

```
numpy.savetxt(file.txt, arr, delimiter  
= ',')
```

[0]	[1]	[2]	[3]	[4]
73	98	86	61	96



Code example

```
numpy.genfromtxt(file.txt, arr, delimiter  
= ',')
```



[0]	[1]	[2]	[3]	[4]
73	98	86	61	96

Note: Same functions are used for .csv files



## **Q1. Problem Statement: Understanding File Operations**

Write a Python program that reads text from a *poem.txt* file (Provided on the LMS) in read-only mode and prints it line-by-line.

### **Input Format:**

You do not need to read any input in this problem.

### **Sample Output:**

```
Jack and Jill went up the hill  
To fetch a pail of water.  
Jack fell down and broke his crown,  
And Jill came tumbling after.  
Then up got Jack and said to Jill,  
As in his arms he took her,  
"Brush off that dirt for you're not hurt,  
Let's fetch that pail of water."  
So Jack and Jill went up the hill  
To fetch the pail of water,  
And took it home to Mother dear,  
Who thanked her son and daughter.
```



## **Q2. Problem Statement: Modifying Text File using Python**

Write a Python program that opens the *poem.txt* file (Provided on the LMS).

Replace all the words in the poem, starting from “J” to “K,” and print the new version of the poem.

### **Input Format:**

You do not need to read any input in this problem.

### **Sample Output:**

```
Kack and Kill went up the hill
To fetch a pail of water.
Kack fell down and broke his crown,
And Kill came tumbling after.
Then up got Kack and said to Kill,
As in his arms he took her,
“Brush off that dirt for you’re not hurt,
Let’s fetch that pail of water.”
So Kack and Kill went up the hill
To fetch the pail of water,
And took it home to Mother dear,
Who thanked her son and daughter.
```



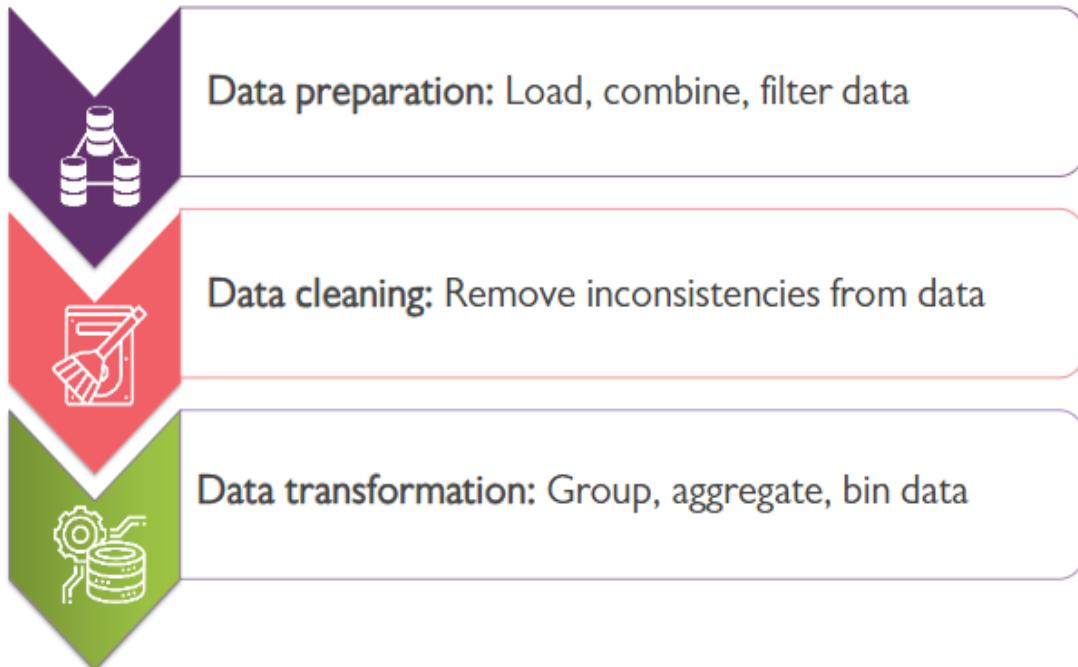
# Introduction to Pandas Library in Python



# Data Manipulation – Brushing up

Data manipulation is the process of performing various operations on data in order to make it ready for further analysis

Data manipulation involves:



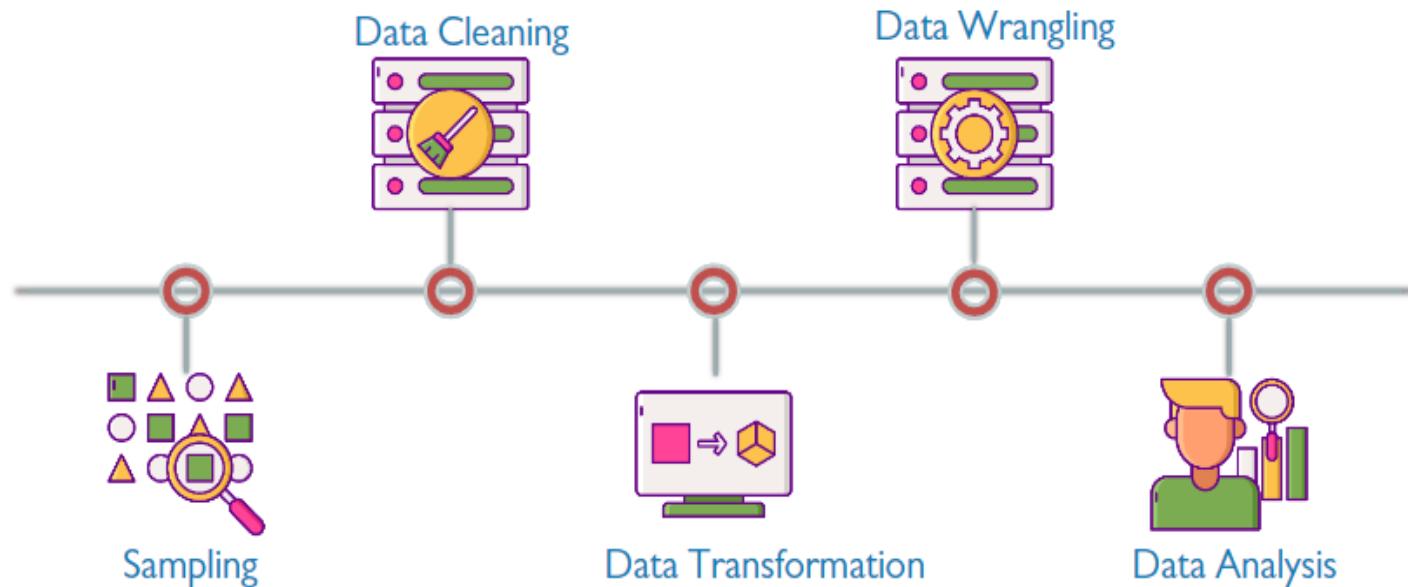
# What is Pandas?

Pandas is an open-source library that provides high-performance data structures to perform efficient data manipulation and analysis in Python



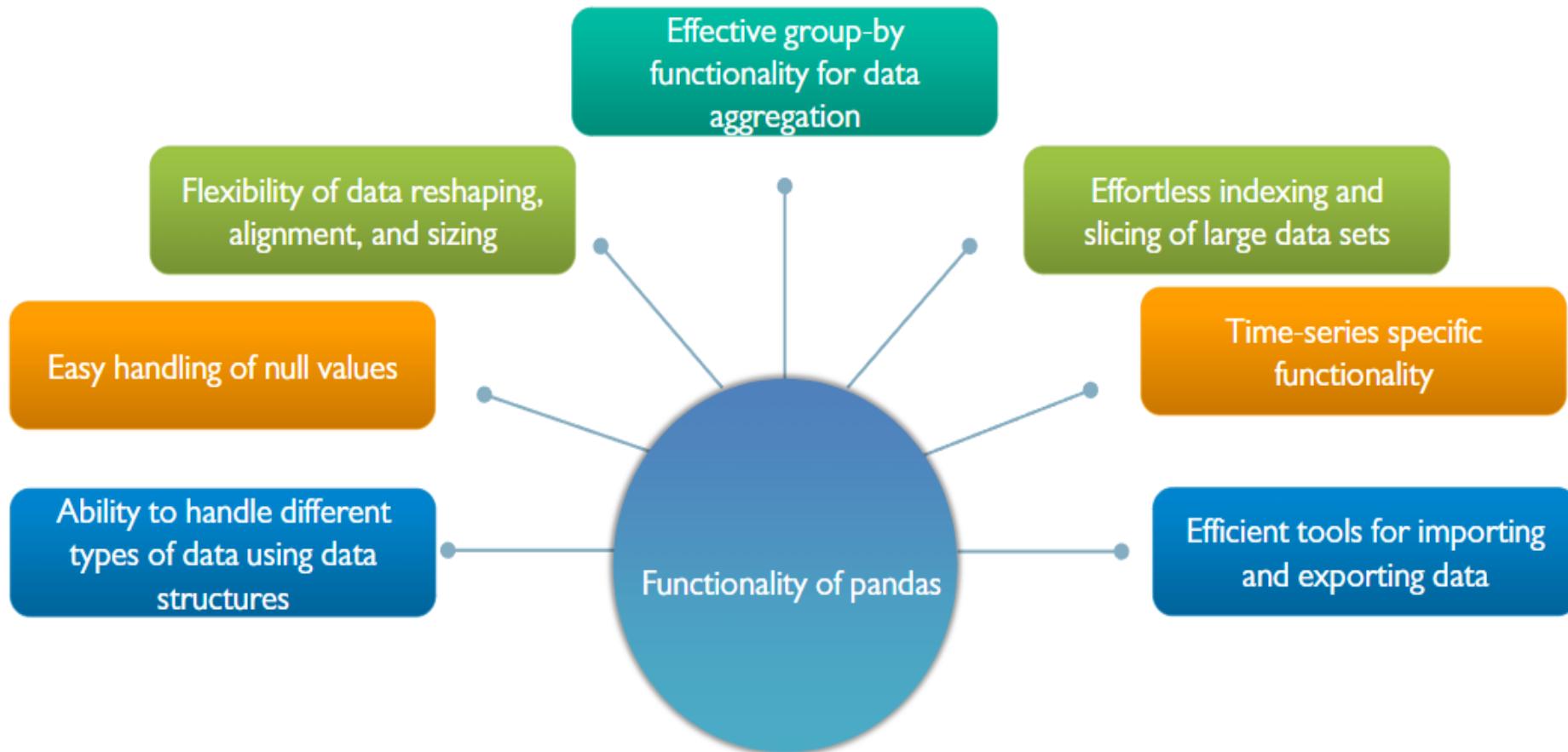
Pandas term is derived from panel (pan) data (da)

Examples of data manipulation tasks performed using pandas



# Functionality of Pandas

---

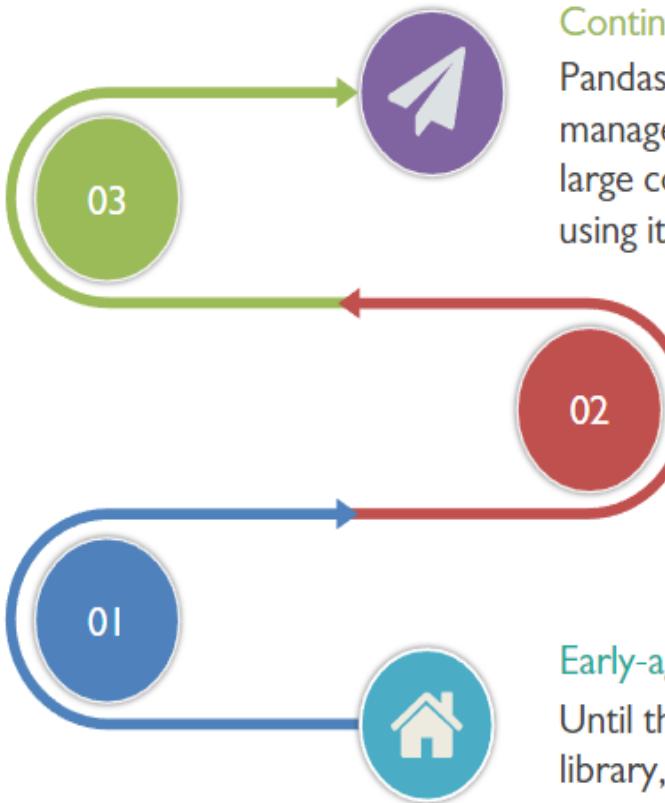


# Significance of Pandas in Python

---

**Pandas' significance in Python**  
Pandas is a powerful data analysis library which makes Python comparable to other tools, such as, R which are specifically designed for statistical programming

**Use of early-age Python**  
Python was good for data preparation but less so for data analysis



**Continuously developed**  
Pandas is being continuously managed and developed by a large community of programmers using it

**Pandas functionality**  
Pandas fills this gap between data preparation and analysis enabling the user to perform the entire operation in one single platform

**Early-age Python**  
Until the development of pandas library, Python was primarily a general purpose programming language



# Installing Pandas

Pandas can be installed on Python Shell using pip installer using the command: >`pip install pandas`

Run the command to  
install pandas in  
command prompt

1

```
c:\ Command Prompt  
Microsoft Windows [Version 10.0.18362.239]  
(c) 2019 Microsoft Corporation. All rights reserved.  
C:\Users\██████████>pip install pandas
```

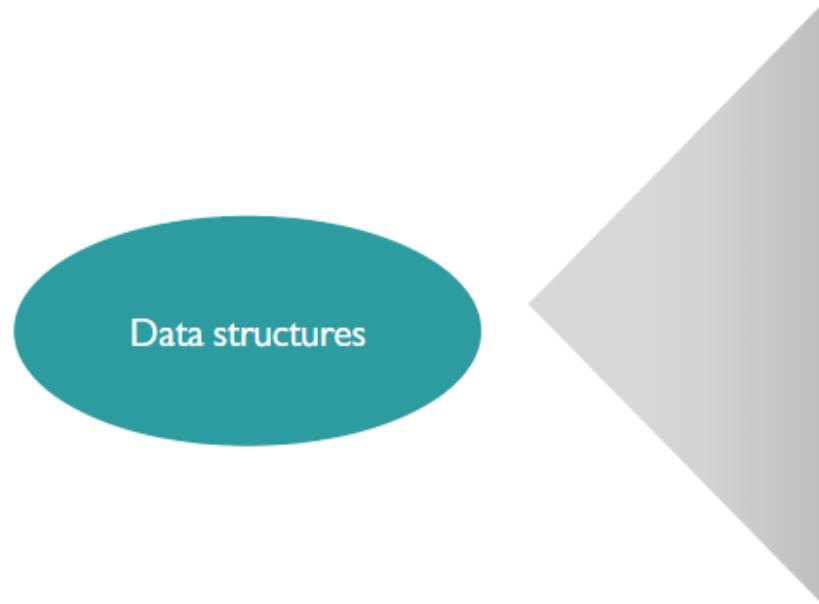
```
Downloading https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721c0a  
c0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-none-any.w  
h1  
Installing collected packages: numpy, six, python-  
dateutil, pytz, pandas  
Successfully installed numpy-1.17.0 pandas-0.25.0  
python-dateutil-2.8.0 pytz-2019.2 six-1.12.0
```

2  
Ensure successful  
installation of the  
included packages

# Pandas Data Structure



# Data Structures in Pandas



## Series

⌚ 1-D labeled array

⌚ Example: Array of temperatures on different days in a week:  
[30, 32, 34.5, 32.1, 30.2, 29.8, 30.5]

## DataFrame

⌚ 2-D labeled structure of rows & columns

⌚ Example: Tabular structure containing columns such as month, week, average temperature, and so on

Sample series & DataFrame

Month	Week	Temp_Avg ( °C )
Jan	1	27.5
Jan	2	30.1
Jan	3	29.7
Jan	4	29.4

DataFrame

Series

# Pandas Series

Series is a one-dimensional labelled array which can hold data of different types such as integer, string, float, Python objects, and so on

Code example

```
s = pandas.Series(data, index=index)
```

Series function of  
pandas module

Array, dictionary, scalar value, etc.

Optional axis  
labels

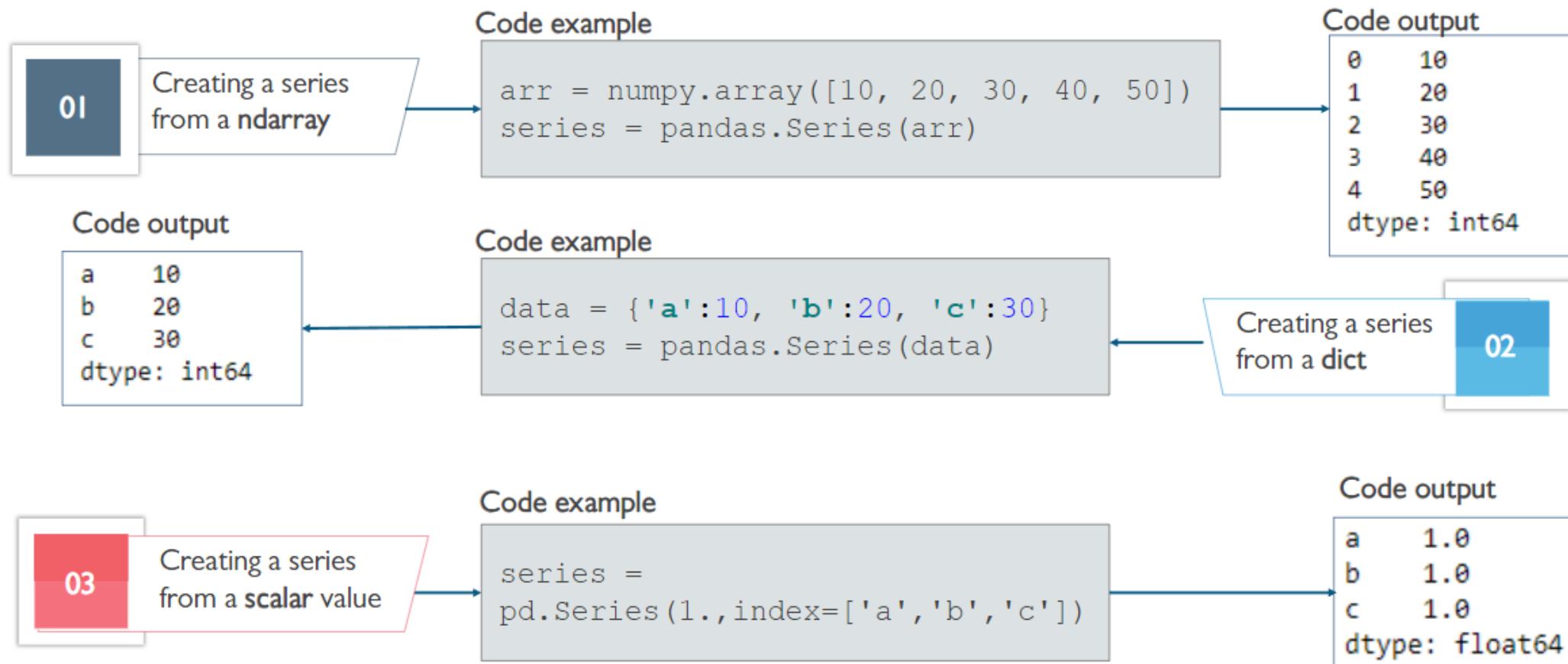
Code output

```
0    30.0
1    32.0
2    34.5
3    32.1
4    30.2
5    29.8
6    30.5
dtype: float64
```

Note: For creating the series displayed in the sample, data is a simple array of float values



# Creating a Series



# Pandas DataFrame

DataFrame is a two-dimensional labelled data structure with columns of different types

Code example

```
df = pandas.DataFrame(data, index=index)
```

DataFrame function  
of pandas module

1-D array, series, 2-D  
NumPy array, DataFrame  
and so on

Optional  
row labels

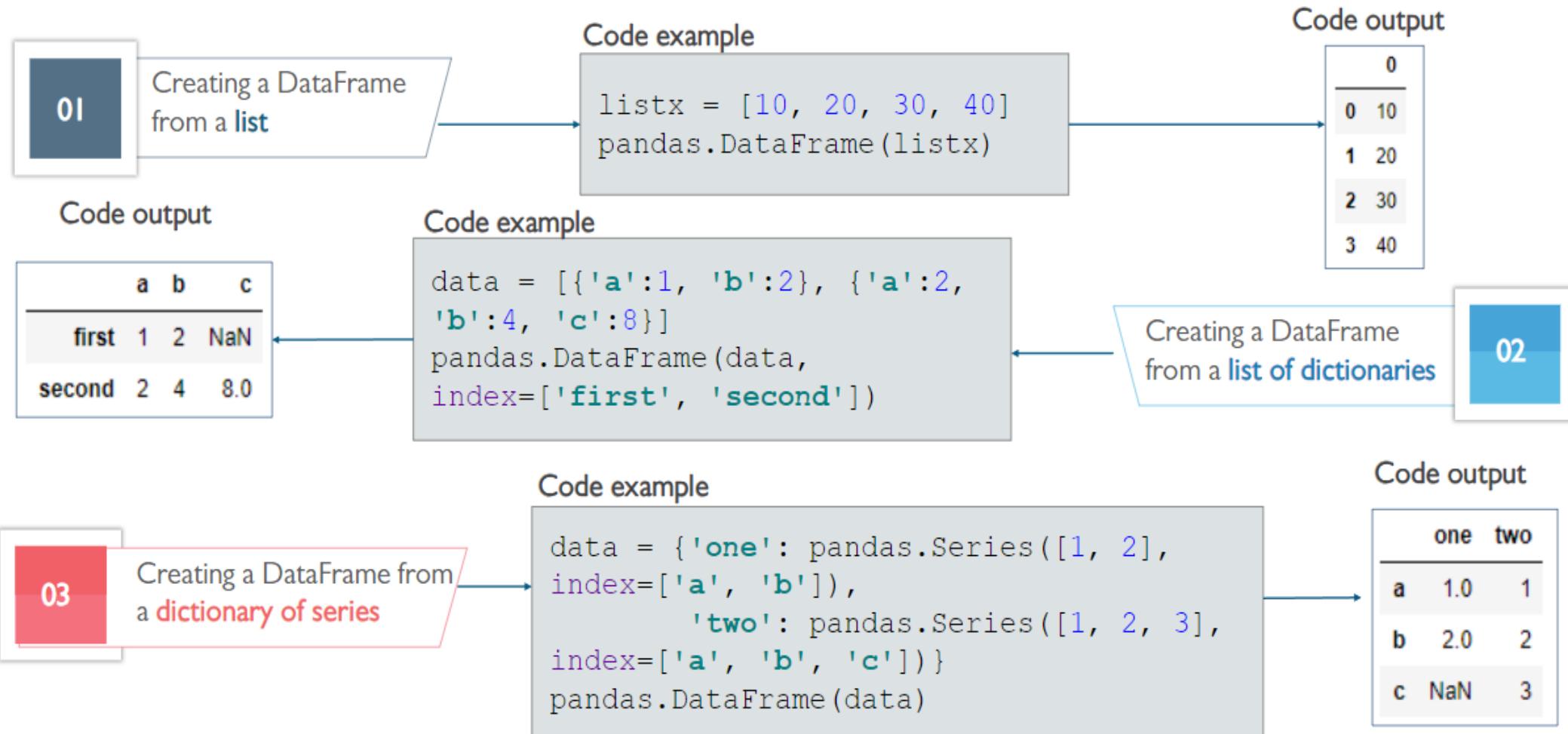
Sample DataFrame

	Week1	Week2	Week3	Week4
0	30.0	32.0	29.0	30.3
1	32.0	31.5	31.7	32.1
2	34.5	30.5	32.7	34.3
3	32.1	32.8	34.1	31.1
4	30.2	30.9	31.2	31.2
5	29.8	28.8	29.6	30.8
6	30.5	31.5	31.3	31.2

Note: For creating the DataFrame displayed in the sample, data consists of 4 series, each containing temperature values for 7 days of a week



# Creating a DataFrame



## **Q1. Problem Statement: Creating Pandas Series**

Write a Python program to create two lists - one for countries and another for country codes. Create a pandas series from these two lists with the codes as index. Print the length, shape and size of the series too.

### **Input Format:**

You do not need to read any input in this problem.

### **Sample Output:**

```
The series is:  
IND          India  
CAN          Canada  
AUS          Australia  
JAP          Japan  
GER          Germany  
FRA          France  
dtype: object  
Length of the series is:  6  
Shape of the series is:  (6,)  
Size of the series is:  6
```

## Q2. Problem Statement: Creating Pandas DataFrame

Write a Python program that reads Name, Age, and Roll no. as input from the user and stores it into a DataFrame, and print the DataFrame you have created.

### Input Format:

```
Enter the number of Student records you want to store: 2
Enter the Name of the student: Shubham
Enter the Age of the student in years: 9
Enter the Roll no. of the student 56
Enter the Name of the student: Harshit
Enter the Age of the student in years: 8
Enter the Roll no. of the student 34
```

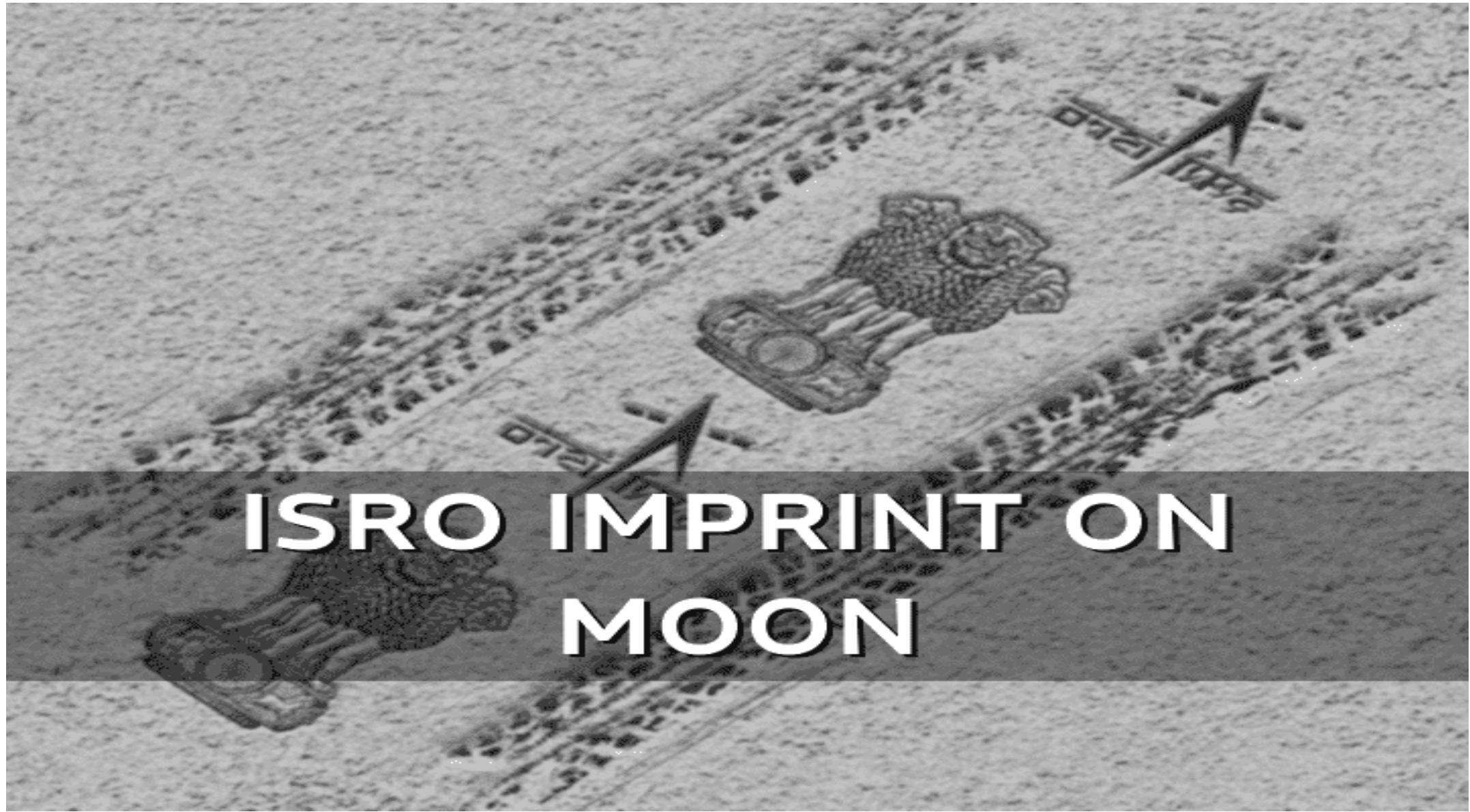
### Sample Output:

```
The DataFrame is-
  Name  Age  Roll No.
0  Shubham    9        56
1  Harshit    8        34
```



# Importing and Exporting Data using Pandas





**ISRO IMPRINT ON  
MOON**



# Reading Data from CSV

- Data from csv files can be read using `read_csv( )` as follows:
- For example - if the input file `purchase_data.csv` contains headers, the data can be imported as :

Code example

```
import pandas as pd  
DF = pd.read_csv("purchase_data.csv")  
print(DF)
```

Code output

	Item	UnitPrice	Quantity	TotalCost
0	Shirts	100	4	400
1	Jeans	300	2	600
2	Shoes	250	1	250
3	Tops	150	3	450

# Reading Data from CSV (contd.)

- ☞ If the input file does not contain headers, the previous function will be inappropriate.

Code example

```
DF = pd.read_csv("data_without_headers.csv")  
print(DF)
```

Code output

	Shirts	100	4	400
0	Jeans	300	2	600
1	Shoes	250	1	250
2	Tops	150	3	450

First row is incorrectly interpreted as Header Row

- ☞ Use headers option as shown below:

Code example

```
DF = pd.read_csv("data_without_headers.csv", header=None)  
print(DF)
```

Code output

	0	1	2	3
0	Shirts	100	4	400
1	Jeans	300	2	600
2	Shoes	250	1	250
3	Tops	150	3	450



# Reading Data from Excel and JSON

- Data from an excel file can be read using `read_excel()` as follows:
- Assume that the input file `quarterly_sales.xlsx` contains sales entries with headers for quarters, then :

Code example

```
DF = pd.read_excel('quarterly_sales.xlsx')  
print(DF)
```

Code output

	Q1	Q2	Q3	Q4
0	120	80	150	100
1	200	180	150	120
2	175	175	200	130

- If the same data is available in JSON format as a result of an output from another application in the organization, use function `read_json()` as follows:

Code example

```
DF = pd.read_json('quarterly_sales.json')  
print(DF)
```

Code output

	Q1	Q2	Q3	Q4
Entry1	120	80	150	100
Entry2	200	180	150	120
Entry3	175	175	200	130





# Exporting Data



# Writing DataFrame Data to CSV and Excel

Consider a pandas dataframe containing temperature data in a month

	Week1	Week2	Week3	Week4
0	30.0	32.0	29.0	30.3
1	32.0	31.5	31.7	32.1
2	34.5	30.5	32.7	34.3
3	32.1	32.8	34.1	31.1
4	30.2	30.9	31.2	31.2
5	29.8	28.8	29.6	30.8
6	30.5	31.5	31.3	31.2

Code example

```
month.to_csv("Temp1.csv")
```

Code output

	A	B	C	D	E
1		Week1	Week2	Week3	Week4
2	0	30	32	29	30.3
3	1	32	31.5	31.7	32.1
4	2	34.5	30.5	32.7	34.3
5	3	32.1	32.8	34.1	31.1
6	4	30.2	30.9	31.2	31.2
7	5	29.8	28.8	29.6	30.8
8	6	30.5	31.5	31.3	31.2

Code example

```
month.to_excel("Temp2.xlsx")
```

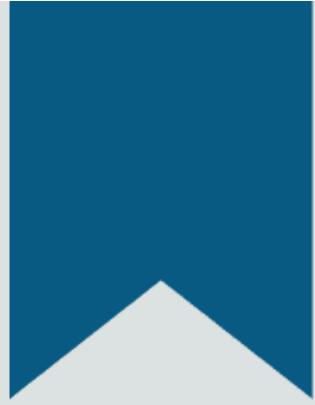
Code output

	A	B	C	D	E
1		Week1	Week2	Week3	Week4
2	0	30	32	29	30.3
3	1	32	31.5	31.7	32.1
4	2	34.5	30.5	32.7	34.3
5	3	32.1	32.8	34.1	31.1
6	4	30.2	30.9	31.2	31.2
7	5	29.8	28.8	29.6	30.8
8	6	30.5	31.5	31.3	31.2



# Functionality of Pandas Series

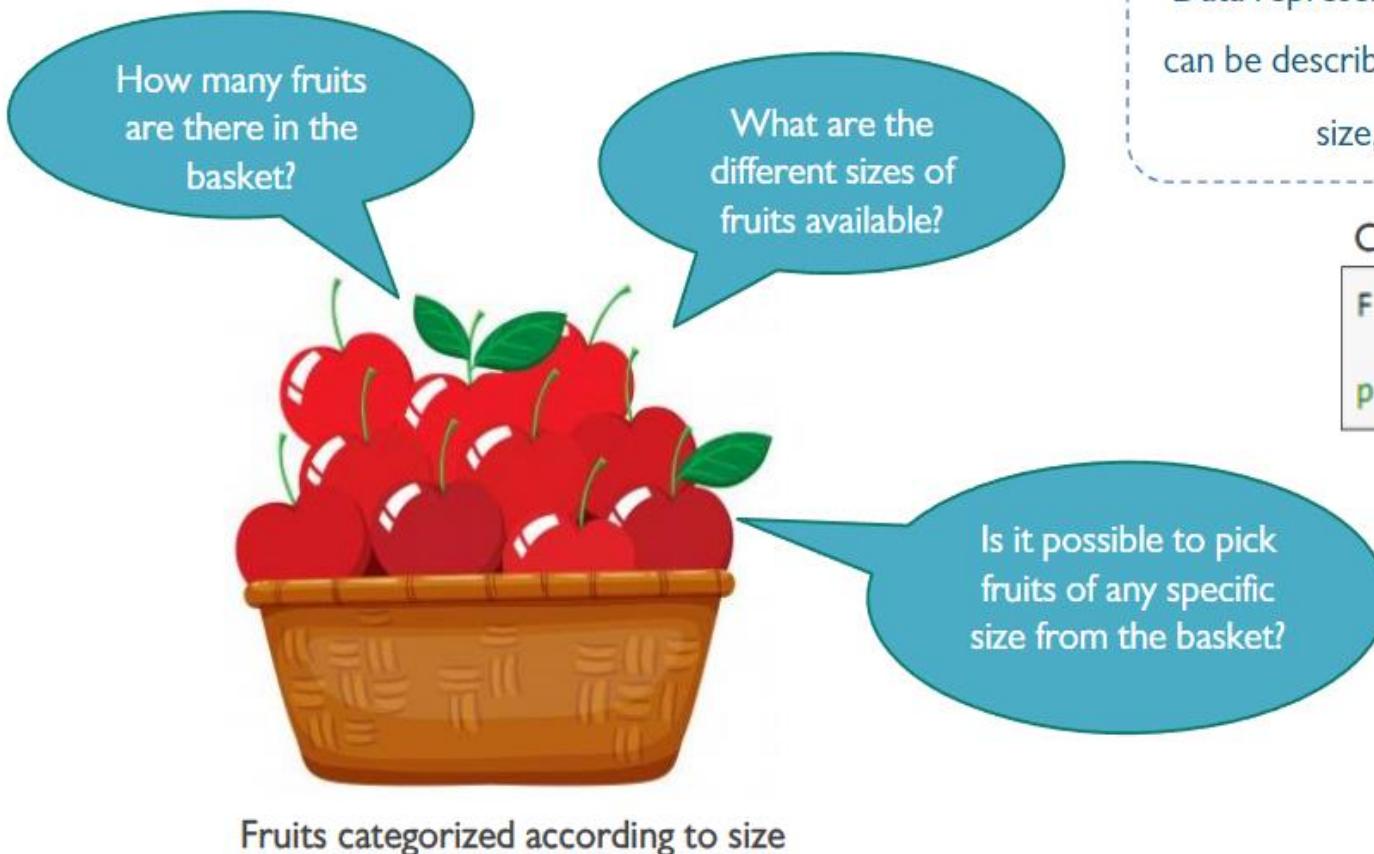




# Essential Functionality of Series and DataFrame



# Representing Features of Data



Data represented using a pandas series and DataFrame can be described in terms of various features such as its size, shape, index, values, and so on

Code example

```
Fruits=pd.Series([18,23,15],  
                 index=['small','medium','large'])  
print(Fruits)
```

small	18
medium	23
large	15
dtype:	int64

Code Output

Sample data showing count of fruits under three size groups

# Functionality of Series

## Code example

```
Fruits.values  
Fruits.index  
Fruits.size  
Fruits.shape
```

## Code Output

```
[18 23 15]  
Index(['small', 'medium', 'large'], dtype='object')  
3  
(3,)
```

### .values

Provides a list of values in the series

### .index

Index values can be retrieved using this property

### .size

Provides number of items in the series

### .shape

Provides shape as a two value tuple where the first value represents the size

Note: Many more features are available, most of which are discussed with respect to DataFrames, in subsequent slides.



## Q1. Problem Statement: Working with DataFrames - I

Write a Python program that reads the “*iris.csv*” dataset (Available on LMS) into a DataFrame, filters the rows on the condition that “*sepal length*” should be more than or equal to 6cm. Write the filtered rows in a new DataFrame, and save the new DataFrame as a CSV file.

### Dataset:

Original DataFrame is:						
						Species
	ID	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

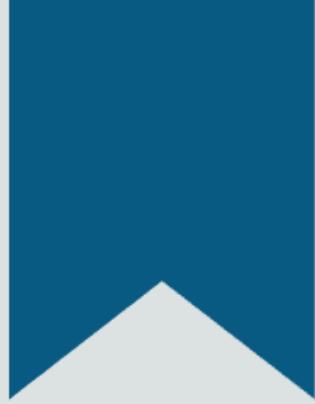
### Output:

Flowers with morethan 6 cm sepal length are:						
						Species
	ID	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
51	52	6.4	3.2	4.5	1.5	Iris-versicolor
52	53	6.9	3.1	4.9	1.5	Iris-versicolor
54	55	6.5	2.8	4.6	1.5	Iris-versicolor
56	57	6.3	3.3	4.7	1.6	Iris-versicolor
...	...	...	...	...	...	...
144	145	6.7	3.3	5.7	2.5	Iris-virginica
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica

67 rows × 6 columns

# Functionality of Pandas DataFrames - I



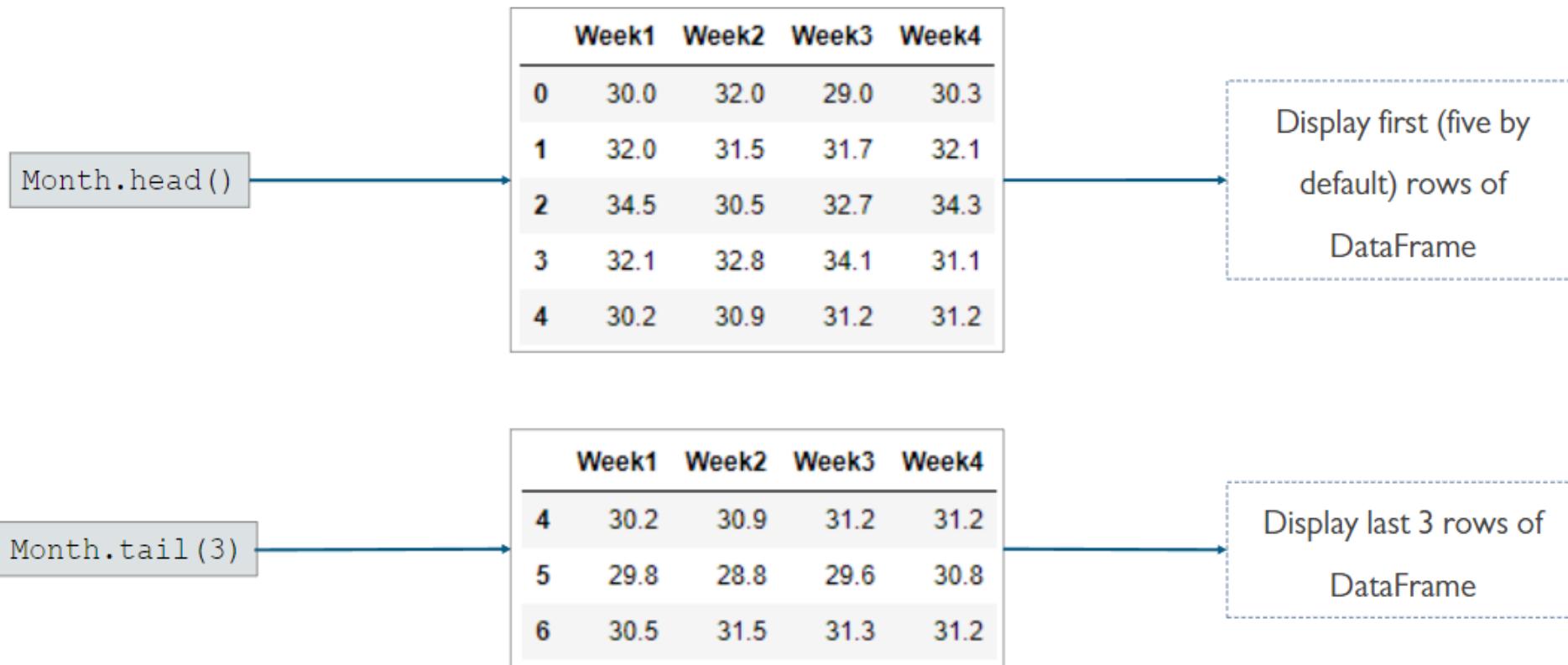


# Understanding Functionality of Pandas DataFrames



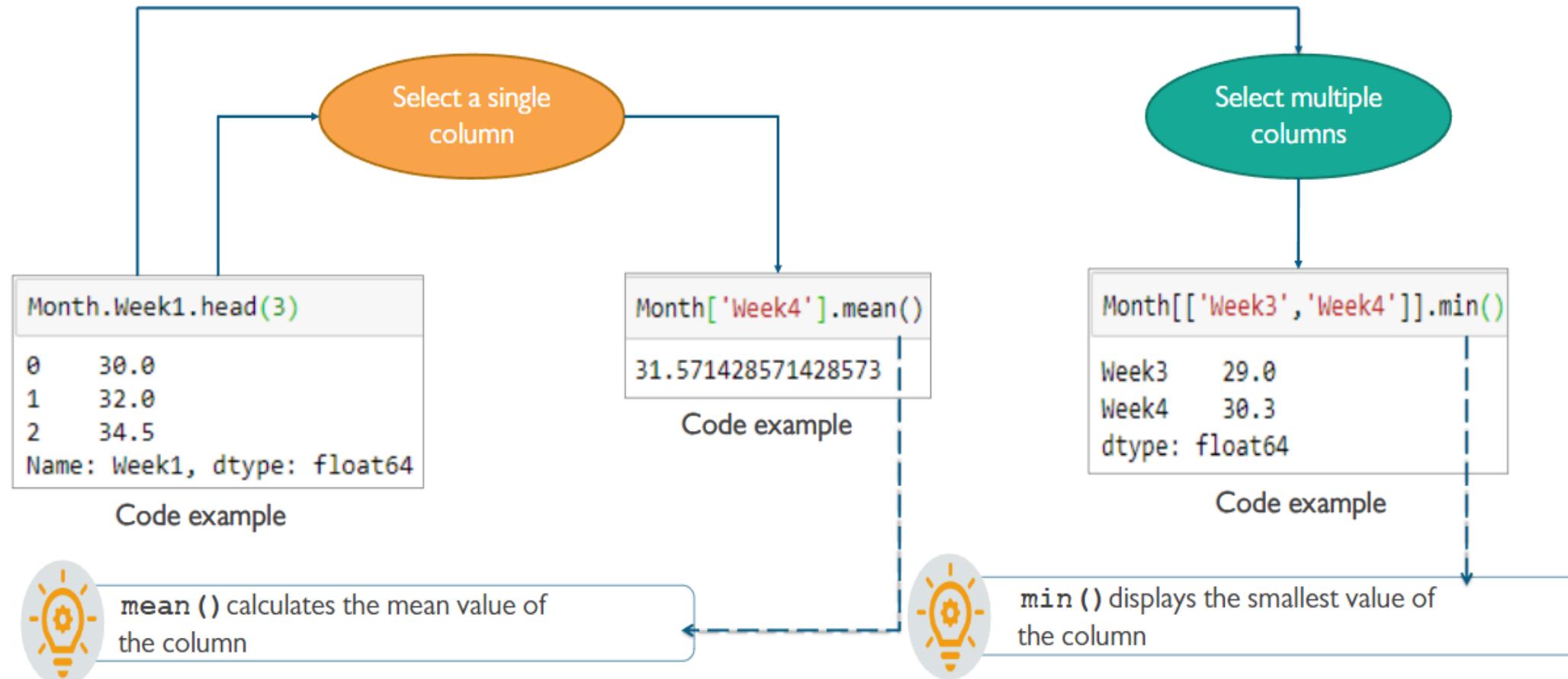
# Introduction to Functionality of Pandas DataFrame

Fetching the first or last few rows of a DataFrame for inspection can be done using the below functions:



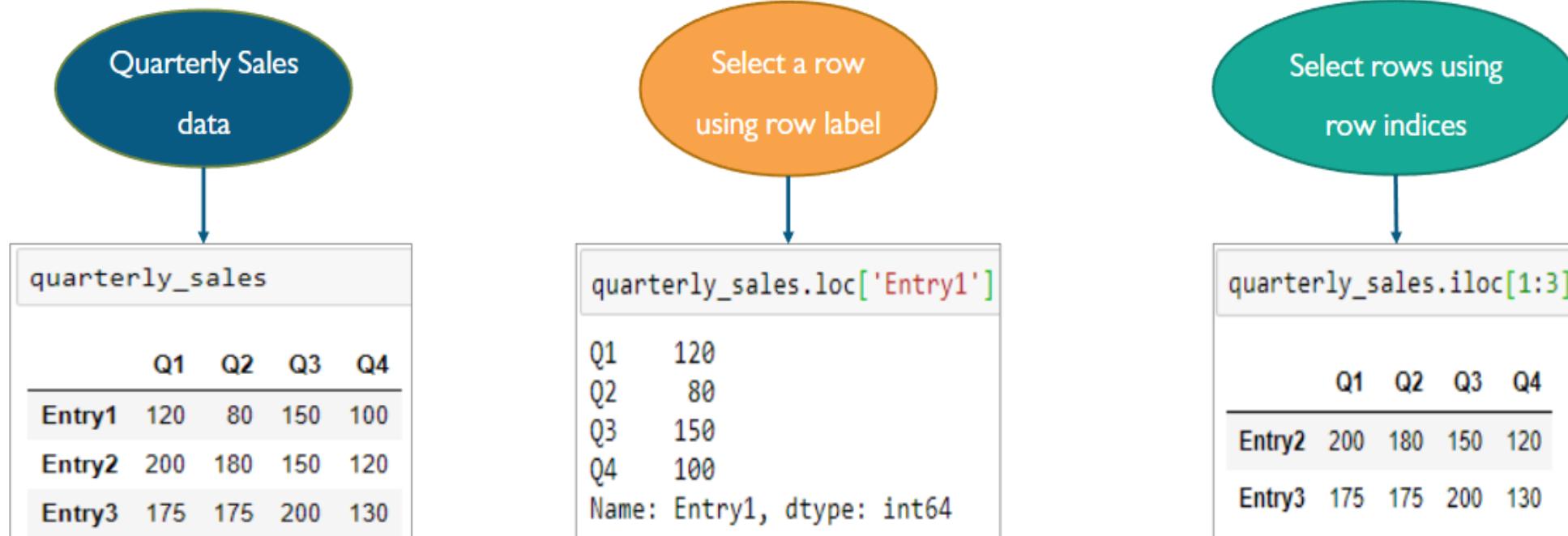
# Selecting Columns of a DataFrame

Selecting specific columns of a DataFrame to perform operations on them can be achieved as shown below:



# Selecting Rows of a DataFrame

Rows can be selected using a row label or row index as shown below:



# Adding Columns

Pandas allow to add and remove columns in existing DataFrame, which can be helpful for purposes such as summarizing data, making data easy to read, reducing processing time, and so on

Example: Display sum of multiple rows in a separate column

quarterly_sales				
	Q1	Q2	Q3	Q4
Entry1	120	80	150	100
Entry2	200	180	150	120
Entry3	175	175	200	130



Code example

```
quarterly_sales['Annual Sale'] =  
    quarterly_sales['Q1'] +  
    quarterly_sales['Q2'] +  
    quarterly_sales['Q3'] +  
    quarterly_sales['Q4']  
quarterly_sales
```

	Q1	Q2	Q3	Q4	Annual Sale
Entry1	120	80	150	100	450
Entry2	200	180	150	120	650
Entry3	175	175	200	130	680



# Removing Columns

The ability to remove columns from a DataFrame in pandas removes unnecessary columns that are not required for data analysis thus, it helps in data cleaning.

Example: If the 'Annual Sale' data is no longer required, it can be removed as shown below:

Code example

```
quarterly_sales.drop(['Annual Sale'], axis='columns')
```

Code example

	Q1	Q2	Q3	Q4
Entry1	120	80	150	100
Entry2	200	180	150	120
Entry3	175	175	200	130

Note: drop () function does not remove Columns/Rows from the original DataFrame: thus, it must be explicitly saved into another DataFrame



# Removing Rows

Removing a specific row or a set of rows from a DataFrame allows filtering out unnecessary data; this results in reduced processing time and more accurate results.

Removing a row  
using index label

```
quarterly_sales.drop(['Entry2'])
```

	Q1	Q2	Q3	Q4	Annual Sale
Entry1	120	80	150	100	450
Entry3	175	175	200	130	680

Code example

Removing multiple rows  
using index range

```
quarterly_sales.drop(quarterly_sales.index[1:3])
```

	Q1	Q2	Q3	Q4	Annual Sale
Entry1	120	80	150	100	450
Entry3	175	175	200	130	680

Code example



# Functionality of Pandas DataFrames - II



# Updating Cells



# Updating a Cell's value

Updating values from a DataFrame provides flexibility; it is helpful to replace erroneous values or update data based on new data received.

Example: `set_value()`: Sets value of cell specified by index and column

Code example

```
quarterly_sales.set_value('Entry1', 'Q1', 0)
```

`set_value()`  
helps to update a  
cell with a scalar  
value

	Q1	Q2	Q3	Q4
Entry1	0	80	150	100
Entry2	200	180	150	120
Entry3	175	175	200	130



# Updating a Cell's value (contd.)

- Another approach to update data in a DataFrame is using another DataFrame
- Example scenario: A Bank has opted for a new system, and all the accounts have old numbering formats. We need to update account numbers for all existing customers with new numbers. This can be done as shown below:

customers  
DataFrame

The diagram illustrates the update operation. It starts with two DataFrames: 'customers' and 'new\_accounts'. An arrow points from the 'customers' DataFrame to its corresponding table below. Another arrow points from the 'new\_accounts' DataFrame to its corresponding table below. The 'customers' table has columns 'Name' and 'AccountNo', and rows C1 through C4. The 'new\_accounts' table has columns 'AccountNo' and rows C1 through C4.

Name	AccountNo
C1	Jude
C2	Madan
C3	Rafael
C4	Edvina

new\_accounts  
DataFrame

AccountNo
C1
C2
C3
C4

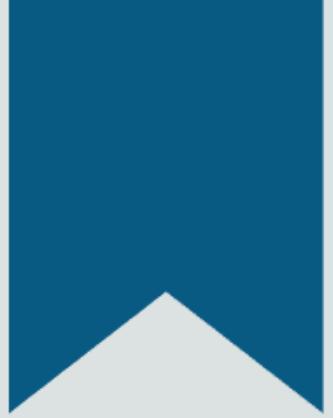
Code example

```
customers.update(new_accounts)
```

The diagram shows the result of the update operation. An arrow points from the code example to the resulting 'customers' DataFrame. The resulting DataFrame has columns 'Name' and 'AccountNo', and rows C1 through C4. The 'AccountNo' values have been updated to 1001, 1002, 1003, and 1004 respectively.

Name	AccountNo
C1	1001
C2	1002
C3	1003
C4	1004





# Filtering A DataFrame



# Introduction to Filtering a DataFrame

Filtering allows to reduce the size of the data used for analysis based on a condition, thereby improving the speed and accuracy of the result.

Example scenario: Consider a data set containing purchase details from a chain of retail stores. Following details are provided in the data set:

Code example

	CustID	City	Store Number	Sales (INR)	Customer Age	Gender
0	1001	Mumbai	10	123	38	M
1	1003	Bangalore	23	543	40	M
2	1006	Delhi	15	220	30	F
3	1007	Chennai	17	190	45	F
4	1009	Mumbai	11	269	28	F

Purchases DataFrame



# Filtering using a Single Value

Example: Display records of only “Male” customers are shown below:

Code example

```
purchases[purchases['Gender']=='M']
```

Code output

	CustID	City	Store Number	Sales (INR)	Customer Age	Gender
0	1001	Mumbai	10	123	38	M
1	1003	Bangalore	23	543	40	M
5	1010	Bangalore	21	130	56	M
6	1011	Mumbai	13	175	43	M
7	1012	Delhi	16	210	36	M
8	1013	Delhi	18	120	25	M
10	1015	Mumbai	11	160	49	M
13	1020	Mumbai	10	273	55	M



# Filtering using a List

Example: Display records showing purchase details from “Bangalore” and “Chennai” are given below:

Code example

```
cities=['Bangalore', 'Chennai']  
purchases[purchases.City.isin(cities)]
```

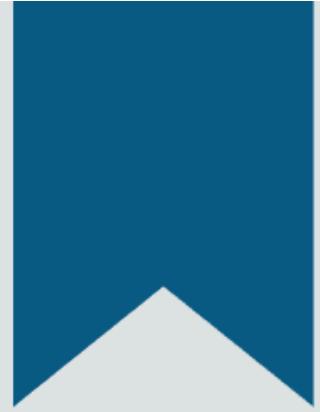
Code output

	CustID	City	Store Number	Sales (INR)	Customer Age	Gender
1	1003	Bangalore	23	543	40	M
3	1007	Chennai	17	190	45	F
5	1010	Bangalore	21	130	56	M
9	1014	Bangalore	21	150	29	F
12	1019	Chennai	17	242	62	F



# Combining Data using Pandas - I





# Concatenate and Merge Data



# Exploring Data to be Combined

A company has the data of its employees stored in separate datasets which needs to be combined for better storage and analysis

Consider the various data sets as shown below:

employees1

	EmplID	Name	Location	Age
0	E5651	Payal	Pune	35
1	E5652	Edvina	Bangalore	30
2	E5653	Madan	Bangalore	45
3	E5654	Rafael	Chennai	40
4	E5655	Gopal	Chennai	32

employees2

	EmplID	Name	Location
0	E4501	Sumeet	Mumbai
1	E4502	Nikhil	Mumbai
2	E4503	Paul	Bangalore

service

	YearsOfService
0	10
1	5
2	17
3	12
4	7



# Concatenate Rows

- Concatenation is the process of combining two or more data structures
- Let's see how to combine rows from two data sets employees1 and employees2

Code example

```
pd.concat([employees1, employees2])
```

Code output

	Age	EmplID	Location	Name
0	35.0	E5651	Pune	Payal
1	30.0	E5652	Bangalore	Edvina
2	45.0	E5653	Bangalore	Madan
3	40.0	E5654	Chennai	Rafael
4	32.0	E5655	Chennai	Gopal
0	NaN	E4501	Mumbai	Sumeet
1	NaN	E4502	Mumbai	Nikhil
2	NaN	E4503	Bangalore	Paul



# Concatenate Columns

`concat()` function allows combining columns from two DataFrames containing similar index values such as in case of employees1 and service DataFrames

Code example

```
pd.concat([employees1,service], axis=1)
```

Code output

	EmplID	Name	Location	Age	YearsOfService
0	E5651	Payal	Pune	35	10
1	E5652	Edvina	Bangalore	30	5
2	E5653	Madan	Bangalore	45	17
3	E5654	Rafael	Chennai	40	12
4	E5655	Gopal	Chennai	32	7



# Append Rows – `append()`

`append()` function appends the rows to a DataFrame resulting in a new DataFrame

Code example

```
employees1.append(employees2)
```

Code output

	Age	EmpID	Location	Name
0	35.0	E5651	Pune	Payal
1	30.0	E5652	Bangalore	Edvina
2	45.0	E5653	Bangalore	Madan
3	40.0	E5654	Chennai	Rafael
4	32.0	E5655	Chennai	Gopal
0	NaN	E4501	Mumbai	Sumeet
1	NaN	E4502	Mumbai	Nikhil
2	NaN	E4503	Bangalore	Paul

Note: `append()` can be thought of as a specific case of `concat()`. `concat()` provides more flexibility



# Merge Operation

`merge()` is a pandas operation that performs database-style joins on columns. Merge operation is performed on all the common rows of given DataFrames

Consider `years` data set which needs to be merged with `employees1` data. This can be done as shown below-

years		
	EmplID	YearsOfService
0	E5651	10
1	E5652	5
2	E5653	17
3	E5654	12
4	E5660	7

employees1				
	EmplID	Name	Location	Age
0	E5651	Payal	Pune	35
1	E5652	Edvina	Bangalore	30
2	E5653	Madan	Bangalore	45
3	E5654	Rafael	Chennai	40
4	E5655	Gopal	Chennai	32



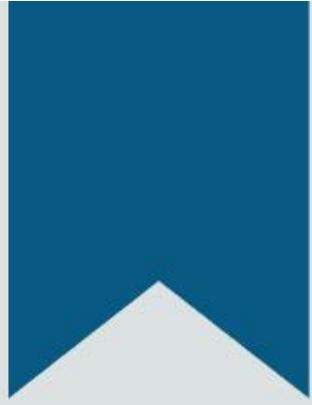
Code example					
employees1.merge(years)					
EmplID	Name	Location	Age	YearsOfService	
0 E5651	Payal	Pune	35	10	
1 E5652	Edvina	Bangalore	30	5	
2 E5653	Madan	Bangalore	45	17	
3 E5654	Rafael	Chennai	40	12	

Code output



# Combining Data using Pandas - II



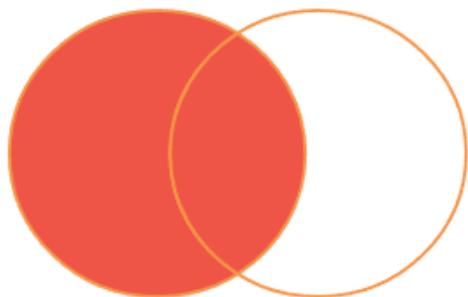


# Join Data



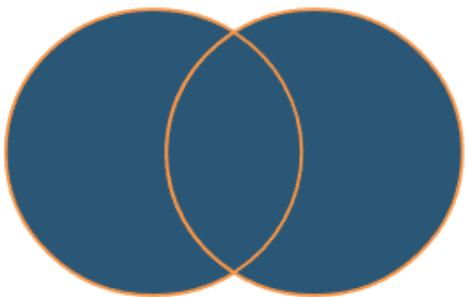
# Join Operation

Join operation on DataFrames can be performed in four different ways as follows:



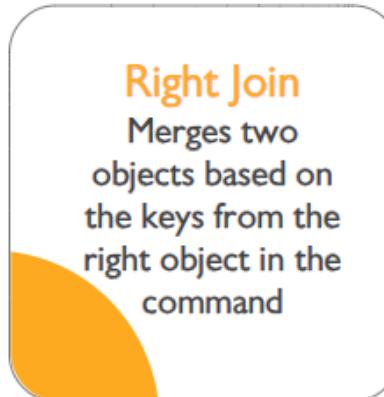
## Left Join

Merges two objects based on the keys from the left object in the command



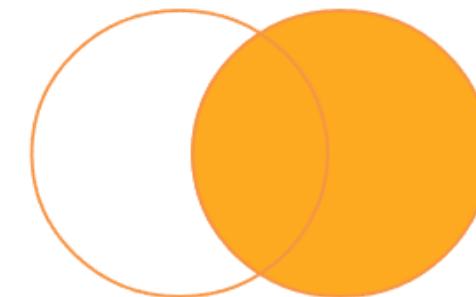
## Outer Join

Merges two objects based on full union of keys from both the objects



## Right Join

Merges two objects based on the keys from the right object in the command



## Inner Join

Merges two objects based on intersection of keys from both the objects



# Left Join

- Consider the same data sets used for merge operation : employees1 and years
- There are some common “EmpIDs” in both data sets, some EmpIDs exclusive to employees1 and some EmpIDs exclusive to years data set

## Code example

```
DF = pd.merge(employees1, years, on='EmpID', how='left')  
print(DF)
```

## Code output

	EmplD	Name	Location	Age	YearsOfService
0	E5651	Payal	Pune	35	10.0
1	E5652	Edvina	Bangalore	30	5.0
2	E5653	Madan	Bangalore	45	17.0
3	E5654	Rafael	Chennai	40	12.0
4	E5655	Gopal	Chennai	32	NaN

Left Join includes all keys from left (employees1) DataFrame

# Right Join

---

## Code example

```
DF = pd.merge(employees1, years, on='EmpID', how='right')
print(DF)
```

## Code output

	EmpID	Name	Location	Age	YearsOfService
0	E5651	Payal	Pune	35.0	10
1	E5652	Edvina	Bangalore	30.0	5
2	E5653	Madan	Bangalore	45.0	17
3	E5654	Rafael	Chennai	40.0	12
4	E5660	NaN	NaN	NaN	7

Right Join includes all  
keys from right  
(years) DataFrame



# Outer Join

---

## Code example

```
DF = pd.merge(employees1, years, on='EmpID', how='outer')  
print(DF)
```

## Code output

	EmplID	Name	Location	Age	YearsOfService
0	E5651	Payal	Pune	35.0	10.0
1	E5652	Edvina	Bangalore	30.0	5.0
2	E5653	Madan	Bangalore	45.0	17.0
3	E5654	Rafael	Chennai	40.0	12.0
4	E5655	Gopal	Chennai	32.0	NaN
5	E5660	NaN	NaN	NaN	7.0

Outer join includes all  
keys from both  
DataFrames



# Inner Join

---

## Code example

```
DF = pd.merge(employees1, years, on='EmpID', how='inner')  
print(DF)
```

## Code output

	EmplID	Name	Location	Age	YearsOfService
0	E5651	Payal	Pune	35	10
1	E5652	Edvina	Bangalore	30	5
2	E5653	Madan	Bangalore	45	17
3	E5654	Rafael	Chennai	40	12

Inner Join includes  
only the common  
keys from both  
DataFrames



# Data Cleaning using Pandas - I





# Data Cleaning using Pandas – Use Case



# Restaurant Customer Rating Data (Use Case)

- Consider a data set containing ratings provided by customers for Delicious Restaurant
- This data is imported into a pandas DataFrame and displayed as shown below
- Observe the various inconsistencies in this data:



us_ID	ID	placeID	food_rating	service_rating
0	U1077	135085	5.0	3
1	U1068	132630	NaN	3
2	U1067	132584	4.0	5
3	U1067	132733	1.0	1
4	U1103	132733	4.0	3
5	U1107	132660	3.0	4
6	U1107	132660	3.0	4
7	U1044	132583	3.0	-999
8	U1070	132608	4.0	3

Inconsistencies in data set

Duplicate IDs

Missing value

Duplicate Rows

Garbage value



# Handling Missing Data Using Pandas

- Missing data is represented using NaN (Not a Number) in pandas
- Missing values can be handled in the following ways in pandas:

`dropna()`

Deletes rows/ columns having null values

`fillna()`

Fills in the missing data using some value or an interpolation method

`isnull()`

Returns False/ True based on whether the value is missing or not

`notnull()`

Returns values which are not null



# Check For Missing Values

`isnull()` function allows checking whether a value is present or not for the entire DataFrame or individual columns of the DataFrame

Code example

```
ratings.isnull()
```

Code output

	userID	placeID	food_rating	service_rating
0	False	False	False	False
1	False	False	True	False
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False
5	False	False	False	False
6	False	False	False	False
7	False	False	False	False
8	False	False	False	False

Code example

```
ratings['food_rating'].  
    .isnull()
```

Code output

```
0  False  
1  True  
2  False  
3  False  
4  False  
5  False  
6  False  
7  False  
8  False  
Name: food_rating, dtype: bool
```



# Handle Missing Values

- Based on the requirement set as per the designed Data Science process, one can choose to drop rows containing missing data or fill some value at that location

Code example

```
ratings.dropna()
```

Code output

userID	placeID	food_rating	service_rating
0	U1077	135085	5.0
2	U1067	132584	4.0
3	U1067	132733	1.0
4	U1103	132733	4.0
5	U1107	132660	3.0
6	U1107	132660	3.0
7	U1044	132583	3.0
8	U1070	132608	4.0

Code example

```
ratings.fillna(0)
```

Code output

userID	placeID	food_rating	service_rating
0	U1077	135085	5.0
1	U1068	132630	0.0
2	U1067	132584	4.0
3	U1067	132733	1.0
4	U1103	132733	4.0
5	U1107	132660	3.0
6	U1107	132660	3.0
7	U1044	132583	3.0
8	U1070	132608	4.0

Code example

```
ratings.fillna(ratings['food_rating'].mean())
```

Code output

userID	placeID	food_rating	service_rating
0	U1077	135085	5.000
1	U1068	132630	3.375
2	U1067	132584	4.000
3	U1067	132733	1.000
4	U1103	132733	4.000
5	U1107	132660	3.000
6	U1107	132660	3.000
7	U1044	132583	3.000
8	U1070	132608	4.000



# Data Cleaning using Pandas - II





# Data Cleaning



# Inspecting and Removing Duplicates

Duplicate rows of a DataFrame can be handled using the below functions of pandas DataFrame:

## 1. Inspecting for Duplicate Rows

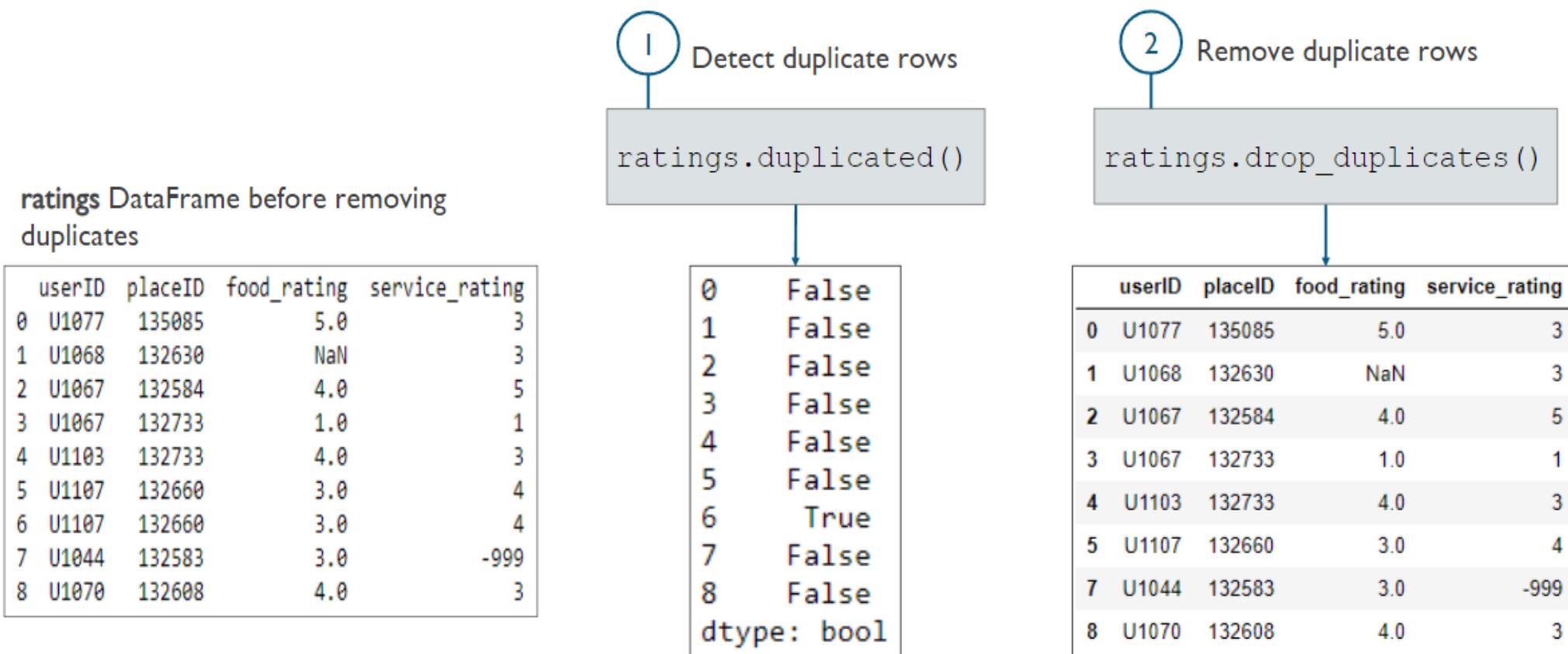
`.duplicated()` function returns a Boolean string indicating whether each row has been encountered before or not

## 2. Remove duplicate rows

`.drop_duplicates( )` function returns a dataframe by removing one of the duplicate rows

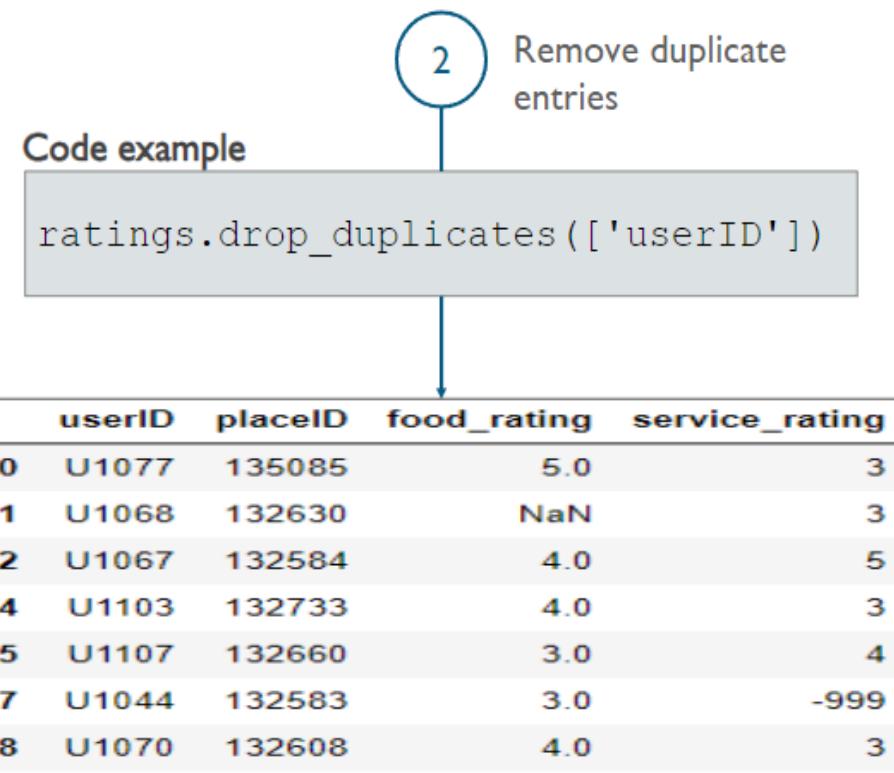
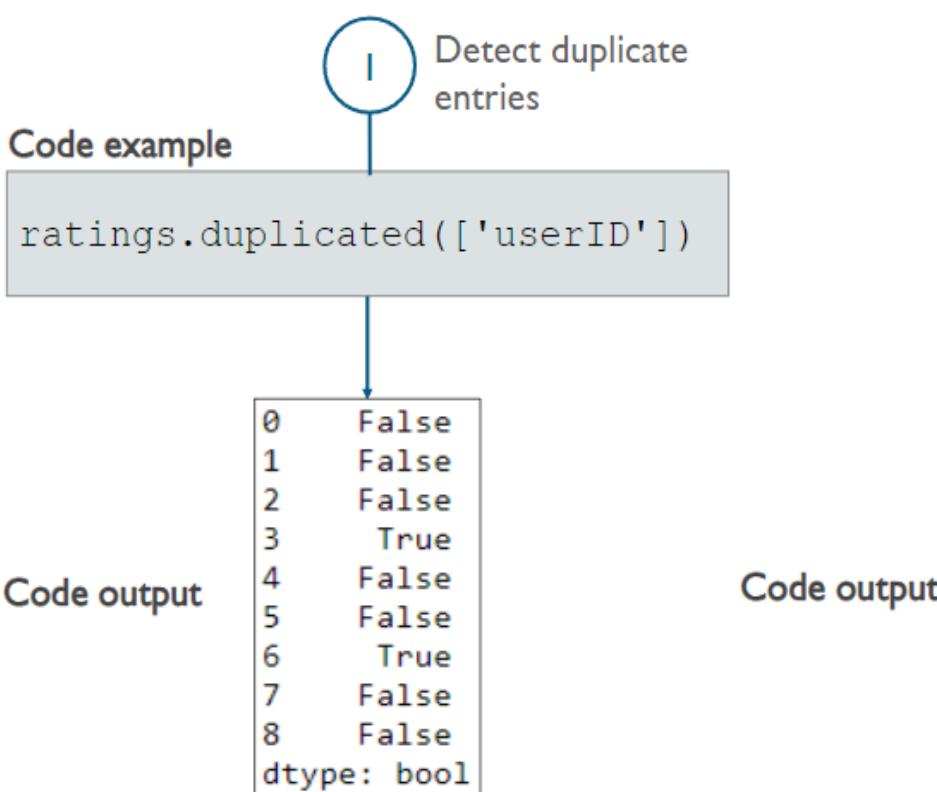
# Removing Duplicate Row

- Sometimes, data may contain exact copies of some records due to some errors in data acquisition
- Such copies can be detected and then removed as follows:



# Removing Duplicates Based On Specific Column

- Data may contain duplicate entries for a specified key due to some errors while entering the data
- Example: In case of “ratings” data, user U1067 has two entries, one of which needs to be removed



# Replacing Values

- replace() function allows to replace values from a Series or a column of a DataFrame
- Example: Data sets may contain null values which are represented using values such as -999.0, -1000.0 etc. To handle such types of missing values, replace() function will be useful

## Code example

```
ratings.replace(-999, np.nan)
```

service_rating
3
3
5
1
3
4
4
-999
3



## Code output

	userID	placeID	food_rating	service_rating
0	U1077	135085	5.0	3.0
1	U1068	132630	NaN	3.0
2	U1067	132584	4.0	5.0
3	U1067	132733	1.0	1.0
4	U1103	132733	4.0	3.0
5	U1107	132660	3.0	4.0
6	U1107	132660	3.0	4.0
7	U1044	132583	3.0	NaN
8	U1070	132608	4.0	3.0



The NaN value can then be handled using techniques discussed earlier



# Grouping Data using Pandas - I



# Groups and Aggregation

- Gives frequency distribution
- Analyze the results of different groups like the highest selling Product Type
- Pandas Group Functions:

Function	Description
<a href="#"><code>DataFrame.groupby()</code></a>	Group DataFrame using a mapper or by a Series of columns
<a href="#"><code>GroupBy.groups</code></a>	Dict {group name -> group labels}
<a href="#"><code>GroupBy.get_group()</code></a>	Construct DataFrame from group with provided name
<a href="#"><code>GroupBy.agg()</code></a>	Aggregate using one or more operations

ProductType	mean	min	max
Cardigan	11.333333	6.0	16.0
Coats/Jackets	12.653846	5.0	20.0
Dresses	13.473684	5.0	20.0
Hats	13.000000	7.0	20.0
Jeans	11.200000	5.0	18.0
Pants	14.485185	9.0	20.0
Shirts/Tops	12.500000	5.0	20.0

How can I calculate the average shipping time for each Product Type?

Simple! Group the data based on Product Type and find mean...



# groupby() Use Case I

Consider a dataset `worldcup`, containing Cricket world cup winning teams and the respective years. It is observed that there are multiple entries for the same teams in the data as shown:

	Team	Rank	Year
0	West Indies	7	1975
1	West Indies	7	1979
2	India	2	1983
3	Australia	1	1987
4	Pakistan	6	1992
5	Sri Lanka	4	1996
6	Australia	1	1999
7	Australia	1	2003
8	Australia	1	2007
9	India	2	2011
10	Australia	1	2015

Original data set imported into a pandas DataFrame

These entries can be grouped into a single entry using one of the columns as **Key**

This type of grouping is useful for performing aggregations; for example- number of times a team has won the world cup



# groupby () (Displaying Grouped Indices)

---

## Code example

```
data = pd.read_csv("worldcup.csv")
print(data.groupby('Team').groups)
```

## Code output

```
{'Australia': [3, 6, 7, 8, 10], 'India': [2, 9], 'Pakistan': [4], 'Sri Lanka': [5], 'West Indies': [0, 1]}
```



# groupby () (Counting Entries In Each Group)

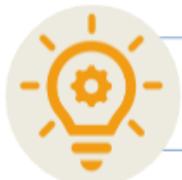
---

Code example

```
print(data.groupby('Team') ['Year'].count())
```

Code output

Team	
Australia	5
India	2
Pakistan	1
Sri Lanka	1
West Indies	2
Name: Year, dtype: int64	



count () calculates the number of values in a group



# Grouping Data using Pandas - II



# groupby () (Multiple Columns) (Use Case 2)

In this operation, hierarchical grouping takes place based on two or more columns

- Consider a data set containing sales data of chain of Retail stores Xtra\_Mart
- It has multiple stores across various cities in India.
- The data is stored in a DataFrame as shown below:

	CustID	City	Store Number	Sales (INR)	Customer	Age	Gender
0	1001	Mumbai	10	123		38	M
1	1003	Bangalore	23	543		40	M
2	1006	Delhi	15	220		30	F
3	1007	Chennai	17	190		45	F
4	1009	Mumbai	11	269		28	F
5	1010	Bangalore	21	130		56	M
6	1011	Mumbai	13	175		43	M
7	1012	Delhi	16	210		36	M
8	1013	Delhi	18	120		25	M
9	1014	Bangalore	21	150		29	F
10	1015	Mumbai	11	160		49	M
11	1017	Delhi	16	170		75	F
12	1019	Chennai	17	242		62	F
13	1020	Mumbai	10	273		55	M



# groupby () (Multiple Columns) (Use Case 2) (contd.)

- This feature is useful when there is a logical hierarchy of columns in the data set

## Code Example

```
print(purchases.groupby(['City', 'Store  
Number'])['Sales (INR)'].mean())
```



City	Store Number	Sales (INR)
Bangalore	21	140.0
	23	543.0
Chennai	17	216.0
	15	220.0
Delhi	16	190.0
	18	120.0
Mumbai	10	198.0
	11	214.5
	13	175.0
Name: Sales (INR), dtype: float64		



Here, `mean()` calculates the mean sales at each store grouped by City



# groupby () (Fetching Data of a Single Group)

A single group can be selected using `get_group()` function

## Code Example

```
print(purchases.groupby('City').get_group("Bangalore"))
```



	CustID	City	Store Number	Sales (INR)	Customer Age	Gender
1	1003	Bangalore	23	543	40	M
5	1010	Bangalore	21	130	56	M
9	1014	Bangalore	21	150	29	F

# Data Visualization Library – Matplotlib





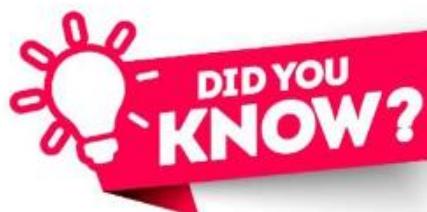
# Data Visualization



# Benefits of Data Visualization

- Visualization allows visual access to vast amounts of data in an easily understandable manner
- A visual summary of data makes it easier to identify trends and extract useful hidden insights

DATA  
VISUALIZATION

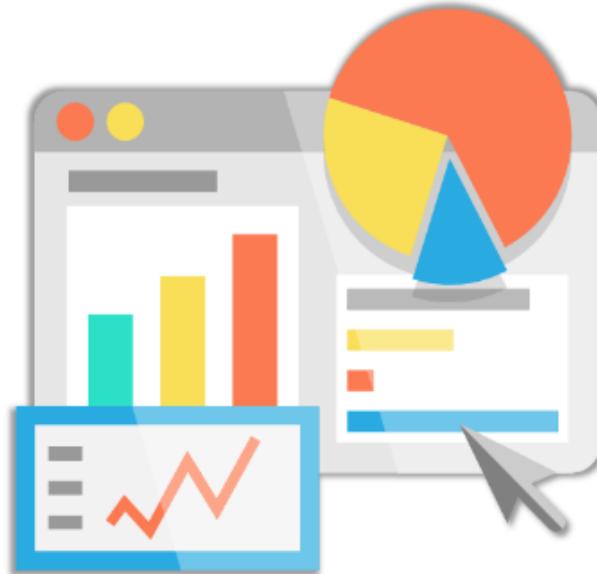


According to a study conducted by [MIT](#), 90 % of data transmitted to the human brain is visual



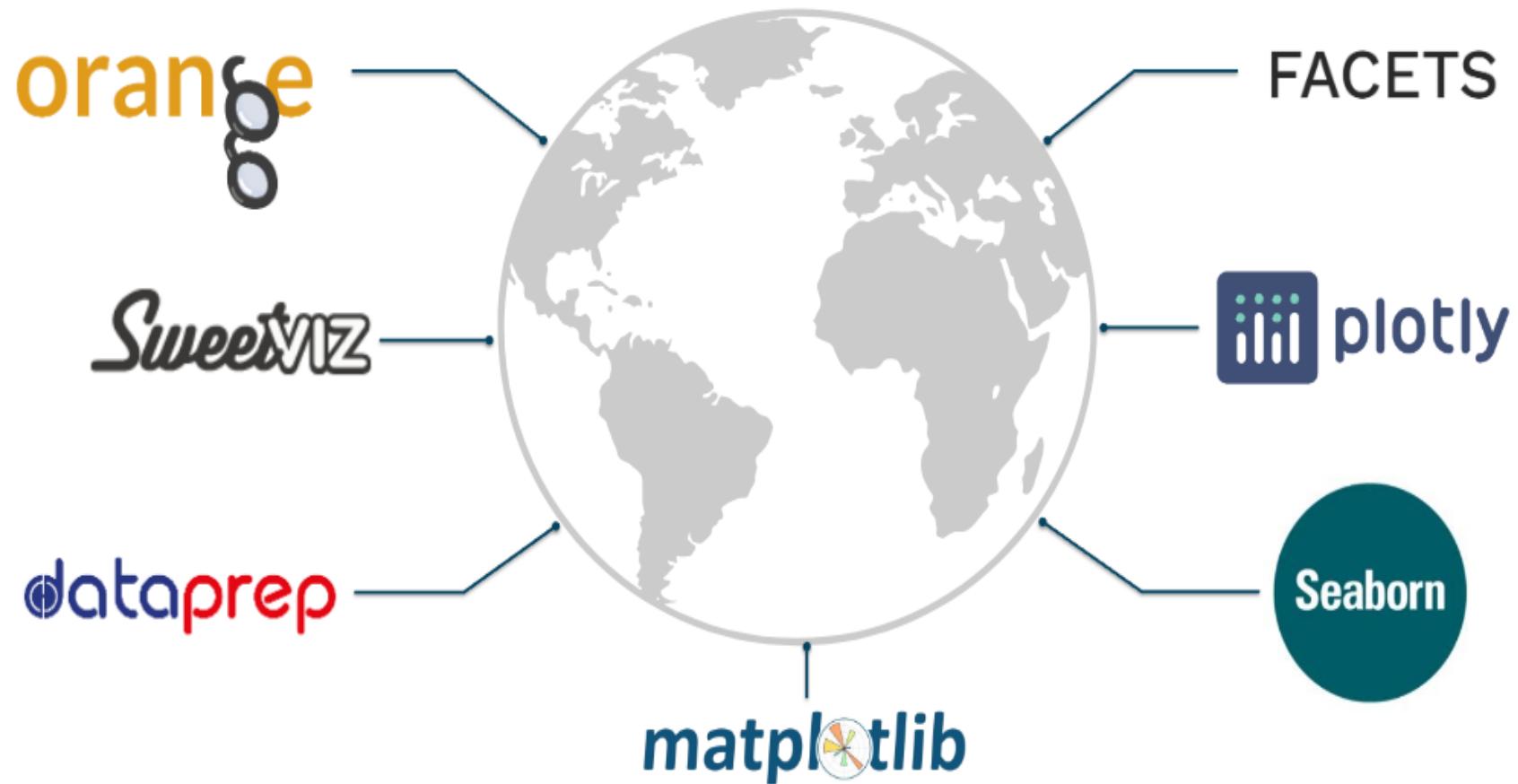
# Benefits of Data Visualization (contd.)

- It is a pictorial or graphical representation of data
- Enables decision-makers to see patterns, trends, and correlations that might go undetected in text-based data

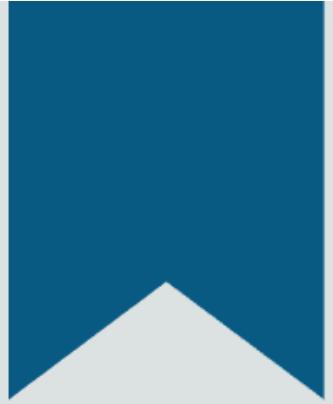


# Libraries and Tools for Data Visualization

---



# Matplotlib

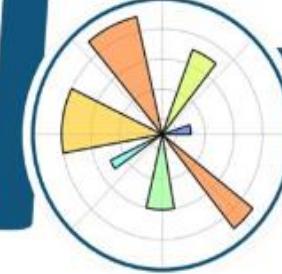


# Introduction to Matplotlib

---

- Matplotlib is a python library that is specially designed for the development of graphical elements such as plots and charts for interactive data visualization
- Matplotlib is inspired from the MATLAB software and reproduces many of its features

matplotlib



# Installing Matplotlib

Open the terminal on your system and install matplotlib using pip command

```
C:\Users\████████>pip install matplotlib
Collecting matplotlib
  Downloading https://files.pythonhosted.org/packages/71/13/0720e50bd8988299137fd7e936e4d494b45a473c5fe70d72cd6c1bd79163/matplotlib-3.1.1-cp37-cp37m-win32.whl (8.9MB)
    ██████████████████████ | 8.9MB 595kB/s
Collecting cycler>=0.10 (from matplotlib)
  Downloading https://files.pythonhosted.org/packages/f7/d2/e07d3ebb2bd7af696440ce7e754c59dd546ffe1bbe732c8ab68b9c834e61/cycler-0.10.0-py2.py3-none-any.whl
Requirement already satisfied: numpy>=1.11 in e:\softwares\python\3.7.3\lib\site-packages (from matplotlib) (1.17.0)
Collecting kiwisolver>=1.0.1 (from matplotlib)
  Downloading https://files.pythonhosted.org/packages/20/6a/e5fff2ed776ab0cd11d7c1d5d3e5e549952464a6f1b9084b7ecbd8341352/kiwisolver-1.1.0-cp37-none-win32.whl (44kB)
    ██████████ | 51kB 59kB/s
Requirement already satisfied: python-dateutil>=2.1 in e:\softwares\python\3.7.3\lib\site-packages (from matplotlib) (2.8.0)
Collecting pyparsing!=2.0.4,!>=2.1.2,!>=2.1.6,>=2.0.1 (from matplotlib)
  Downloading https://files.pythonhosted.org/packages/11/fa/0160cd525c62d7abd076a070ff02b2b94de589f1a9789774f17d7c54058e/pyparsing-2.4.2-py2.py3-none-any.whl (65kB)
    ██████████ | 71kB 38kB/s
Requirement already satisfied: six in e:\softwares\python\3.7.3\lib\site-packages (from cycler>=0.10->matplotlib) (1.12.0)
Requirement already satisfied: setuptools in e:\softwares\python\3.7.3\lib\site-packages (from kiwisolver>=1.0.1->matplotlib) (40.8.0)
Installing collected packages: cycler, kiwisolver, pyparsing, matplotlib
Successfully installed cycler-0.10.0 kiwisolver-1.1.0 matplotlib-3.1.1 pyparsing-2.4.2
```



# Types of Plots and Charts



# Line Plot

- Line plots are data points connected with a line
- Each datapoint is located using x, y value-pair
- Datapoints are plotted y versus x using plot () function
- Consider the rainfall\_in\_kerala data below:

Rainfall in Kerala - dataset

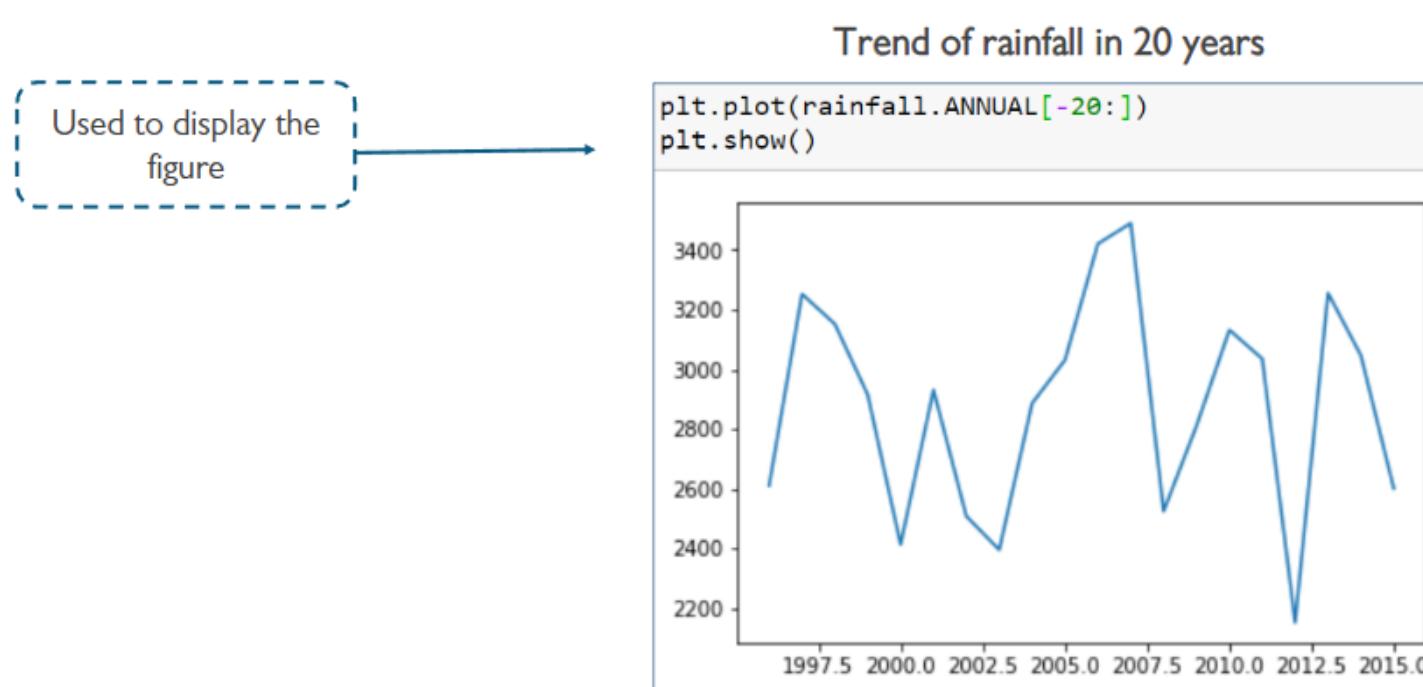
	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL
1901	28.7	44.7	51.6	160.0	174.7	824.6	743.0	357.5	197.7	266.9	350.8	48.4	3248.6
1902	6.7	2.6	57.3	83.9	134.5	390.9	1205.0	315.8	491.6	358.4	158.3	121.5	3326.6
1903	3.2	18.6	3.1	83.6	249.7	558.6	1022.5	420.2	341.8	354.1	157.0	59.0	3271.2
1904	23.7	3.0	32.2	71.5	235.7	1098.2	725.5	351.8	222.7	328.1	33.9	3.3	3129.7
1905	1.2	22.3	9.4	105.9	263.3	850.2	520.5	293.6	217.2	383.5	74.4	0.2	2741.6
1906	26.7	7.4	9.9	59.4	160.8	414.9	954.2	442.8	131.2	251.7	163.1	86.0	2708.0
1907	18.8	4.8	55.7	170.8	101.4	770.9	760.4	981.5	225.0	309.7	219.1	52.8	3671.1
1908	8.0	20.8	38.2	102.9	142.6	592.6	902.2	352.9	175.9	253.3	47.9	11.0	2648.3
1909	54.1	11.8	61.3	93.8	473.2	704.7	782.3	258.0	195.4	212.1	171.1	32.3	3050.2

How can I find the trend of annual rainfall?



# Line Plot (contd.)

- Line Plots are used to monitor trends or changes over a period
- Plot () function is used to plot a simple line plot which takes y as the required argument
- Datapoints are located using x, y value-pair

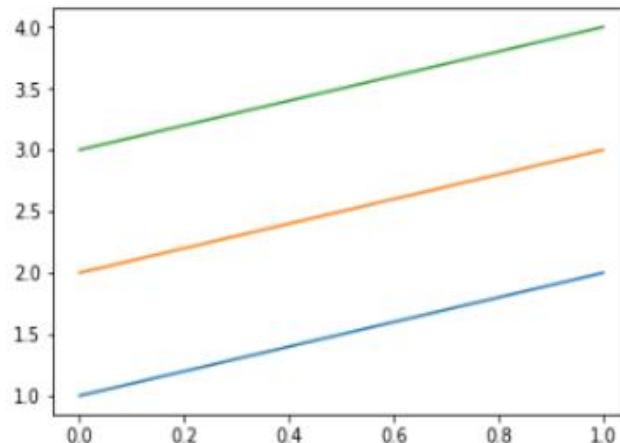


# Line Plot (contd.)

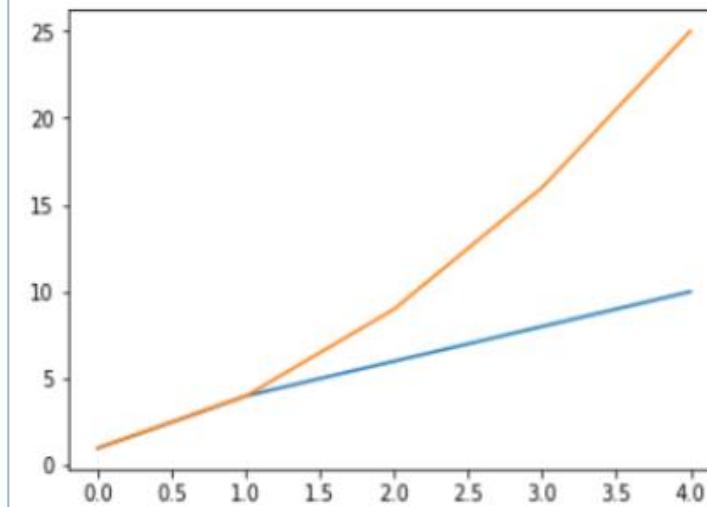
- Plot function supports NumPy arrays too
- Labelled data can also be plotted from DataFrames

Examples of line plot

```
arr=np.array([[1,2,3],[2,3,4]])
...
shape = (2,3)
where 2 is number of datapoints
and 3 is number of lines
...
plt.plot(arr) #Three parallel lines
plt.show()
```



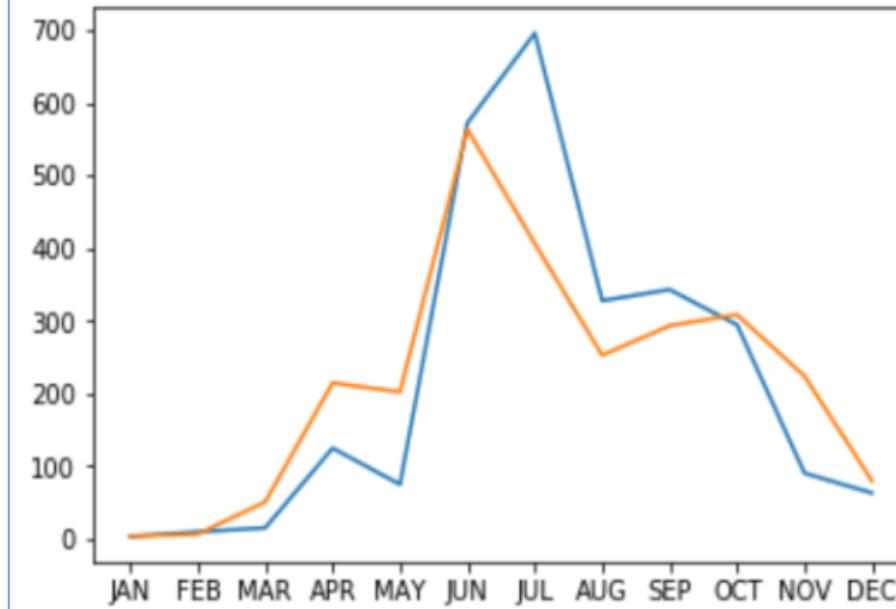
```
data={'y1':[1,4,6,8,10], 'y2':[1,4,9,16,25]}
df=pd.DataFrame(data)
plt.plot('y1',data=df)
plt.plot('y2',data=df)
plt.show()
```



## Line Plot (contd.)

Monthly trend of rainfall in the year of 1996 and 2015

```
plt.plot(rainfall.loc[1996, 'JAN':'DEC'])  
plt.plot(rainfall.loc[2015, 'JAN':'DEC'])  
plt.show()
```



Which line represents which year?

This will be covered in styling section



# Bar Chart

- Bar charts are used to visually compare two or more values using rectangular bars
- They are used to compare aggregate values across categorical data
- `bar()` function is used to plot a bar plot
- Consider "Students" dataset as shown below:

"Students" dataset

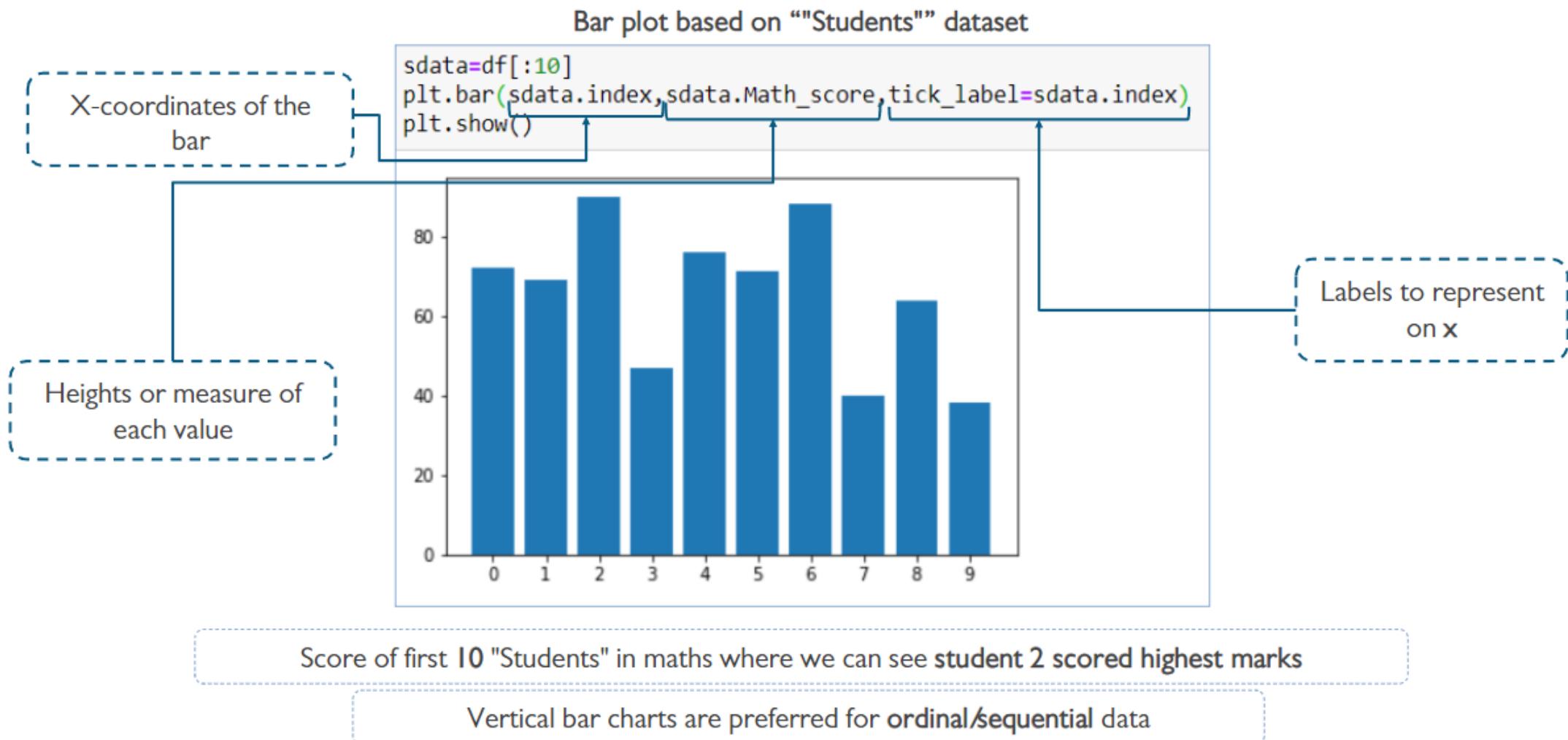


Can you find which student scored highest in mathematics?

	Group	Qualification	Math_score	Reading_score	Writing_score
0	group B	bachelor's degree	72	72	74
1	group C	some college	69	90	88
2	group B	master's degree	90	95	93
3	group A	associate's degree	47	57	44
4	group C	some college	76	78	75
5	group B	associate's degree	71	83	78
6	group B	some college	88	95	92
7	group B	some college	40	43	39
8	group D	high school	64	64	67
9	group B	high school	38	60	50



# Bar Chart (contd.)

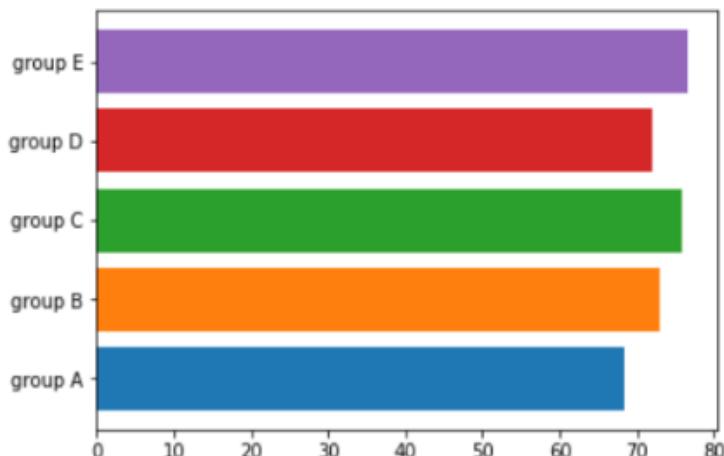


# Horizontal Bar Chart

- Horizontal bar charts can be represented using the `barrh()` function
- Here, longer bars represent greater values

Horizontal bar chart based on "Students" dataset

```
for group,data in df.groupby('Group'):
    plt.barh(group,data[data.Qualification=='bachelor\'s degree'].mean())
plt.show()
```



Horizontal bar charts are preferred for nominal/categorical data. here, we are showing number of bachelors in each group

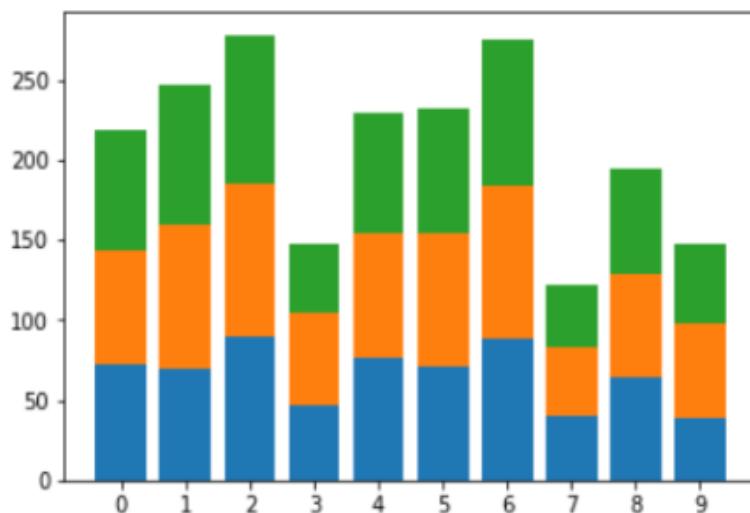


# Stacked Bar Chart

Using bottom parameter, we can create stacked vertical bar charts

Stacked bar chart based on "Students" dataset

```
plt.bar(sdata.index,sdata.Math_score,tick_label=sdata.index)
plt.bar(sdata.index,sdata.Reading_score,bottom=sdata.Math_score)
plt.bar(sdata.index,sdata.Writing_score,bottom=sdata.Math_score+sdata.Reading_score)
plt.show()
```



Scores of first 10 "Students" for math, reading and writing



# Stacked Bar Chart (contd.)

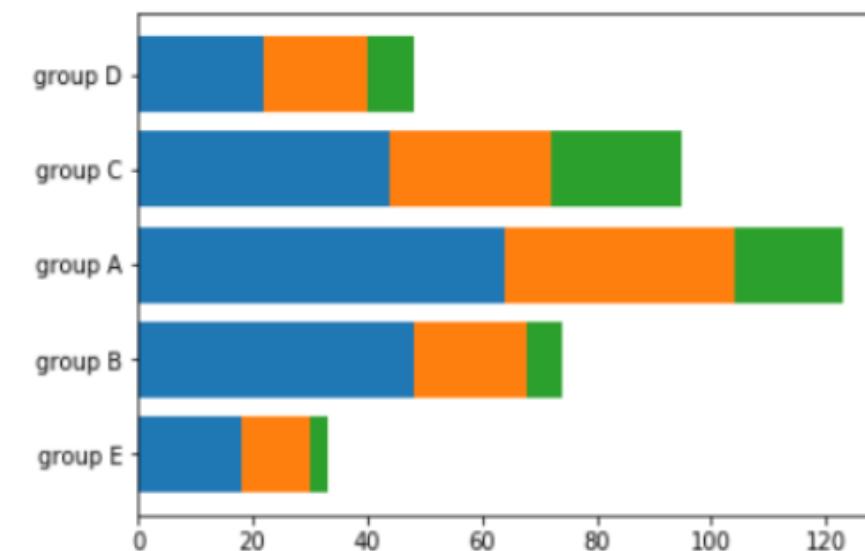
Using left parameter, we can create stacked horizontal bar charts

Code example (Stacked bar)

```
highschool,bach,masters=[],[],[]
group=list(set(df.Group))
#Filtering qualifications of students in each group
for grp,data in df.groupby('Group'):
    highschool.append(len(data[data.Qualification=='high school']))
    bach.append(len(data[data.Qualification=='bachelor\'s degree']))
    masters.append(len(data[data.Qualification=='master\'s degree']))

plt.barh(group,highschool)
plt.barh(group,bach,left=highschool)
plt.barh(group,masters,left=np.array(bach)+np.array(highschool))
plt.show()
```

Code output



Qualifications of students in each group

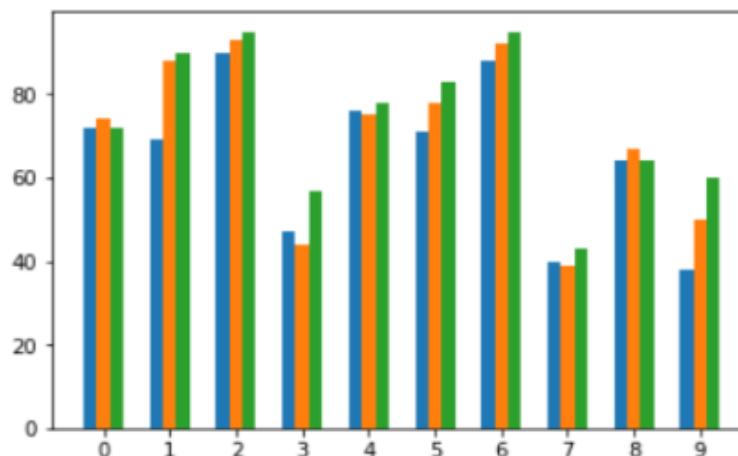


# Grouped Bar Chart

Width parameter is used to adjust positions of each group value in vertical bar charts

Grouped bar chart based on "Students" dataset

```
grp_width=0.2  
plt.bar(sdata.index-grp_width,sdata.Math_score,width=grp_width)  
plt.bar(sdata.index,sdata.Writing_score,tick_label=sdata.index,width=grp_width)  
plt.bar(sdata.index+grp_width,sdata.Reading_score,width=grp_width)  
plt.show()
```



Scores of first 10 students for math, reading, and writing



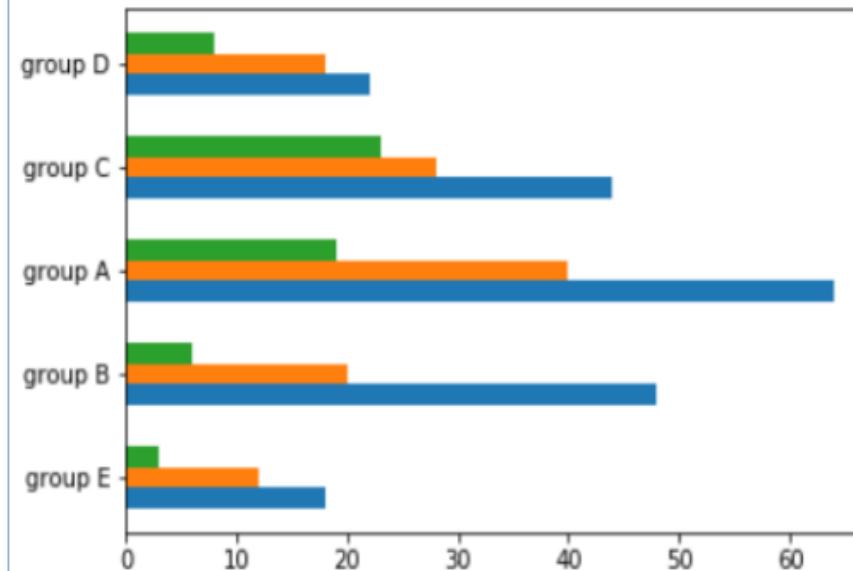
# Grouped Bar Chart (contd.)

Height parameter is used to adjust positions of each group value in horizontal bar charts

Code example (grouped bar)

```
grp_height=0.2  
...  
cannot use mathematical operations on list of string  
using numpy array instead  
...  
grp_ind=np.arange(len(group))  
plt.barh(grp_ind-grp_height,highschool,height=grp_height)  
plt.barh(grp_ind,bach,tick_label=group,height=grp_height)  
plt.barh(grp_ind+grp_height,masters,height=grp_height)  
plt.show()
```

Code output



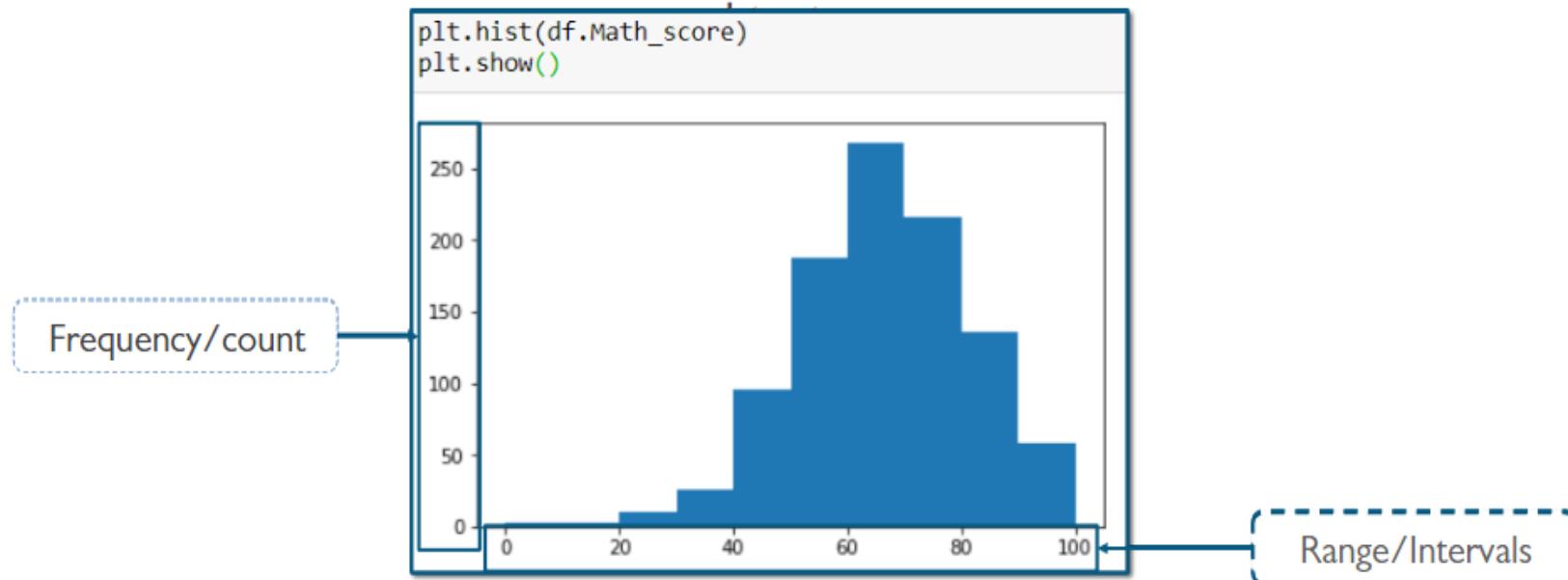
Qualifications of "Students" in each group



# Histogram

- ☞ Histograms display the frequency values into discrete intervals called bins
- ☞ Frequency count is kept on Y-axis whereas intervals on X-axis
- ☞ `hist()` function is used to plot histograms which takes array or sequence of arrays

Histogram example based on "Students"



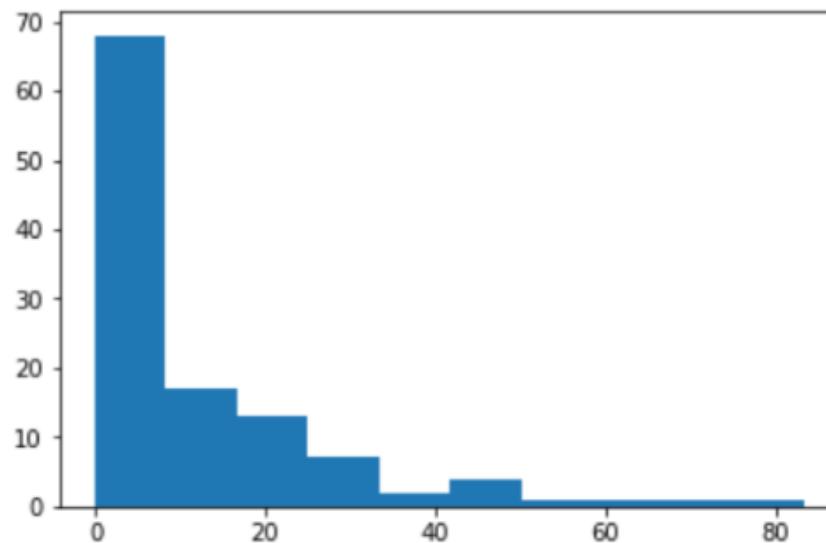
The above histogram shows that maximum number of students got a score between 60-70



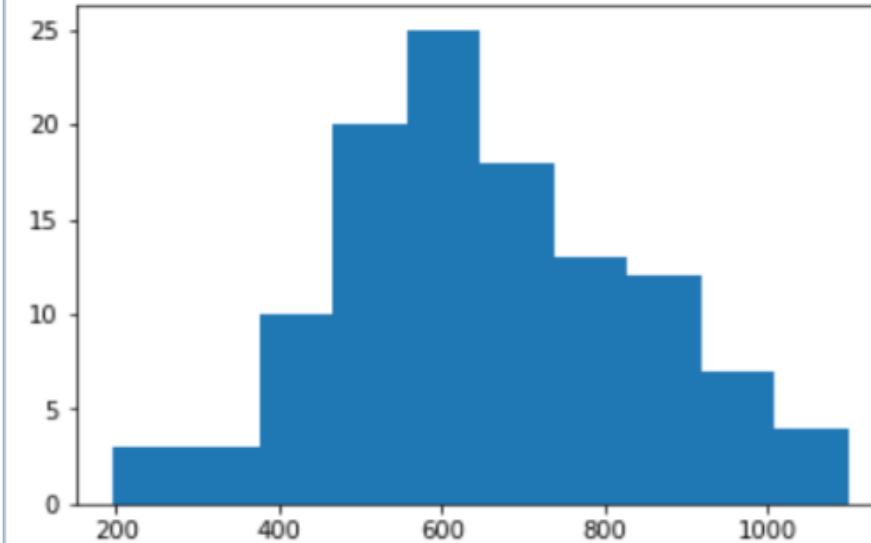
# Histogram (contd.)

Histogram example based on rainfall dataset

```
plt.hist(rainfall.JAN)  
plt.show()
```



```
plt.hist(rainfall.JUN)  
plt.show()
```



Above histograms shows the frequency of rainfall in the month of January and June, respectively. June is the month with good amount of rainfall



# Data Visualization Library – Seaborn



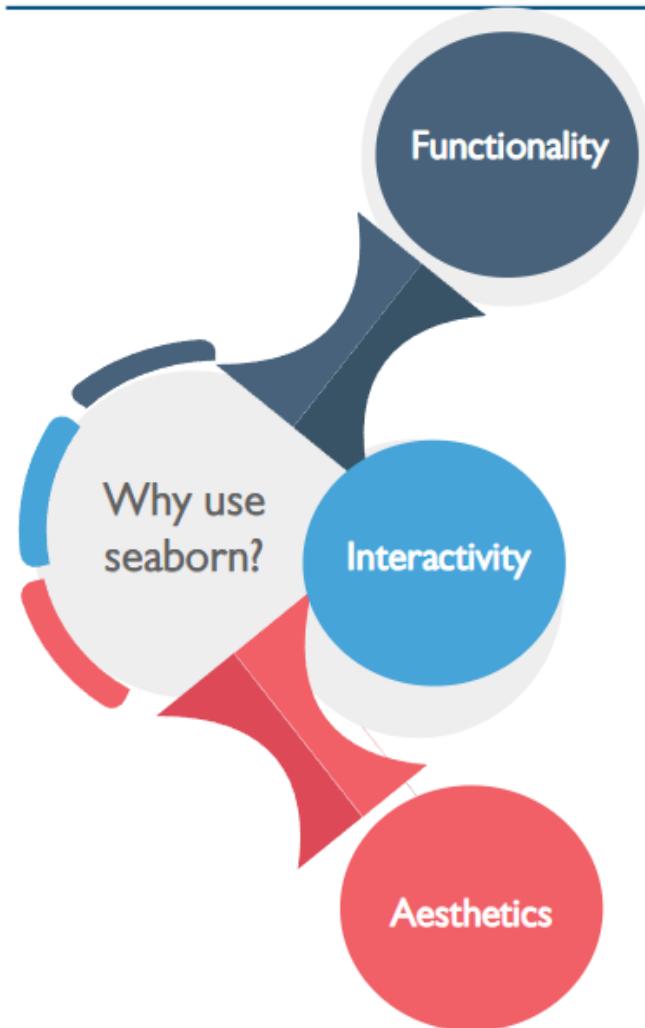


# Data Visualization Using Seaborn



# Benefits of Seaborn

---



Uses fewer syntax and has easy and interesting default themes

Provides interactive visualization

Has customizable aesthetics



# Data Visualization using Seaborn

---

Seaborn is built on top of matplotlib, used for drawing attractive and informative statistical graphics.

- 
- 1 Close integration with the pandas DataFrame
  - 2 Specialize support for categorical variables
  - 3 Tools for selecting colour palettes that reveal hidden patterns in the data



# Matplotlib vs. Seaborn

---

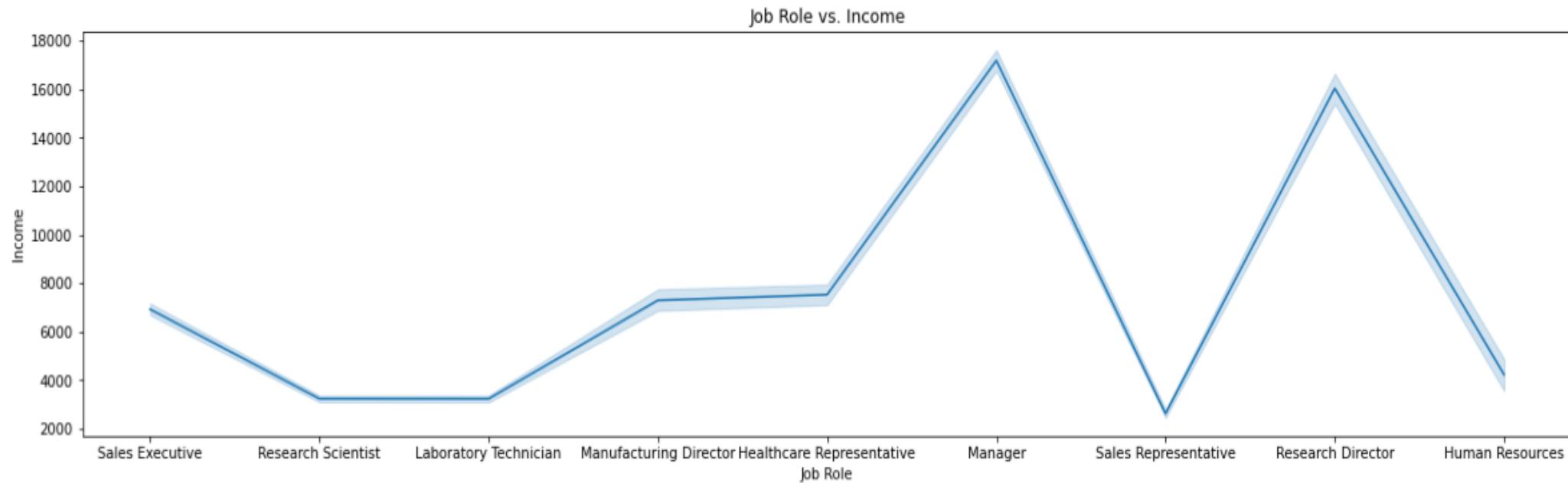
Features	Matplotlib	Seaborn
Functionality	Mainly used for basic plotting using bar plots, scatter plots, etc.	Specializes in statistical data visualizations
Visualization	Well-integrated with the numpy and pandas library of python	More integrated for working with pandas DataFrames
Flexibility	Highly customizable and powerful	Provides default themes which are most used
Interactivity	Not available	Available
Functionality	Mainly used for basic plotting using bar plots, scatter plots, etc.	Specializes in statistical data visualizations



# Line Plot

Task : What is the salary trend based on their job role?

Line plot example



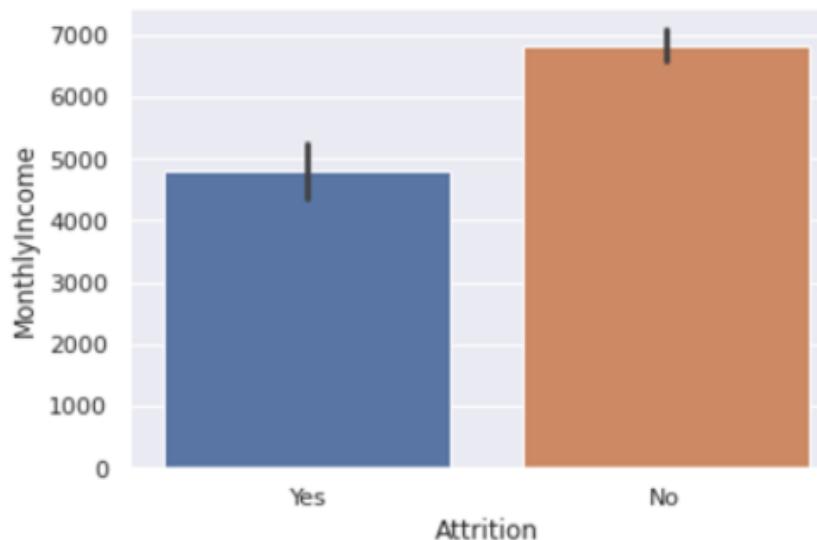
Line plot usually shows trend over time



# Bar Plot

Task : How does the average monthly income affect the attrition rate?

Bar plot example



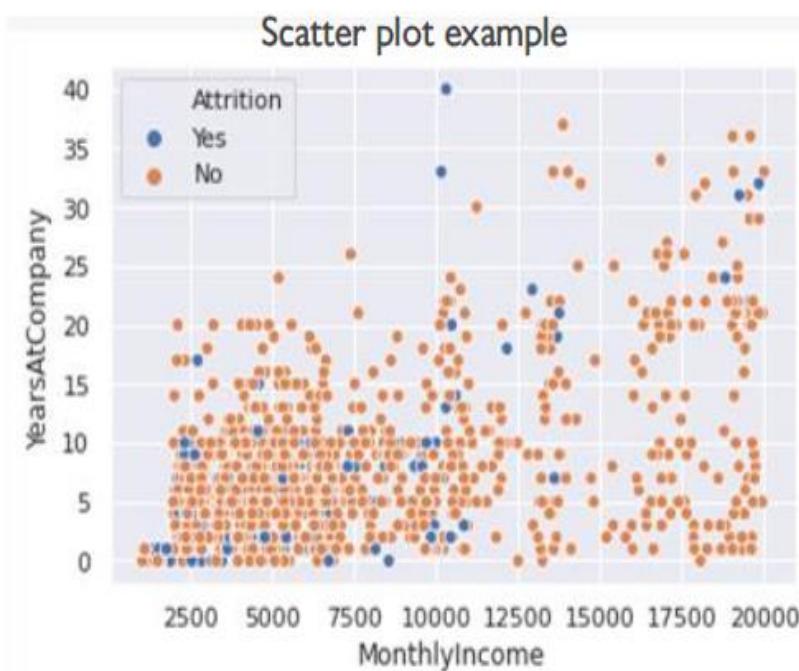
A bar plot or chart is a plot that shows the relationship between a numerical variable and a categorical variable



# Scatter Plot

---

Task : Which employees are more likely to leave? senior employees with higher monthly income or recent joiners with low monthly income



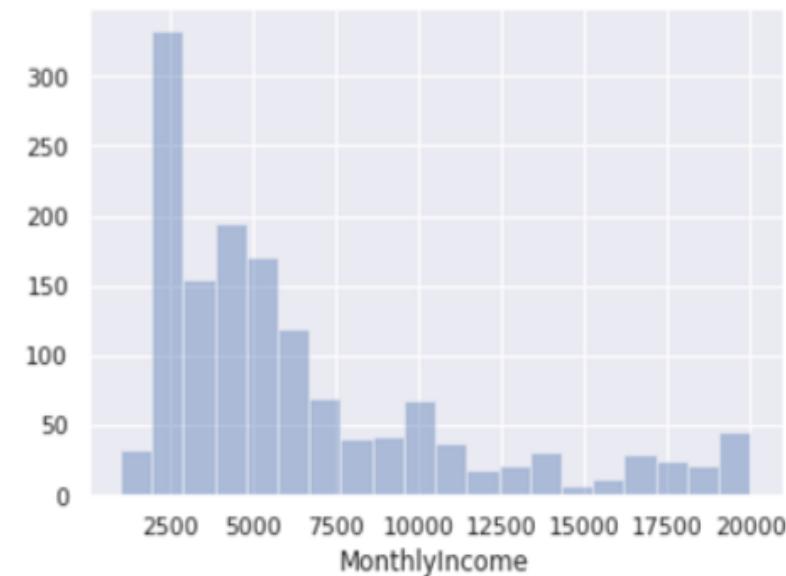
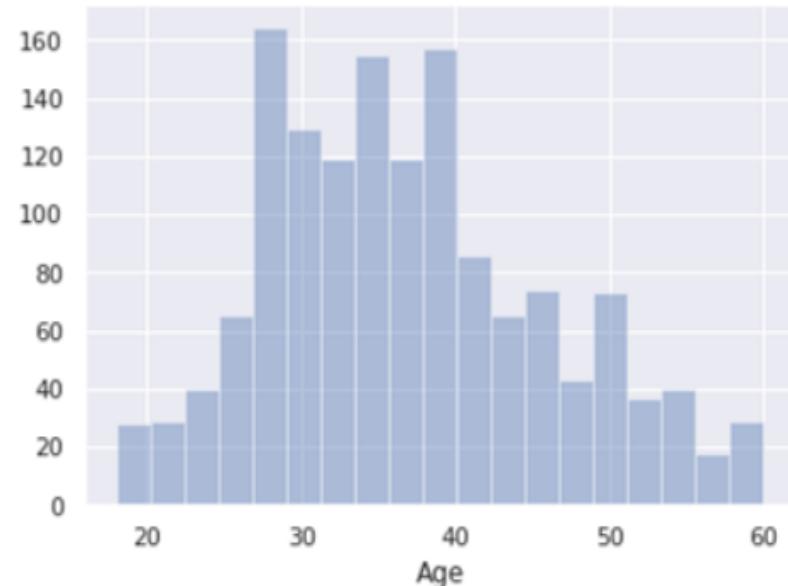
A scatter plot is used to analyze the relationship between two numerical variables graphically. It is also efficient in detecting the outliers in the dataset.



# Histogram

Task : What is the distribution of the age & monthly income of all the Employees

Histogram example



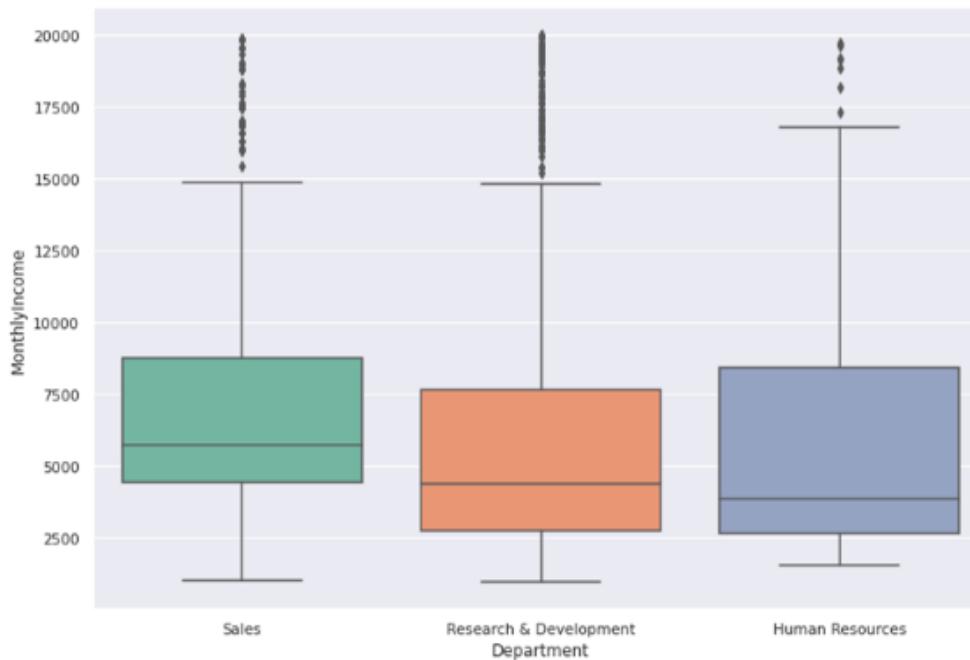
A histogram gives the distribution of data. used to visualize continuous data such as sales of a company segregated by month



# Box Plot

Task : Analyze the monthly income of employees in various departments such as sales, research & development, and human resource

Box plot example



Box Plot gives a statistical summary of one or more numerical variables

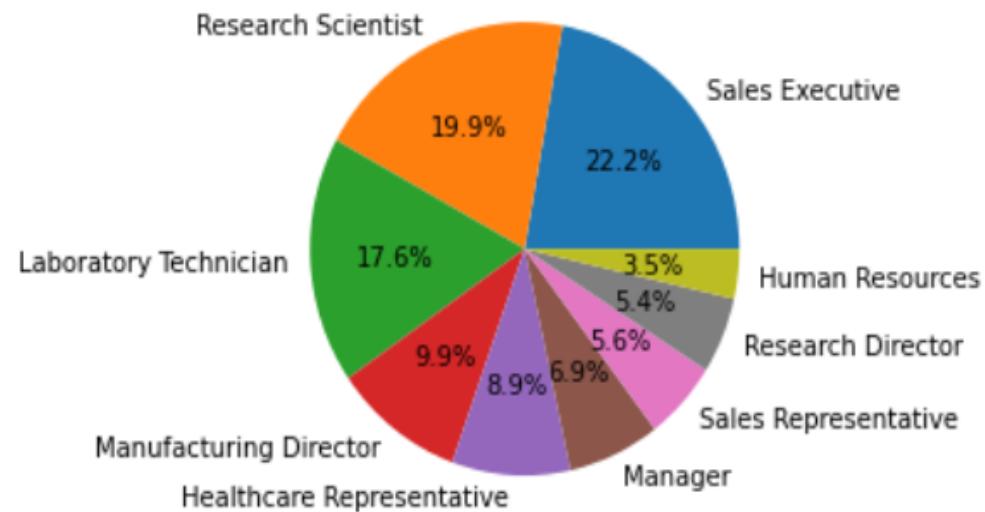


# Pie Chart

Task : What is the percentage of research scientist in IBM

Pie chart example

Percentage of Research Scientist



Pie charts are used to represent grouped or categorized data



# Visualizing Matplotlib Plots and Charts





# Matplotlib Plots and Charts



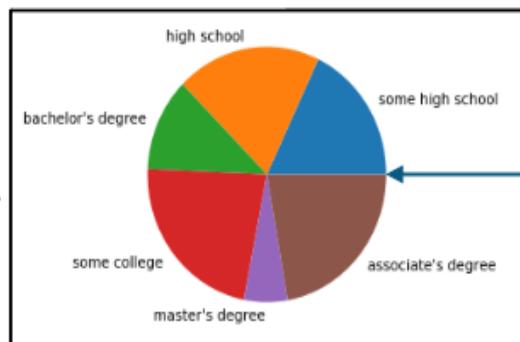
# Pie Chart

- Pie charts are used to represent a group of values as parts of a whole.
- Each value is represented by a slice of a pie.
- `pie()` function is used to plot pie charts which takes an array to represent slices.
- Pie charts are used to represent grouped or categorized data.

## Code example

```
q_label=list(set(df.Qualification))
q_count=[]
for qualification in q_label:
    q_count.append(len(df[df.Qualification==qualification]))
plt.pie(q_count,labels=q_label)
plt.show()
```

Code output



Pie charts starts plotting from 0°

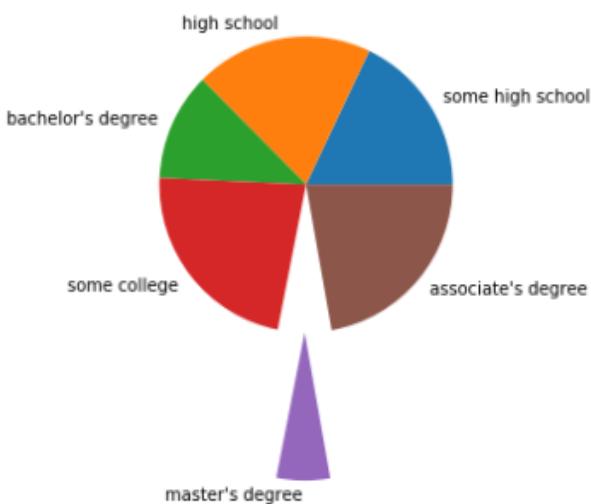


# Pie Chart (contd.)

- Using explode parameter, a slice can be separated from the pie chart.
- It takes an iterable or list of values for a slice to explode.

Code example and code output

```
min_qual=q_count.index(min(q_count))
q_explode=np.zeros(len(q_count))
q_explode[min_qual]=1
plt.pie(q_count,labels=q_label,explode=q_explode)
plt.show()
```



List should be equal to length of data, where 1 represents value to explode and all other points are zero.



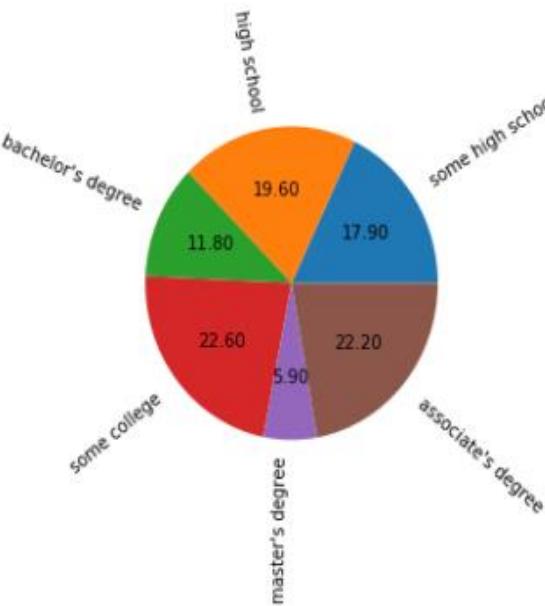
# Formatting in Pie Chart

---

- Parameter `rotatelabels` is used to rotate labels with respect to the pie.
- Parameter `autopct` is used to show percentage occupied by the pie.

Code example and code output

```
plt.pie(q_count,labels=q_label,rotatelabels=True,autopct='%0.2f')
plt.show()
```

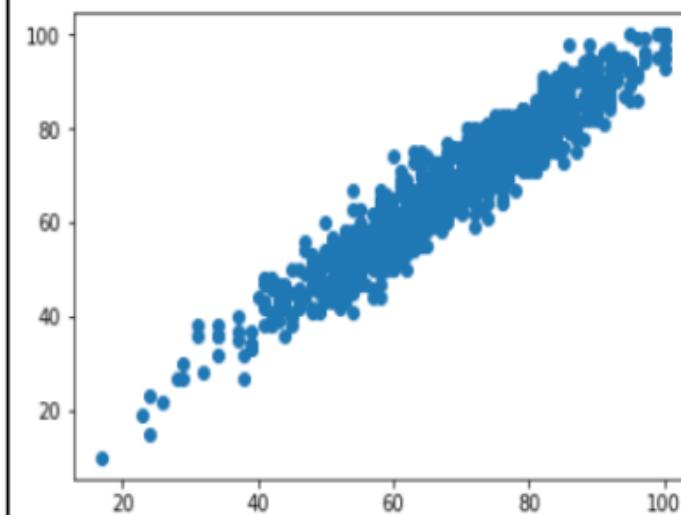


# Scatter Plot

- It represents data as collection of points.
- Order of data in scatter plot is insignificant.
- The plot is also known as correlation plot as it shows relationship between 2 variables.

Code example (scatter plot)

```
plt.scatter(df.Reading_score,df.Writing_score)  
plt.show()
```



Students who get good marks in **Reading** section  
are likely to get good marks in **Writing** section

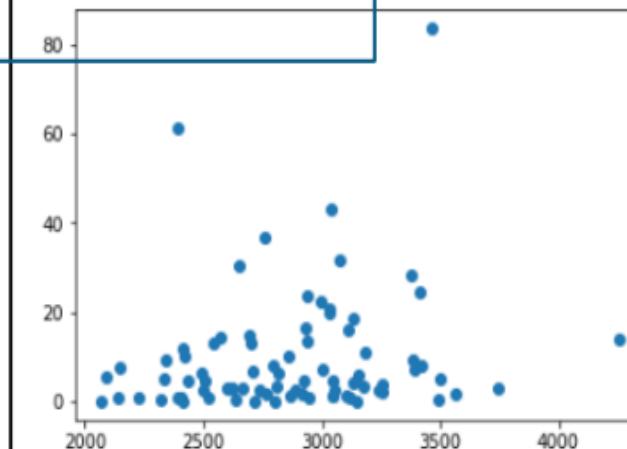


# Scatter Plot (contd.)

Code example (scatter plot)

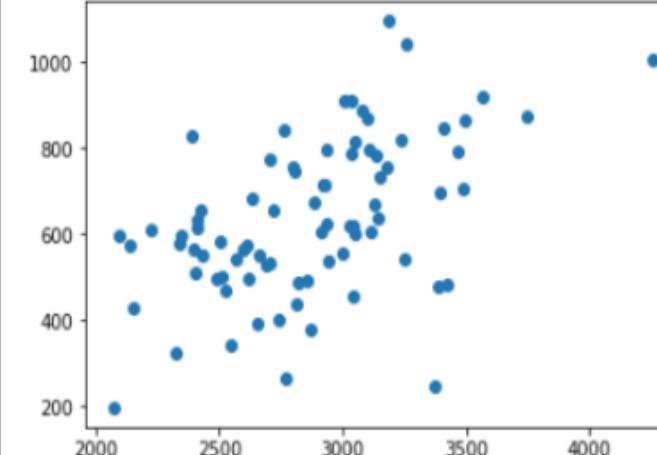
```
plt.scatter(rainfall.ANUAL, rainfall.JAN)  
plt.show()
```

Takes two iterable  
arguments



Code example (scatter plot)

```
plt.scatter(rainfall.ANUAL, rainfall.JUN)  
plt.show()
```

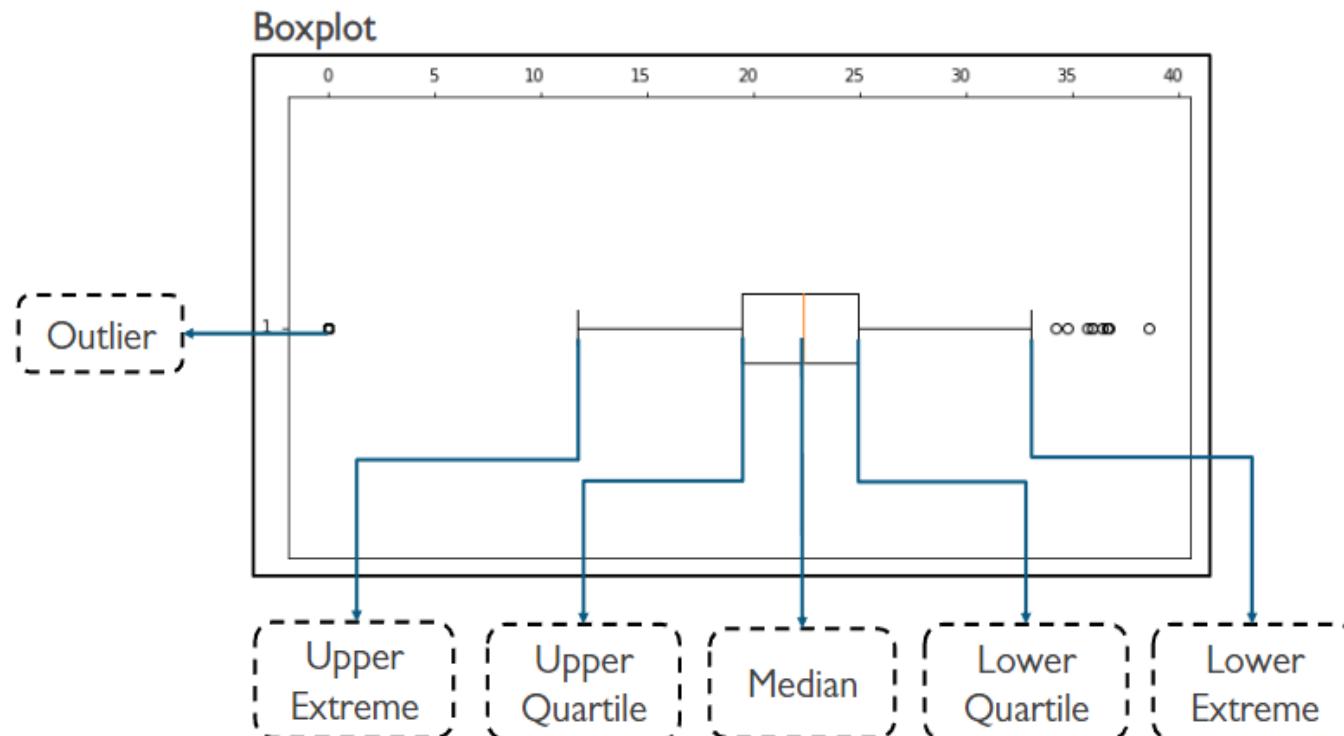


June is more correlated with annual rainfall



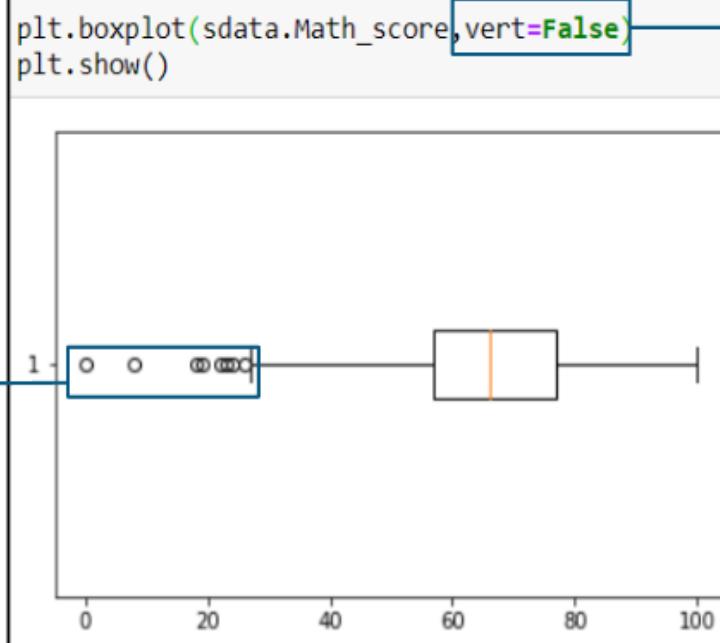
# Boxplot

- It used to display data distribution through quartiles.
- Quartiles are four equal groups where the data is distributed.
- Median separates low half and upper half.



# Boxplot (contd.)

Code example (boxplot)



Outliers indicate that very few students got low marks

If `vert` is `True`, then the plot will be vertical

Boxplot represents the score in marks of students in math section

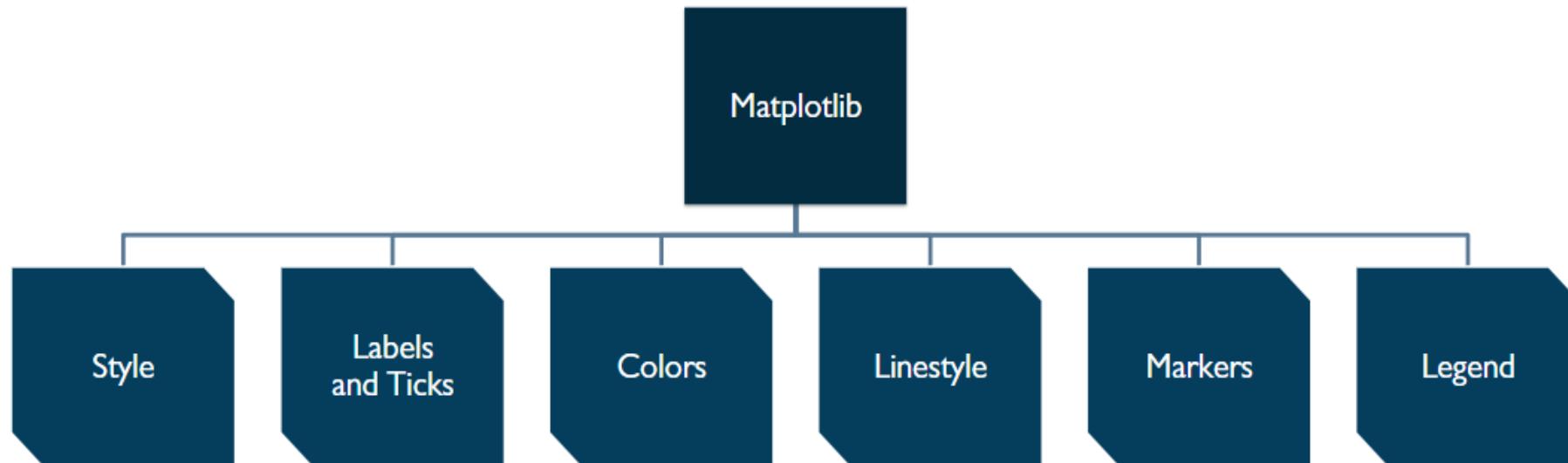
# Customizing Visualizations and Saving Plots



# Introduction to Customizing Visualizations

---

Matplotlib allows us to customize our plot in following ways:

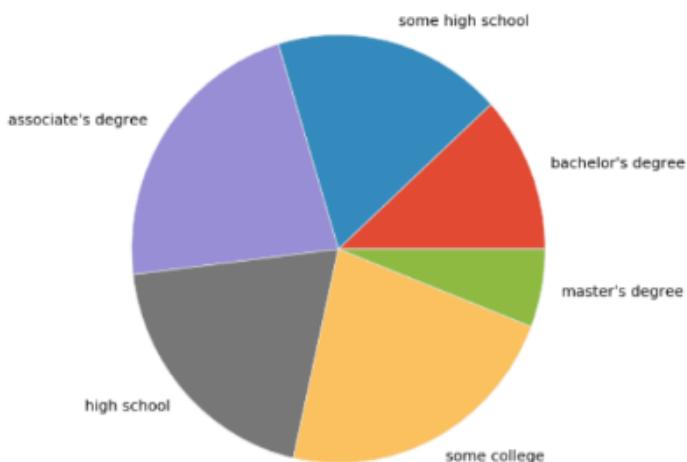


# Style

Matplotlib provides various types of styles for charts.

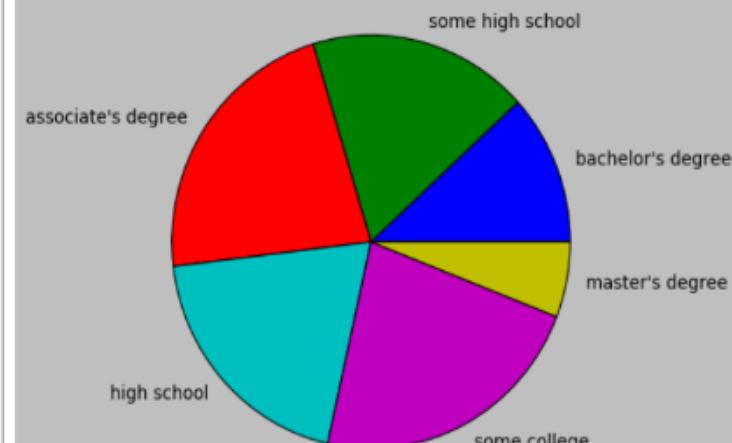
Code example (Pie Chart)

```
plt.style.use('ggplot')
plt.pie(Qualification_Count,labels=Qualification)
plt.show()
```



Code example (Pie Chart)

```
plt.style.use('classic')
plt.pie(Qualification_Count,labels=Qualification)
plt.show()
```



Use `plt.style.available` to see the available styles



# Labels and Ticks

---

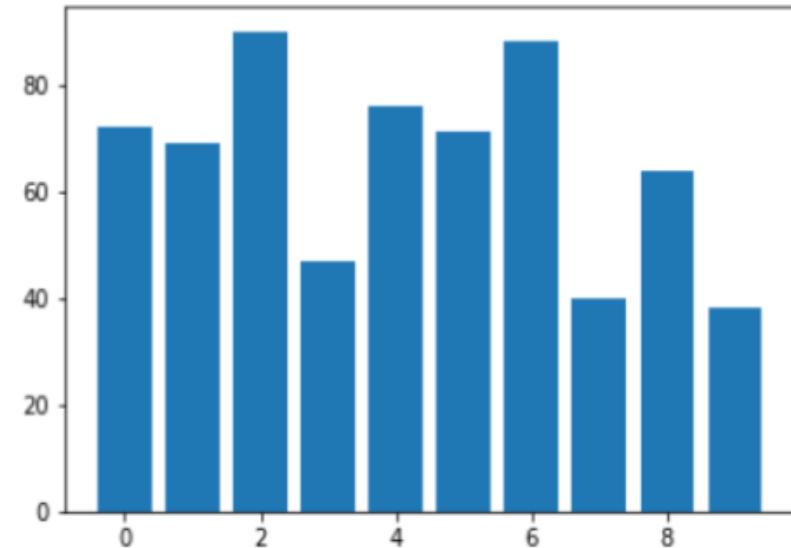
- ↳ Titles and axis labels can be created using `xlabel()`, `ylabel()`, and `title()` functions which takes a string parameter
- ↳ Ticks are pointers to values on axes.
- ↳ Ticks are set by default, but they can be changed.

Code example

```
1 plt.bar(sdata.index,sdata.Math_score)  
2 plt.show()
```



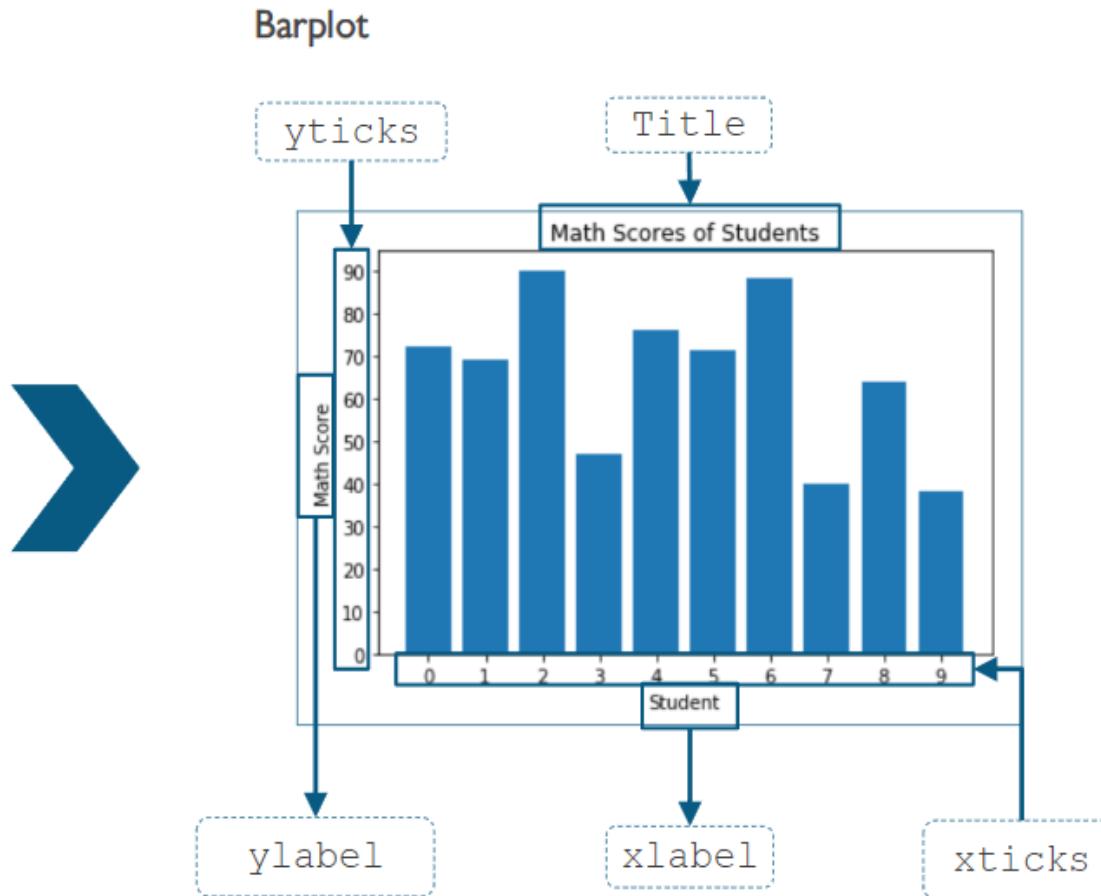
Barplot



# Labels and Ticks (contd.)

Code example

```
1 plt.bar(sdata.index,sdata.Math_score)
2 plt.ylabel('Math Score')
3 plt.xlabel('Student')
4 plt.title('Math Scores of Students')
5 plt.xticks(sdata.index)
6 plt.yticks(np.arange(0,100,10))
7 plt.show()
```



# Colors

Matplotlib allows to choose custom colors for plots.

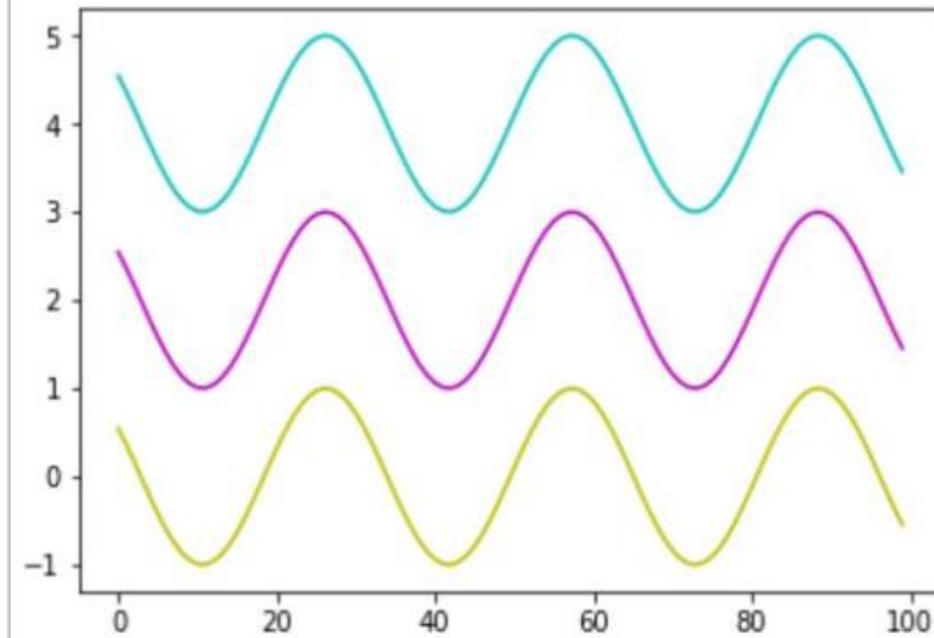
Code example

```
1 x=np.linspace(-10,10,100)
2 y = np.sin(x)
3 plt.plot(y, 'y')
4 plt.plot(y+2, 'm')
5 plt.plot(y+4, 'c')
6 plt.show()
```



Specifying line  
colors

Plot



# Color Codes

---

Color code	Color
b	Blue
c	Cyan
g	Green
k	Black
m	Magenta
r	Red
w	White
y	Yellow



# Linestyle

---

Matplotlib allows different linestyles for plots

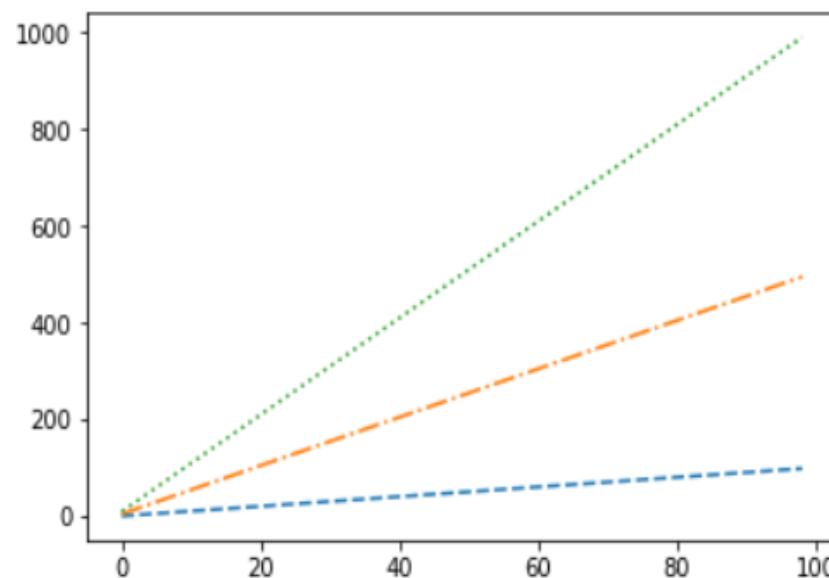
## Code example

```
1 y = np.arange(1, 100)
2 plt.plot(y, y*5, y*10, y*15)
3 plt.show()
```

Specifying linestyling



Plot

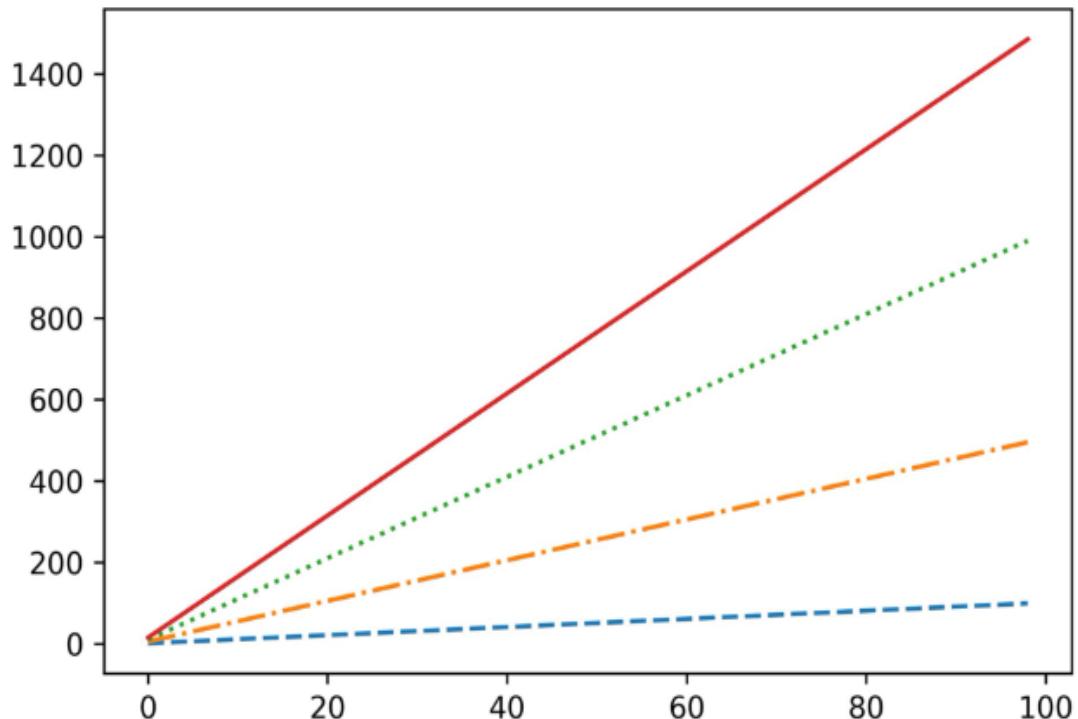


# Labels and Ticks

---

Style	Style Name
-	Solid line
--	Dashed line
-.	Dash-Dot line
:	Dotted line

Plot



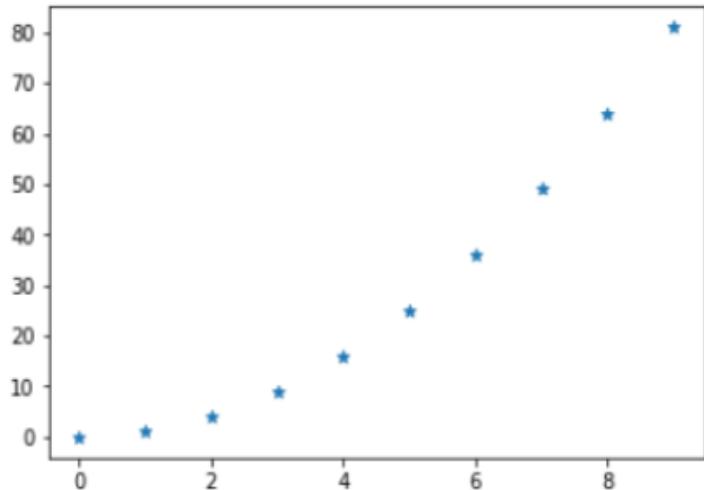
# Markers

---

- Markers are used to indicate datapoints.
- Parameter helps to define markers in Matplotlib.

Code example and plot

```
x = np.arange(0,10,1)
y = x*x
plt.scatter(x,y,marker='*')
plt.show()
```



'\*' is used to display five-point star markers.



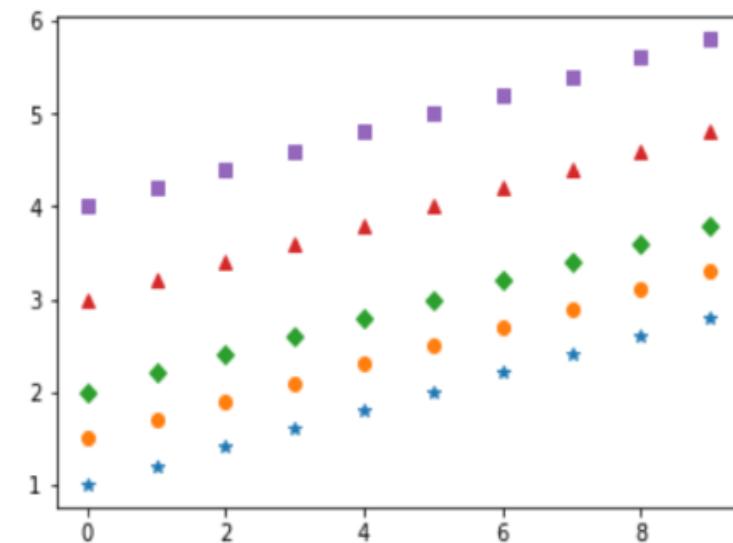
# Types of Markers

Different types of markers in matplotlib

Code example

```
1 | y = np.arange(1, 3, 0.2)
2 | plt.plot(y, '*', y+0.5, 'o', y+1, 'D', y+2, '^', y+3, 's')
3 | plt.show()
```

Plot



# Legend

Legends explain the meaning of each line in the graph.

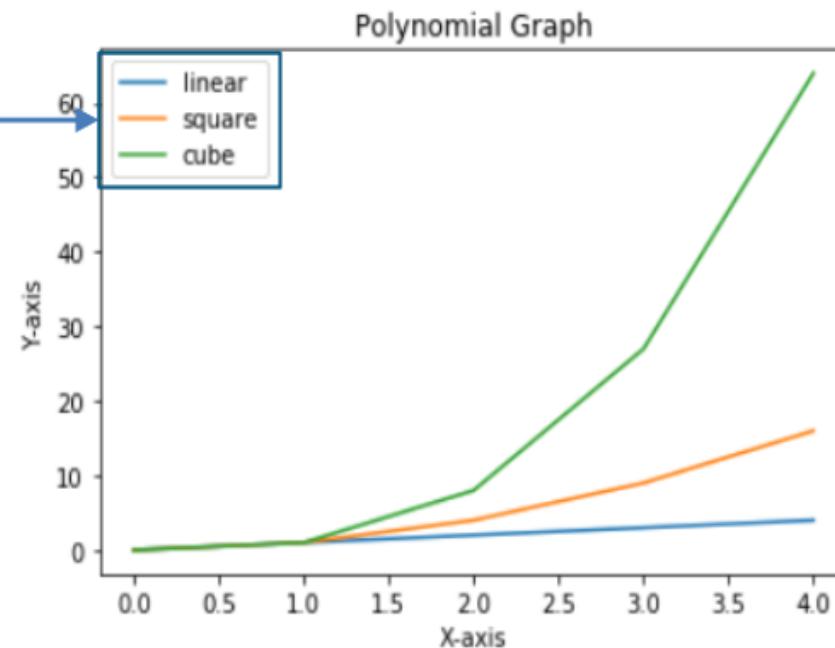
## Code example

```
1 x = np.arange(5)
2 plt.plot(x, x, label='linear')
3 plt.plot(x, x*x, label='square')
4 plt.plot(x, x*x*x, label='cube')
5 plt.xlabel('X-axis')
6 plt.ylabel('Y-axis')
7 plt.title('Polynomial Graph')
8 plt.legend()
9 plt.show()
```

`legend()` displays the legend on the plot.



Plot with a title and legend



## Legend (contd.)

---

- Legend function can also be used to define legend for the plot.
- Parameter loc defines the location of the legend.
- Consider student's data set and rainfall\_in\_kerala data set as shown below:

	Group	Qualification	Math_score	Reading_score	Writing_score
0	group B	bachelor's degree	72	72	74
1	group C	some college	69	90	88
2	group B	master's degree	90	95	93
3	group A	associate's degree	47	57	44
4	group C	some college	76	78	75
5	group B	associate's degree	71	83	78
6	group B	some college	88	95	92
7	group B	some college	40	43	39
8	group D	high school	64	64	67
9	group B	high school	38	60	50



## Legend (contd.)

---

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL
1996	2.8	9.1	14.4	124.3	74.3	572.4	696.0	327.4	342.7	294.1	89.9	62.5	2610.0
1997	2.1	1.5	36.1	60.6	133.6	544.2	970.5	536.0	292.2	288.9	298.4	88.4	3252.4
1998	6.0	2.1	8.1	61.1	151.6	732.5	641.4	371.8	517.6	444.8	135.0	79.4	3151.5
1999	1.8	23.8	21.4	111.6	453.2	607.3	700.4	266.3	88.0	567.9	68.1	4.9	2914.6
2000	11.7	57.8	21.5	96.3	124.5	633.8	343.2	566.5	195.8	214.2	78.1	69.1	2412.6
2001	16.5	28.3	7.0	238.0	238.6	715.3	598.5	361.3	216.8	319.6	181.0	10.1	2931.1
2002	4.7	8.7	35.7	117.3	330.8	503.1	318.7	438.2	99.0	511.7	137.5	2.1	2507.4
2003	0.7	50.9	82.1	134.4	91.0	566.7	532.0	350.3	93.6	407.0	76.4	9.7	2394.9
2004	2.4	8.1	37.9	113.2	610.9	673.4	385.4	417.9	192.8	320.6	120.7	2.7	2886.1
2005	10.2	7.0	25.3	205.0	134.8	610.2	832.7	201.0	414.7	240.1	184.3	56.4	3031.1

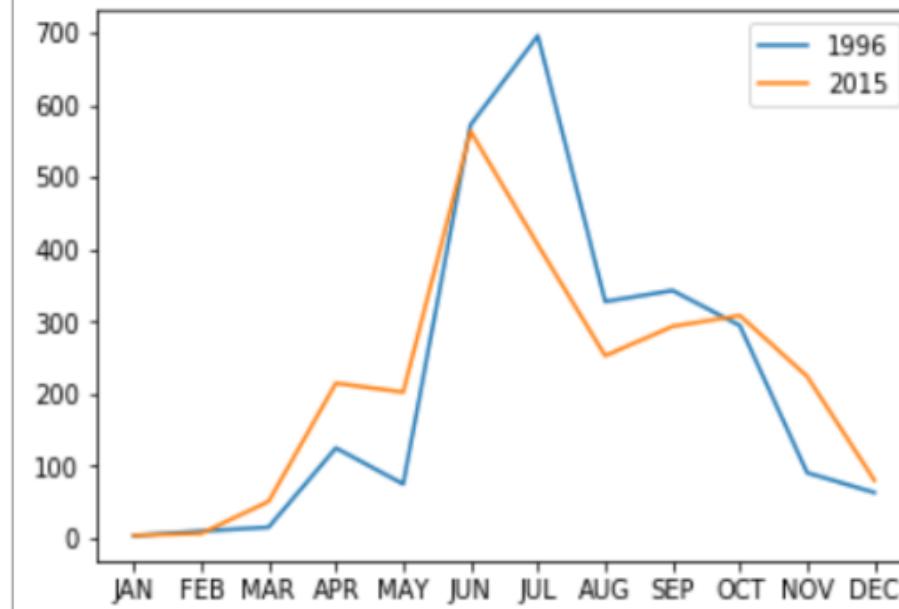


# Legend (contd.)

---

Code example and Line plot

```
plt.plot(rainfall.loc[1996,'JAN':'DEC'])
plt.plot(rainfall.loc[2015,'JAN':'DEC'])
plt.legend(['1996','2015'])
plt.show()
```





# Saving Plots



# Operation of Saving Plots

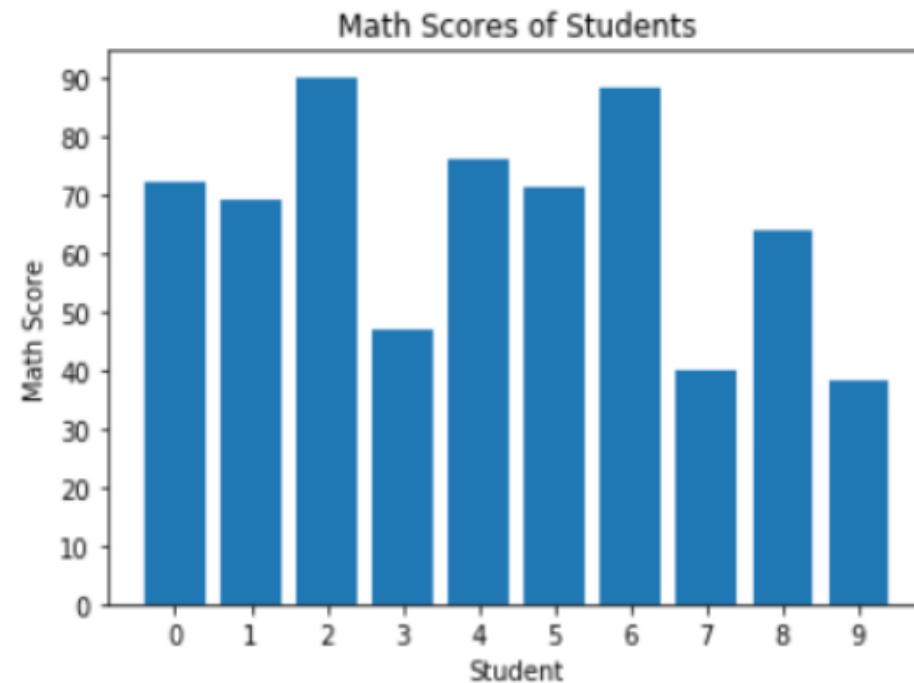
Plots can be saved using `savefig()` function.

Code example

```
1 plt.bar(sdata.index,sdata.Math_score)
2 plt.ylabel('Math Score')
3 plt.xlabel('Student')
4 plt.title('Math Scores of Students')
5 plt.xticks(sdata.index)
6 plt.yticks(np.arange(0,100,10))
7 plt.savefig('Math_Marks.png')
8 plt.show()
```

Saves an image named '`Math_Marks.png`' in the current directory

Bar chart





# Subplots



# Introduction to Subplots

---

- Function subplot() is used to plot multiple plots in the same figure.
- It uses two parameters to specify the number of rows and columns while the third parameter for index.
- Consider cereals data set:

		name	calories	sodium	potass	rating
0		100% Bran	70	130	280	68.402973
1		100% Natural Bran	120	15	135	33.983679
2		All-Bran	70	260	320	59.425505
3		All-Bran with Extra Fiber	50	140	330	93.704912
4		Almond Delight	110	200	-1	34.384843
5		Apple Cinnamon Cheerios	110	180	70	29.509541
6		Apple Jacks	110	125	30	33.174094
7		Basic 4	130	210	100	37.038562
8		Bran Chex	90	200	125	49.120253
9		Bran Flakes	90	210	190	53.313813

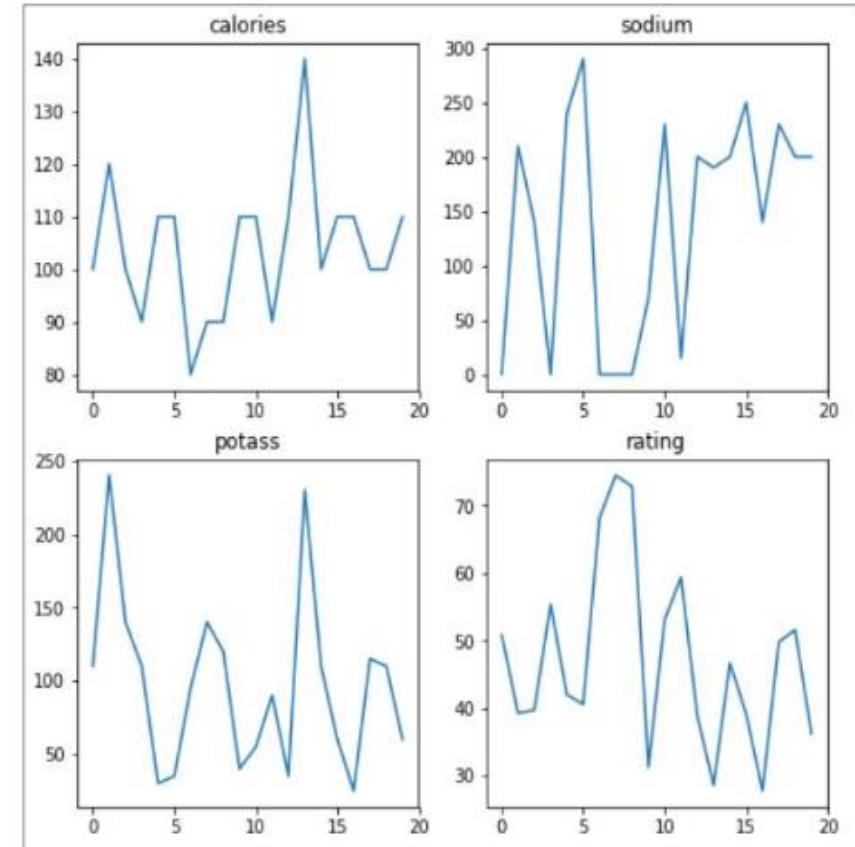


# Operation of Subplots

Code example

```
1 index=1 #index starts from 1 for subplot
2 → plt.figure(figsize=(8,8))
3 for col in cereal.columns[1:]:
4     plt.subplot(2,2,index)
5     plt.plot(cereal[col],)
6     plt.title(col)
7     index=index+1
```

To set figure size in form of tuple (width, height)



The above plot has 4 simple line plots of calories, sodium, potassium, and rating of cereals.



# Grid

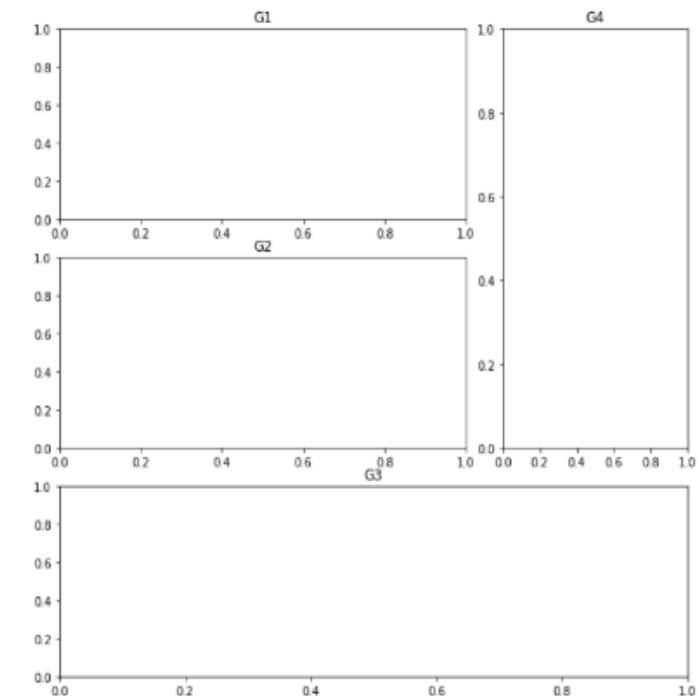
- With GridSpec() function, we can create complex grids.
- The subplots can be divided into different sizes and orientations.
- Each grid is defined using X and Y range/value separated by a comma.

## Code example

```
1 my_grid=plt.GridSpec(3,3)
2 my_fig=plt.figure(figsize=(10,10))
3 my_fig.add_subplot(my_grid[0,:2])
4 plt.title('G1')
5 my_fig.add_subplot(my_grid[1,:2])
6 plt.title('G2')
7 my_fig.add_subplot(my_grid[2,:])
8 plt.title('G3')
9 my_fig.add_subplot(my_grid[:2,2])
10 plt.title('G4')
11 plt.show()
```



Grid



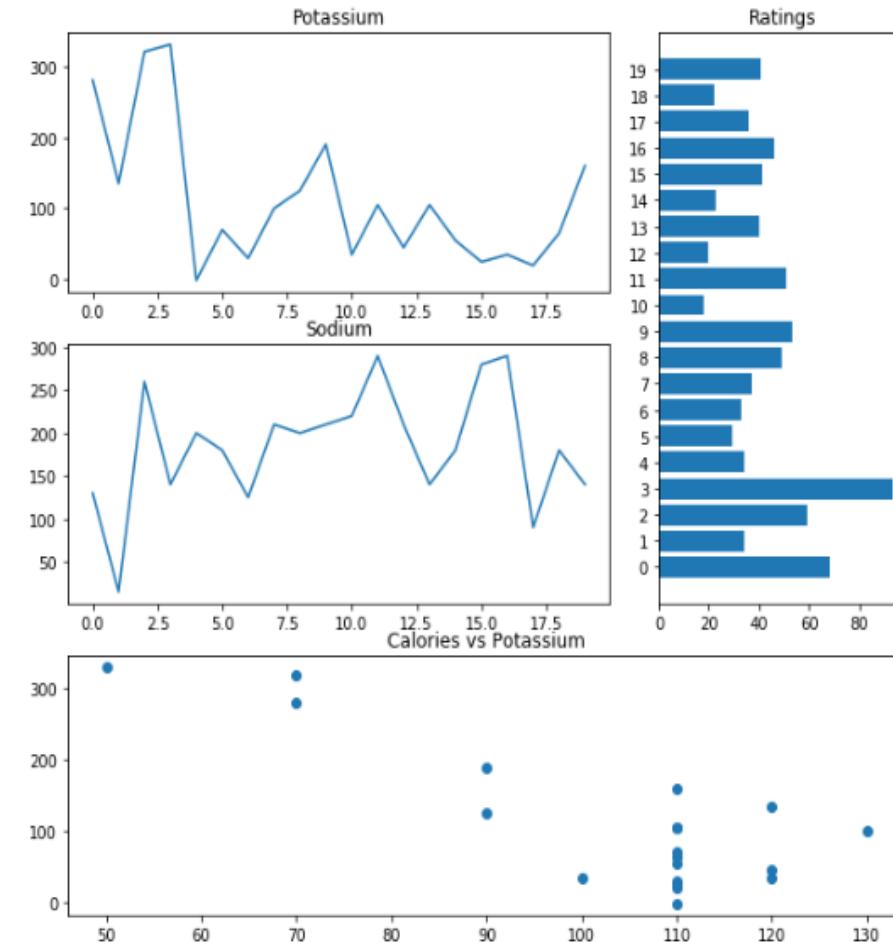
# Grid (contd.)

## Code example

```
1 my_grid=plt.GridSpec(3,3)
2 my_fig=plt.figure(figsize=(10,10))
3 my_fig.add_subplot(my_grid[0,:2])
4 plt.title('Potassium')
5 plt.plot(cereal.potass)
6 my_fig.add_subplot(my_grid[1,:2])
7 plt.title('Sodium')
8 plt.plot(cereal.sodium)
9 my_fig.add_subplot(my_grid[2,:])
10 plt.title('Calories vs Potassium')
11 plt.scatter(cereal.calories,cereal.potass)
12 my_fig.add_subplot(my_grid[:2,2])
13 plt.barh(cereal.index,cereal.rating)
14 plt.yticks(cereal.index)
15 plt.title('Ratings')
16 plt.show()
```



The plot has 4 different plots using GridSpec().



# Introduction to Web Scraping





# Basics of Web Scraping



# Understanding Web Scraping

---

- It is the process of extracting data from websites without human intervention.
- The construction of an agent to download, parse, and organize data from the web in an automated manner is web scraping.
- Web scraping is also known as web harvesting, web data extraction, or even web data mining.



# Use Cases of Web Scraping

---



Google translate utilises text stored on the web to train and improve itself.



hiQ sells employee data analyses by collecting and examining public profile information.



Tinder and Instagram Images are used to predict whether images will be popular or not.



Lyst scraped the web for information about fashion products to present a clean and elegant website.

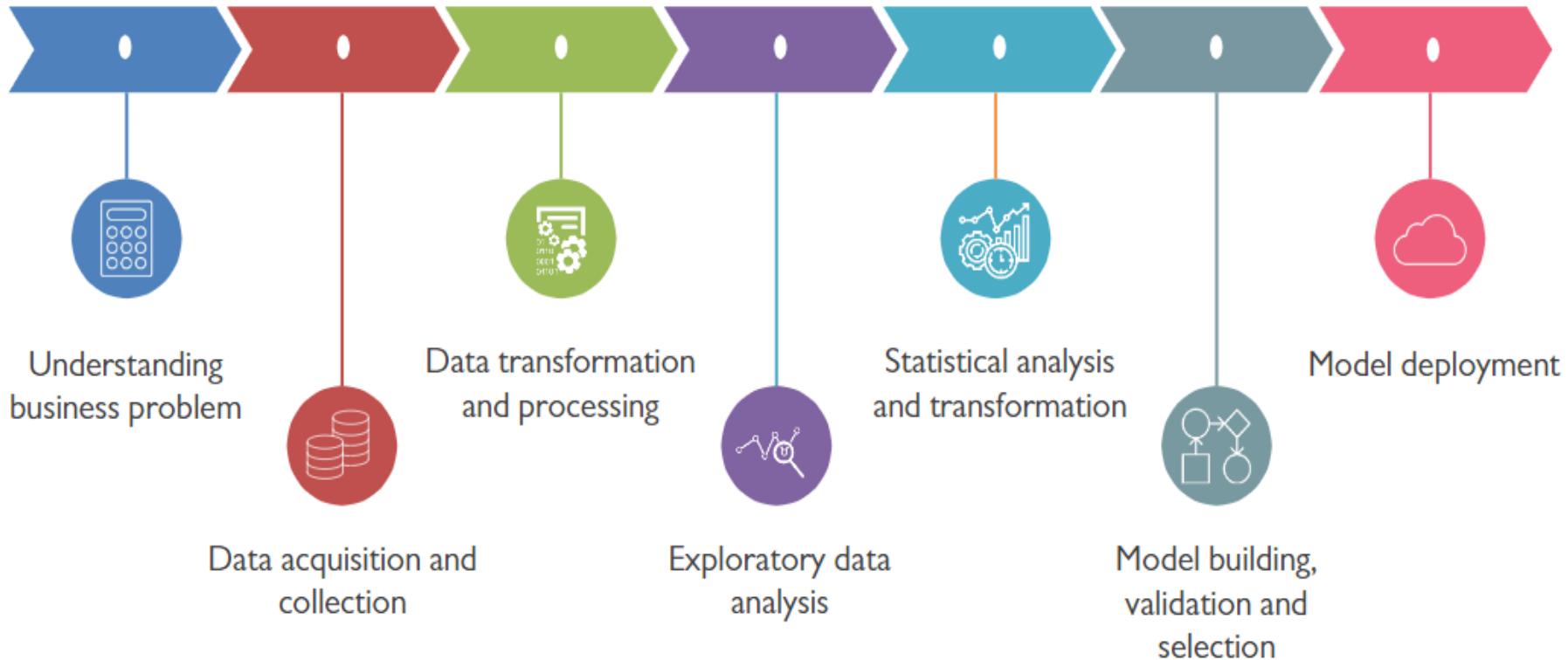


Emmanuel sales also scraped Twitter to make sense of his own social circle and timeline of posts.



# Need for Web Scraping in Data Science

---



Web scraping is part of data acquisition and collection where we collect data from various websites



# Need for Web Scraping in Data Science (contd.)

---



Essential step to gather data from various sources like websites, databases, and pdfs.



Utilize the unstructured data like comments using web scrapping

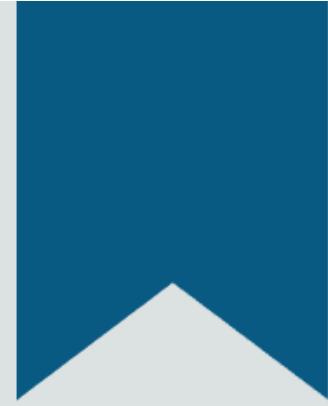


Incorporate additional features like local traffic, average temperature in your dataset



Use social network analytics of your company to enrich growth



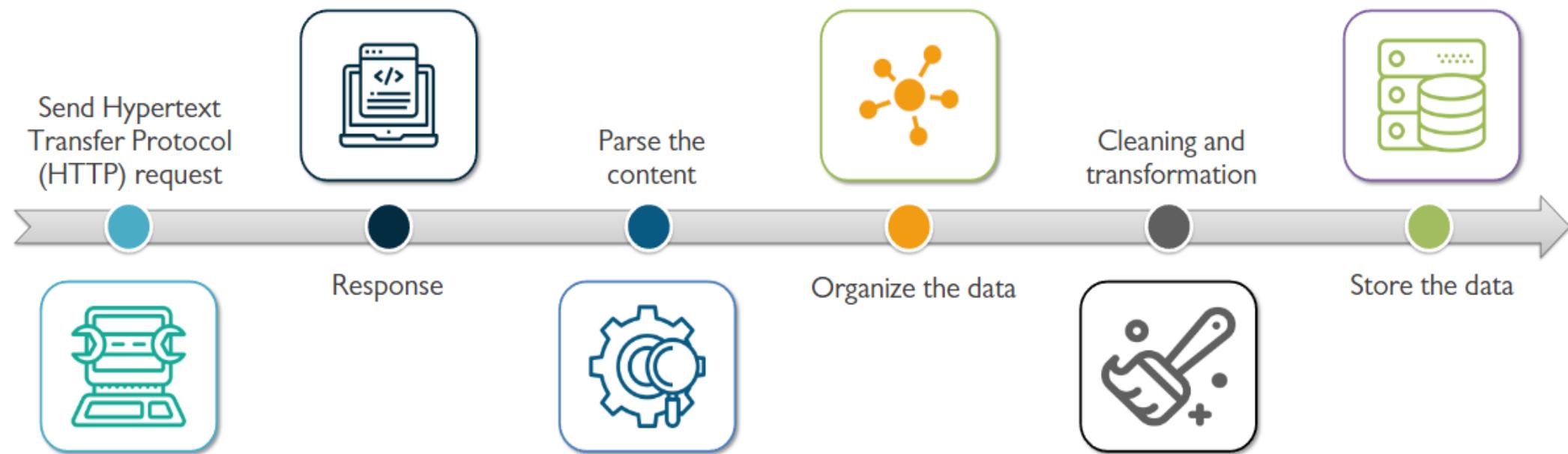


# Web Scraping Process Flow



# Illustration of Web Scraping Process Flow

---



# Send HTTP Requests



HTTP request is sent to URL of a webpage.

Client side system



Server



HTTP request  
<http://quotes.toscrape.com>



# Response



The server responds with HTML contents of the webpage.

## HTML code example

```
<!doctype html>
<html lang="en" class="gr__quotes_tosrape_com">
  <head>
    <meta charset="UTF-8">
    <title>Quotes to Scrape</title>
    <link rel="stylesheet" href="/static/bootstrap.min.css">
    <link rel="stylesheet" href="/static/main.css">
  </head>
  <body data-gr-c-s-loaded="true">
    <div class="container">
      <div class="row header-box">
        <div class="col-md-8">
          <h1>
            <a href="/" style="text-decoration: none">Quotes to Scrape</a>
          </h1>
        </div>
      </div>
    </div>
  </body>
</html>
```

## Server



Response

# Parsing Content



Collect the required data from the response content

Web listing

## Quotes to Scrape

"The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking."  
by [Albert Einstein](#) (about)

Tags: [change](#) [deep-thoughts](#) [thinking](#) [world](#)

"It is our choices, Harry, that show what we truly are, far more than our abilities."  
by [J.K. Rowling](#) (about)

Tags: [abilities](#) [choices](#)

"There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle."  
by [Albert Einstein](#) (about)

Tags: [inspirational](#) [life](#) [live](#) [miracle](#) [miracles](#)

Author  
Names

Extracted author names

Albert Einstein  
J.K. Rowling  
Albert Einstein  
Jane Austen  
Marilyn Monroe  
Albert Einstein  
André Gide  
Thomas A. Edison  
Eleanor Roosevelt  
Steve Martin

# Organize The Data

---



Organize the data into structured format

## Author Names

Author Names	Tags
Steve Martin	Humour, obvious, simile
André Gide Eleanor	Life, love
Thomas A. Edison	Failure, inspirational, paraphrased
Albert Einstein	Life change, deep-thoughts, thinking, world
J.K. Rowling	Abilities, choices
Albert Einstein	Inspirational, life, live, miracle, miracle
Jane Austen	Aliteracy, books, classic, humour
Marilyn Monroe	Be-yourself, inspirational
Albert Einstein	Adulthood, success, value



# Cleaning and Transformation



Eliminate unnecessary, repetitive, or missing data

## Author Names

Steve Martin

André Gide Eleanor

Thomas A. Edison

Albert Einstein

J.K. Rowling

Albert Einstein

Jane Austen

Marilyn Monroe

Albert Einstein

## Tags

Humour, obvious, simile

Life, love

Failure, inspirational, paraphrased

Life change, deep-thoughts, thinking, world

Abilities, choices

Inspirational, life, live, miracle, miracle

Aliteracy, books, classic, humour

Be-yourself, inspirational

Adulthood, success, value

Repetitive Author Name



# Storing Data



Storing data in the respective formats like Comma Separated Values (CSV), JavaScript Object Notation (JSON), and eXtensible Markup Language (XML)

## Author Names

Steve Martin

Humour, obvious, simile

André Gide Eleanor

Life, love

Thomas A. Edison

Failure, inspirational, paraphrased

Albert Einstein

Life change, deep-thoughts, thinking, world, adulthood, success, value,  
Inspirational, life, live, miracle

J.K. Rowling

Abilities, choices

Jane Austen

Aliteracy, books, classic, humour

Marilyn Monroe

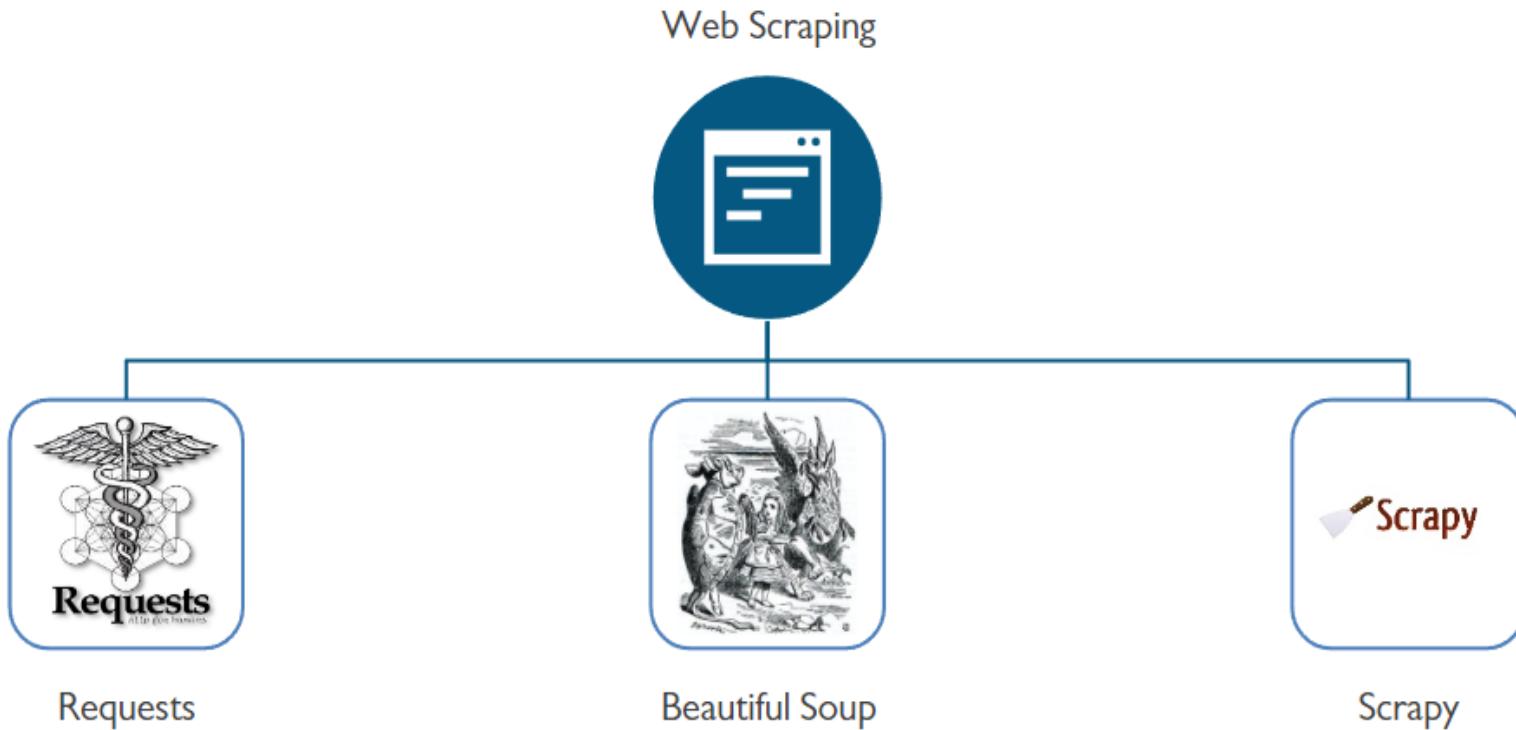
Be-yourself, inspirational

## Tags

Merged all the related tags to author

# Popular Tools for Web Scraping in Python

---



# Requests

- Requests is a Python library that is widely used for sending and receiving information over HTTP.
- Requests can be installed using the pip command in shell/bash.
- With requests, we can fetch the HTML content of web pages to create beautiful soup object.

## Code example

```
1 import requests  
2  
3 r=requests.get("https://www.free-ebooks.net/best-books")  
4 c=r.content  
5  
6 print(c)
```

```
b'<!DOCTYPE HTML>\n<html lang="en-US">\n<head>\n\t<!-- Go\n>\n\t<script>(function(w,d,s,l,i){w[1]=w[1]||[];w[1].push({\n
```

r is the requests object that contains the data received by get ()

URL of the webpage to be downloaded



# Web Scraping Using Beautiful Soup





# Beautiful Soup



# Introduction to Beautiful Soup

---

- ☞ A python library to extract data from Hypertext Markup Language (HTML) and eXtensible Markup Language (XML) files.
- ☞ It can work with malformed HTML documents.
- ☞ It creates a parse tree from parsed HTML documents to extract data.
- ☞ Beautiful soup cannot fetch pages on its own; it needs external library-like requests.



# Installation of Beautiful Soup

Beautiful soup can be installed using `pip` command in bash/shell.

## Code example

```
C:\Users\sumeet_kumar_jain>pip install bs4
Collecting bs4
  Downloading https://files.pythonhosted.org/packages/10/ed/7e8b97591f6f456174139ec089c76
  9f89a94a1a4025fe967691de971f314/bs4-0.0.1.tar.gz
Requirement already satisfied: beautifulsoup4 in c:\users\sumeet_kumar_jain\appdata\local
\programs\python\python37-32\lib\site-packages (from bs4) (4.8.1)
Requirement already satisfied: soupsieve>=1.2 in c:\users\sumeet_kumar_jain\appdata\local
\programs\python\python37-32\lib\site-packages (from beautifulsoup4->bs4) (1.9.5)
Installing collected packages: bs4
  Running setup.py install for bs4 ... done
Successfully installed bs4-0.0.1
```

# Creating Soup

- To extract data from HTML content, we create a beautiful soup object.
- The object can be created by passing the Uniform Resource Locator (URL) content.

Code example

```
1 from bs4 import BeautifulSoup  
2 import requests  
3 req=requests.get('https://www.wikipedia.org/')  
4 soup = BeautifulSoup(req.content,'html.parser')  
5 soup.prettify()
```

Beautiful soup method `prettify()` is used to convert beautiful soup tree in formatted unicode string.

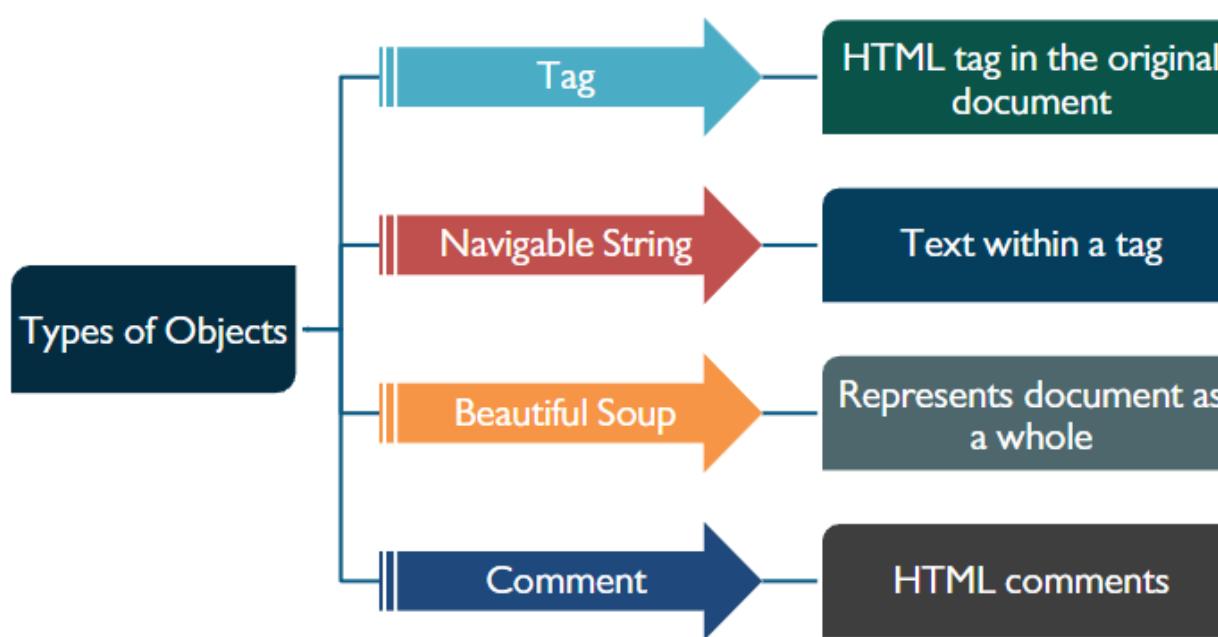
Code output

```
<!DOCTYPE html>  
<html class="no-js" lang="mul">  
<head>  
<meta charset="utf-8"/>  
<title>  
 Wikipedia  
</title>  
<meta content="Wikipedia is a free online encyclopedia, created and edited by volunteers around the world and hosted by the Wikimedia Foundation." name="description"/>  
<if gt IE 7?>  
<script>
```

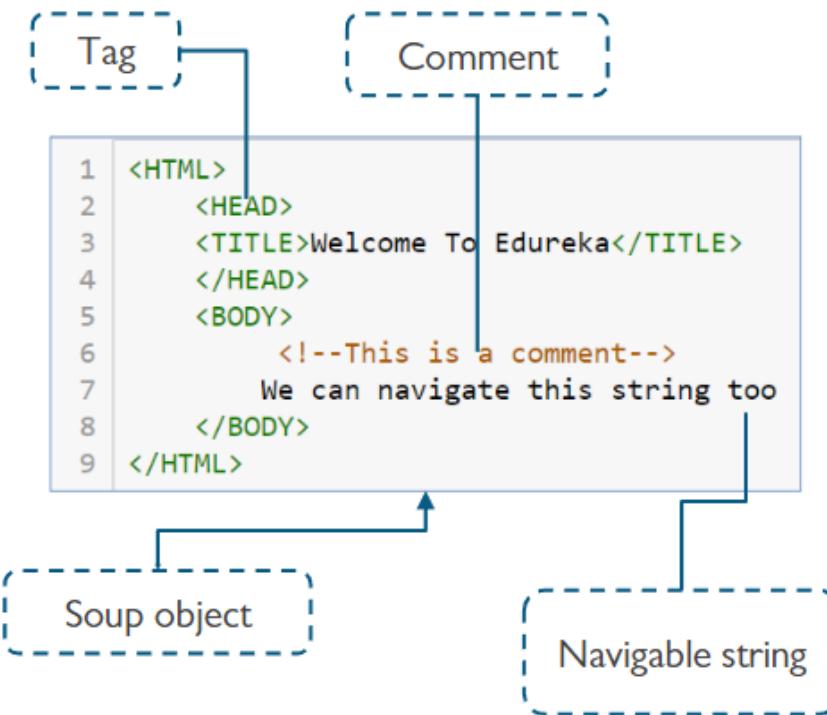
Beautiful soup object created using content from request and HTML parser library

# Types of Objects

Beautiful soup transforms a complex HTML document into a complex tree of python objects.



Code example



# Types of Objects (contd.)

Code example

```
1 tag=soup.title
2 print(type(tag))
3 print(type(tag.string))
4 print(type(soup))
5
6 mysoup=BeautifulSoup('<b><!-- Hey! edureka...--></b>')
7 comment=mysoup.b.string
8 print(type(comment))
```

Code output

```
<class 'bs4.element.Tag'>
<class 'bs4.element.NavigableString'>
<class 'bs4.BeautifulSoup'>
<class 'bs4.element.Comment'>
```





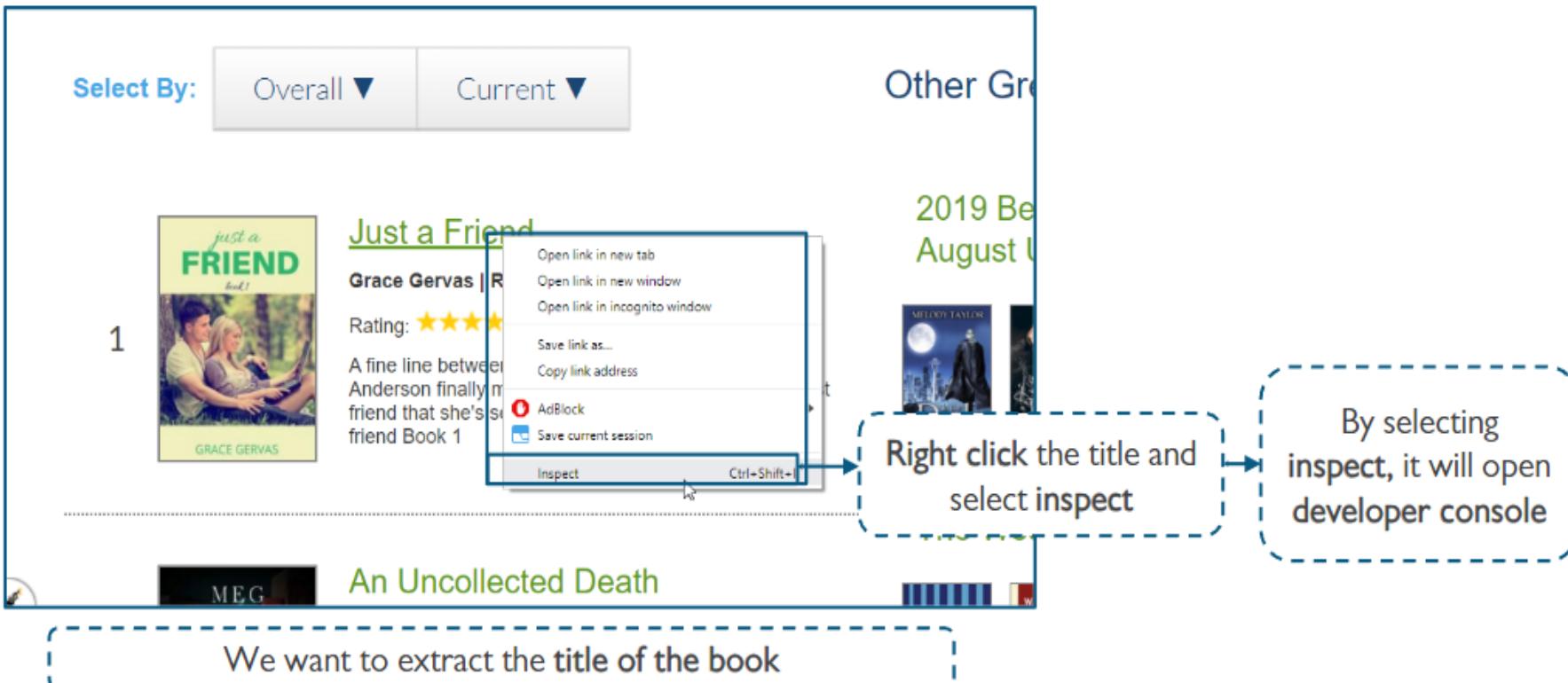
# Inspecting a Web Page



# Scrutinizing Webpage

- Before filtering content, we need to inspect our webpage.
- Hover over the element you want to extract from the website.

## Inspecting elements of a webpage



# Scrutinizing Webpage (contd.)

- As the developer console starts, it will highlight the selected element's HTML code.
- If you hover over the highlighted tag, it will highlight elements on the webpage.

Inspecting elements of a webpage

The screenshot shows a webpage with a list of books. A specific book entry for "Just a Friend" by Grace Gervas is highlighted with a blue dashed box. An arrow points from this highlighted area to the developer console on the right, which is also highlighted with a blue dashed box. The developer console displays the HTML code for the selected element, specifically the [Just a Friend](#) link. The code includes attributes like href, class, and title. Below the developer console, another blue dashed box highlights the "Developer console" tab.

Highlighted title of book

Overall ▾

1 Just a Friend  
Grace Gervas | Romance

Rating: ★★★★☆

A fine line between love and friendship. Would Hanna Anderson finally manage to capture the heart of her best friend that she's secretly in love with? Find out in Just A friend Book

An Uncollected Death

MEG WOLFE

Developer console

```
<div class="col-sm-16">
  <div class="mr">...</div>
  <div class="row mt20">
    ::before
    <div class="col-sm-24 ptb10">
      <span class="num">1</span>
      
      <h4 class="tlc">
        <a href="/romance/Just-a-Friend" class="title">Just a Friend</a> == $0
      </h4>
      <p>...</p>
      <p>...</p>
      <p class="book-description">...</p>
    </div>
  </div>
</div>
```

Elements Console Sources

Styles Computed Event Listeners DOM Breakpoints

Ancestors All Framework listeners

Console

# Scrutinizing Webpage (contd.)

In this case, title comes under a tag where the class is title.

Inspecting elements of a webpage

The screenshot shows a webpage listing a book titled "Just a Friend" by Grace Gervas. The book cover features a couple sitting outdoors. The listing includes the title, author, genre (Romance), and a 5-star rating. A callout box highlights the title "Just a Friend" with the class "a.title". To the right, a browser's developer tools are open, displaying the HTML structure. An 

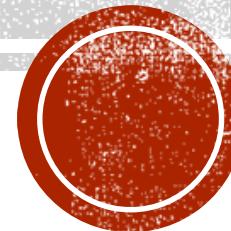
#### element with the class "tlc" contains the title, which is also wrapped in an anchor tag with the href "/romance/Just-a-Friend" and the class "title". The entire listing is enclosed in a section element.

```
alt="Just a Friend"
▼ <h4 class="tlc">
...
<a href="/romance/Just-a-Friend" class="title">Just a
Friend</a> == $0
</h4>
▶ <br> <br>
...
section div b div div div h4 a.title
```

# NUMPY CASE STUDY



Week 2 Mini project





## Case Study – Movie Rating System



# NumPy Case Study – Movie Rating System

**Problem Statement:** A movie review website needs to create a database to store their data and perform operations on it.

Create a dummy model with 100 users and 1000 movies to explain how it will work.

Topics Covered:

- NumPy
  - NumPy ndarray
  - Statistical operations using NumPy
  - Mathematical operations using NumPy



Check out the steps in  
the demo file

Matrix 1

Movie ID	Movie 1301	Movie 1302	...	Movie 2299
User 1	-1	9	...	3
User 2	2	7	...	-1
User 100	-1	0	...	8

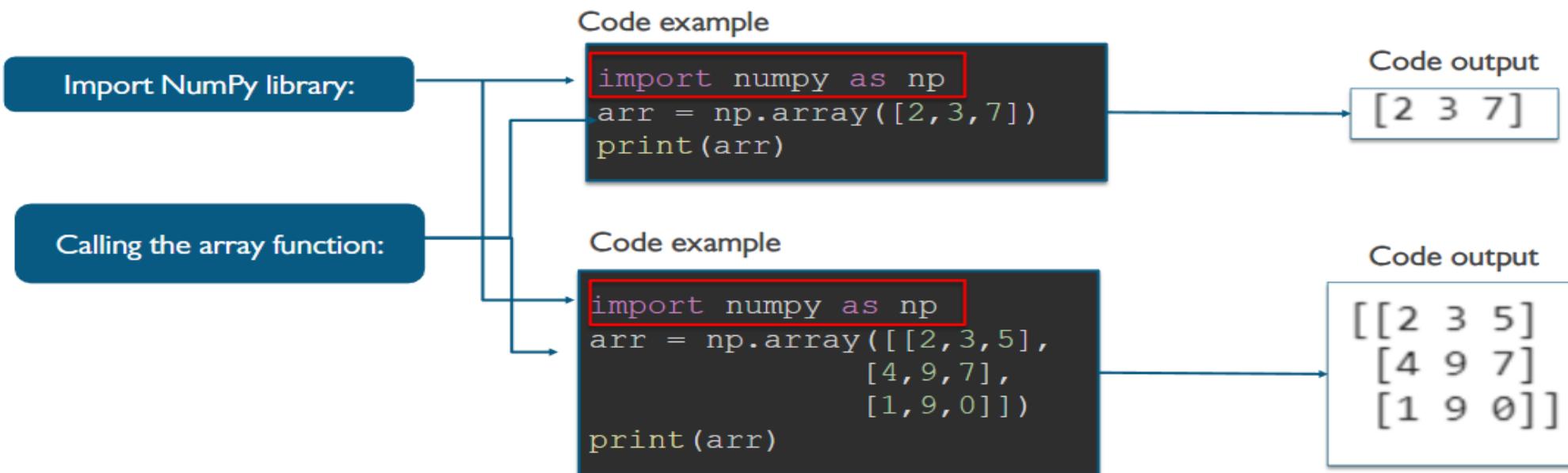
Matrix 2

Movie ID	Movie Rating	Number of ratings	Deviation of ratings
1301	6.47	72	2.39
1302	7.6	57	3.2



# How to Create NumPy Arrays?

Creating single dimension and multi-dimensional arrays:



**NOTE:** Check out more ways of creating arrays in the demo file



# Creating a NumPy Array

---

**Task 1:** Generate 1000 movie ids starting from 1301

Code example

```
movie_id = np.arange(1301,2301)
```

 **arange()** is another method of creating NumPy arrays that returns a range of specified values in an array

Code output

```
[1301 1302 1303 1304 1305 1306 1307 1308 1309 1310]  
1000
```



# Creating a NumPy Array (contd.)

## Task 2.1: Create a matrix to store user ratings

For 1000 movies, we will first create an array of size 1000 filled with values -1 (not rated).

Then, we will fill the movies index with the random ratings from a user.

Code example

```
movies_rated_by_me = np.full(1000, -1)
```

This method will create a list of ratings of 1000 movies by a user

Code example

```
movies_rated_by_me[index] = random.randint(0,10)
```

## Creating a NumPy Array (contd.)

**Task 2.2:** Create matrix `movies_matrix` such that each of the 100 users can review as many movies as they want

There are 1000 columns, each representing a movie, and there are 100 rows, each representing a user.

We will append the list of ratings by each user in a Python list and convert it to NumPy array.

Code examples

```
movie_matrix.append(movies_rated_by_me)
```

```
movie_matrix = np.array(movie_matrix)
```

Code output

```
[[[-1 -1 -1 ... -1 -1 -1]
 [ 1 -1  7 ...  0  2 -1]
 [-1 -1 -1 ...  1 -1 -1]
 ...
 [-1 -1 -1 ... -1  4 -1]
 [-1 -1 -1 ...  9 -1  8]
 [ 7 10  4 ...  0  6  3]]]
```



# Merging two NumPy Arrays

**Task 3:** We have 10 movie experts to provide their reviews. Also, 50 new movies have to be added with reviews

Code example

```
movie_matrix = np.vstack([movie_matrix, expert_matrix])
movie_matrix = np.hstack([movie_matrix, new_movies_matrix])

print(movie_matrix.shape)
```

Code output  
`(110, 1050)`

To add expert reviews, we are using `vstack()`, it adds arrays vertically.

We see that 10 more rows have been added.

To add new movies reviews, we are using `hstack()`, it adds arrays horizontally.

We see that 50 more columns have been added.



# Indexing and Slicing a NumPy Array

Indexing and slicing are used to access a subset of the data. Indexing in NumPy is identical to Python's Indexing scheme

Code example

```
a = np.arange(1,10)
print('Given Array:',a)
print('Indexed Element:',a[-1])
```

Code output

```
Given Array: [1 2 3 4 5 6 7 8 9]
Indexed Element: 9
```

Code example

```
a = np.arange(1,10)
print('Given Array:',a)
print('Indexed Element:',a[5])
```

Code output

```
Given Array: [1 2 3 4 5 6 7 8 9]
Indexed Element: 6
```

NOTE: Indexing in Python starts from 0



# Slicing a NumPy Array

**Task 4:** Create, `final_movie_rating` matrix with 4 columns i.e., 'Movie ID', 'Average rating', ' Number of ratings', and 'Standard deviation of ratings'

Code example

```
m = movie_matrix[:, i]
```

We are taking all rows and ith column from matrix I. The ith column represents the ith movie and all the rows have their corresponding user ratings for that movie

To calculate average rating, we need to collect all the values except -1, then we will calculate mean, standard deviation, and the count of non-negative values

Code example

```
m = m[m>=0]
```



# Statistical Calculation using NumPy

**Task 4:** Create, `final_movie_rating` matrix with 4 columns i.e. 'Movie ID', 'Average rating', ' Number of ratings', and 'Standard deviation of ratings'

Code examples

```
rating = m.mean()
```

```
standard_deviation = m.std()
```

```
total_num_rating = m.size
```

Code output

```
[1301, 4.846153846153846, 52, 2.9960526298692995],  
[1302, 4.4666666666666667, 45, 2.729265265394496],  
[1303, 5.125, 40, 4.038486721533203],  
[1304, 4.28, 50, 3.2251511592481994],  
[1305, 6.314285714285714, 35, 2.8261172614578833],  
[1306, 3.702127659574468, 47, 2.3239263695171317],  
[1307, 4.930232558139535, 43, 3.4801453380316874],
```

# Mathematical Operation using NumPy

**Task 5:** Convert the final movie ratings to have range from 0 to 10, such that the minimum rating converts to 0 and maximum to 10 , and the other values in between

Code example

```
avg_movie_rating[:,1] = ((x - x.min()) * (new_range/old_range) + 0)
```

Here,  $x$  is an array, and is being subtracted with a scalar value, NumPy broadcasts the scalar value over array  $x$  and subtract it with all the elements in  $x$

# Sorting a NumPy Array

**Task 6:** Display the movies rating-wise, highest to lowest

Code output

```
[[1971.000 10.000 51.000 3.038]
 [2091.000 9.544 54.000 2.867]
 [1416.000 9.303 59.000 2.797]
 ...
 [1326.000 0.189 51.000 3.022]
 [2018.000 0.135 54.000 2.504]
 [1771.000 0.000 55.000 2.979]]
```



To sort an array, two NumPy methods, `sort()` and `argsort()` are used.

- ☞ `sort()` method returns the sorted array
- ☞ `argsort()` method returns the indices of the sorted array