# Basic Programming

Lesson 10. Analyzing dataframes. Pandas and Matplotlib Python libraries

Alexandra Ciobotaru

17 dec 2021

# Syllabus

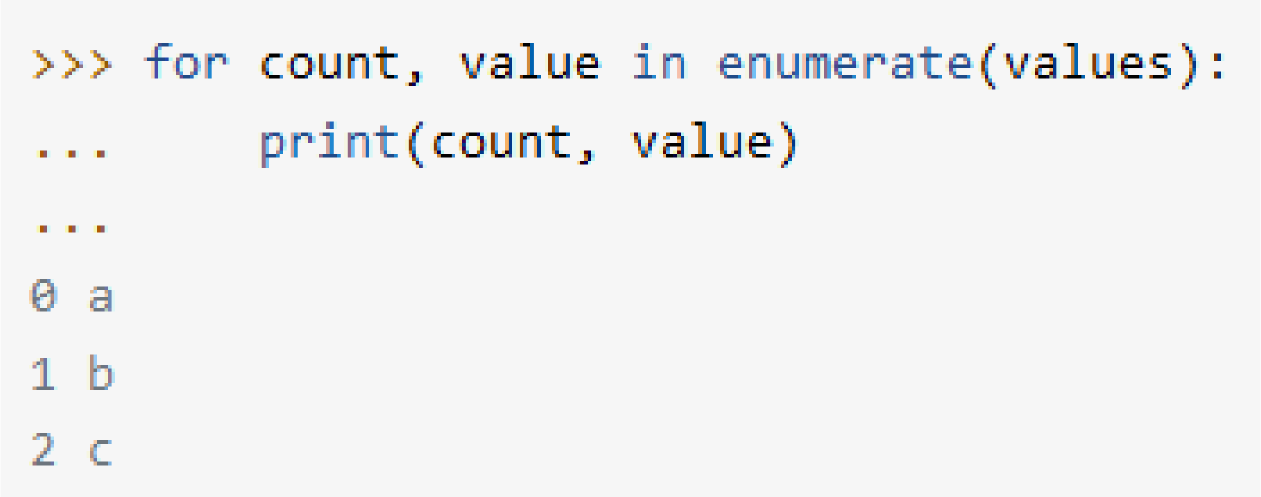
- **Lesson 1. Computers, Programming and Cognitive Science. From pseudocode to programming languages.**
- **Lesson 2. Variables in Python. Basic calculus. Using Math library, type() and help() functions.**
- **Lesson 3. Collections: Lists, Tuples.**
- **Lesson 4. Strings. Working with strings.**
- **Lesson 5. Branching and decisions: Logical operators, If-Statements, Nested conditions. Loops: For and While**
- **Lesson 6**. **Lesson 5 continued** Quiz 1 (25%).
- **Lesson 7. Creating functions. Recursive functions. Matrices**.
- **Lesson 8. Collections: Dictionaries. JSON construction.**
- **Lesson 9. Working with files. Reading and Writing.**
- **Lesson 10. Analyzing dataframes. Pandas and Matplotlib Python libraries.**
- Lesson 12. Object-Oriented Programming: Encapsulation, Inheritance and Polymorphism. Quiz 2 (25%).
- Lesson 13. Object-Oriented Programming (cont.). Error handling. Best practices when programming.
- Lab (20%) + final exam (30%).

# What you’ll learn today:

- 1. What is Pandas
  - 1.1. Pandas Series
  - 1.2. Creating a dataframe from series
  - 1.3. Add series to dataframe
- 2. Dataframes
  - 2.1. Creating dataframes
  - 2.2. Loading csv data into dataframes
  - 2.3. Selecting specific elements from dataframe
  - 2.4. Dataframe statistics
  - 2.5. Writing dataframe to csv
  - 2.6. Read and write to excel
- 3. Matplotlib library
  - 3.1. Bar plots
  - 3.2. Line plots
  - 3.3. Pie chart

# But first

…A small remaining from last courses: Python **enumerate()**



```
>>> for count, value in enumerate(values):
...     print(count, value)
...
0 a
1 b
2 c
```

When you use enumerate(), the function gives you back two loop variables:

- The **count** of the current iteration
- The **value** of the item at the current iteration

Of course, as always, you can name your variables how you wish.

# What is Pandas?

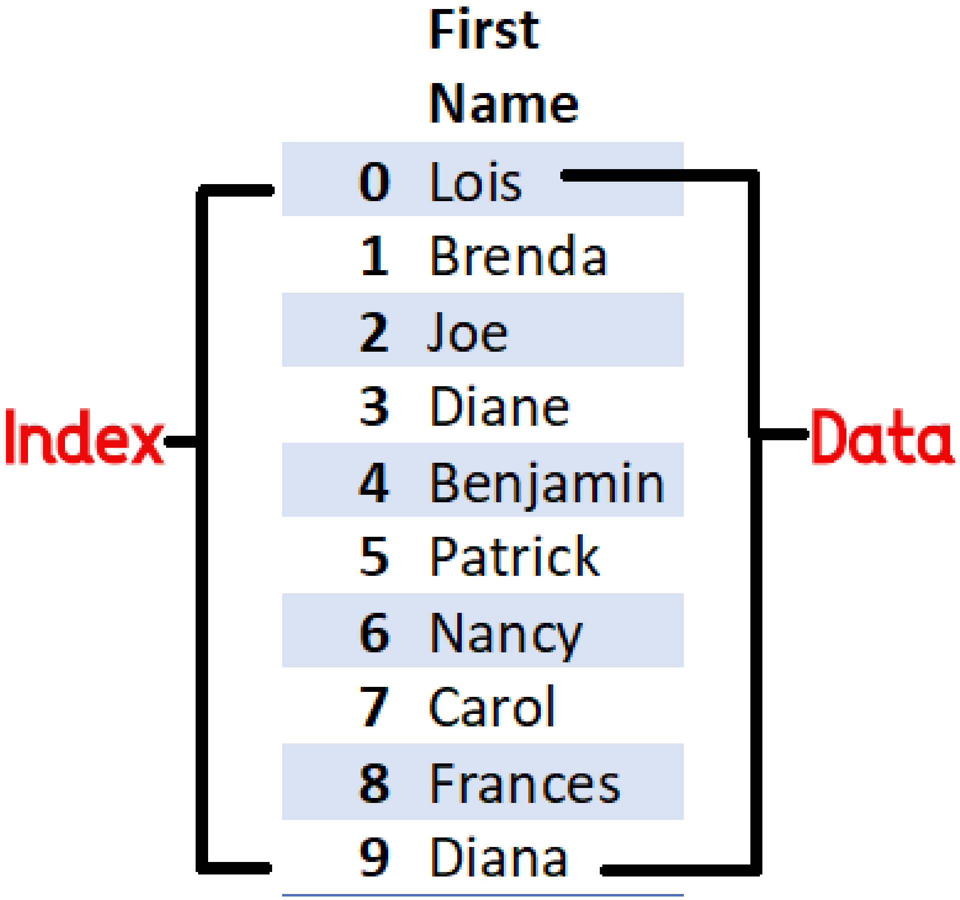
pandas

- A library for manipulating spreadsheets (tabular data)
- Useful for data preparation, data cleaning and analysis
- Installation: pip install pandas
- Importing: import pandas as pd

# Pandas Series

- One dimensional datastructures
- A pandas series returns an object in the form of a list, having index starting from 0 to n
- Creating a Series from a list  (by default, indexing is done from 0)

```
list_1 = ['a','b','c','d']
ser = pd.Series(list_1)
print(ser)
```



| | First Name |
|---|---|
| 0 | Lois |
| 1 | Brenda |
| 2 | Joe |
| 3 | Diane |
| 4 | Benjamin |
| 5 | Patrick |
| 6 | Nancy |
| 7 | Carol |
| 8 | Frances |
| 9 | Diana |

- Create Series with custom index

```
labels = [1,2,3,4]
ser_1 = pd.Series(data=list_1, index=labels)
```

- Create series from numpy array

```
arr_1 = np.array([1,2,3,4])
ser_2 = pd.Series(arr_1)
```

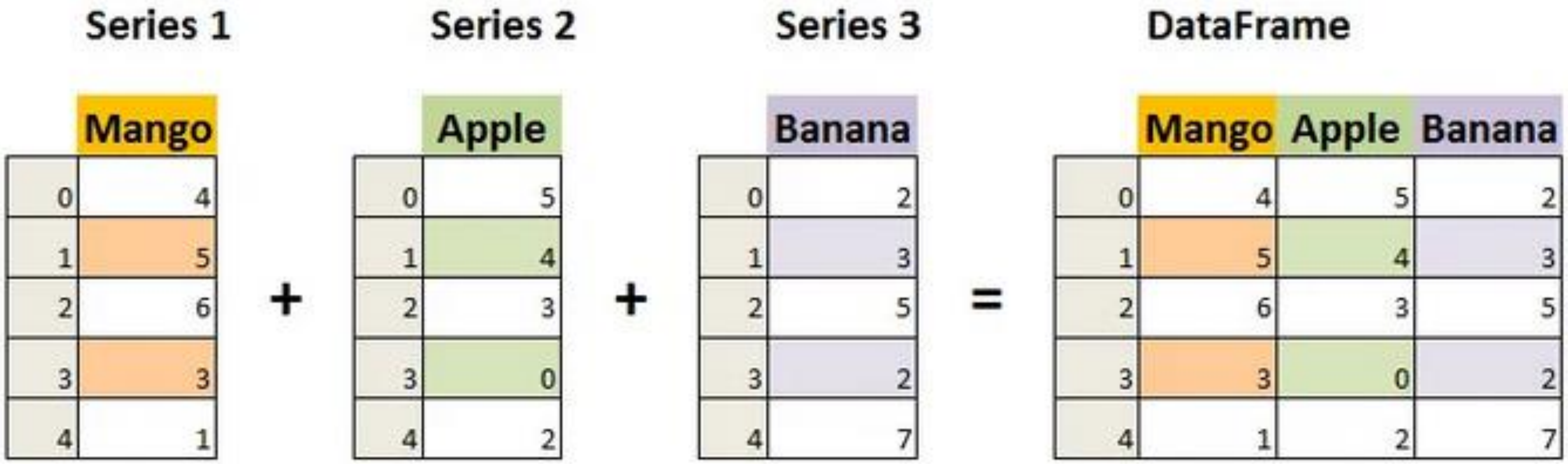
- Create series from dictionary

```
dict_1 = {'f_name':'Derek', 'l_name':'Barnabas', 
'age':44}
ser_3 = pd.Series(dict_1)
```

- Create series from scalar

```
s = pd.Series(7, index=[0, 1, 2, 3])
```

# Creating a dataframe from multiple series



Series 1

| | Mango |
|---|---|
| 0 | 4 |
| 1 | 5 |
| 2 | 6 |
| 3 | 3 |
| 4 | 1 |

+

Series 2

| | Apple |
|---|---|
| 0 | 5 |
| 1 | 4 |
| 2 | 3 |
| 3 | 0 |
| 4 | 2 |

+

Series 3

| | Banana |
|---|---|
| 0 | 2 |
| 1 | 3 |
| 2 | 5 |
| 3 | 2 |
| 4 | 7 |

=

DataFrame

| | Mango | Apple | Banana |
|---|---|---|---|
| 0 | 4 | 5 | 2 |
| 1 | 5 | 4 | 3 |
| 2 | 6 | 3 | 5 |
| 3 | 3 | 0 | 2 |
| 4 | 1 | 2 | 7 |

Image source: https://www.datasciencemadesimple.com/create-series-in-python-pandas/

# Creating a dataframe from multiple series

```
mango = [4, 5, 6, 3, 1]
apple = [4, 4, 3, 0, 2]
banana = [2, 3, 5, 2, 7]
mango_series = pd.Series(mango)
apple_series = pd.Series(apple)
banana_series = pd.Series(banana)
frame = {'Mango': mango_series, 'Apple': apple_series,
'Banana': banana_series}
result = pd.DataFrame(frame)
print(result)
```

# Add series to dataframe (add a column)

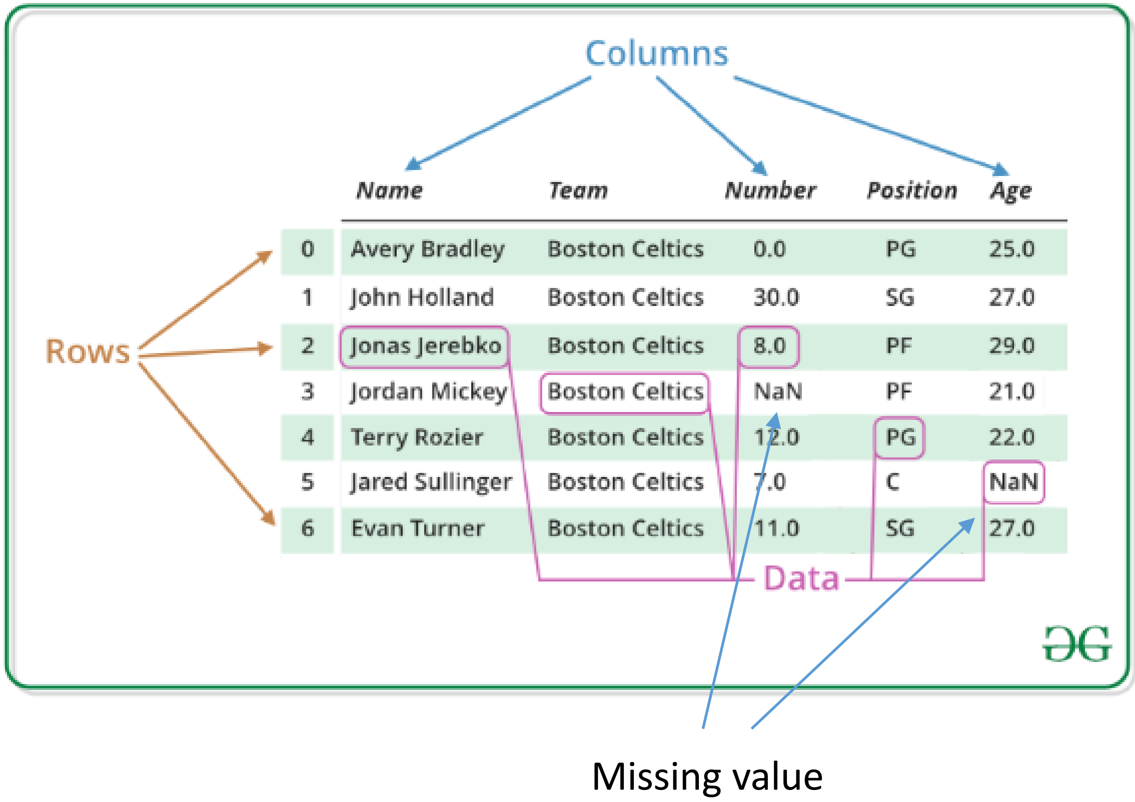
```
rotten = [0, 1, 0, 1, 0]
result['Rotten'] = rotten
print(result)
```

```
 Mango  Apple  Banana
0    4      4       2
1    5      4       3
2    6      3       5
3    3      0       2
4    1      2       7
```

```
 Mango  Apple  Banana  Rotten
0    4      4       2       0
1    5      4       3       1
2    6      3       5       0
3    3      0       2       1
4    1      2       7       0
```

# Dataframes

- **Pandas DataFrame** is two-dimensional size, mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).
- Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.
- We can perform many operations on these datasets like arithmetic operation, columns/rows selection, columns/rows addition etc.



| | *Name* | *Team* | *Number* | *Position* | *Age* |
|---|---|---|---|---|---|
| 0 | Avery Bradley | Boston Celtics | 0.0 | PG | 25.0 |
| 1 | John Holland | Boston Celtics | 30.0 | SG | 27.0 |
| 2 | Jonas Jerebko | Boston Celtics | 8.0 | PF | 29.0 |
| 3 | Jordan Mickey | Boston Celtics | NaN | PF | 21.0 |
| 4 | Terry Rozier | Boston Celtics | 12.0 | PG | 22.0 |
| 5 | Jared Sullinger | Boston Celtics | 7.0 | C | NaN |
| 6 | Evan Turner | Boston Celtics | 11.0 | SG | 27.0 |

# Creating a dataframe

- Creating an empty dataframe

```
df = pd.DataFrame()
```

- Creating a dataframe from a dictionary

```
# Let’s define a dictionary containing employee data
data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],
        'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}
# Convert the dictionary into DataFrame
df = pd.DataFrame(data)
print(df)
```

- You can see the first 5 rows of the dataset:

```
print(df.head())
```

- Values in dictionary must have the same length, otherwise the program will throw an error.
- Other methods that show basic info about dataframes:

```
df.shape -> (rows, columns)

df.index -> describes index

df.count() -> no of non-NA values
```

# Loading data from a csv into a dataframe

- CSV – Comma Separated files
- Loading csv data into a dataframe is done using **.read_csv** method:

```
data = pd.read_csv("nba.csv")
```

- You can also specify the column for indexing:

```
data = pd.read_csv("nba.csv", index_col ="Name")
```

- View all columns:

```
for col in data.columns:
    print(col)
```

- Delete rows with empty values:

```
data = data.dropna()
```

- **Exercise**: can you define a list ‘col’ that holds all column names using accumulation? Can you do it also using list comprehension?
- Another way of doing this is using column.values method:

```
print(list(data.columns.values))
```

# Selecting specific elements in dataframe

- You can select rows using **.loc** method on the index elements:

```
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]
print(first)
print(second)
```

**loc** -> selects [row_label, column_label]
**iloc** -> selects [row_position, column_position]

- Select element at specific row & column:

```
df = pd.read_csv('nba.csv', index_col='Name')
print(df.loc['Avery Bradley', 'Weight'])
```

- You can select elements by indexes using **.iloc** method:

```
print(df.iloc[2,6])
```

- You can also select by condition:

```
new_df = df[df["Team"] == "Boston Celtics"]
```

# Selecting specific columns in csv

- We can select columns in dataframe by calling them by their column name, using the [] operator

```
df = pd.read_csv('nba.csv')
```

- We can print the columns just to remember what's inside

```
print(df.columns)
```

- Print the contents of column Name:

```
print(df["Name"])
```

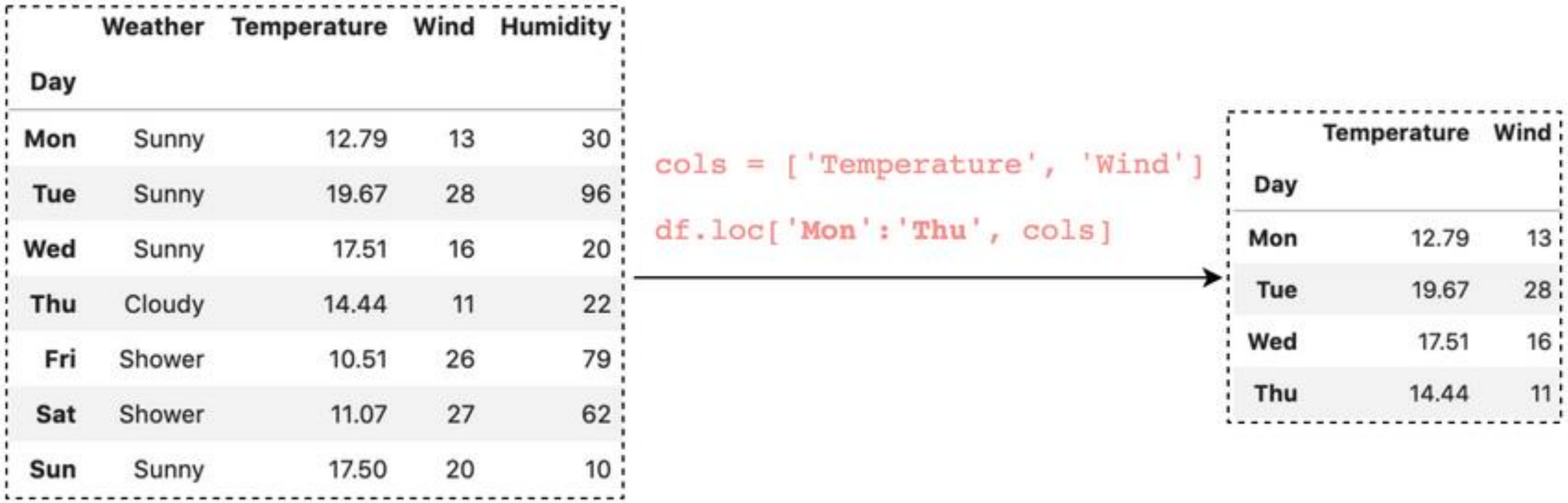
- Selecting more than one column is done using [[]] operator
- Print the contents of columns Name and Age:

```
print(df[["Name", "Age"]])
```

- Or, we can select specific columns when loading the csv:

```
df = pd.read_csv('nba.csv')
df = pd.DataFrame(df, columns=['Name', 'Team'])
```

# Selecting with slices



|  | Weather | Temperature | Wind | Humidity |
|---|---|---|---|---|
| **Day** |  |  |  |  |
| **Mon** | Sunny | 12.79 | 13 | 30 |
| **Tue** | Sunny | 19.67 | 28 | 96 |
| **Wed** | Sunny | 17.51 | 16 | 20 |
| **Thu** | Cloudy | 14.44 | 11 | 22 |
| **Fri** | Shower | 10.51 | 26 | 79 |
| **Sat** | Shower | 11.07 | 27 | 62 |
| **Sun** | Sunny | 17.50 | 20 | 10 |

```
cols = ['Temperature', 'Wind']

df.loc['Mon':'Thu', cols]
```

|  | Temperature | Wind |
|---|---|---|
| **Day** |  |  |
| **Mon** | 12.79 | 13 |
| **Tue** | 19.67 | 28 |
| **Wed** | 17.51 | 16 |
| **Thu** | 14.44 | 11 |

df

Image source: https://towardsdatascience.com/how-to-use-loc-and-iloc-for-selecting-data-in-pandas-bd09cb4c3d79

- Selecting rows with slices:
- `df = pd.read_csv('nba.csv')`
- `new_df = df[:5]` # will select the first 5 rows and put them in `new_df`
- **Sorting**: `df.sort_values(by=‘Salary’)`

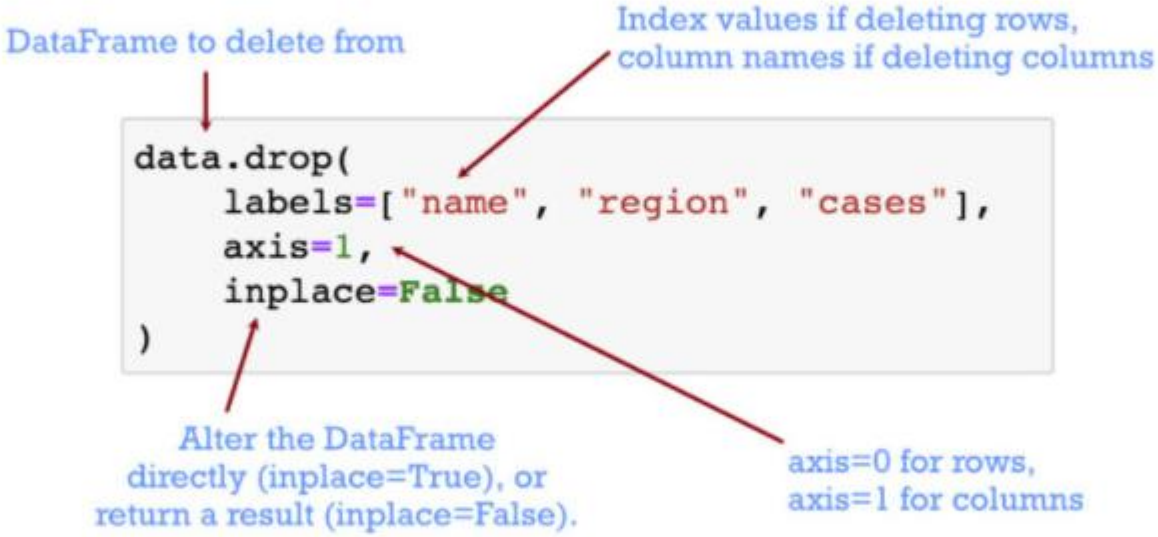
# Deleting specific columns or rows

- In Order to delete a column in Pandas DataFrame, we can use the drop() method.

  ```
  df.drop(["Team", "Weight"], axis = 1,
  inplace=True)
  ```

- Inplace = True means that the changes are made in the original dataframe; False means it creates a new dataframe
- Axis = 1 means that dropping is done on columns
- Axis = 0 means that dropping is done on rows, by index
- Delete a few specified rows at index values 1, 2, 4.

  ```
  # Note that the index values do not
  always align to row numbers.
  data = data.drop(labels=[1,2,4], axis=0)
  ```



```
data.drop(
    labels=["name", "region", "cases"],
    axis=1,
    inplace=False
)
```

DataFrame to delete from
Index values if deleting rows, column names if deleting columns
Alter the DataFrame directly (inplace=True), or return a result (inplace=False).
axis=0 for rows, axis=1 for columns

Image source: https://www.shanelynn.ie/pandas-drop-delete-dataframe-rows-columns/

# Dataframe statistics

- If we have the dataframe:

```
data = pd.read_csv("nba.csv", index_col ="Name")
```

- The following methods compute on all possible columns:

`data.sum()` -> sum of values

`data.min() / data.max()` -> min/max values

`data.idxmin() / data.idxmax()` -> min/max index value

`data.describe()` -> summary statistics

`data.mean()` -> computes mean on all possible columns

`data.median()` -> computes median on all possible columns

# Writing dataframe to csv

- After you modified the dataframe, you can write it to a csv using **.to_csv** method:
- # do something to df dataframe and then save it
- df.to_csv(‘name_of_csv.csv’)
- Encoding parameter – useful to set it to ‘utf-8’ when dealing with Romanian, both for reading and for writing csvs.

# Read and write to Excel

- You can open an excel file using **.read_excel** method:

```
df = pd.read_excel(‘file.xlsx’)
```

- You can create an excel file from a dataframe using .to_excel method:

```
df.to_excel(‘path_to_excel_file.xlsx’, sheet_name =
‘Sheet1’)
```

- You can read each sheet in a different dataframe:

```
df = pd.read_excel(xlsx, ‘Sheet1’)
```

# What is Matplotlib?

matplotlib

- A comprehensive library for creating static, animated, and interactive visualizations in Python
- Installation: pip install matplotlib
- Importing: import matplotlib.pyplot as plt

- You can create plots with: matplotlib, seaborn and pandas as well!

# Bar plot

- You can use **.plot.bar()** method to plot the graph vertically in form of rectangular bars

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

- Create a dataframe from a random numpy array

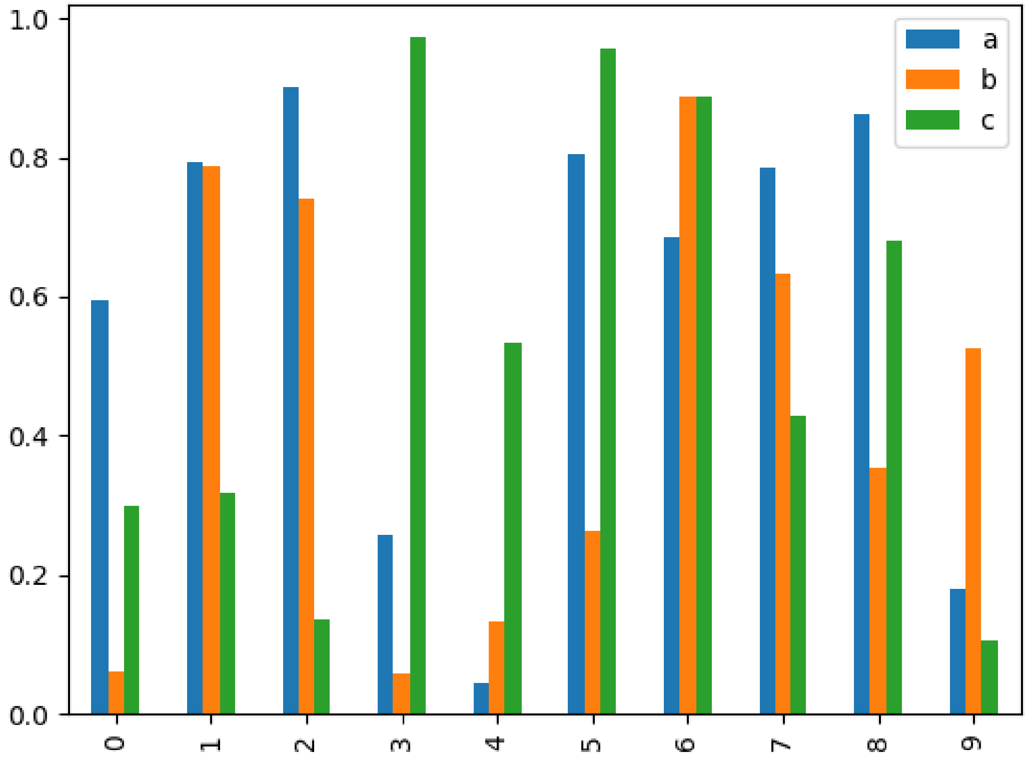
```
rnd_arr = np.random.rand(10, 3)
print(rnd_arr)
df = pd.DataFrame(rnd_arr, columns=['a', 'b', 'c'])
df.plot.bar()
plt.show()
```

- Plotting two columns from dataframe:

```
df.plot.bar(x = ‘name_of_column1’, ‘name_of_column2’)
```

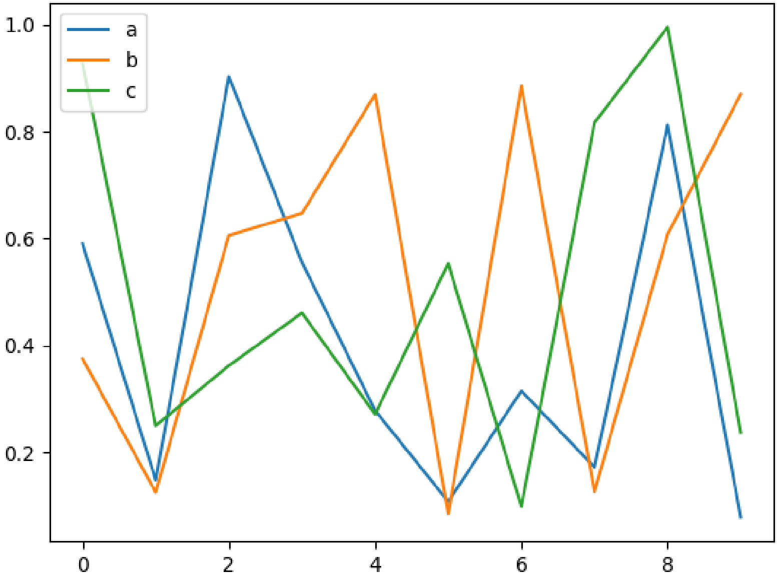
# Bar plot

```
 a       b       c
0  0.594736  0.060920  0.299605
1  0.792587  0.788698  0.317061
2  0.900321  0.740707  0.136934
3  0.257342  0.058175  0.972296
4  0.045137  0.131768  0.534247
5  0.803600  0.262312  0.957628
6  0.684835  0.888481  0.887564
7  0.784256  0.633894  0.429599
8  0.862750  0.354024  0.679965
9  0.181168  0.524034  0.106147
```



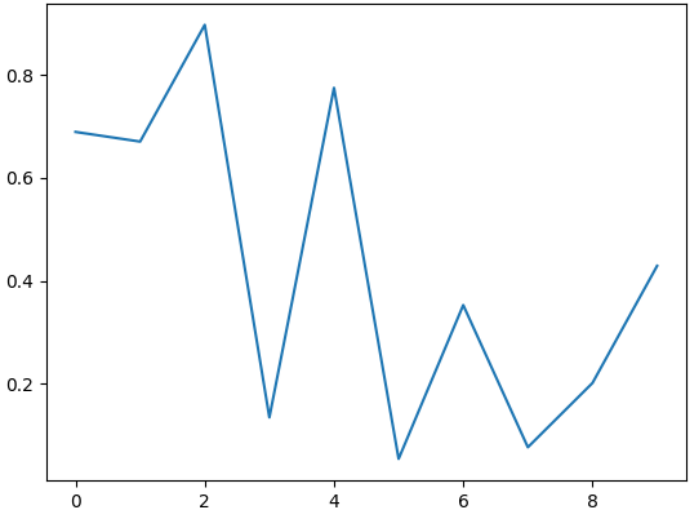
# Line plot

- On the same dataframe as previous, if I write df.plot.line() instead of df.plot.bar():



- If I want, I can select only one column to plot:

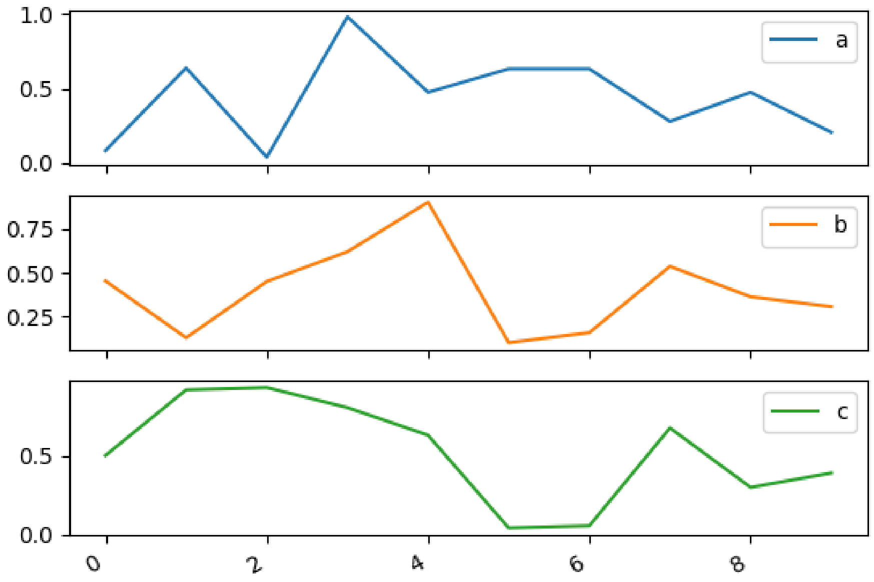
```
df[‘a’].plot.line()
```



Disclaimer: plots are different because of creating another random array on each run

- We can also see each line in dataframe separately:

```
axes = df.plot.line(subplots=True)
```

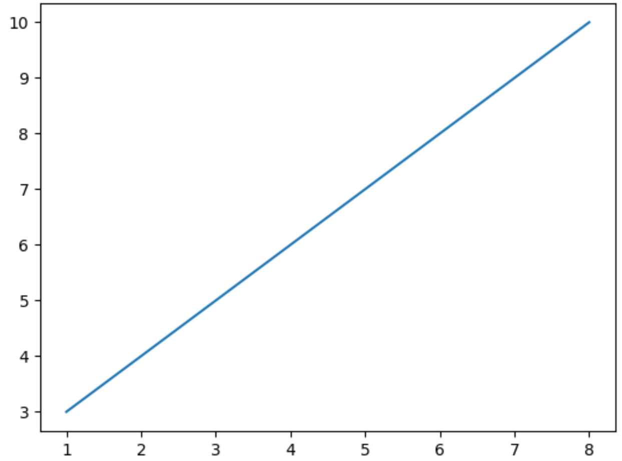


# Creating line plots with matplotlib

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

By setting coordonates for each point:

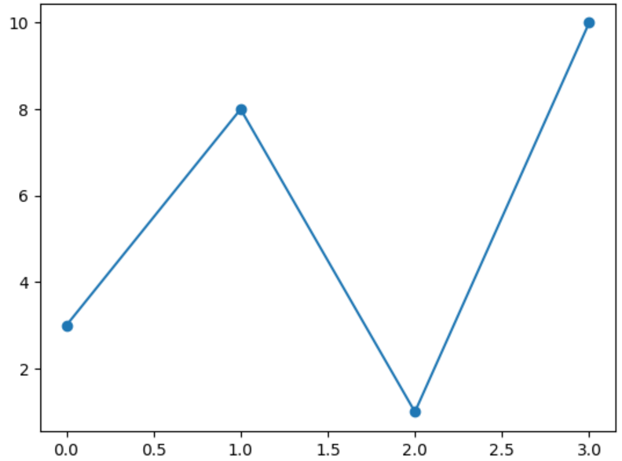
```
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints)
plt.show()
```



By setting only y coordinates.

And you can also use markers.

```
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o')
plt.show()
```



More on matplotlib: https://www.w3schools.com/python/matplotlib_intro.asp

# Pie chart

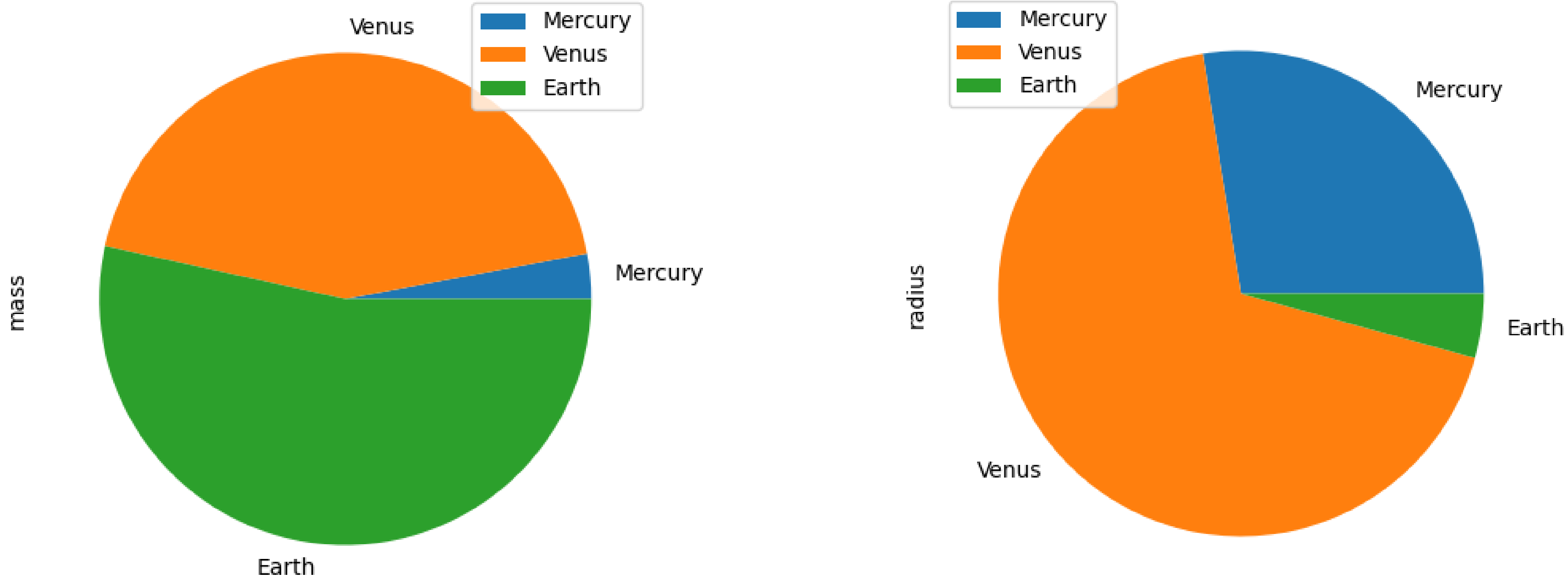
- Here is a dataframe holding information about some planets mass and radius:

```
df = pd.DataFrame({'mass': [0.330, 4.87 , 5.97],
                   'radius': [2439.7, 6051.8, 378.1]},
                  index=['Mercury', 'Venus', 'Earth'])
```

- You can use .plot.pie method to get a pie plot:

```
plot = df.plot.pie(y='mass', figsize=(5, 5))
```

# Pie chart



More about pandas plots: [https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html)

# Thank you!