***NumPy***

* NumPy is a python library
* Used for working with arrays
* NumPy stands for Numerical Python
* Was created in 2005 by Travis Oliphant
* It is an open source project and you can use it freely
* It also has functions for working in domain of linear algebra, fourier transform, and matrices

**Lists VS NumPy**

NumPy aims to provide an array object which is 50x faster than lists. Lists also used to work with arrays, but are slow to process.

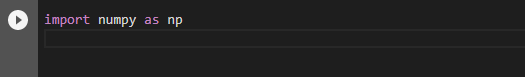
* NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This is called locality of reference in computer science.
* This is the main reason why NumPy is faster than lists. Also it is optimized to work with the latest CPU architectures.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

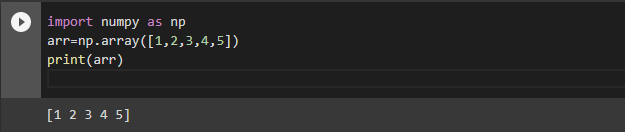
* Import NumPy in your applications by adding the import keyword



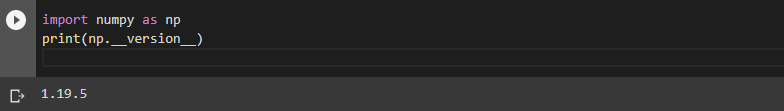
* NumPy is usually imported under the np alias.



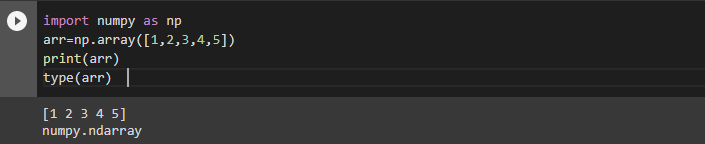
* 1-D Array using array() function



* Checking version using \_\_version\_\_ attribute



The array object in NumPy is called ndarray. We can create a NumPy ndarray object by using the array() function.



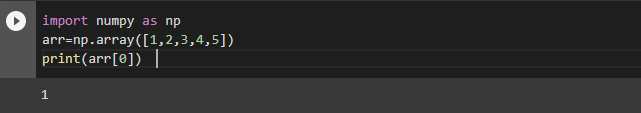
type(): This is a python built-in function which tells us the type of the object.

**INDEXING:**

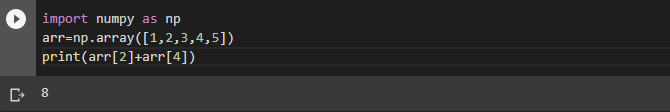
Accessing 1-D Arrays

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

* Printing first element



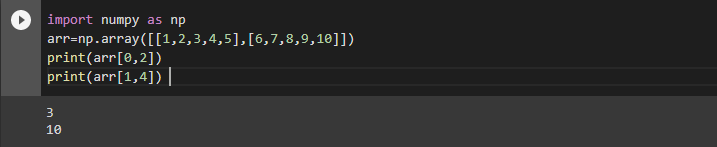
* Printing the sum of given indices



Accessing 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

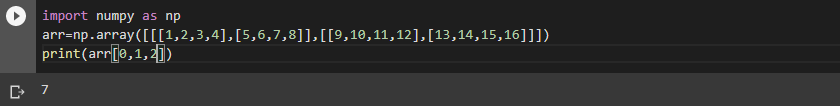
* Printing 3nd element from 1st Dimension and 5th element from 2nd Dimension



Accessing 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

* Printing third element of the second array of the first array.



How indexing works in above experiments

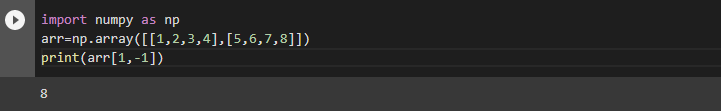
The first number represents the first dimension, which contains two arrays:  
[[1, 2, 3, 4], [5, 6, 7, 8]]  
and:  
[[9, 10, 11, 12], [13, 14, 15, 16]]  
Since we selected 0, we are left with the first array:  
[[1, 2, 3, 4], [5, 6, 7, 8]]

The second number represents the second dimension, which also contains two arrays:  
[1, 2, 3, 4]  
and:  
[5, 6, 7, 8]  
Since we selected 1, we are left with the second array:  
[5, 6, 7, 8]

The third number represents the third dimension, which contains three values:  
 5, 6, 7, 8  
Since we selected 2, we end up with the third value:  
7

**Negative Indexing**

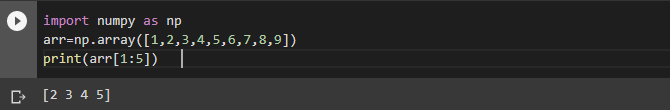
* Printing the last element from 2nd Dimension



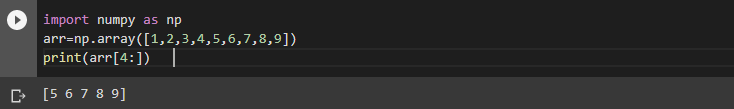
**SLICING:**

Slicing in python means taking elements from one given index to another given index.

* We pass slice instead of index like this: [*start*:*end*].
* We can also define the step, like this: [*start*:*end*:*step*].
* If we don't pass start it is considered 0
* If we don't pass end it is considered length of array in that dimension
* If we don't pass step it is considered 1
* Slicing elements from index 1 to 5

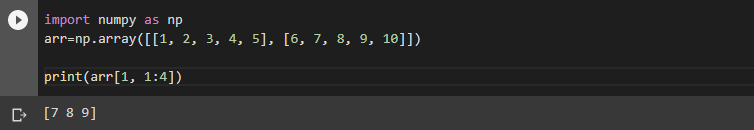


* The result includes the start index but excludes the end index
* Slicing elements from 4th index to end of the array



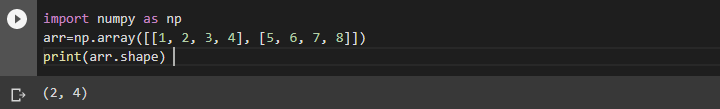
**Slicing in 2-D Arrays**

* From the second Array, slice elements from index 1 to 4



**NUMPY ARRAY SHAPE:**

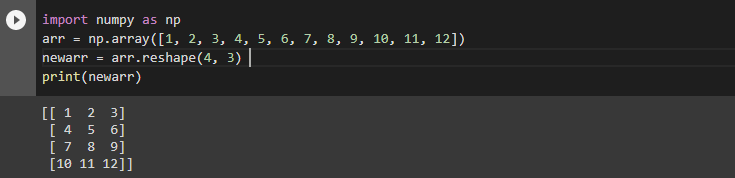
* The shape of an array is the number of elements in each dimension.
* NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.
* Printing the shape of 2-D Array



Here (2,4) means that the array has 2 dimensions, and each dimension has 4 elements.

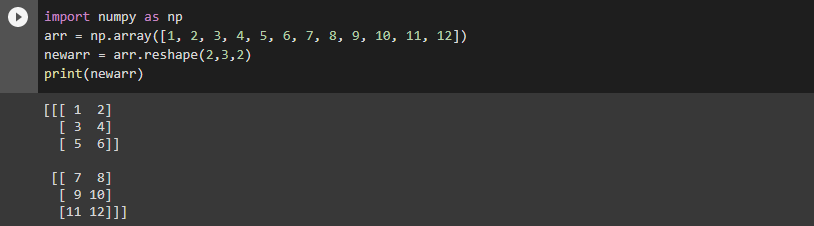
**NUMPY ARRAY RESHAPE:**

* Reshaping means changing the shape of an array.
* The shape of an array is the number of elements in each dimension.
* By reshaping we can add or remove dimensions or change number of elements in each dimension.
* Reshaping 1-D array to 2-D array



The outermost dimension will have 4 arrays, each with 3 elements.

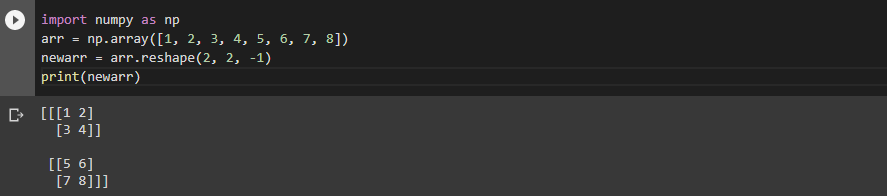
* Reshaping 1-D to 3-D Array



The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements.

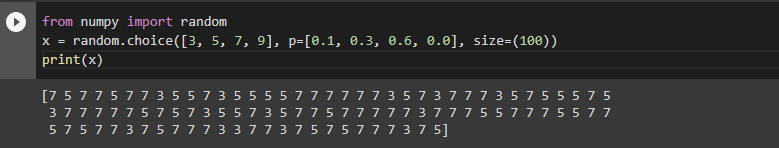
**Unknown Dimension:**

* This means that we do not have to specify an exact number for one of the dimensions in the reshape method.
* Pass -1 as the value, and NumPy will calculate this number.
* Reshaping 1-D Array with 8 elements to 3-D Array with 2x2 elements



**RANDOM DATA DISTRIBUTION:**

* Data Distribution is a list of all possible values, and how often each value occurs.
* Such lists are important when working with statistics and data science.
* The random module offer methods that returns randomly generated data distributions.
* A random distribution is a set of random numbers that follow a certain *probability density function*.



This output represents a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9.

The probability for the value to be 3 is set to be 0.1

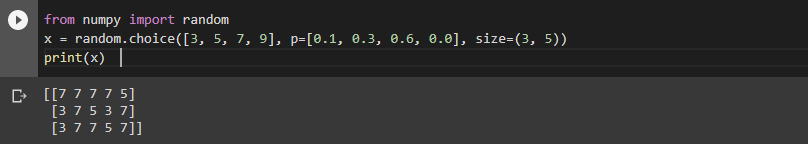
The probability for the value to be 5 is set to be 0.3

The probability for the value to be 7 is set to be 0.6

The probability for the value to be 9 is set to be 0

**NOTE:** The sum of all probability numbers should be 1.

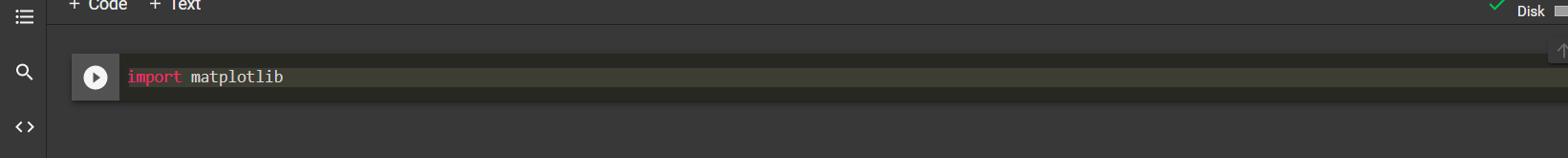
* Returning 2-D Array with 3 rows, each containing 5 values using size parameter



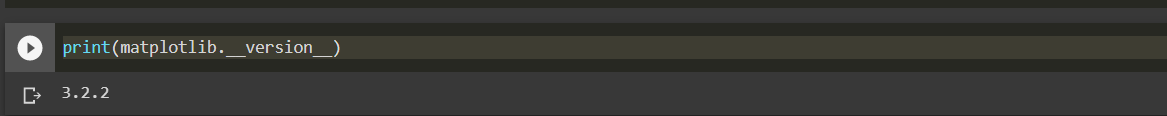
***Matplotlib***

* Matplotlib is a plotting library in python that serves as a visualization utility.
* It is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.
* Matplotlib was created by John D. Hunter.

↪ Import it in your applications by adding the import keyword.

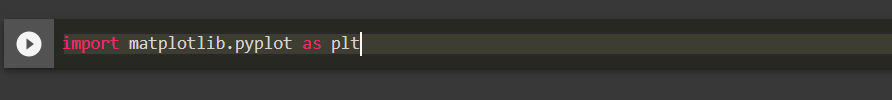


↪ The version string is stored under the \_\_version\_\_ attribute.



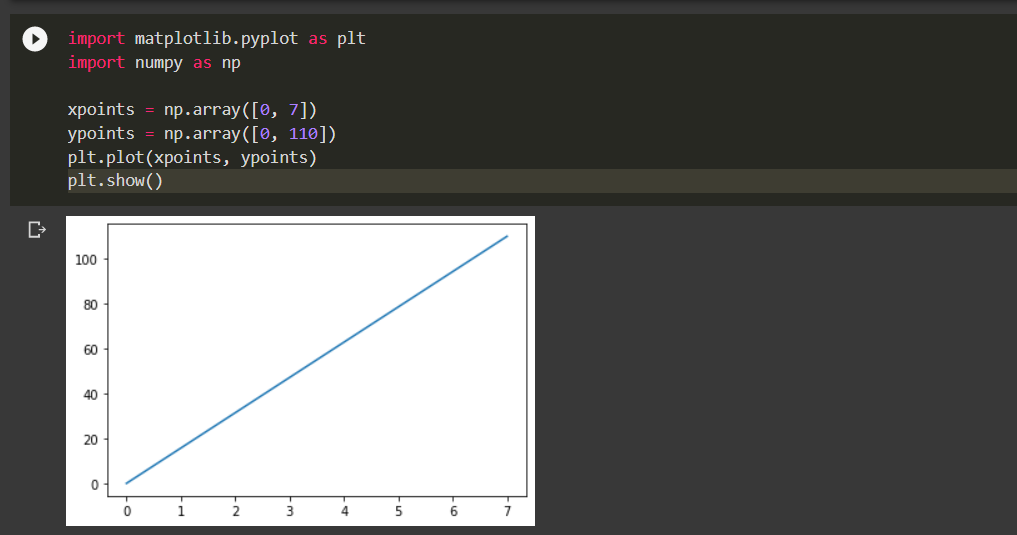
**Pyplot :**

↪ Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt



⟹ Now Pyplot can be referred as plt

⇒ Let us take an example to draw a line from position (0,0) to (7,110) :



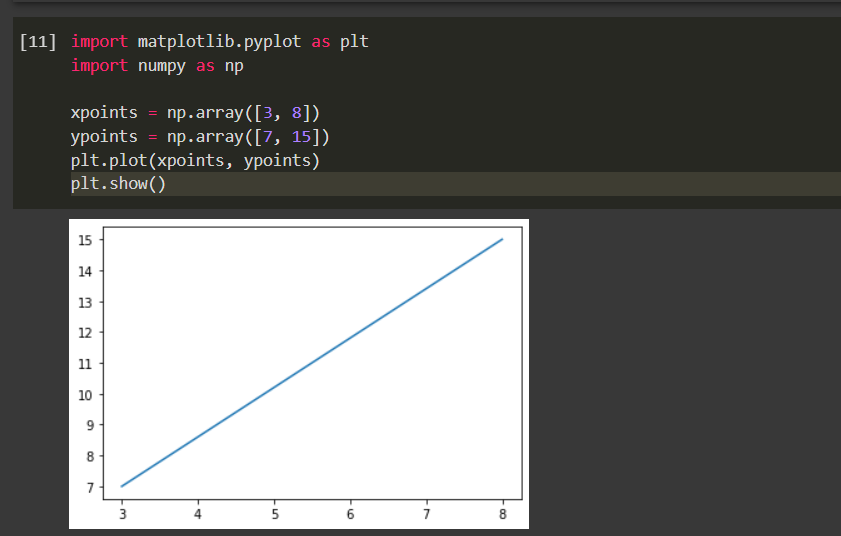
**Matplotlib Plotting :**

↪ The plot() function is used to draw points (markers) in a diagram and it draws line from point to point.

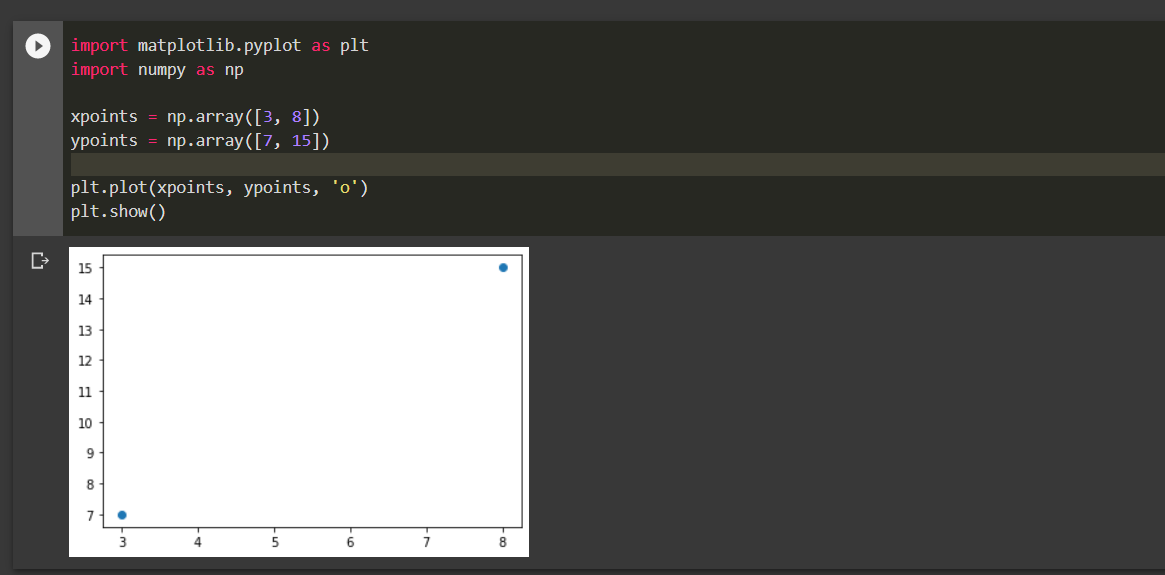
↪ Parameter 1 is an array containing the points on the x-axis.

↪ Parameter 2 is an array containing the points on the y-axis.

⇒ Let us take an example by drawing line from position (3,7) to (8,15)



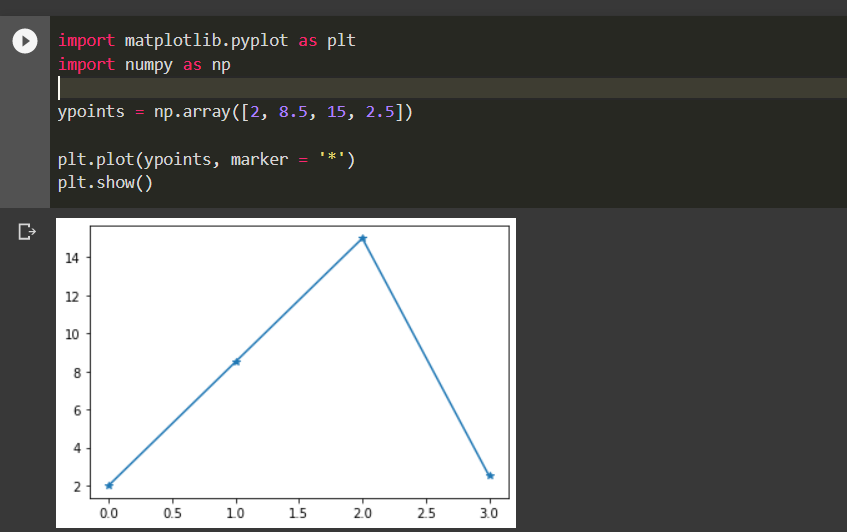
**⇒** we can also plot without line :

****

**Matplotlib Markers :**

✫ You can use the keyword argument marker to emphasize each point with a specified marker :

Marking each point with asterisk.



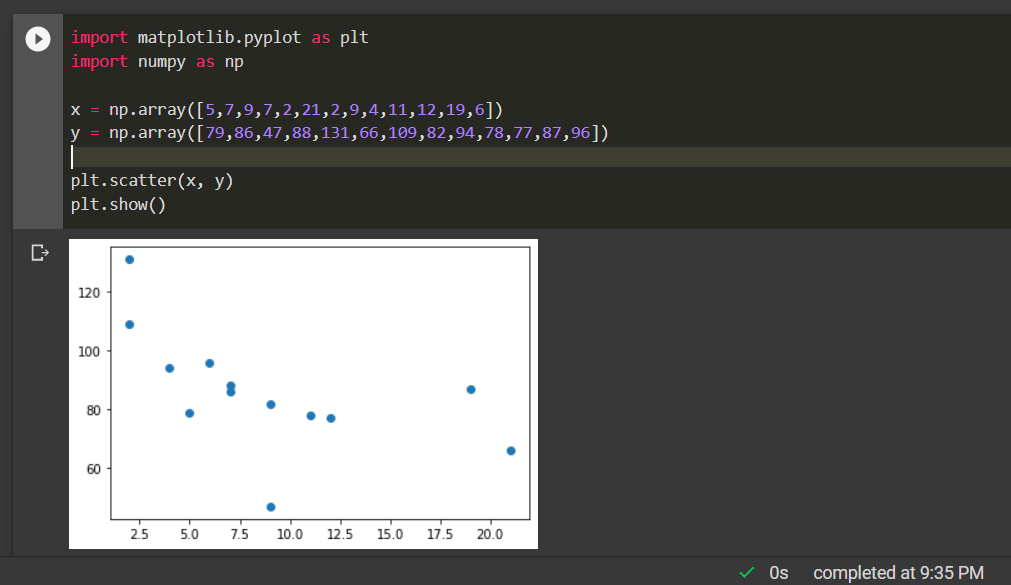
We can also mark points with ‘o’, ‘.’, ‘,’, ‘x’, ‘+’ etc.

**Matplotlib Scatter :**

✫ scatter() function is used to draw a scatter plot, It plots one dot for each observation.

✫ It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis.

A simple scatter plot :



We can also compare two plots by drawing them on same figure using scatter :

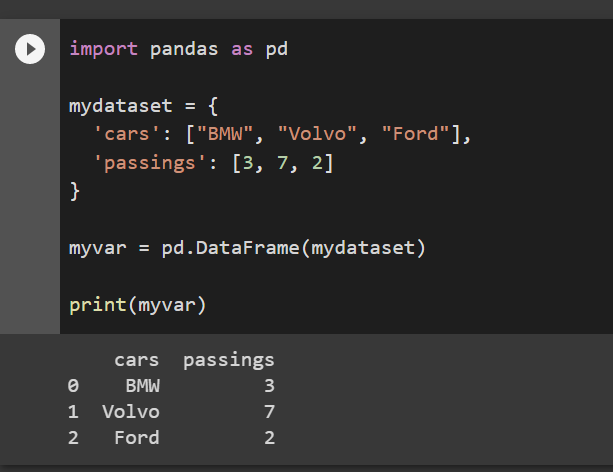


***Pandas***

* Pandas is used to analyse data
* Pandas is a Python library used for working with data sets. It has functions for analysing, cleaning, exploring, and manipulating data.
* Pandas is usually imported under the pd alias.

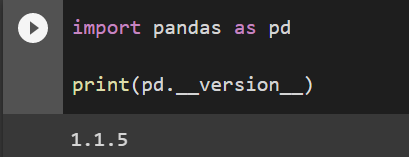


Now the Pandas package can be referred to as pd instead of pandas.



**Checking Pandas Version**

The version string is stored under \_\_version\_\_ attribute.



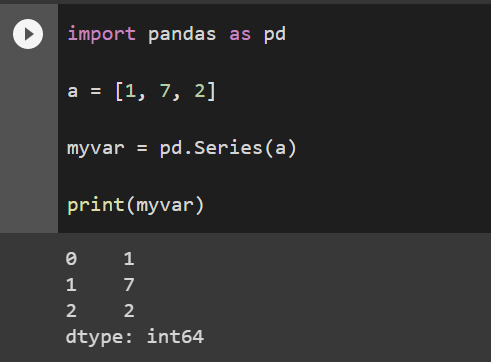
Pandas generally provide two data structure for manipulating data, They are:

* **Series**
* **Data Frame**

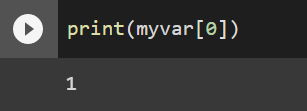
**Series:**

 A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type

Example

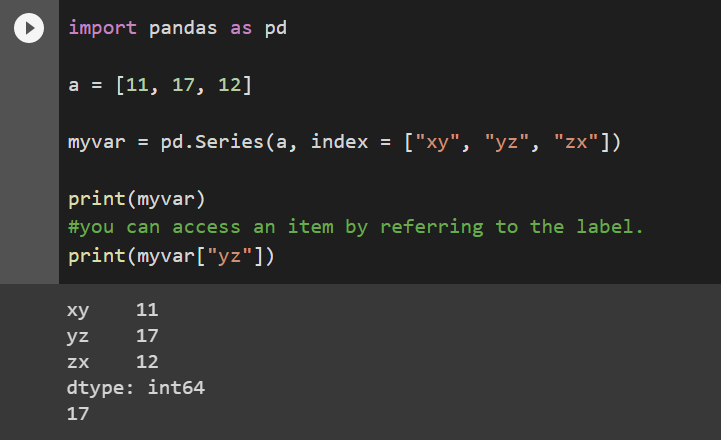
:

label can be used to access a specified value.

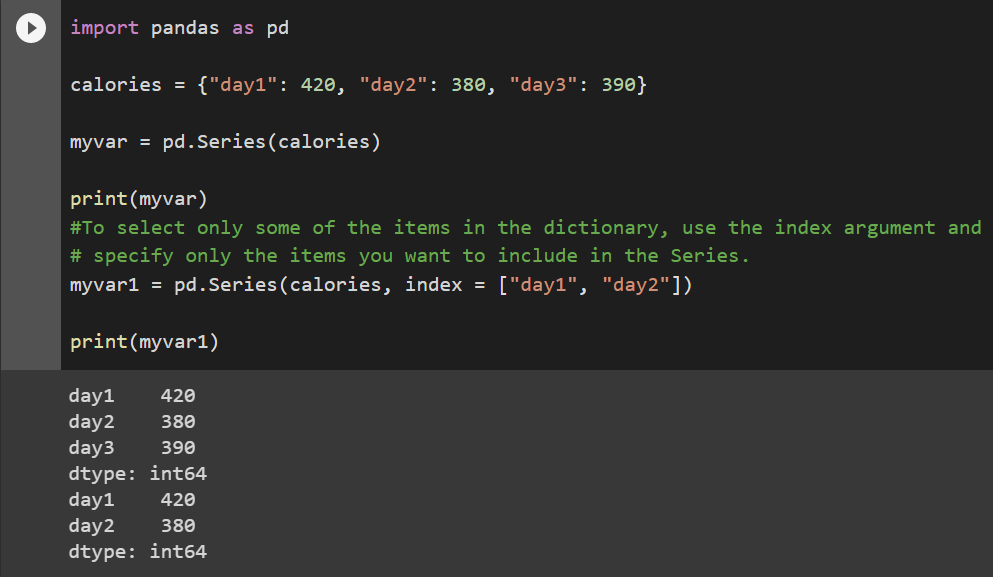


**Create Labels:**

You can create your own labels by using index argument

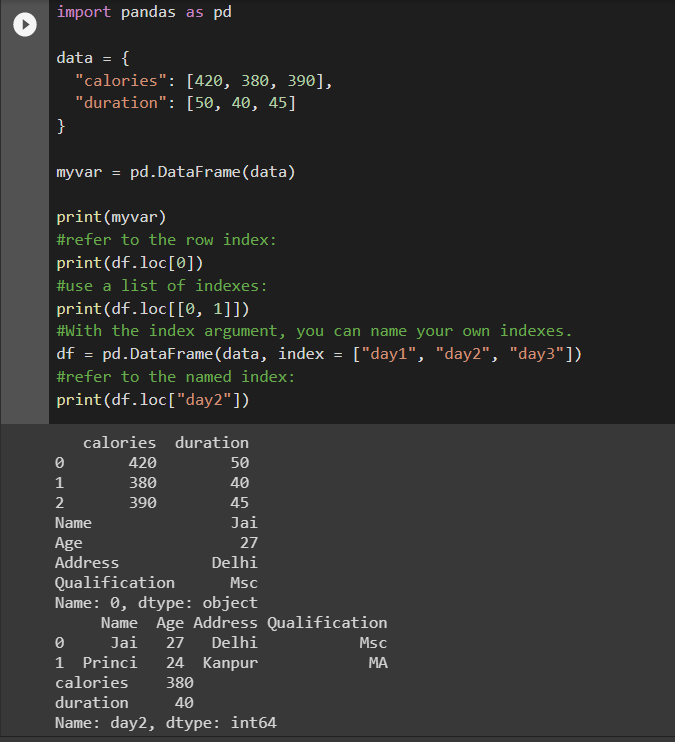


**You can also use key value pairs like in dict.**



**Data Frames**

* Data sets in Pandas are usually multi-dimensional tables, called DataFrames.
* Series is like a column, a DataFrame is the whole table.
* Pandas use the loc attribute to return one or more specified row(s)
* With the index argument, you can name your own indexes.
* Use the named index in the loc attribute to return the specified row(s)



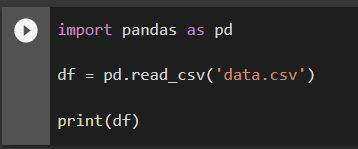
**Load Files Into a DataFrame**

If your data sets are stored in a file, Pandas can load them into a DataFrame.

**Read CSV Files**

A simple way to store big data sets is to use CSV files (comma separated files).

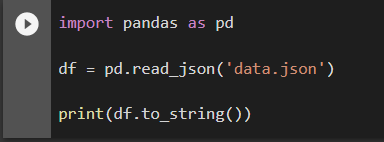
CSV files contains plain text and is a well know format that can be read by everyone including Pandas



* Here we have used **print(df)**
* It will give you only up to 5 rows  and the last 5 rows
* But we use **df.to\_string()** to print entire Data Frame

**Read JSON**

* Big data sets are often stored, or extracted as JSON.
* JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.



use **to\_string()** to print the entire DataFrame.

**Dictionary as JSON**

JSON = Python Dictionary

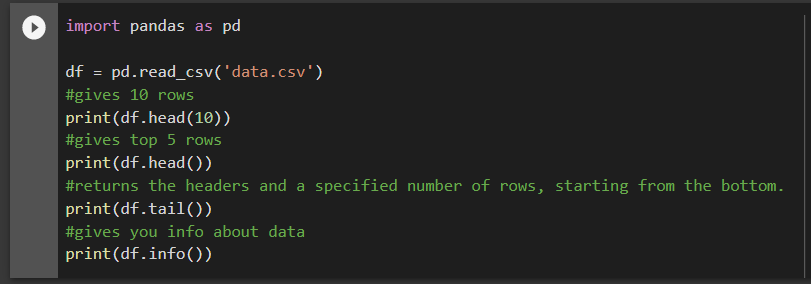
JSON objects have the same format as Python dictionaries.

If your JSON code is not in a file, but in a Python Dictionary, you can load it into a DataFrame directly

Pandas - Analyzing DataFrames

**Viewing the Data**

* The head() method returns the headers and a specified number of rows, starting from the top.
* if the number of rows is not specified, the head() method will return the top 5 rows.
* The tail() method returns the headers and a specified number of rows, starting from the bottom.
* The DataFrames object has a method called info(), that gives you more information about the data set.(like how many rows and columns)
* The info() method also tells us how many Non-Null values there are present in each column



## Data Cleaning

Data cleaning means fixing bad data in your data set.

Bad data could be:

* Empty cells
* Data in wrong format
* Wrong data
* Duplicates

Empty cells can potentially give you a wrong result when you analyse data.

## Remove Rows

By default, the dropna() method returns a new DataFrame, and will not change the original.

If you want to change the original DataFrame, use the inplace = True argument

the dropna(inplace = True) will NOT return a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.

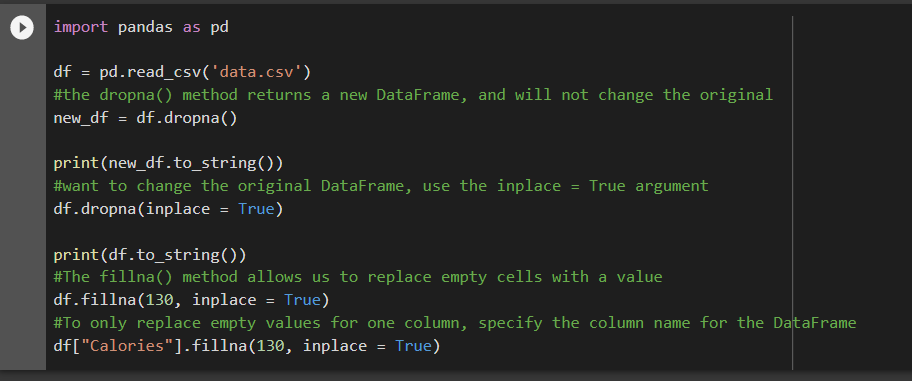
## Replace Empty Values

Another way of dealing with empty cells is to insert a new value instead.

The fillna() method allows us to replace empty cells with a value

This will replaces all empty cells in the whole Data Frame

To only replace empty values for one column, specify the *column name* for the DataFrame



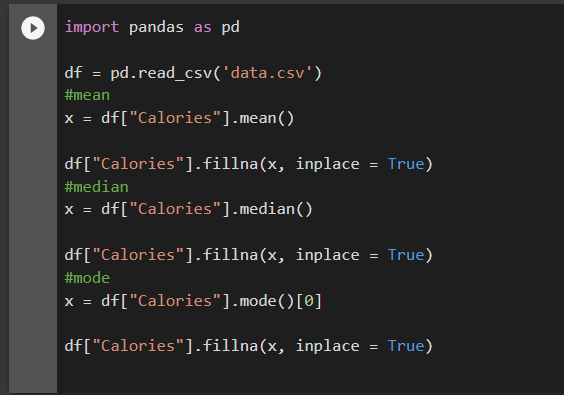
## Replace Using Mean, Median, or Mode

Pandas uses the mean() median() and mode() methods to calculate the respective values for a specified column

Mean = the average value (the sum of all values divided by number of values).

Median = the value in the middle, after you have sorted all values ascending.

Mode = the value that appears most frequently.



## Data of Wrong Format

Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

## Convert Into a Correct Format

## Removing Rows

## The result from the converting in the example above gave us a NaT value, which can be handled as a NULL value, and we can remove the row by using the dropna() method.

## Wrong Data

## "Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

## Replacing Values

One way to fix wrong values is to replace them with something else.

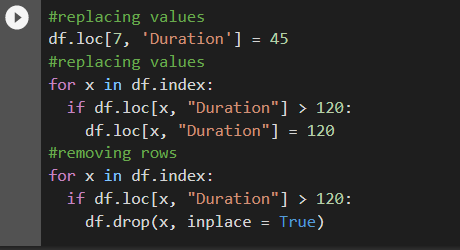
In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7

To replace wrong data for larger data sets you can create some rules, e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

## Removing Rows

Another way of handling wrong data is to remove the rows that contains wrong data.

This way you do not have to find out what to replace them with, and there is a good chance you do not need them to do your analyses.



# Pandas - Removing Duplicates

## Discovering Duplicates:

To discover duplicates, we can use the duplicated() method.

The duplicated() method returns a Boolean values for each row

## Removing Duplicates

To remove duplicates, use the drop\_duplicates() method

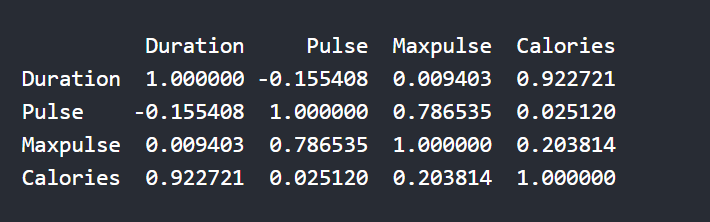
## The (inplace = True) will make sure that the method does NOT return a new DataFrame, but it will remove all duplicates from the original DataFrame.

## 

**Pandas - Data Correlations**

* The corr() method calculates the relationship between each column in your data set.
* The **corr()** method ignores "not numeric" columns.

df.corr()

for this input we get output like:

### Perfect Correlation:

We can see that "Duration" and "Duration" got the number 1.000000, which makes sense, each column always has a perfect relationship with itself.

### Good Correlation:

"Duration" and "Calories" got a 0.922721 correlation, which is a very good correlation, and we can predict that the longer you work out, the more calories you burn, and the other way around: if you burned a lot of calories, you probably had a long workout.

### Bad Correlation:

"Duration" and "Maxpulse" got a 0.009403 correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the workout, and vice versa.

# Pandas – Plotting

# Pandas uses the plot() method to create diagrams.

# 

## Scatter Plot

## Specify that you want a scatter plot with the kind argument:

kind = 'scatter'

A scatter plot needs an x- and a y-axis.

In the example below we will use "Duration" for the x-axis and "Calories" for the y-axis.

Include the x and y arguments like this:

x = 'Duration', y = 'Calories'

# 

# 

## Histogram

Use the kind argument to specify that you want a histogram:

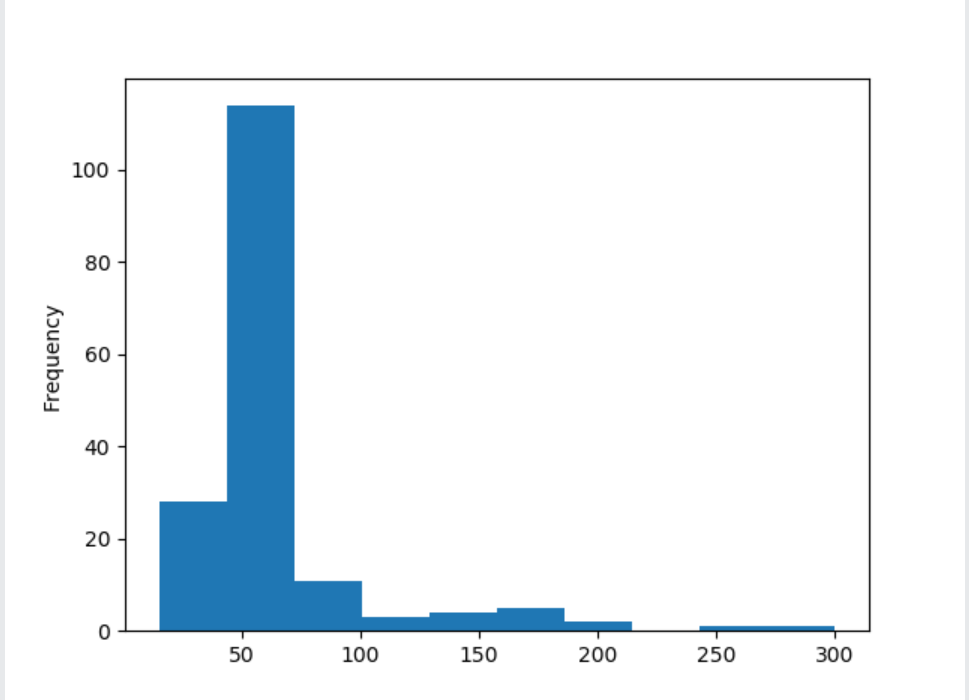
kind = 'hist'

A histogram needs only one column.

A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?

In the example below we will use the "Duration" column to create the histogram:





Github link for the above code :

https://github.com/Yashwanth-23/NumPy-Matplotlib-Pandas