UNIT 3:
Python Programming

Introduction to Python

Python is a general-purpose, high level programming language. It was created by Guido van Rossum, and released in 1991. Python got its name from a BBC comedy series – “Monty Python’s Flying Circus”

Features of Python

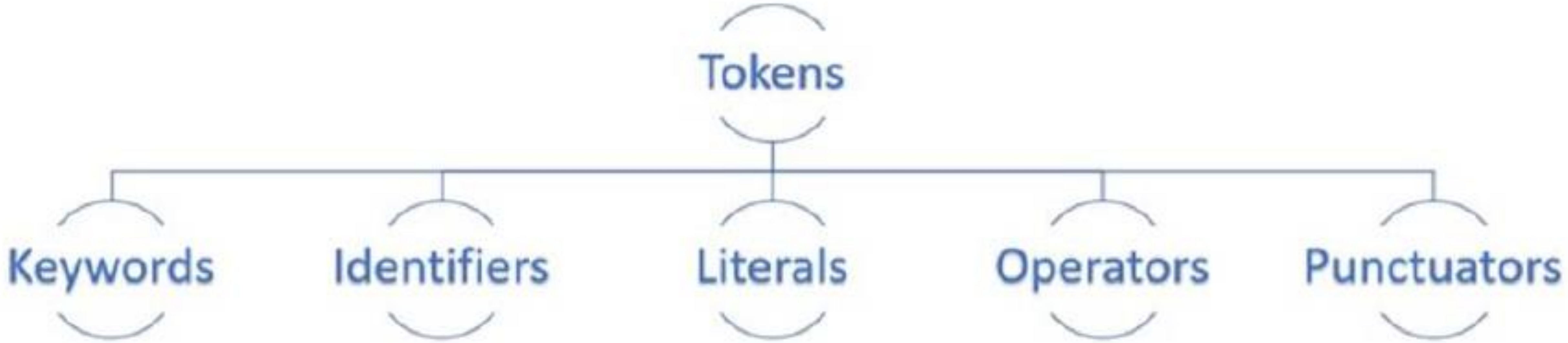
- High Level language
- Interpreted Language
- Free and Open Source
- Platform Independent (Cross-Platform) – runs virtually in every platform if a compatible python interpreter is installed.
- Easy to use and learn – simple syntax similar to human language.
- Variety of Python Editors – Python IDLE, PyCharm, Jupyter, Spyder
- Python can process all characters of ASCII and UNICODE.
- Widely used in many different domains and industries.

Python Editors

- Jupyter Notebook
- PyCharm,
- Spyder,
- IDLE
- Google Collaboratory

Getting Started with Python Programs

- Python program consists of Tokens.
- It is the smallest unit of a program that the interpreter or compiler recognizes.
- Tokens consist of identifiers, keywords, literals, operators, and punctuators.



Tokens

Keywords

Identifiers

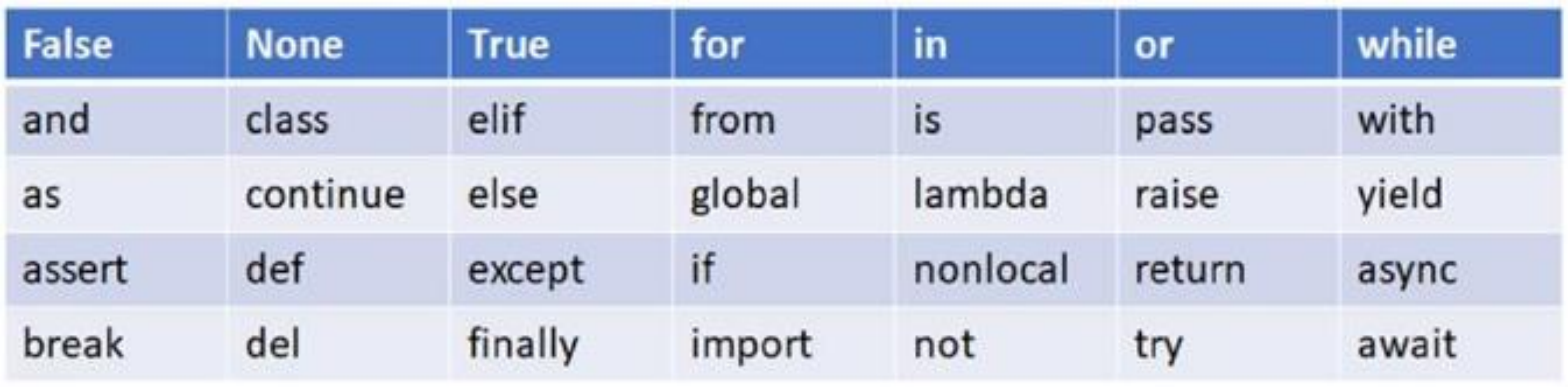
Literals

Operators

Punctuators

Keywords

Reserved words used for special purpose. List of keywords are given below.



False	None	True	for	in	or	while
and	class	elif	from	is	pass	with
as	continue	else	global	lambda	raise	yield
assert	def	except	if	nonlocal	return	async
break	del	finally	import	not	try	await

Identifier

- An identifier is a name used to identify a variable, function, class, module or other object.
- Generally, keywords (list given above) are not used as variables.
- Identifiers cannot start with digit and also it can’t contain any special characters except underscore.

Literals:

- Literals are the raw data values that are explicitly specified in a program.

- Different types of Literals in Python are String Literal, Numeric Literal(Numbers), Boolean Literal(True & False), Special Literal (None) and Literal Collections.

Operators:

Operators are symbols or keywords that perform operations on operands to produce a result. Python supports a wide range of operators:

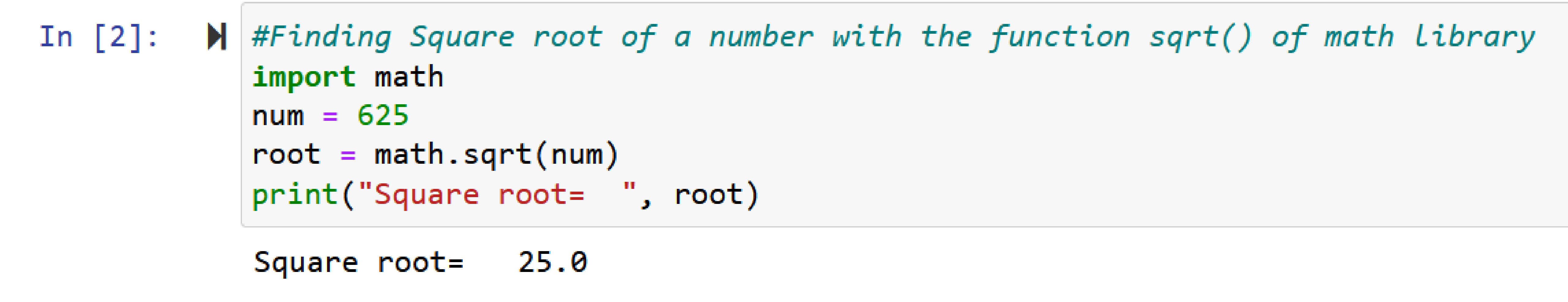
- Arithmetic operators (+, -, *, /, %)
- Relational operators (==, !=, <, >, <=,>=)
- Assignment operators (=, +=, -=)
- Logical operators (and, or, not)
- Bitwise operators (&, |, ^, <<, >>)
- Identity operators (is, is not)
- Membership operators (in, not in)

Punctuators:

Common punctuators in Python include

: ( ) [ ] { } , ; . ` ' ' " " / \ & @ ! ? | ~ etc.

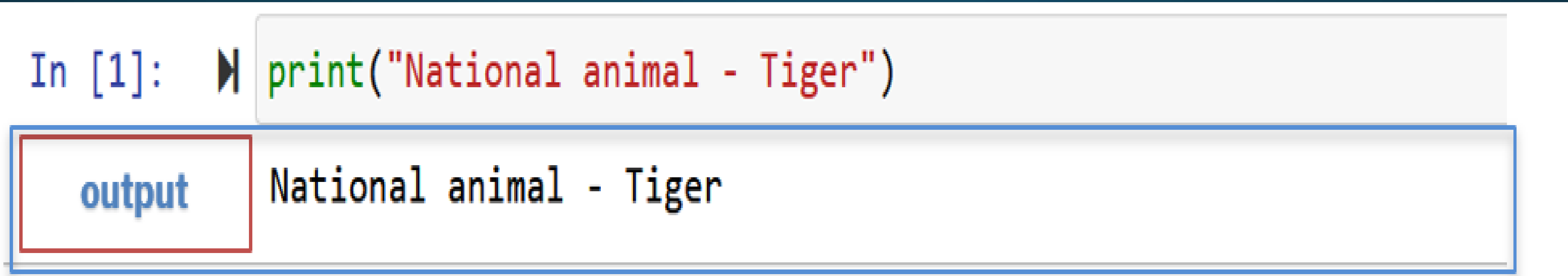
Example



```
#Finding Square root of a number with the function sqrt() of math library
import math
num = 625
root = math.sqrt(num)
print("Square root=  ", root)
```

```
Square root=   25.0
```

Sample Program-1 Display the string “National Animal-Tiger” on the screen



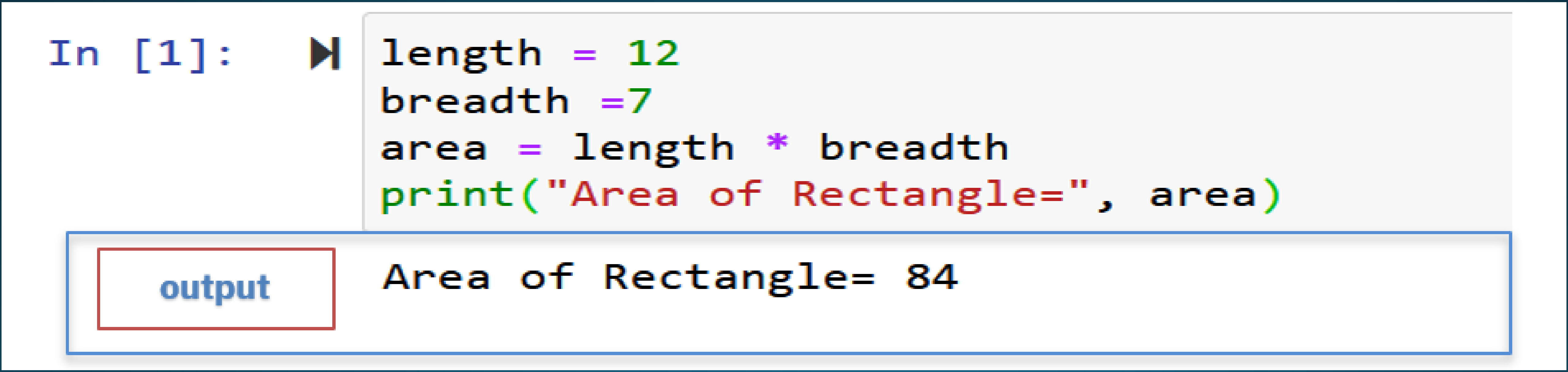
```
In [1]:  print("National animal - Tiger")
```

output

```
National animal - Tiger
```

Sample Program-2

Write a program to calculate the area of a rectangle given the length and breadth are 50 and 20 respectively.

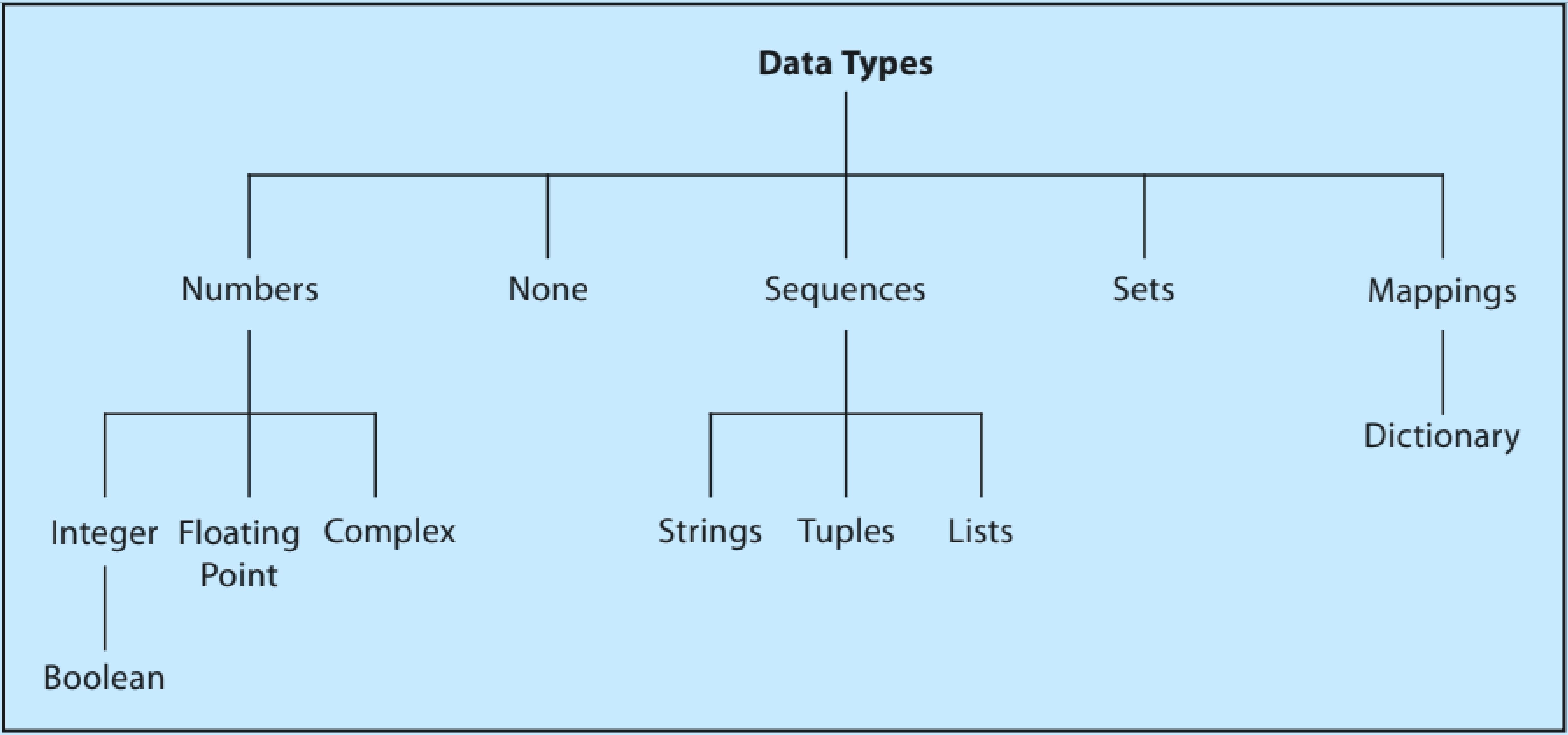


```
length = 12
breadth =7
area = length * breadth
print("Area of Rectangle=", area)
```

```
Area of Rectangle= 84
```

Data Types:

- Data types are the classification or categorization of data items.
- It represents the kind of value that tells what operations can be performed on a particular data.
- Python supports Dynamic Typing.
- A variable pointing to a value of certain data type can be made to point to a value/object of another data type.
- This is called Dynamic Typing.



Data Types

Numbers

None

Sequences

Sets

Mappings

Integer

Floating Point

Complex

Boolean

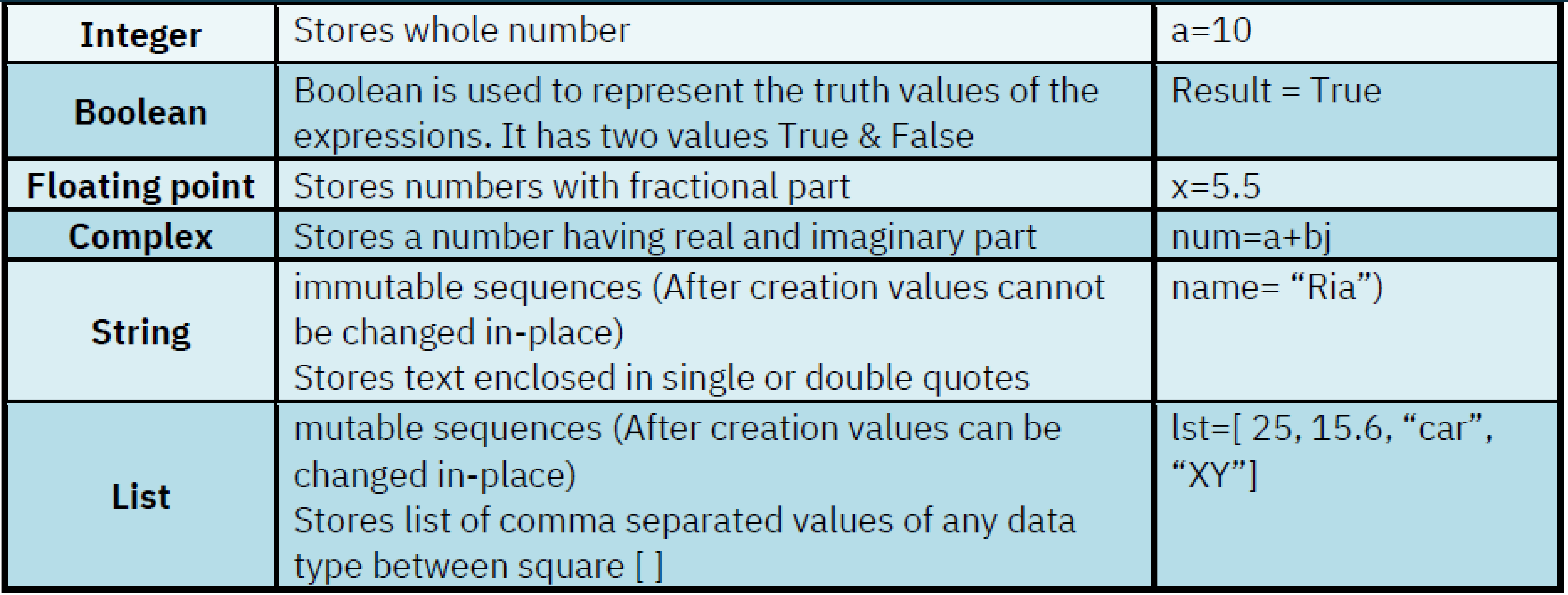
Strings

Tuples

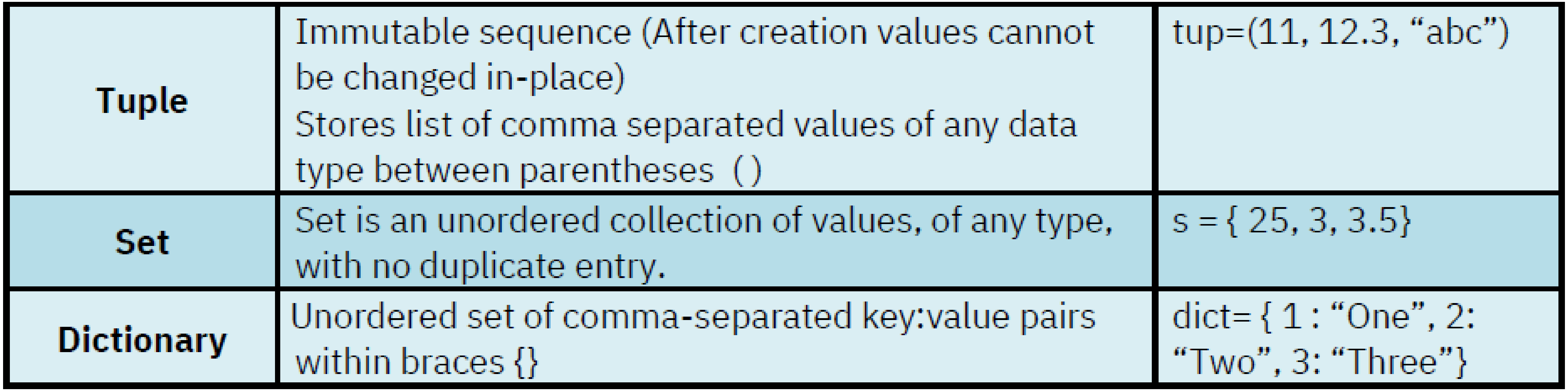
Lists

Dictionary

Data Type Description



Integer	Stores whole number	a=10
Boolean	Boolean is used to represent the truth values of the expressions. It has two values True & False	Result = True
Floating point	Stores numbers with fractional part	x=5.5
Complex	Stores a number having real and imaginary part	num=a+bj
String	immutable sequences (After creation values cannot be changed in-place) Stores text enclosed in single or double quotes	name= “Ria”)
List	mutable sequences (After creation values can be changed in-place) Stores list of comma separated values of any data type between square [ ]	lst=[25, 15.6, “car”, “XY”]

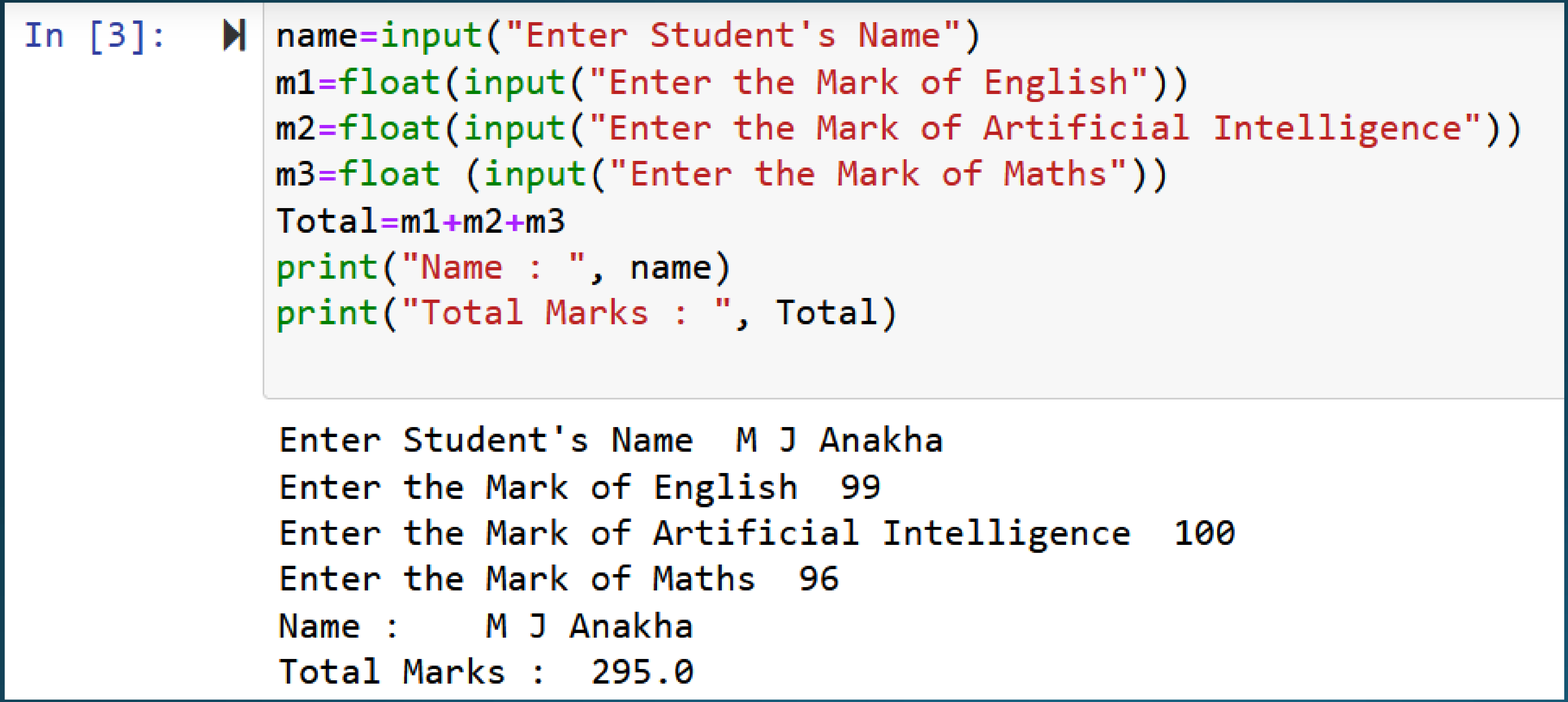


Tuple	Immutable sequence (After creation values cannot be changed in-place) Stores list of comma separated values of any data type between parentheses ( )	tup=(11, 12.3, “abc”)
Set	Set is an unordered collection of values, of any type, with no duplicate entry.	s = { 25, 3, 3.5}
Dictionary	Unordered set of comma-separated key:value pairs within braces {}	dict= { 1 : “One”, 2: “Two”, 3: “Three”}

Accepting values from the user

- The input() function retrieves text from the user by prompting them with a string argument.
- For instance: name = input("What is your name?")
- Return type of input function is string.
- So, to receive values of other types we have to use conversion functions together with input function.

Sample Program-3 Write a program to read name and marks of a student and display the total mark.

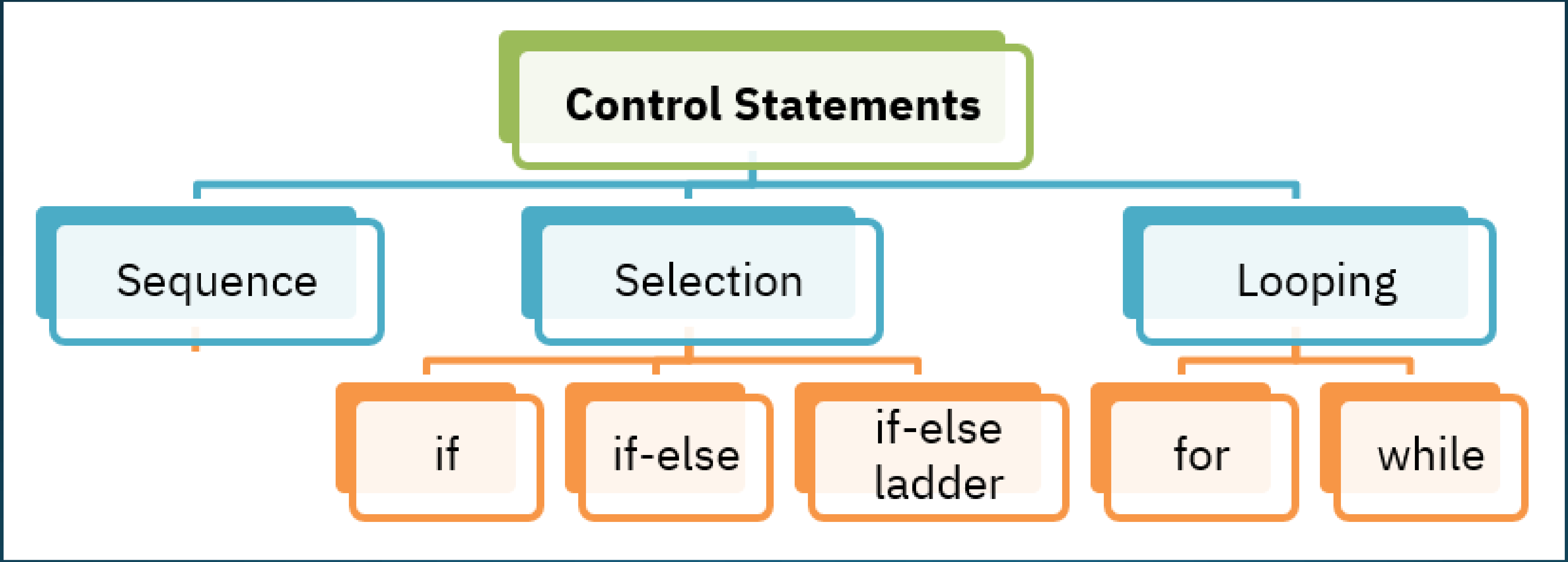


```python
name=input("Enter Student's Name")
m1=float(input("Enter the Mark of English"))
m2=float(input("Enter the Mark of Artificial Intelligence"))
m3=float (input("Enter the Mark of Maths"))
Total=m1+m2+m3
print("Name : ", name)
print("Total Marks : ", Total)
```

```
Enter Student's Name  M J Anakha
Enter the Mark of English  99
Enter the Mark of Artificial Intelligence  100
Enter the Mark of Maths  96
Name :    M J Anakha
Total Marks :  295.0
```

In the above example float( ) is used to convert the datatype into floating point. The explicit conversion of an operand to a specific type is called type casting

Control flow statements in Python



Control Statements

Sequence

Selection

Looping

if

if-else

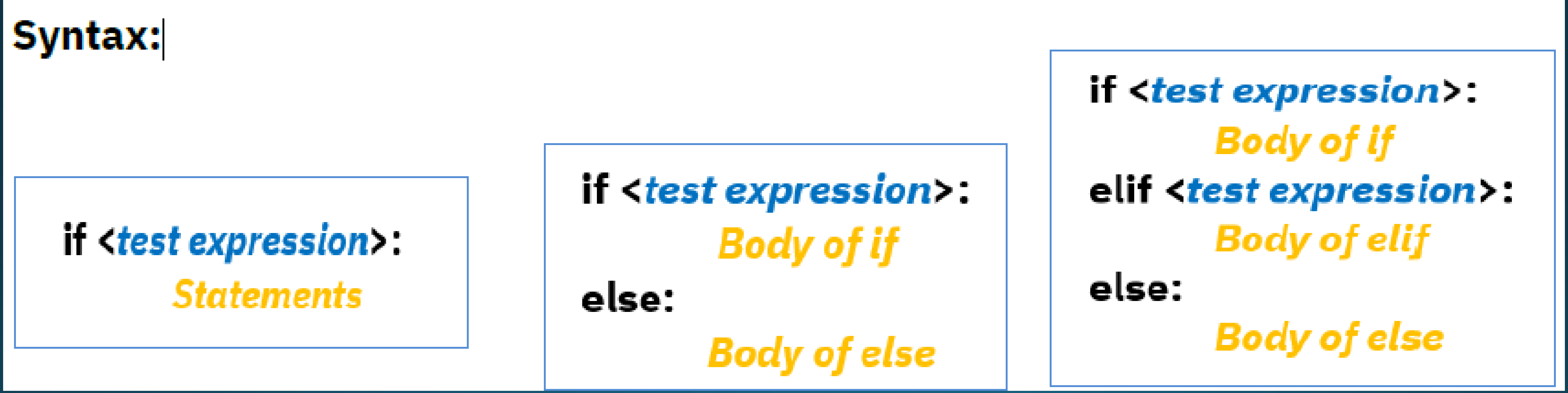
if-else ladder

for

while

Selection Statement

The if/ if..else statement evaluates test expression and the statements written below will execute if the condition is true otherwise the statements below else will get executed. Indentation is used to separate the blocks.



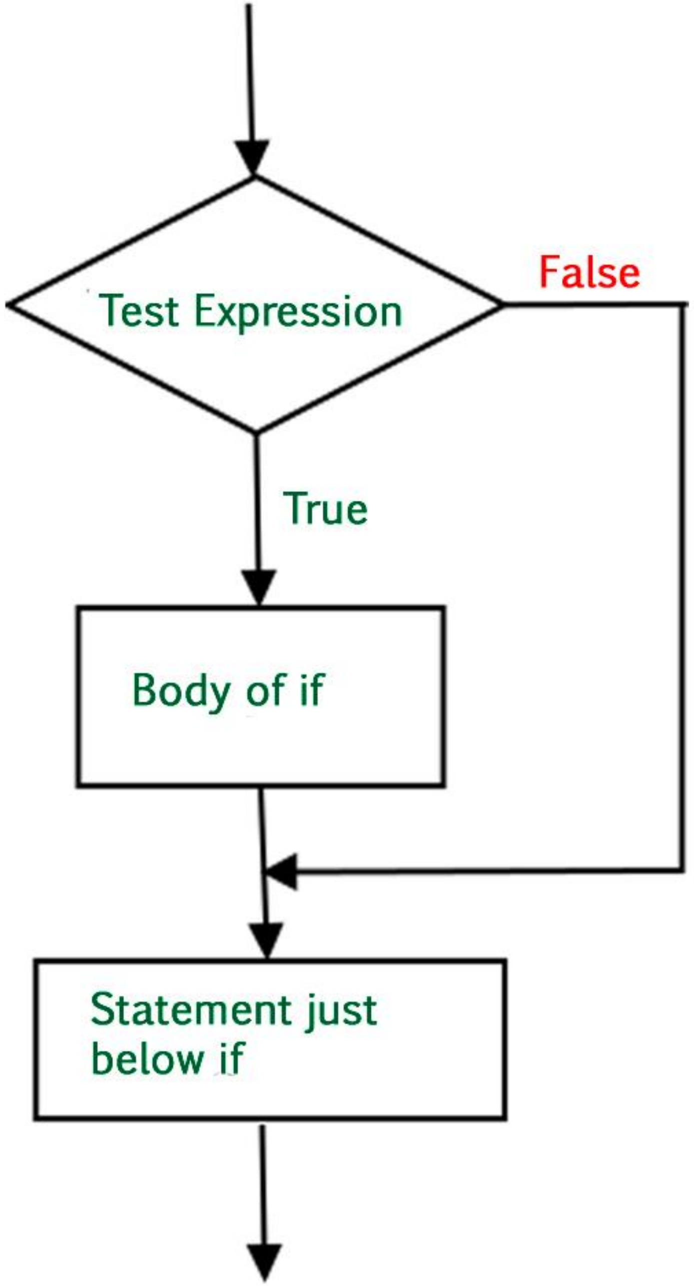
Syntax:

```
if <test expression>:
        Statements
```

```
if <test expression>:
        Body of if
else:
        Body of else
```

```
if <test expression>:
        Body of if
elif <test expression>:
        Body of elif
else:
        Body of else
```

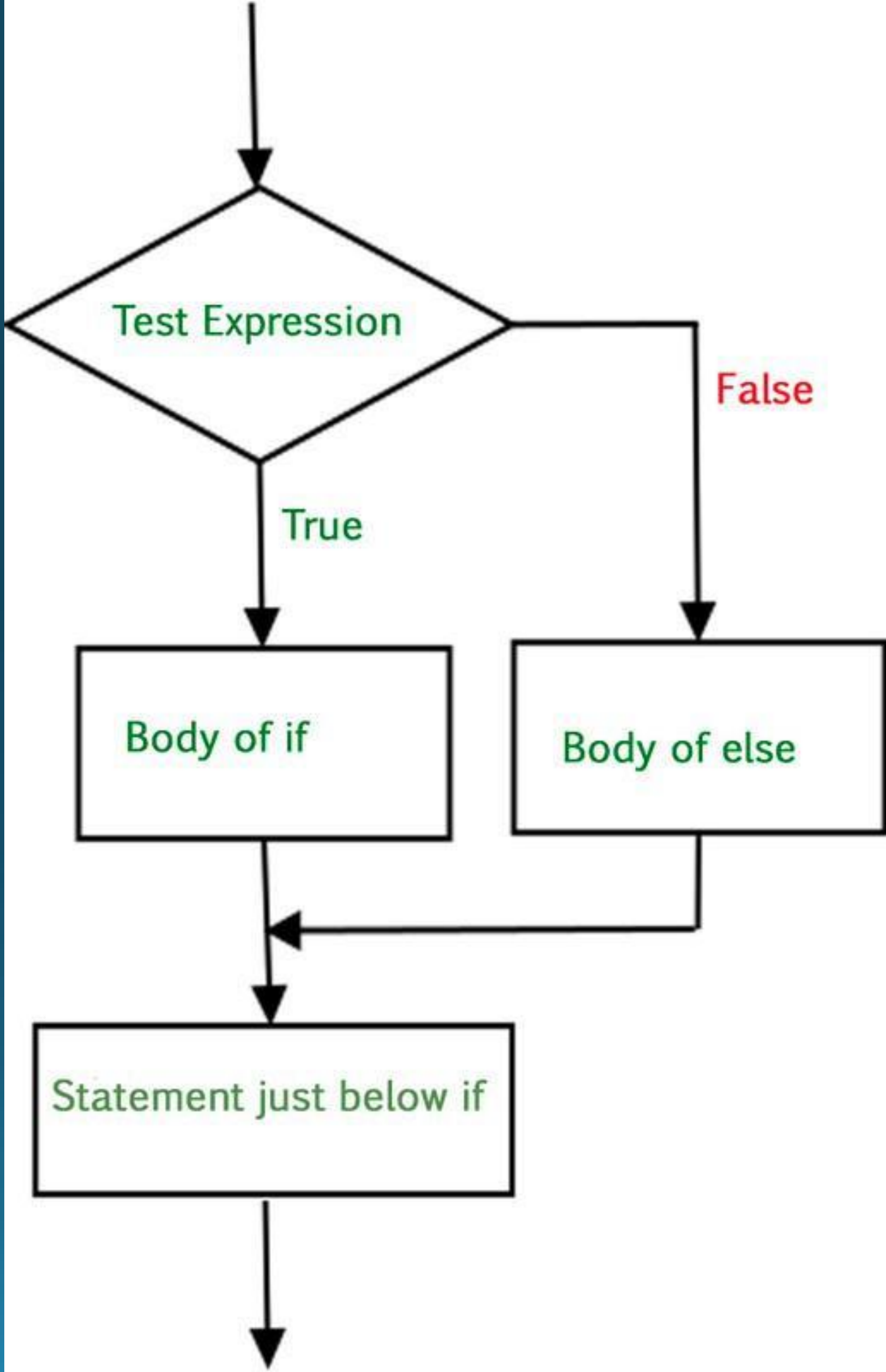
```
#if syntax Python

if condition:
   # Statements to execute if
   # condition is true
```



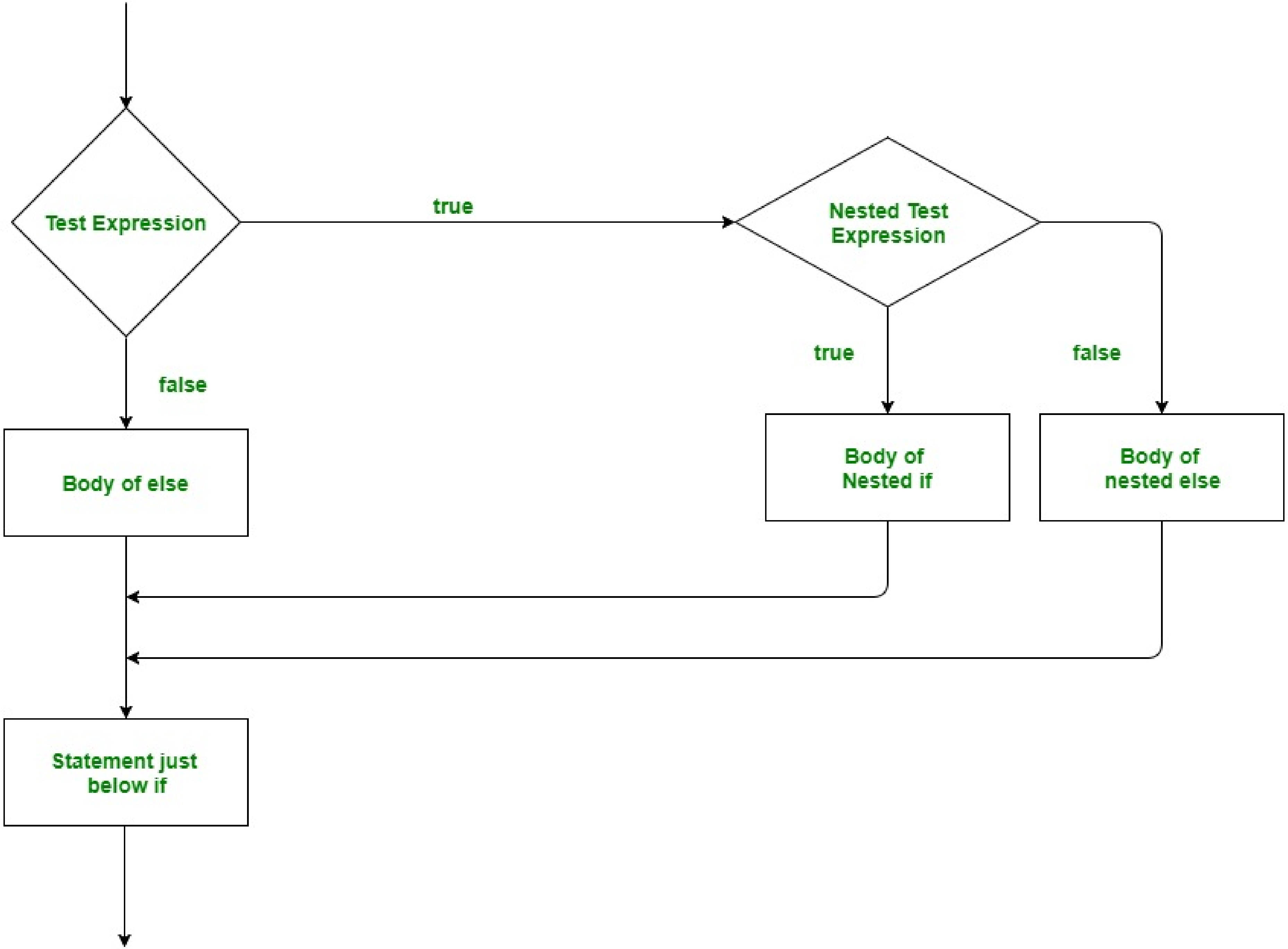
```
# python program to illustrate If statement
i = 10

if (i > 15):
    print("10 is less than 15")
print("I am Not in if")
```

Syntax of If Else in Python

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

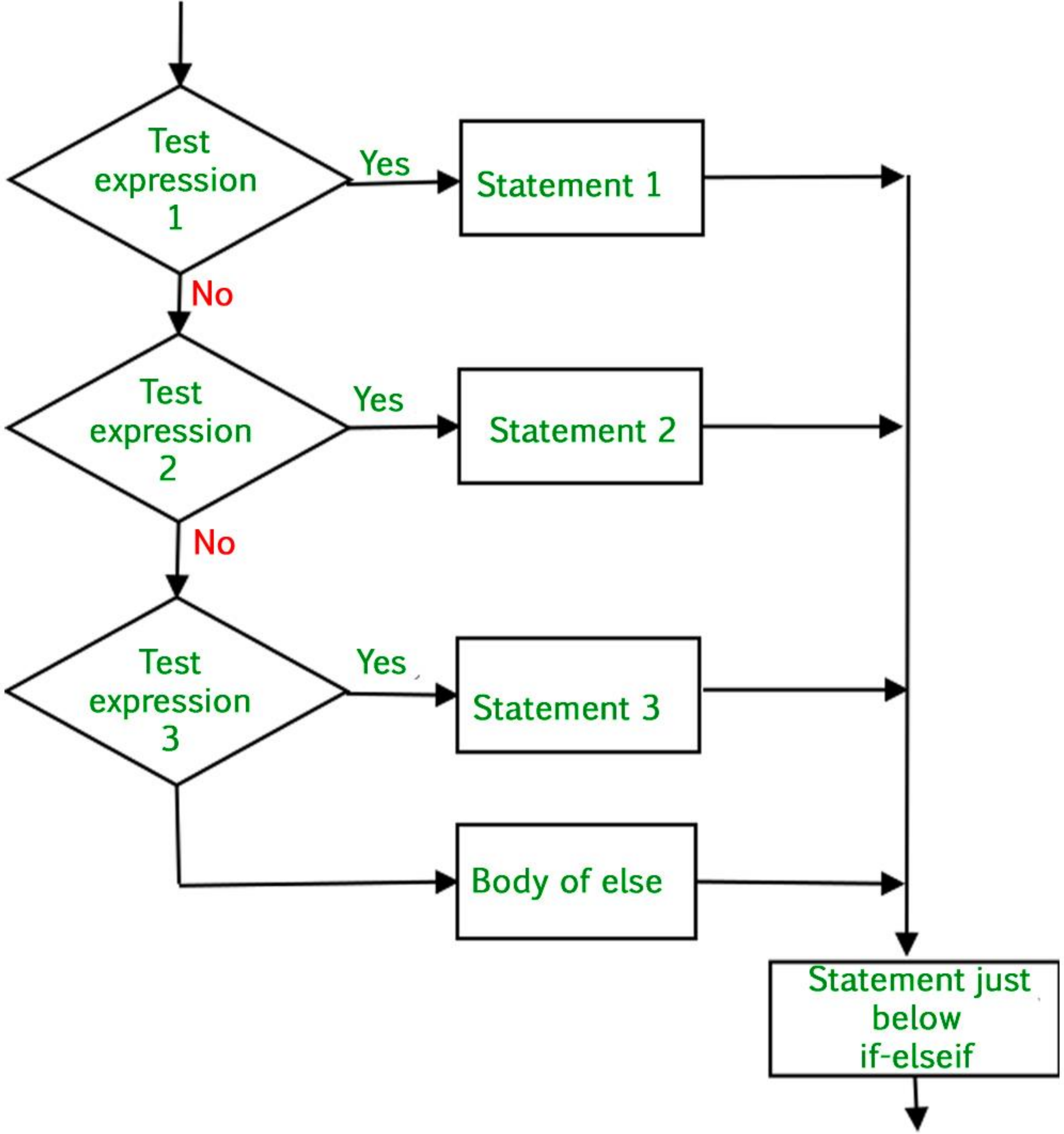


```
# python program to illustrate else if in Python statement

i = 20
if (i < 15):
    print("i is smaller than 15")
    print("i'm in if Block")
else:
    print("i is greater than 15")
    print("i'm in else Block")
print("i'm not in if and not in else Block")
```


Test Expression
true
Nested Test Expression
true
false
false
Body of else
Body of Nested if
Body of nested else
Statement just below if

Syntax:

```
if (condition1):
    # Executes when condition1 is true
    if (condition2):
        # Executes when condition2 is true
    # if Block is end here
# if Block is end here
# python program to illustrate nested If statement

i = 10
if (i == 10):
    #  First if statement
    if (i < 15):
        print("i is smaller than 15")
    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (i < 12):
        print("i is smaller than 12 too")
    else:
        print("i is greater than 15")
```


Test expression 1
Yes
Statement 1
No
Test expression 2
Yes
Statement 2
No
Test expression 3
Yes
Statement 3
Body of else
Statement just below if-elseif

Syntax:

```
if (condition):
    statement
elif (condition):
    statement
.
.
else:
    statement
```

```
# Python program to illustrate if-elif-else ladder

i = 25
if (i == 10):
    print("i is 10")
elif (i == 15):
    print("i is 15")
elif (i == 20):
    print("i is 20")
else:
    print("i is not present")
```

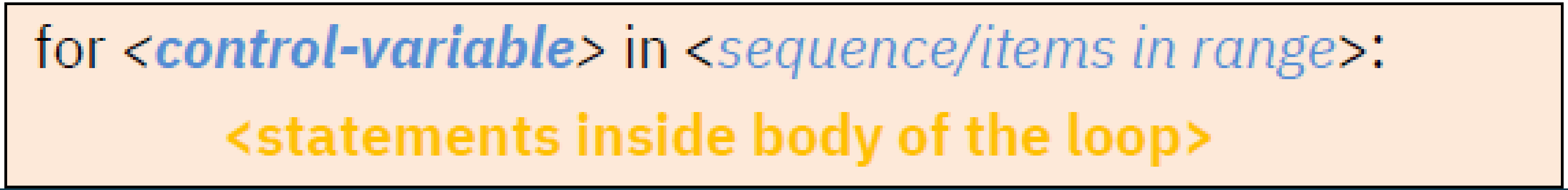
Looping Statements

- Looping statements in programming languages allow you to execute a block of code repeatedly.
- In Python, there are mainly two types of looping statements: for loop and while loop.

For loop

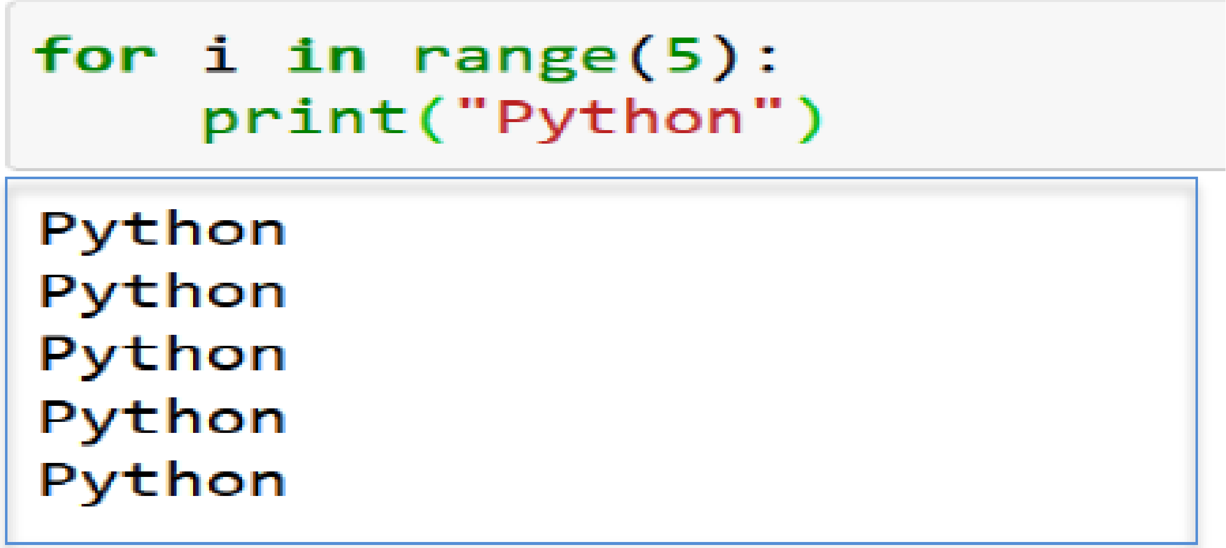
- For loop iterates through a portion of a program based on a sequence, which is an ordered collection of items.
- The “for” keyword is used to start the loop. The loop variable takes on each value in the specified sequence (e.g., list, string, range).
- The colon (:) at the end of the for statement indicates the start of the loop body.
- The statements within the loop body are executed for each iteration.
- Indentation is used to define the scope of the loop body.
- All statements indented under the for statement are considered part of the loop.
- It is advisable to utilize a for loop when the exact number of iterations is known in advance.

Syntax



```
for <control-variable> in <sequence/items in range>:
        <statements inside body of the loop>
```

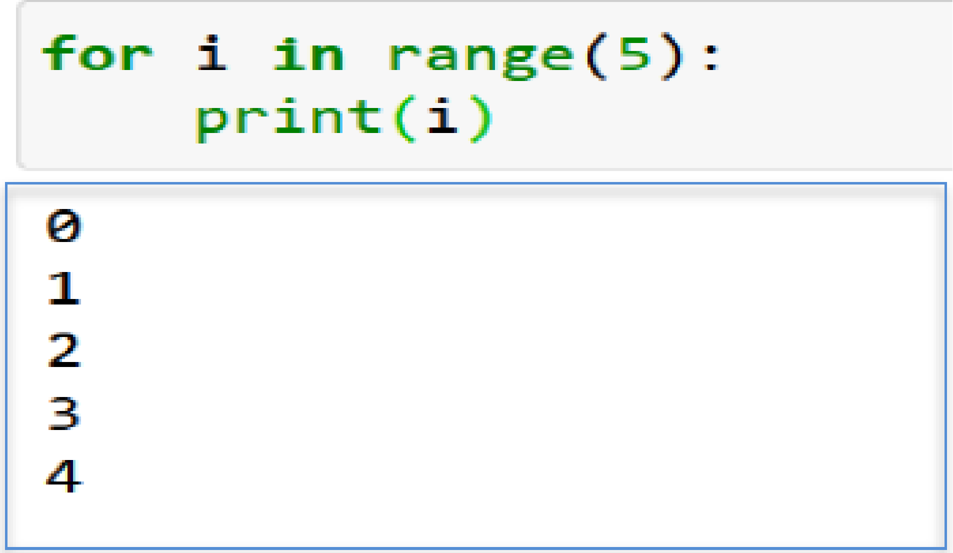
Example -1



```python
for i in range(5):
    print("Python")
```

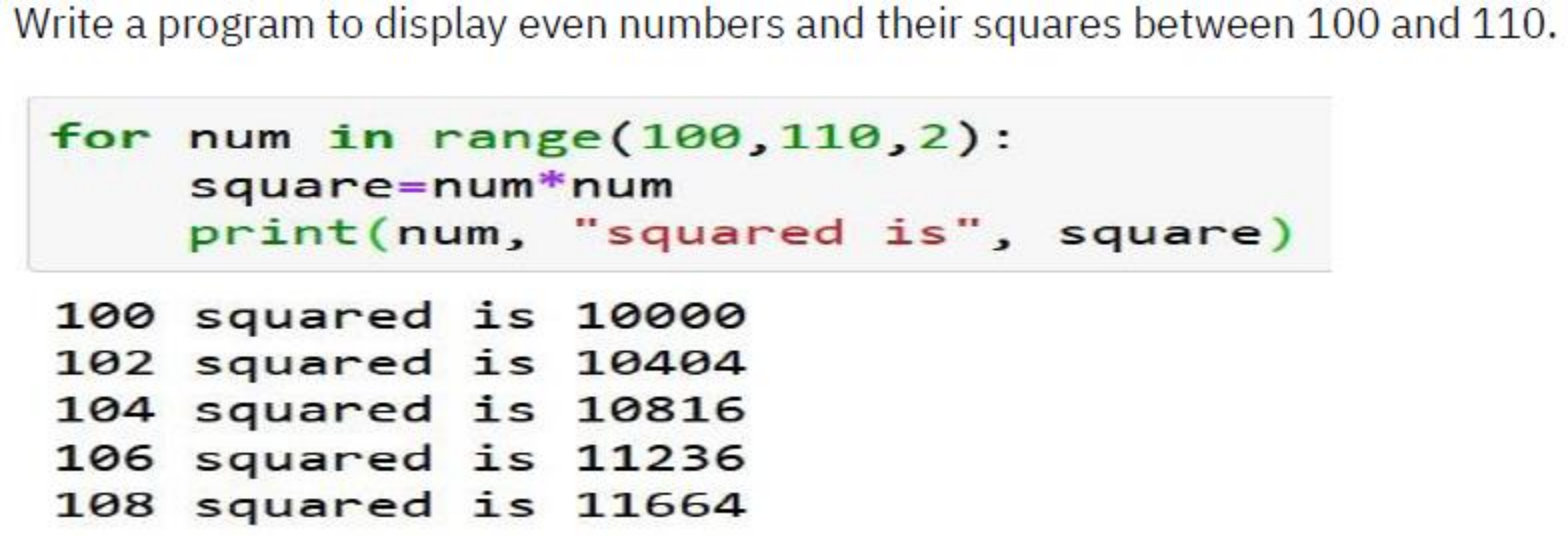
```
Python
Python
Python
Python
Python
```

Example-2



```python
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

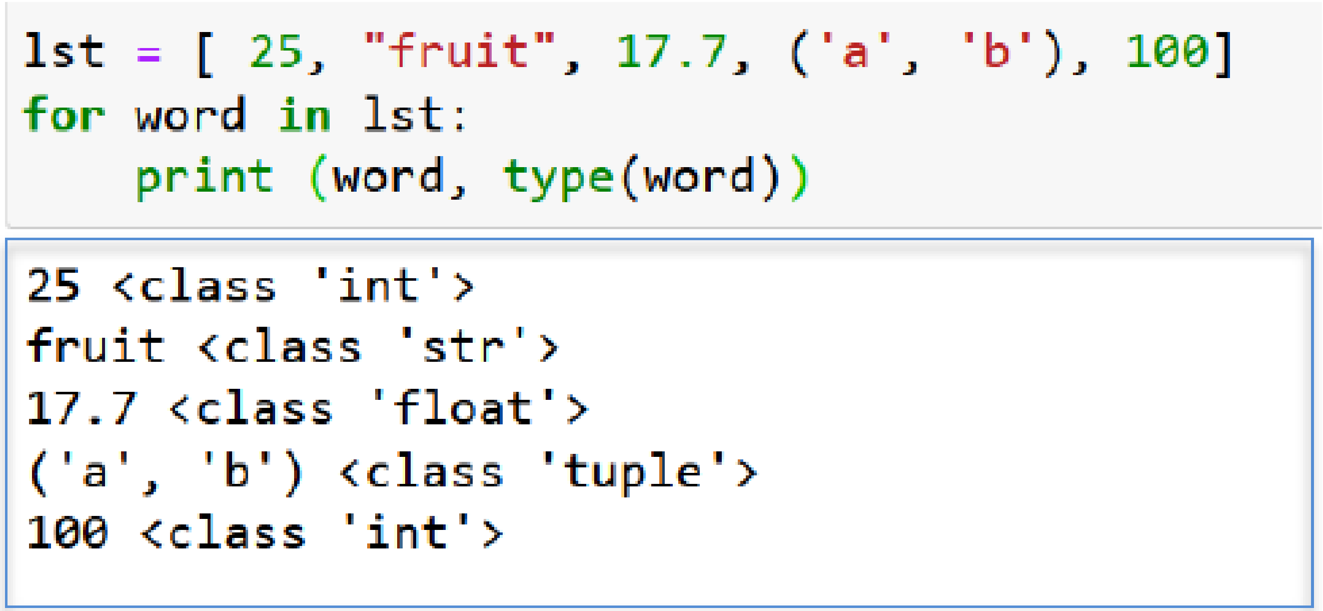


Write a program to display even numbers and their squares between 100 and 110.

```python
for num in range(100,110,2):
    square=num*num
    print(num, "squared is", square)
```

```
100 squared is 10000
102 squared is 10404
104 squared is 10816
106 squared is 11236
108 squared is 11664
```

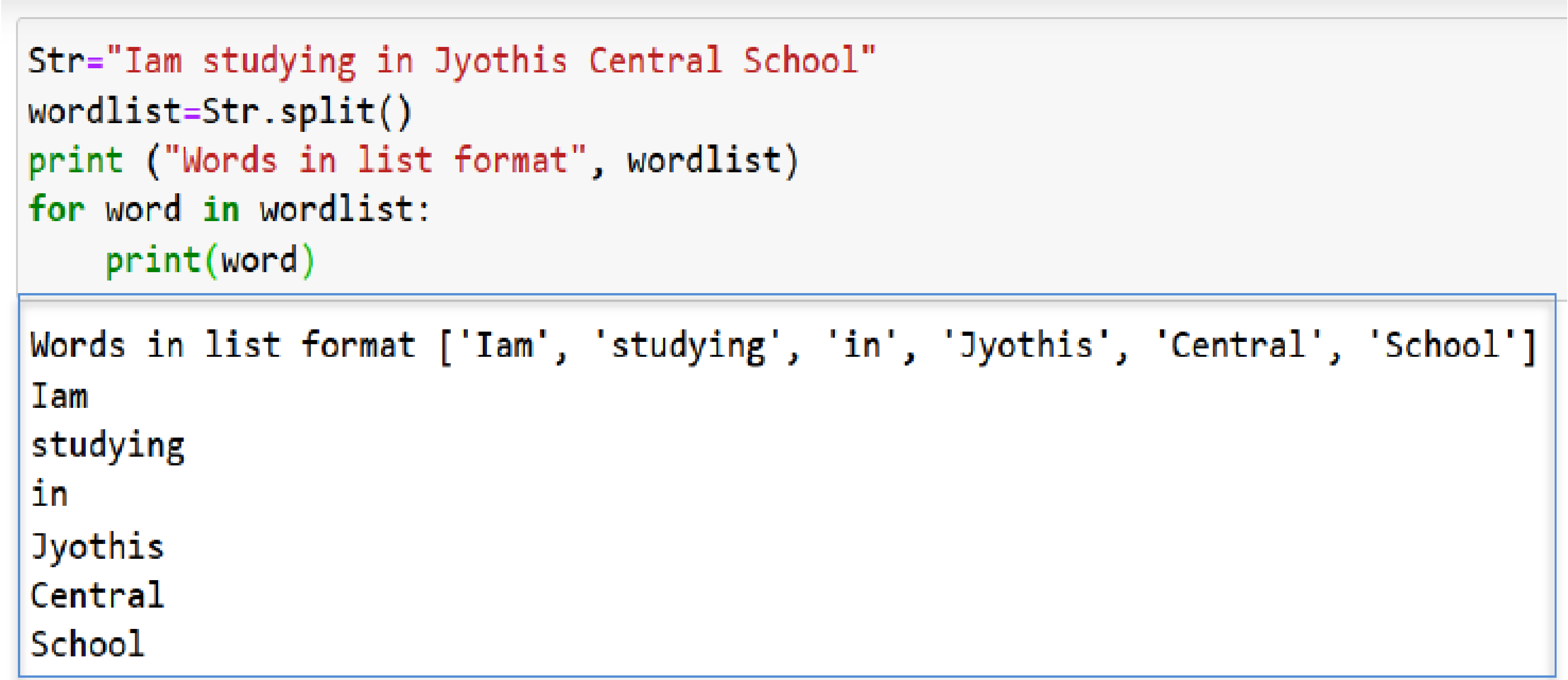
Write a program to read a list, display each element and its type. (*use type( ) to display the data type.*)



```
lst = [ 25, "fruit", 17.7, ('a', 'b'), 100]
for word in lst:
    print (word, type(word))
```

```
25 <class 'int'>
fruit <class 'str'>
17.7 <class 'float'>
('a', 'b') <class 'tuple'>
100 <class 'int'>
```

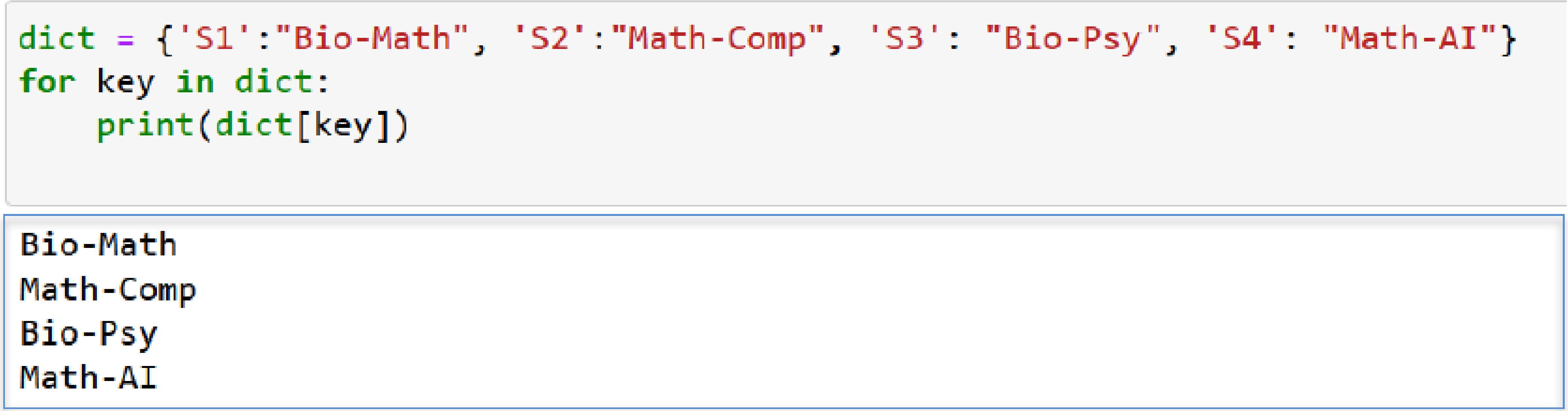
Write a program to read a string. Split the string into list of words and display each word.



```python
Str="Iam studying in Jyothis Central School"
wordlist=Str.split()
print ("Words in list format", wordlist)
for word in wordlist:
    print(word)
```

```
Words in list format ['Iam', 'studying', 'in', 'Jyothis', 'Central', 'School']
Iam
studying
in
Jyothis
Central
School
```

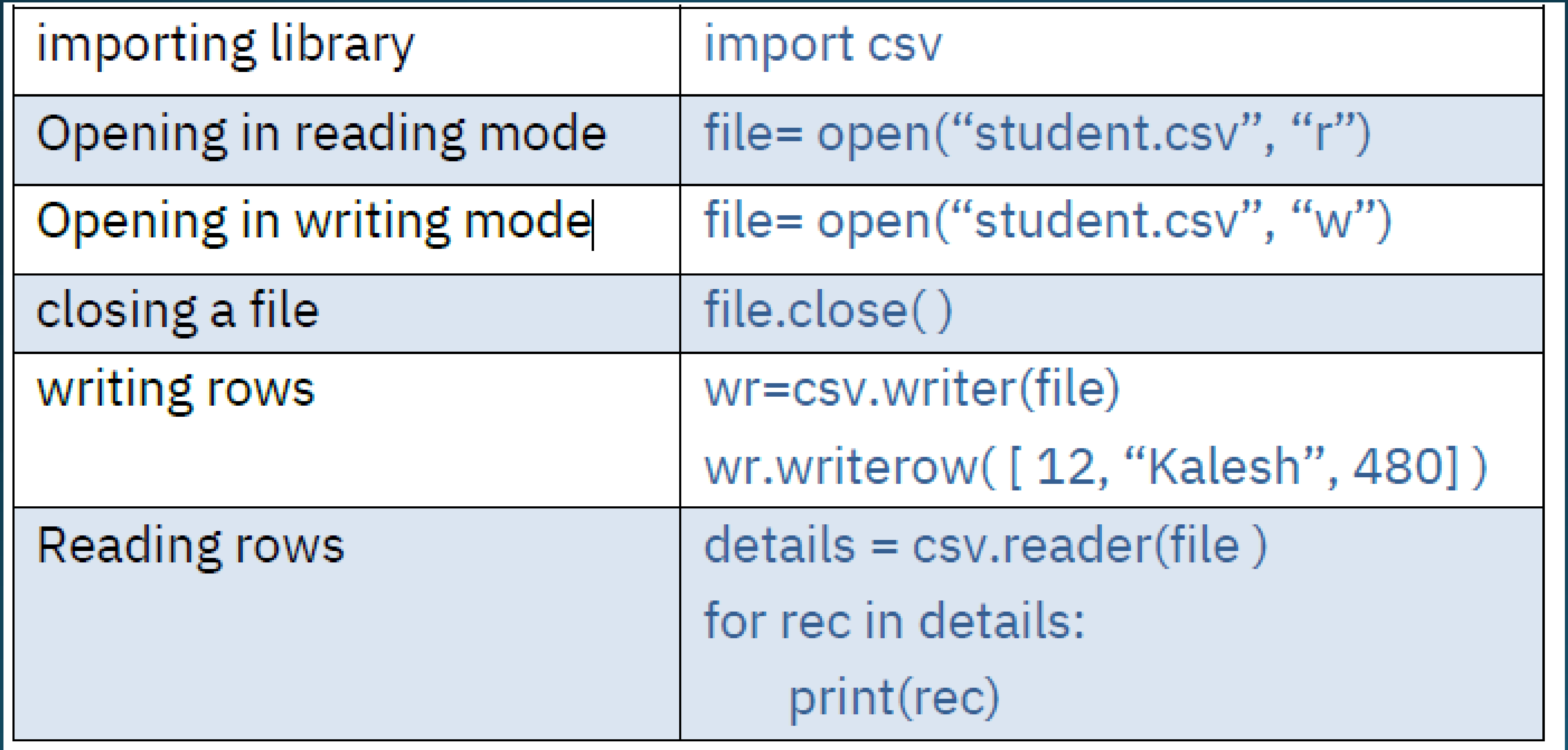
Write a simple program to display the values stored in dictionary



```python
dict = {'S1':"Bio-Math", 'S2':"Math-Comp", 'S3': "Bio-Psy", 'S4': "Math-AI"}
for key in dict:
    print(dict[key])
```

```
Bio-Math
Math-Comp
Bio-Psy
Math-AI
```

UNDERSTANDING CSV file (Comma Separated Values)



importing library	import csv
Opening in reading mode	file= open(“student.csv”, “r”)
Opening in writing mode	file= open(“student.csv”, “w”)
closing a file	file.close( )

writing rows	wr=csv.writer(file)
	wr.writerow([12, “Kalesh”, 480])

Reading rows	details = csv.reader(file)
	for rec in details:
	    print(rec)

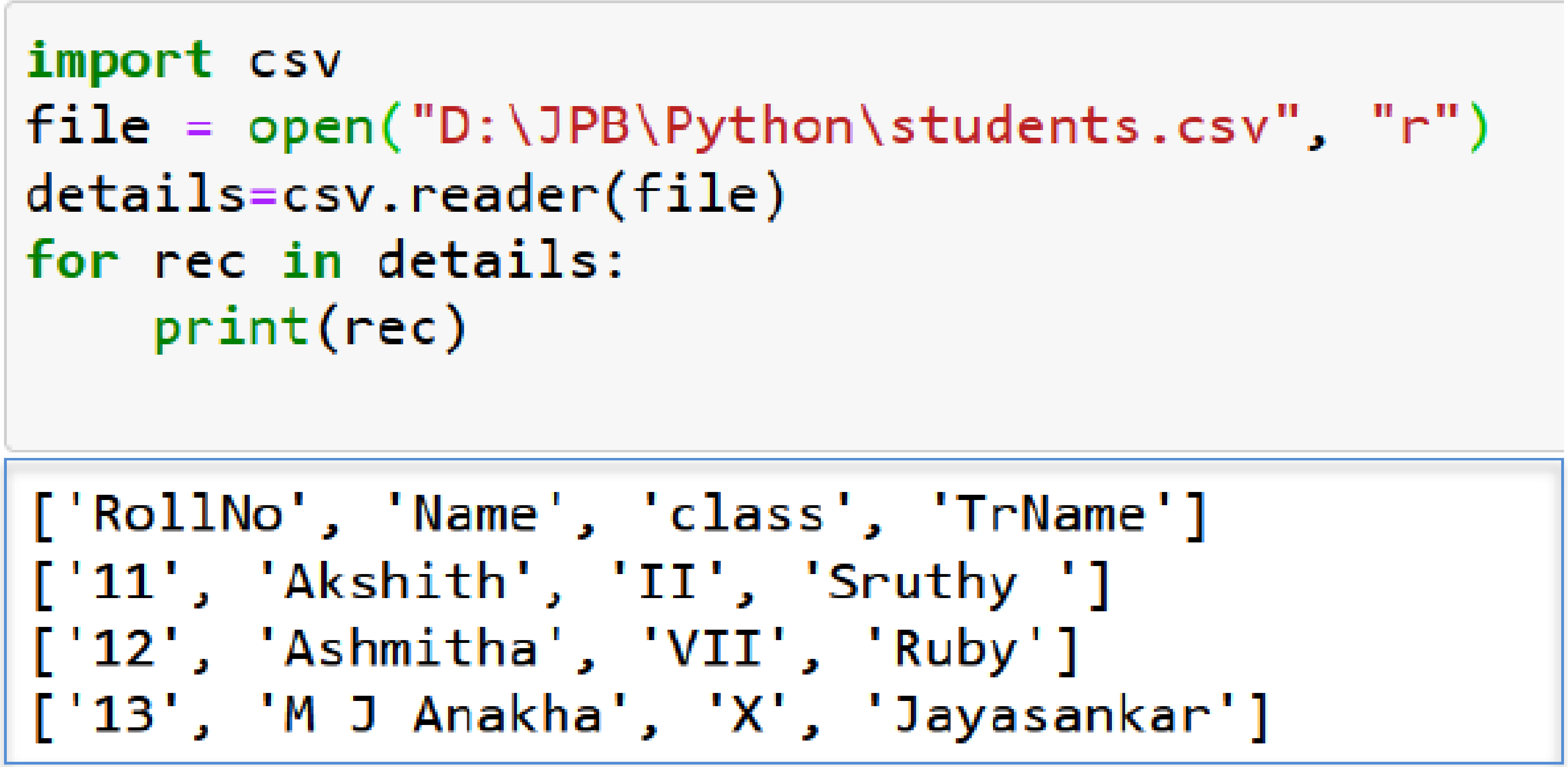
- CSV files are delimited files that store tabular data (data stored in rows and columns). It looks similar to spread sheets, but internally it is stored in a different format. In csv file, values are separated by comma.

- Data Sets used in AI programming are easily saved in csv format. Each line in a csv file is a data record.

- Each record consists of more than one fields(columns).

- The csv module of Python provides functionality to read and write tabular data in CSV format.

Write a Program to open a csv file students.csv and display its details

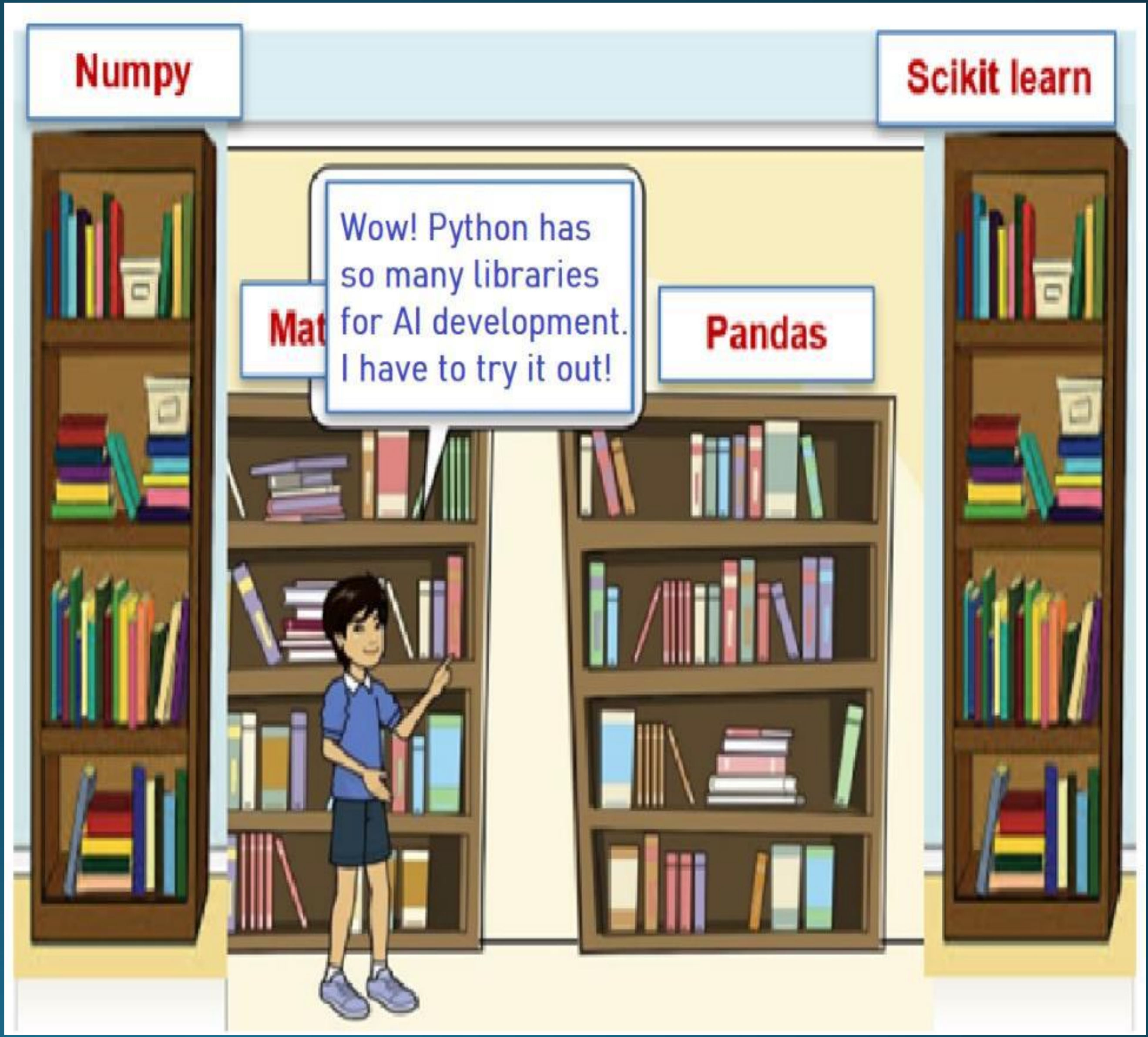


```python
import csv
file = open("D:\JPB\Python\students.csv", "r")
details=csv.reader(file)
for rec in details:
    print(rec)
```

```
['RollNo', 'Name', 'class', 'TrName']
['11', 'Akshith', 'II', 'Sruthy ']
['12', 'Ashmitha', 'VII', 'Ruby']
['13', 'M J Anakha', 'X', 'Jayasankar']
```

INTRODUCING LIBRARIES

- A library in Python typically refers to a collection of reusable modules or functions that provide specific functionality.
- Libraries are designed to be used in various projects to simplify development by providing pre-written code for common tasks. Concept of libraries are very easy to understand.


Numpy
Scikit learn
Mat
Wow! Python has so many libraries for AI development. I have to try it out!
Pandas

- In Python, functions are organized within libraries similar to how library books are arranged by subjects such as physics, computer science, and economics.
- For example, the "math" library contains numerous functions like sqrt(), pow(), abs(), and sin(), which facilitate mathematical operations and calculations.
- To utilize a library in a program, it must be imported.
- For example, if we wish to use the sqrt() function in our program, we include the statement "import math".
- This allows us to access and utilize the functionalities provided by the math library.

NUMPY

- NumPy, which stands for Numerical Python, is a powerful library in Python used for numerical computing.

- It is a general-purpose array-processing package.

- NumPy provides the ndarray (N-dimensional array) data structure, which represents arrays of any dimension.

- These arrays are homogeneous (all elements are of the same data type) and can contain elements of various numerical types (integers, floats, etc.)

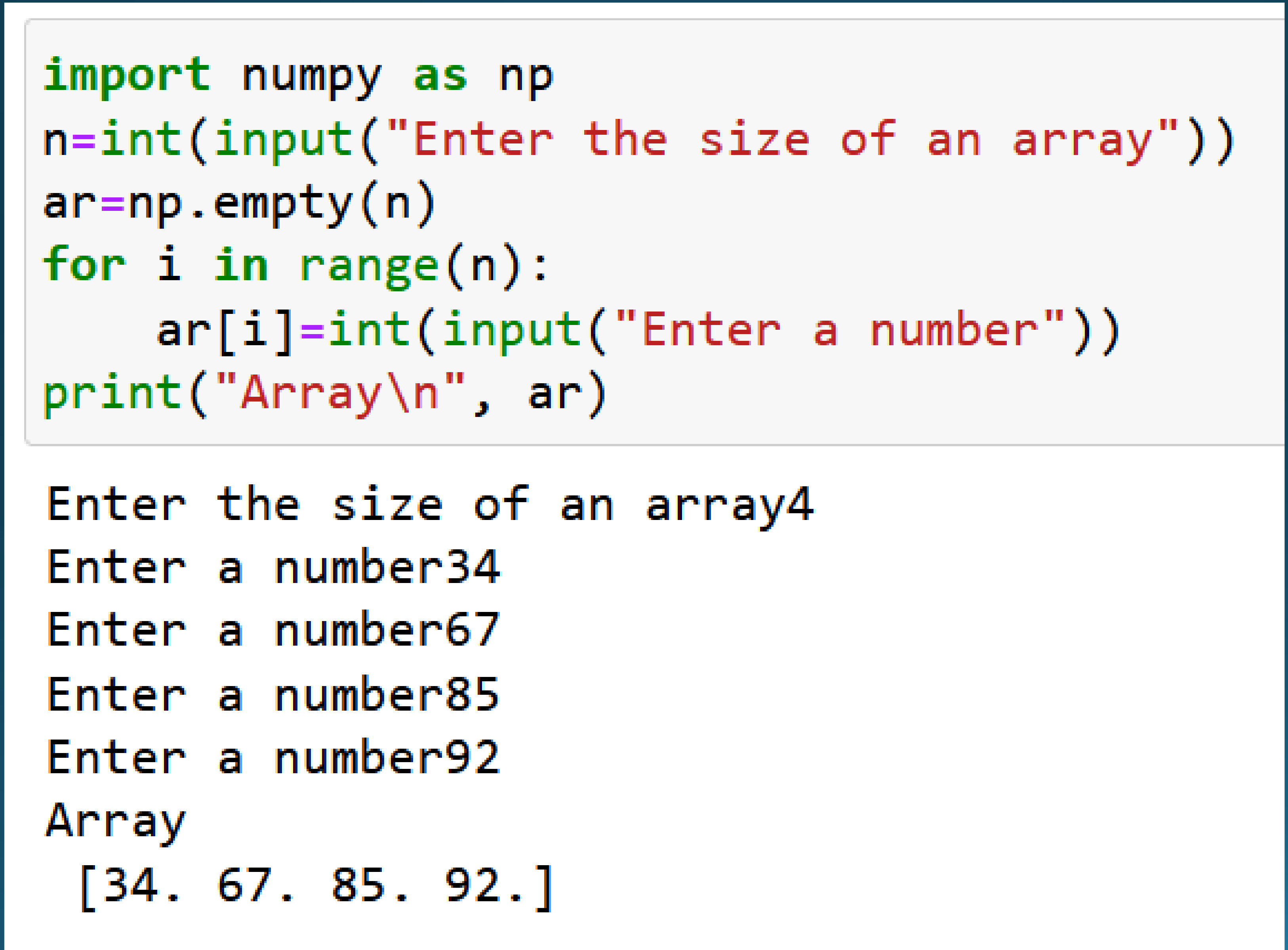
Where and why do we use the Numpy library in Artificial Intelligence?

- Suppose you have a dataset containing exam scores of students in various subjects, and you want to perform some basic analysis on this data.
- You can utilize NumPy arrays to store exam scores for different subjects efficiently.
- With NumPy's array operations, you can perform various calculations such as calculating average scores for each subject, finding total scores for each student, calculating the overall average score across all subjects., identifying the highest and lowest scores.
- NumPy's array operations streamline these computations, making them both efficient and convenient.
- This makes NumPy an indispensable tool for data manipulation and analysis in data science applications.

NumPy can be installed using Python's package manager, pip.

```
pip install numpy
```

Using values from the user (using empty( )-- The empty() function in Python is used to return a new array of a given size)

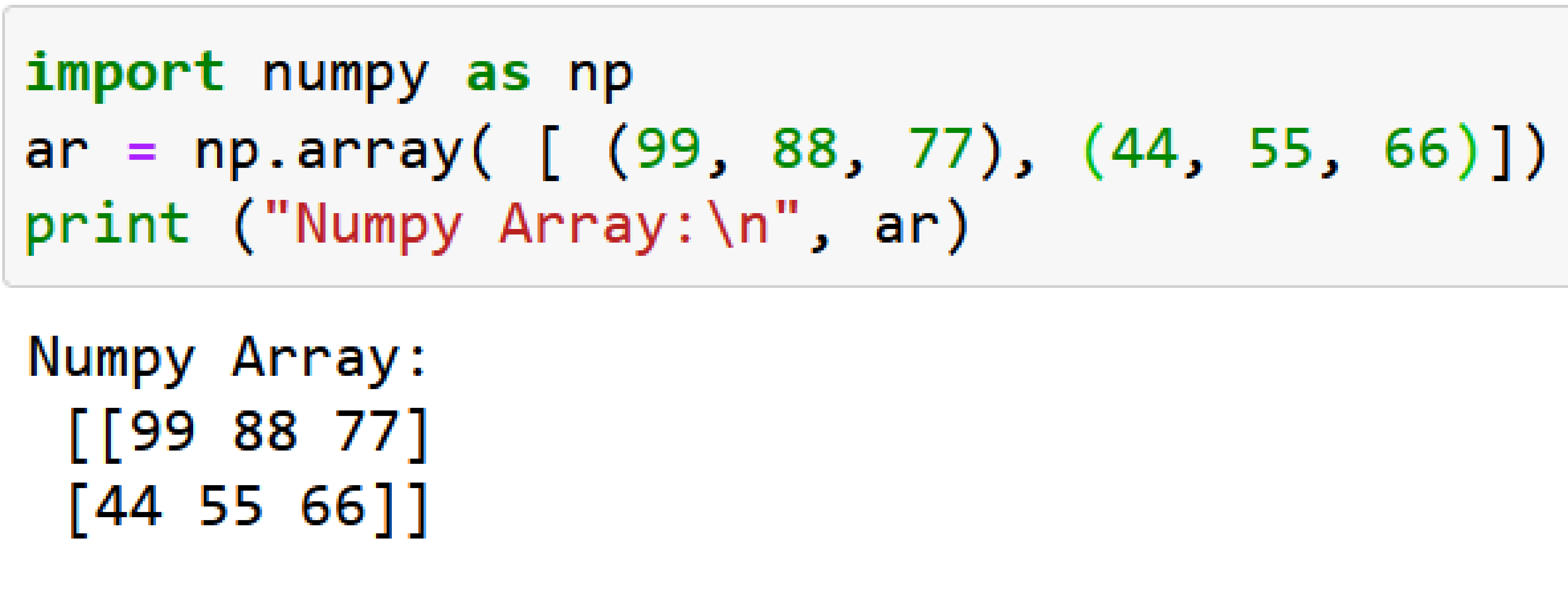


```
import numpy as np
n=int(input("Enter the size of an array"))
ar=np.empty(n)
for i in range(n):
    ar[i]=int(input("Enter a number"))
print("Array\n", ar)
```

```
Enter the size of an array4
Enter a number34
Enter a number67
Enter a number85
Enter a number92
Array
 [34. 67. 85. 92.]
```

Creating a Numpy Array - Arrays in Numpy can be created by multiple ways. Some of the ways are programmed here:

Using List of Tuples NUMPY



```
import numpy as np
ar = np.array( [ (99, 88, 77), (44, 55, 66)])
print ("Numpy Array:\n", ar)
```

```
Numpy Array:
 [[99 88 77]
 [44 55 66]]
```

PANDAS

- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis“ .
- Pandas is a powerful and versatile library that simplifies tasks of data manipulation in Python .
- Pandas is built on top of the NumPy library which means that a lot of structures of NumPy are used or replicated in Pandas and Pandas is particularly well-suited for working with tabular data, such as spreadsheets or SQL tables.
- Its versatility and ease of use make it an essential tool for data analysts, scientists, and engineers working with structured data in Python.

Where and why do we use the Pandas library in Artificial Intelligence?

- Suppose you have a dataset containing information about various marketing campaigns conducted by the company, such as campaign type, budget, duration, reach, engagement metrics, and sales performance.
- We use Pandas to load the dataset, display summary statistics, and perform group-wise analysis to understand the performance of different marketing campaigns.
- We then visualize the sales performance and average engagement metrics for each campaign type using Matplotlib, a popular plotting library in Python.
- Pandas provides powerful data manipulation and aggregation functionalities, making it easy to perform complex analysis and generate insightful visualizations.
- This capability is invaluable in AI and data-driven decision-making processes, allowing businesses to gain actionable insights from their data.

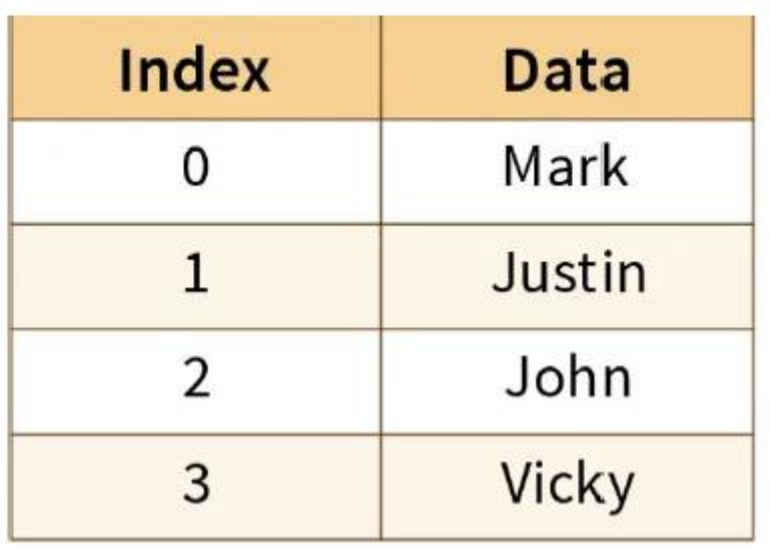
Pandas can be installed using:

```
pip install pandas
```

Pandas generally provide two data structures for manipulating data.
They are: Series and Data Frame.

Series

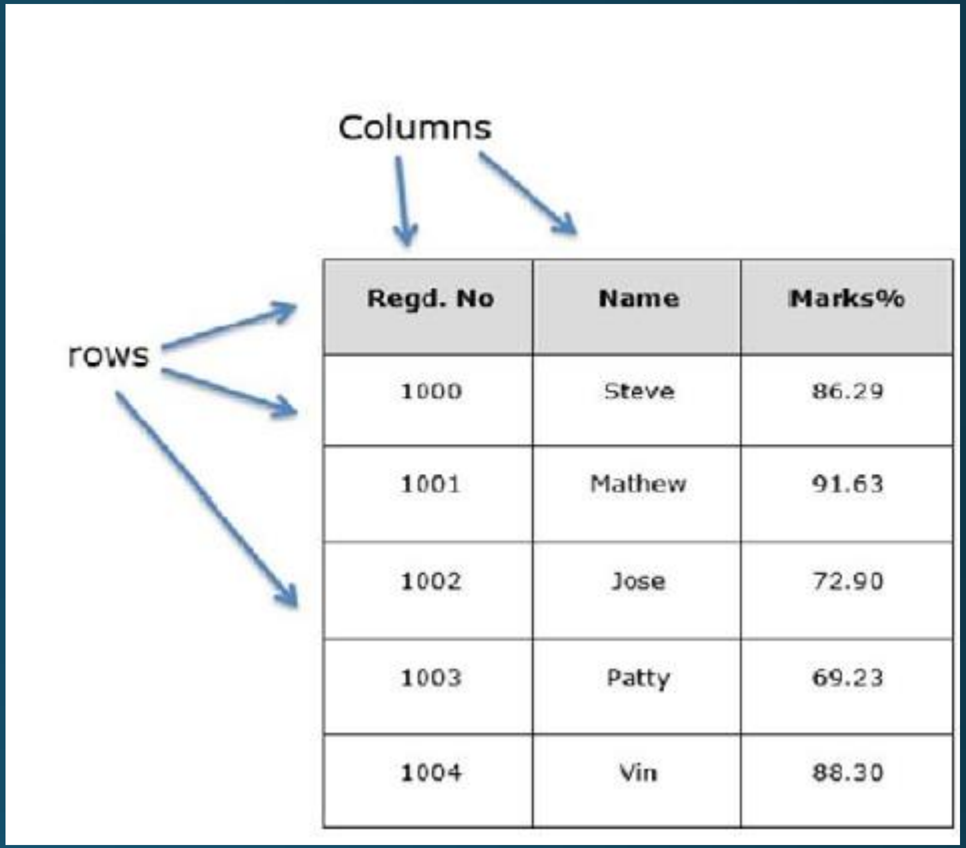
- A Series is a one-dimensional array containing a sequence of values of any data type (int, float, list, string, etc.) which by default have numeric data labels starting from zero.
- The data label associated with a particular value is called its index. We can also assign values of other data types as index.
- We can imagine a Pandas Series as a column in a spreadsheet as given here.



Index	Data
0	Mark
1	Justin
2	John
3	Vicky

- In data science, we often encounter datasets with two-dimensional structures.
- This is where Pandas Data Frames come into play.
- A Data Frame is used when we need to work on multiple columns at a time, i.e., we need to process the tabular data.

- For example, the result of a class, items in a restaurant’s menu, reservation chart of a train, etc.
- A DataFrame is a two-dimensional labeled data structure like a table of MySQL.
- It contains rows and columns, and therefore has both a row and column index.
- Each column can have a different type of value such as numeric, string, boolean, etc., as in tables of a database.

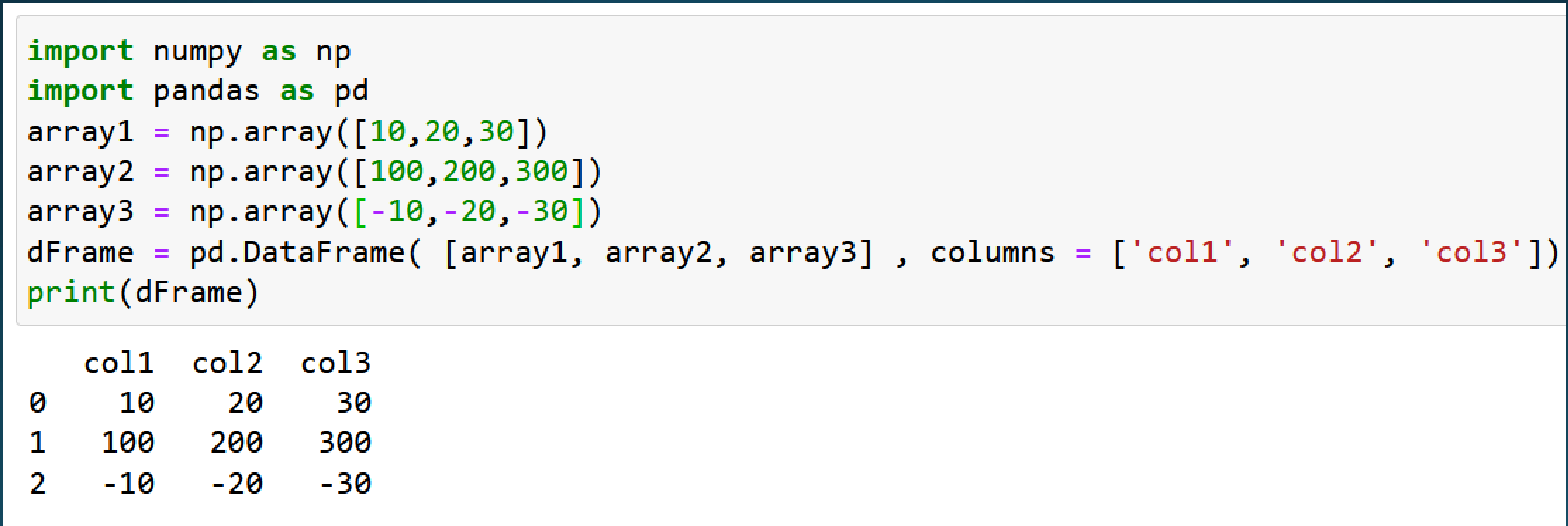


Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

Creation of DataFrame

There are several methods to create a DataFrame in Pandas, but here we will discuss two common approaches:

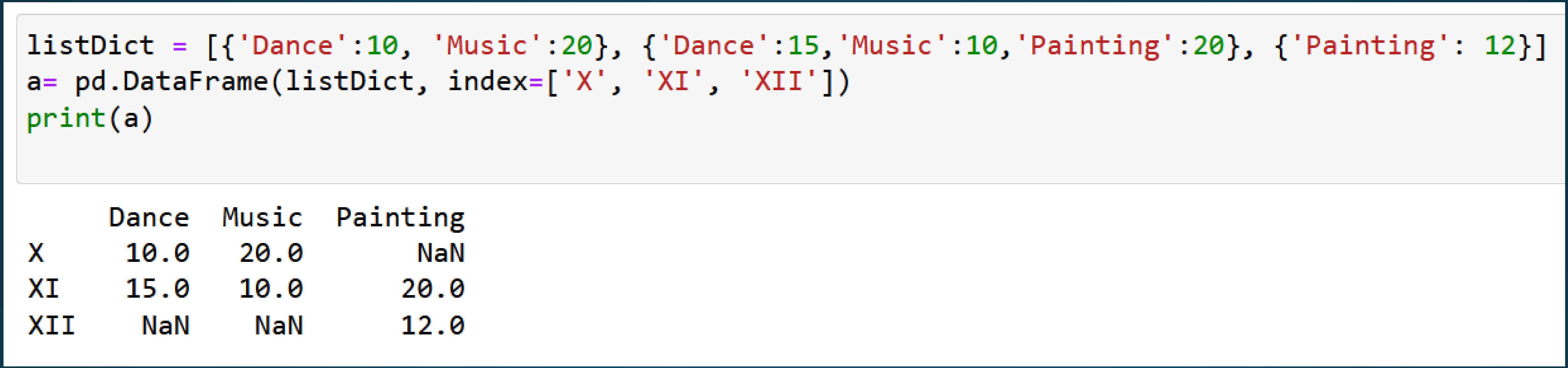
Using NumPy ndarrays-



```python
import numpy as np
import pandas as pd
array1 = np.array([10,20,30])
array2 = np.array([100,200,300])
array3 = np.array([-10,-20,-30])
dFrame = pd.DataFrame( [array1, array2, array3] , columns = ['col1', 'col2', 'col3'])
print(dFrame)
```

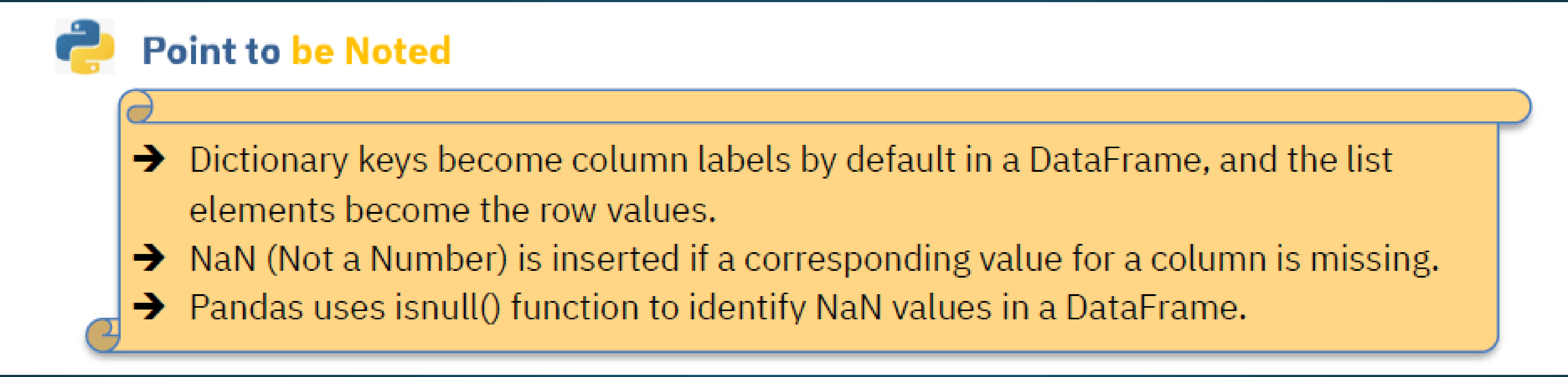
```
   col1  col2  col3
0    10    20    30
1   100   200   300
2   -10   -20   -30
```

Using List of Dictionaries



```
listDict = [{'Dance':10, 'Music':20}, {'Dance':15,'Music':10,'Painting':20}, {'Painting': 12}]
a= pd.DataFrame(listDict, index=['X', 'XI', 'XII'])
print(a)
```

```
     Dance  Music  Painting
X     10.0   20.0       NaN
XI    15.0   10.0      20.0
XII    NaN    NaN      12.0
```



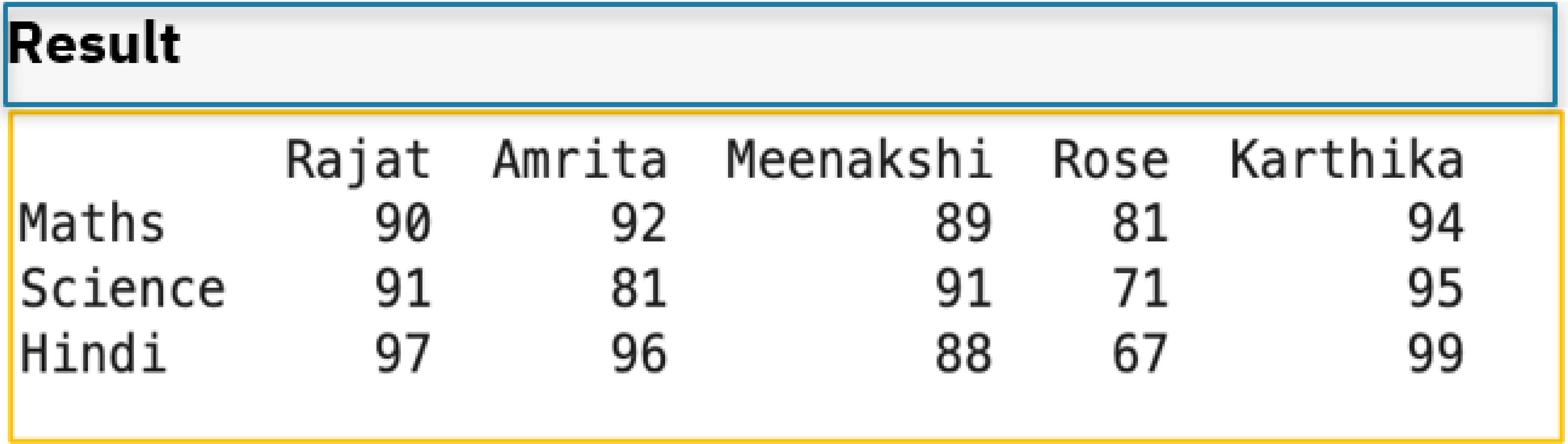
Point to be Noted

- ➔ Dictionary keys become column labels by default in a DataFrame, and the list elements become the row values.
- ➔ NaN (Not a Number) is inserted if a corresponding value for a column is missing.
- ➔ Pandas uses isnull() function to identify NaN values in a DataFrame.

Dealing with Rows and Columns

- Based on the DataFrame 'Result' provided below, we can observe various operations related to rows and columns.
- Each operation statement is accompanied by its corresponding output from the Result DataFrame.

DataFrame: Result

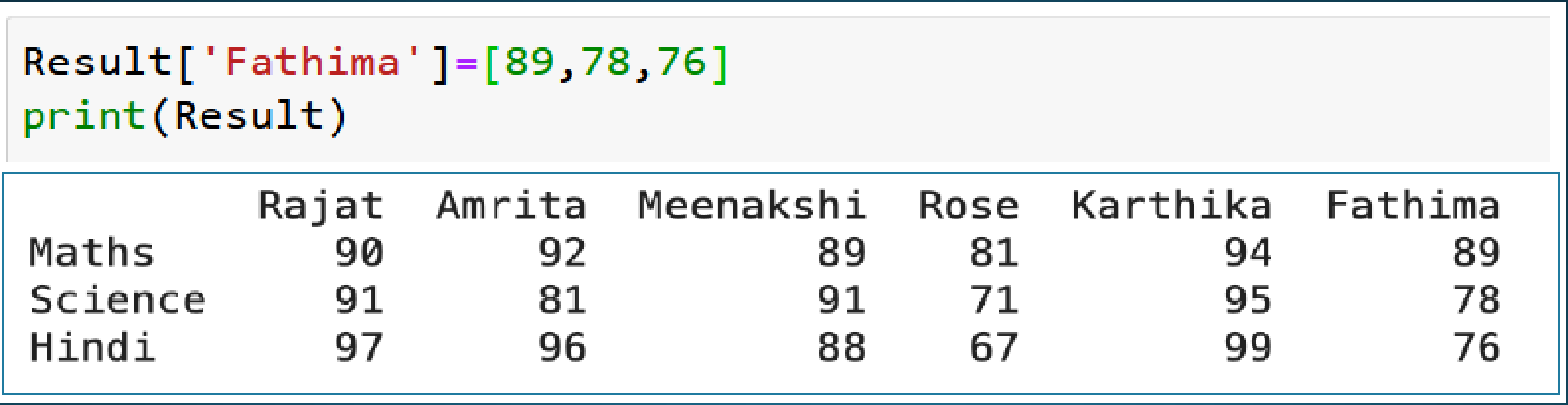


Result

```
         Rajat  Amrita  Meenakshi  Rose  Karthika
Maths       90      92         89    81        94
Science     91      81         91    71        95
Hindi       97      96         88    67        99
```

Adding a New Column to a DataFrame:

We can add a new column ‘Fathima’, by mentioning column name as given below:



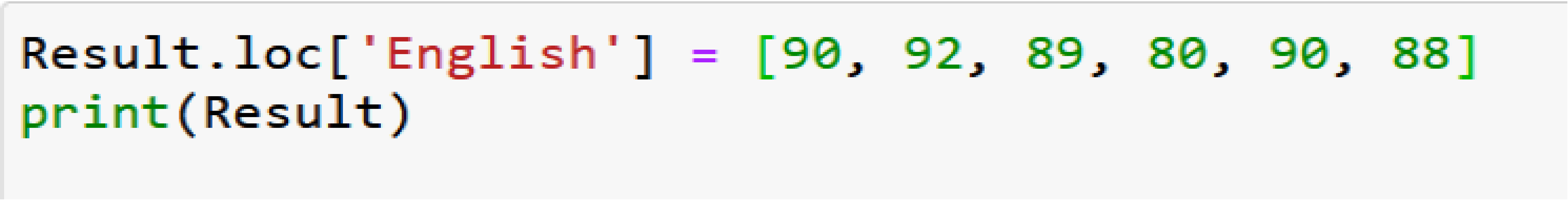
```
Result['Fathima']=[89,78,76]
print(Result)
```

```
         Rajat  Amrita  Meenakshi  Rose  Karthika  Fathima
Maths       90      92         89    81        94       89
Science     91      81         91    71        95       78
Hindi       97      96         88    67        99       76
```

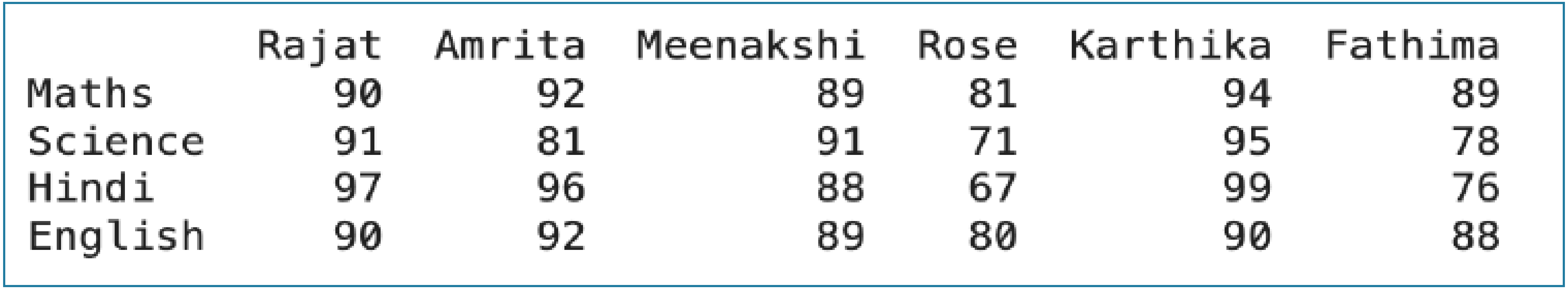
Adding a New Row to a DataFrame:

We can add a new row to a DataFrame using the DataFrame.loc[ ] method.

Let us add marks for English subject in Result ➔



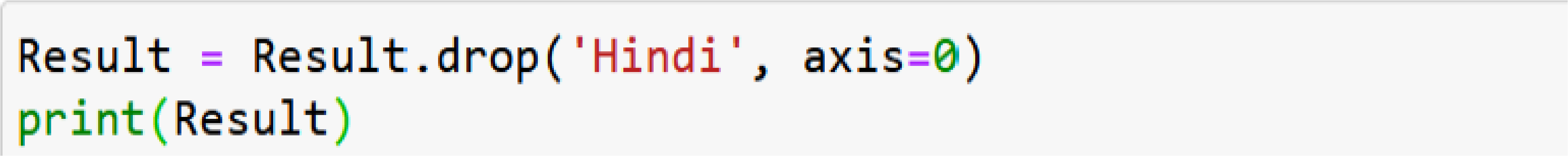
```
Result.loc['English'] = [90, 92, 89, 80, 90, 88]
print(Result)
```



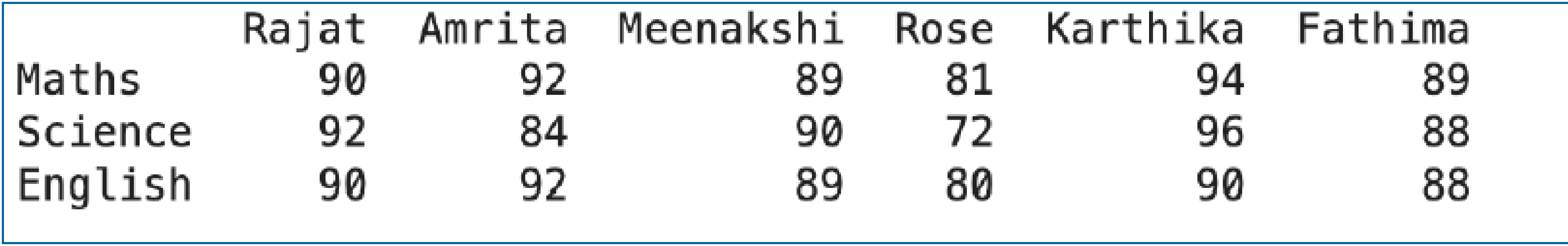
```
         Rajat  Amrita  Meenakshi  Rose  Karthika  Fathima
Maths       90      92         89    81        94       89
Science     91      81         91    71        95       78
Hindi       97      96         88    67        99       76
English     90      92         89    80        90       88
```

Deleting Rows and Columns from a DataFrame:

- We need to specify the names of the labels to be dropped and the axis from which they need to be dropped.
- To delete a row, the parameter axis is assigned the value 0 and for deleting a column, the parameter axis is assigned the value 1.
- Deleting a row “Hindi”

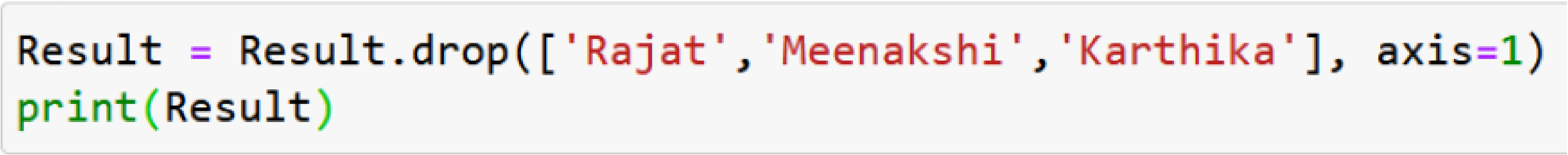


```
Result = Result.drop('Hindi', axis=0)
print(Result)
```

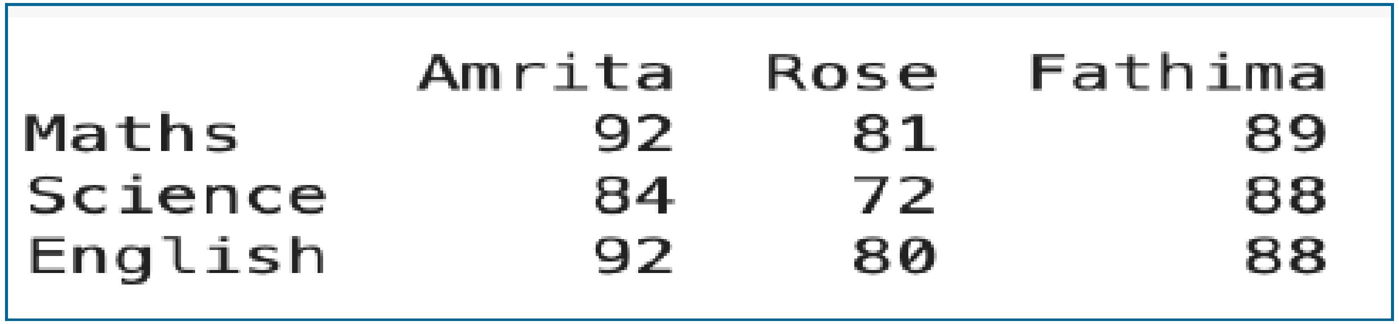


```
         Rajat  Amrita  Meenakshi  Rose  Karthika  Fathima
Maths       90      92         89    81        94       89
Science     92      84         90    72        96       88
English     90      92         89    80        90       88
```

Delete the columns having labels ‘Rajat’, 'Meenakshi' and ‘Karthika”:



```
Result = Result.drop(['Rajat','Meenakshi','Karthika'], axis=1)
print(Result)
```



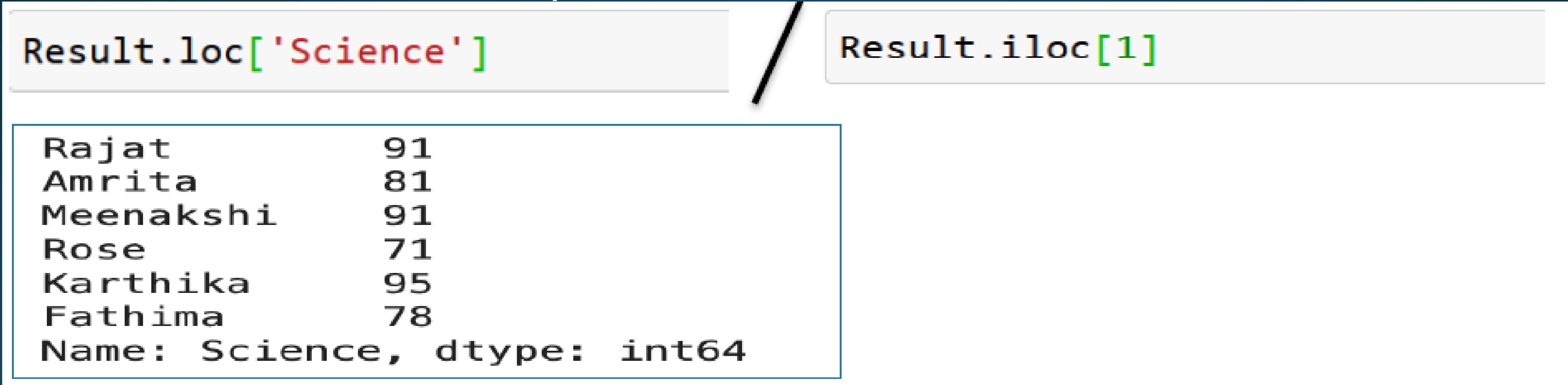
```
         Amrita  Rose  Fathima
Maths        92    81       89
Science      84    72       88
English      92    80       88
```

Point to be Noted

During Data Analysis, DataFrame.drop() method is used to remove the rows and columns.

Accessing DataFrame Elements

- Data elements in a DataFrame can be accessed using different ways.
- Two common ways of accessing are using loc and iloc. DataFrame.loc[ ] uses label names for accessing and DataFrame.iloc[ ] uses the index position for accessing the elements of a DataFrame.
- Let us check an example:



```
Result.loc['Science']
```

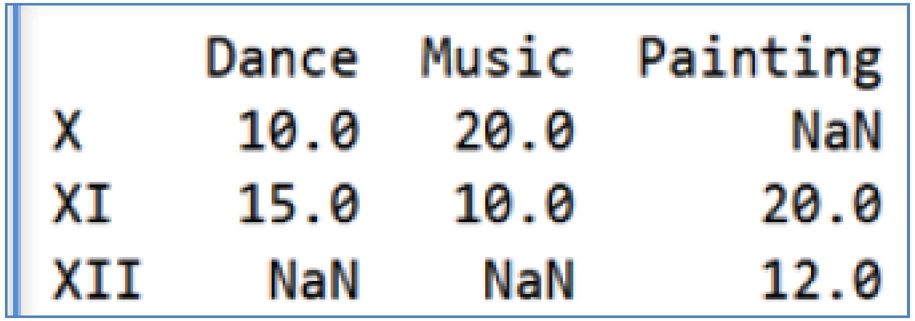
/

```
Result.iloc[1]
```

```
Rajat           91
Amrita          81
Meenakshi       91
Rose            71
Karthika        95
Fathima         78
Name: Science, dtype: int64
```

Understanding Missing Values

- Missing Data or Not Available data can occur when no information is provided for one or more items or for a whole unit.
- During Data Analysis, it is common for an object to have some missing attributes.
- If data is not collected properly it results in missing data.
- In DataFrame it is stored as NaN (Not a Number).
- For example, while collecting data, some people may not fill all the fields while taking the survey. Sometimes, some attributes are not relevant to all.
- Pandas provide a function isnull() to check whether any value is missing or not in the DataFrame.
- This function checks all attributes and returns True in case that attribute has missing values, otherwise returns False.
- Now, we can explore different operations related to missing values based on the DataFrame 'listDict' provided below.



```
      Dance  Music  Painting
X      10.0   20.0       NaN
XI     15.0   10.0      20.0
XII     NaN    NaN      12.0
```

listDict . isnull( )

	Dance	Music	Painting
X	True	True	False
XI	True	Tre	True
XII	False	False	True

Point to be Noted

Finding any missing value in a column ➔ listDict[‘Music’] . isnull() . any( ) →True

Finding total number of NaN ➔ listDict . isnull() . sum() → 3

Deleting entire row with NaN values ➔ listDict . dropna( )

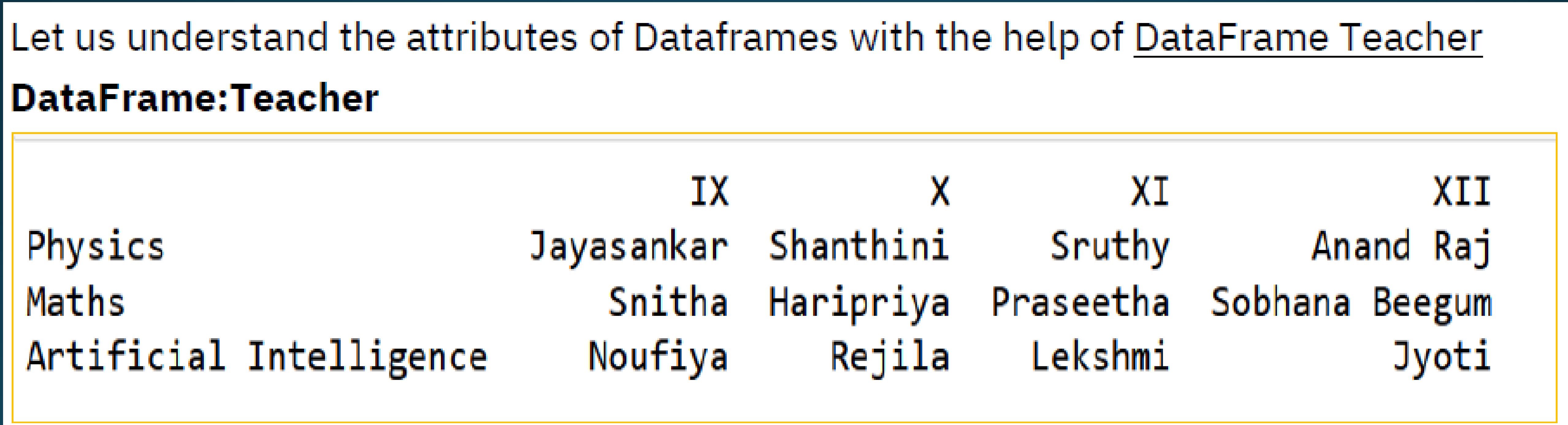
Replacing NaN values (here by 1) ➔ listDict . fillna (1)

Attributes of DataFrames

Attributes are the properties of a DataFrame that can be used to fetch data or any information related to a particular dataframe.

The syntax of writing an attribute is:

DataFrame_name . attribute

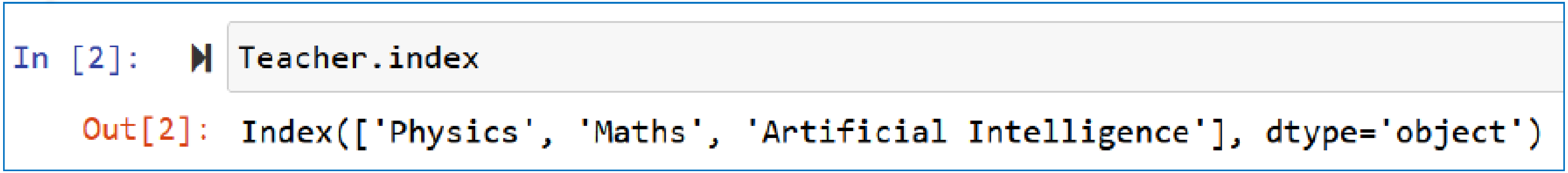


Let us understand the attributes of Dataframes with the help of DataFrame Teacher

DataFrame:Teacher

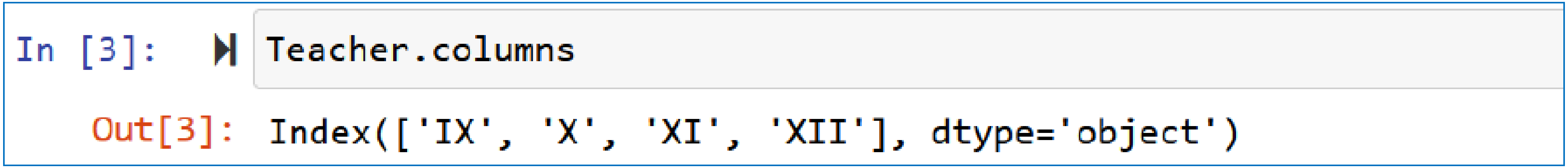
```
                                 IX          X         XI             XII
Physics                  Jayasankar  Shanthini     Sruthy       Anand Raj
Maths                        Snitha  Haripriya  Praseetha  Sobhana Beegum
Artificial Intelligence     Noufiya     Rejila    Lekshmi           Jyoti
```

Displaying Row Indexes - Teacher.index



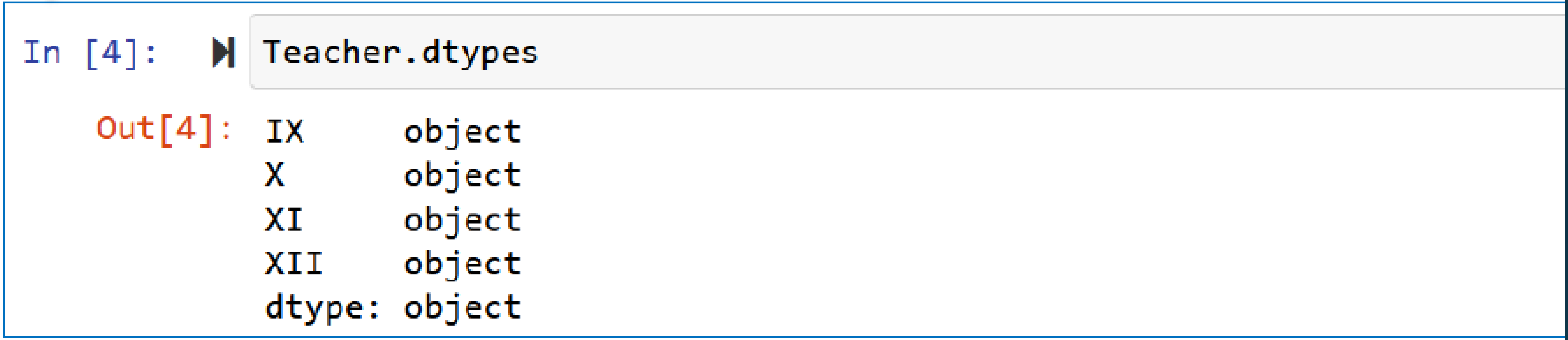
```
In [2]:  Teacher.index

Out[2]:  Index(['Physics', 'Maths', 'Artificial Intelligence'], dtype='object')
```

Displaying column Indexes - Teacher.columns



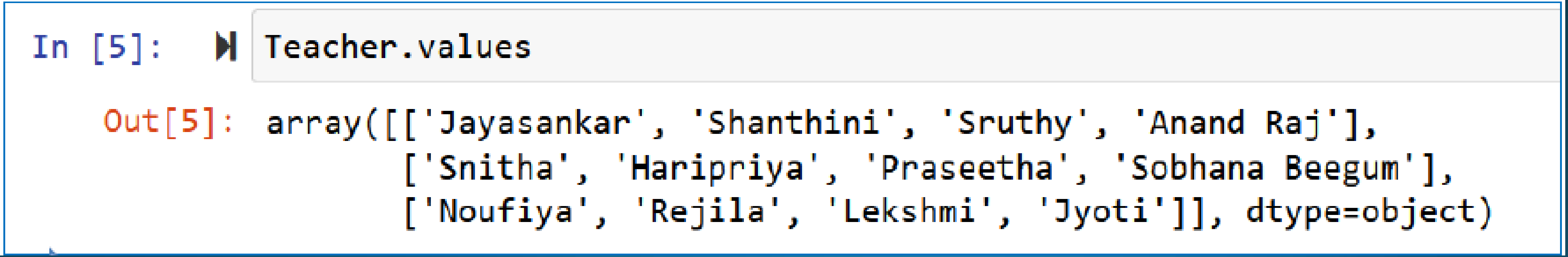
```
In [3]:  Teacher.columns

Out[3]:  Index(['IX', 'X', 'XI', 'XII'], dtype='object')
```

Displaying datatype of each - Teacher.dtypes



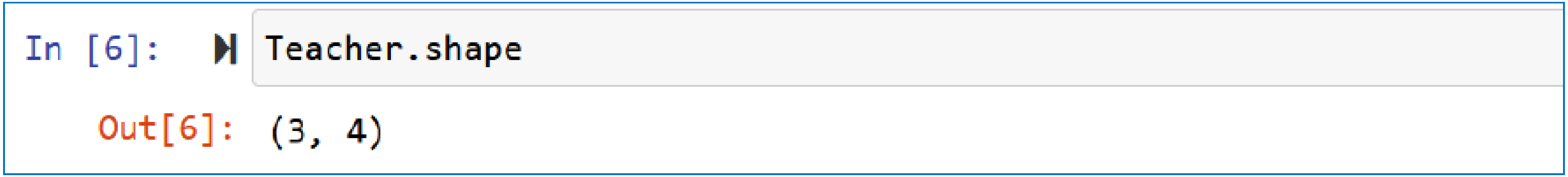
```
In [4]:  Teacher.dtypes

Out[4]: IX       object
        X        object
        XI       object
        XII      object
        dtype: object
```

Displaying data in Numpy Array form - Teacher.values



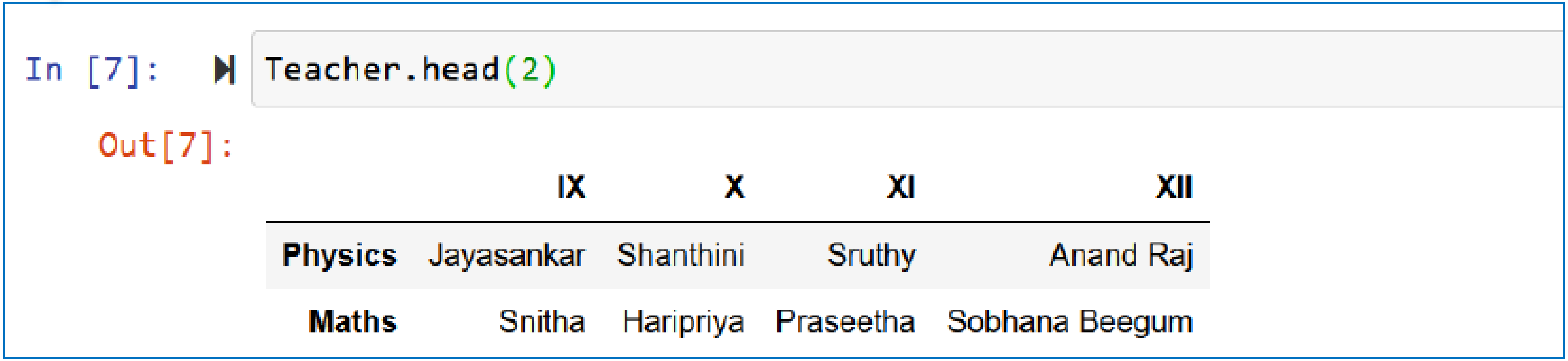
```
In [5]:  Teacher.values

Out[5]: array([['Jayasankar', 'Shanthini', 'Sruthy', 'Anand Raj'],
               ['Snitha', 'Haripriya', 'Praseetha', 'Sobhana Beegum'],
               ['Noufiya', 'Rejila', 'Lekshmi', 'Jyoti']], dtype=object)
```

Displaying total number of rows and columns (row, column) - Teacher.shape



```
In [6]:  Teacher.shape

Out[6]: (3, 4)
```

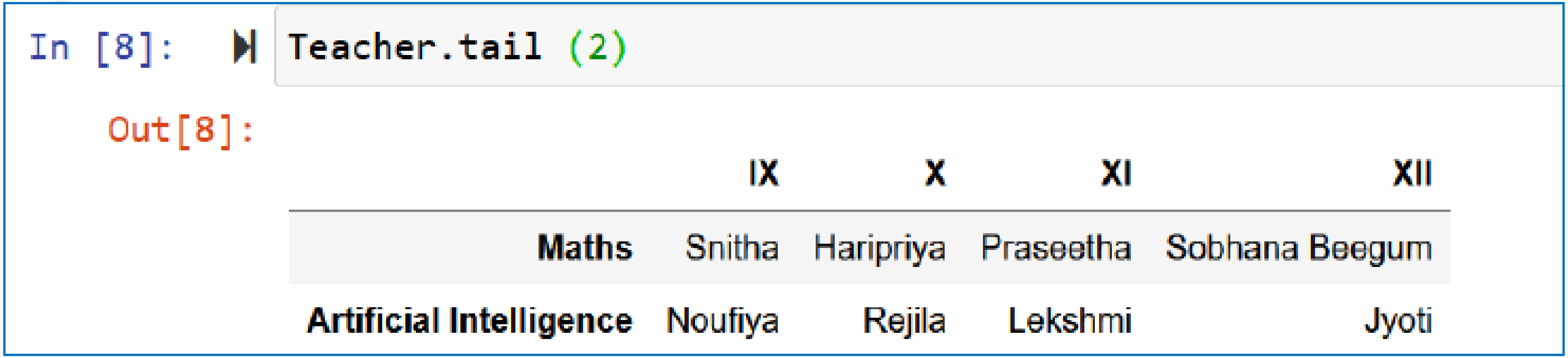
Displaying first n rows (here n = 2) - Teacher. head (2)



```
In [7]:  Teacher.head(2)

Out[7]:
```

	IX	X	XI	XII
Physics	Jayasankar	Shanthini	Sruthy	Anand Raj
Maths	Snitha	Haripriya	Praseetha	Sobhana Beegum

Displaying last n rows (here n = 2) - Teacher. tail (2)



In [8]:

```
Teacher.tail (2)
```

Out[8]:

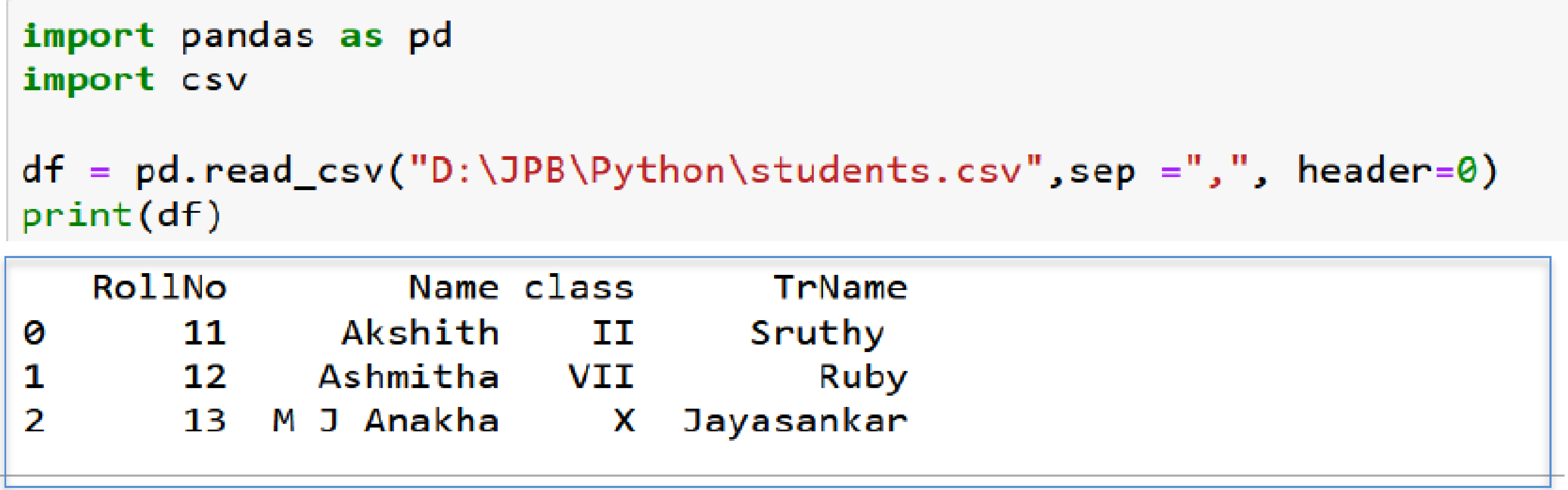
	IX	X	XI	XII
Maths	Snitha	Haripriya	Praseetha	Sobhana Beegum
Artificial Intelligence	Noufiya	Rejila	Lekshmi	Jyoti

Importing and Exporting Data between CSV Files and DataFrames

- We can create a DataFrame by importing data from CSV files.
- Similarly, we can also store or export data in a DataFrame as a .csv file.
- Importing a CSV file to a DataFrame Using the read_csv() function, you can import tabular data from CSV files into pandas dataframe by specifying a parameter value for the file name.

Syntax: pd.read_csv("filename.csv")

Example: Reading file students.csv



```
import pandas as pd
import csv

df = pd.read_csv("D:\JPB\Python\students.csv",sep =",", header=0)
print(df)
```

```
   RollNo         Name class      TrName
0      11      Akshith    II      Sruthy
1      12     Ashmitha   VII        Ruby
2      13  M J Anakha     X  Jayasankar
```

Point to be Noted

- read_csv() is used to read the csv file with its correct path
- sep specifies whether the values are separated by comma, semicolon, tab, or any other character. The default value for sep is a space.
- The parameter header marks the start of the data to be fetched. header=0 implies that column names are inferred from the first line of the file. By default, header=0.

Exporting a DataFrame to a CSV file

- We can use the to_csv() function to save a DataFrame to a text or csv file.
- For example, to save the DataFrame Teacher into csv file resultout, we should write

```
Teacher.to_csv(path_or_buf='C:/PANDAS/resultout.csv', sep=',')
```

- When we open this file in any text editor or a spreadsheet, we will find the above data along with the row labels and the column headers, separated by comma.

Scikit-learn (Learn) (This topic can be taught after teaching the Machine Learning Unit.)

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python.

It provides a selection of efficient tools for machine learning and statistical modeling via a consistent interface in Python. Sklearn is built on (relies heavily on) NumPy, SciPy and Matplotlib. .

Key Features:

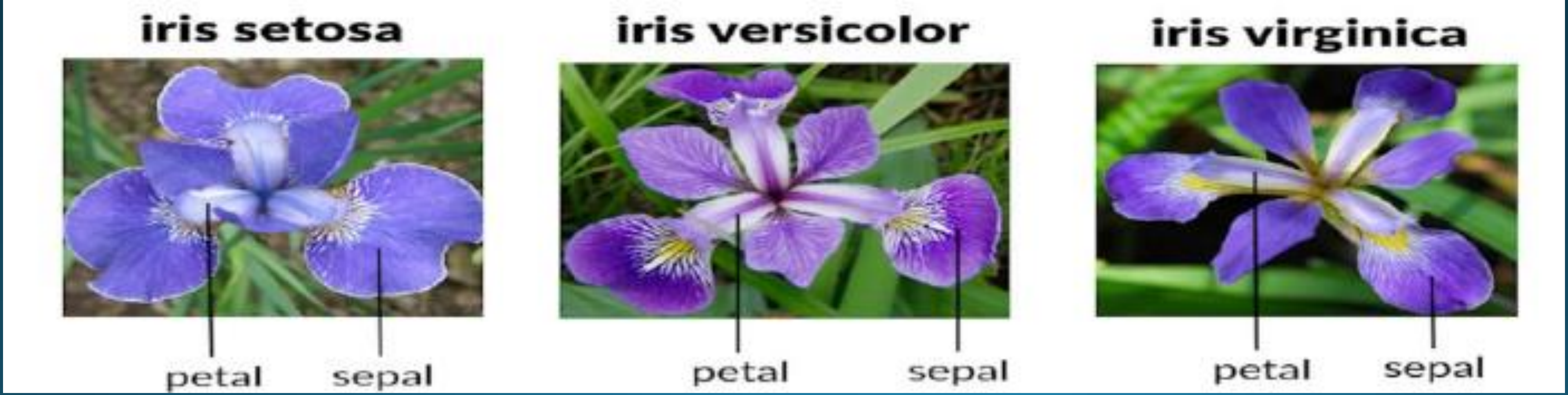
- Offers a wide range of supervised and unsupervised learning algorithms.
- Provides tools for model selection, evaluation, and validation.
- Supports various tasks such as classification, regression, clustering, dimensionality reduction, and more.
- Integrates seamlessly with other Python libraries like NumPy, SciPy, and Pandas.

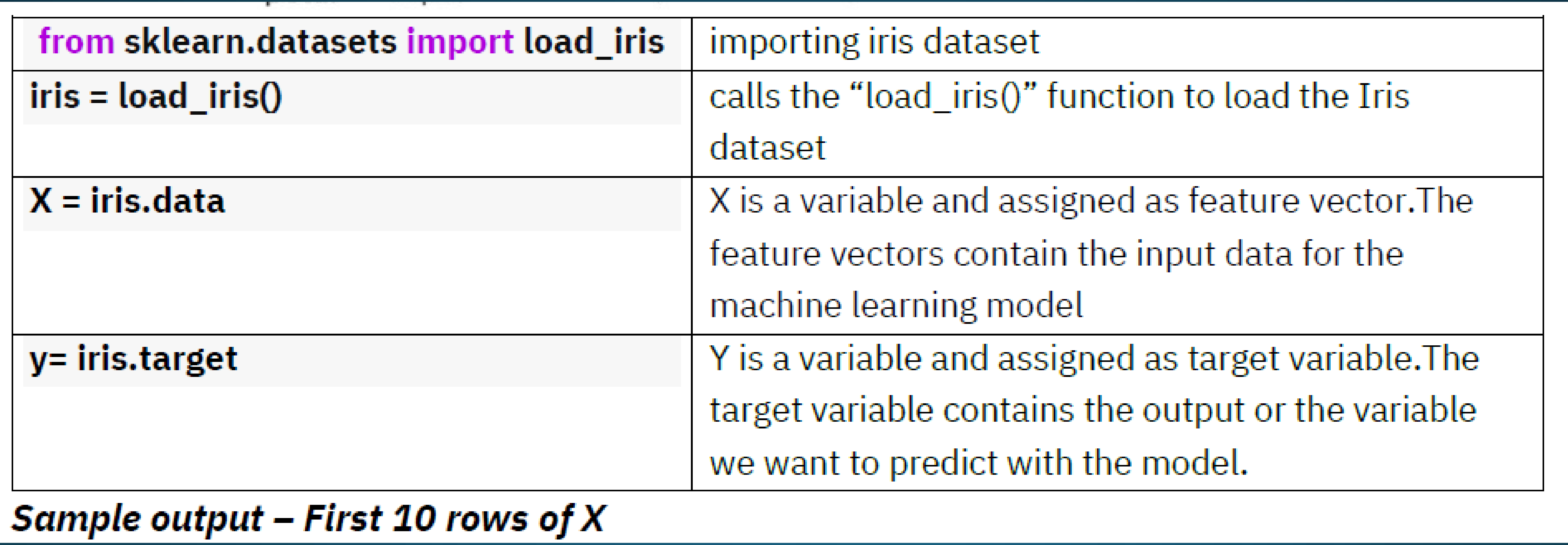
Install scikit-learn using the statement

```
pip install scikit-learn
```

load_iris (In sklearn.datasets)

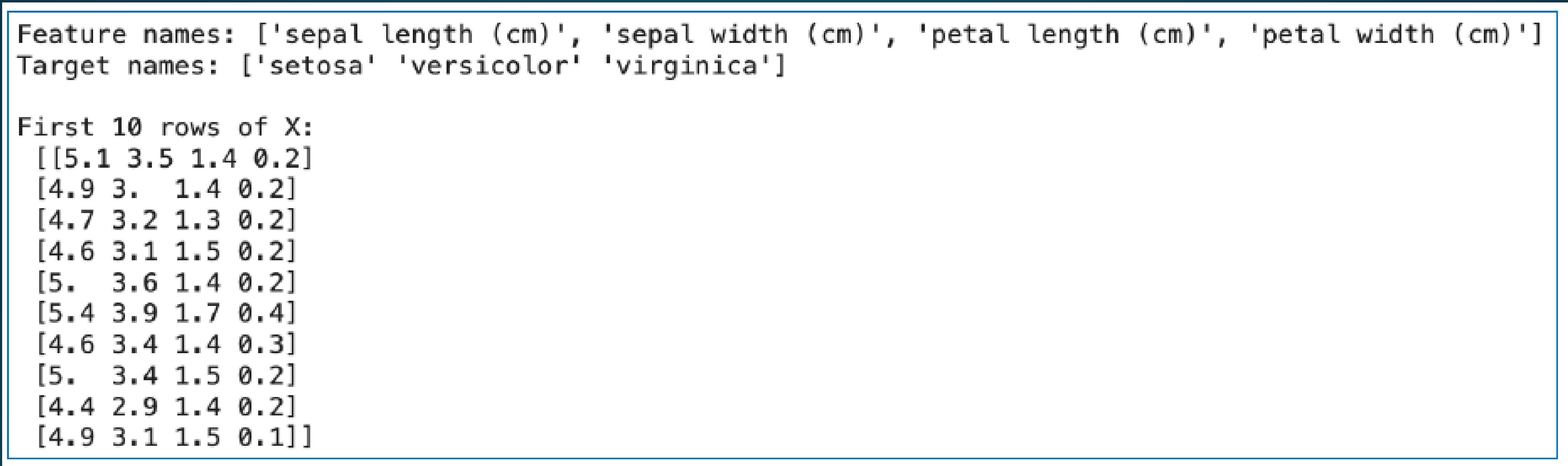
- The Iris dataset is a classic and widely used dataset in machine learning, particularly for classification tasks.
- It comprises measurements of various characteristics of iris flowers, such as sepal length, sepal width, petal length, and petal width, along with the corresponding species of iris to which they belong.
- The dataset typically includes three species: setosa, versicolor, and virginica.





from sklearn.datasets import load_iris	importing iris dataset
iris = load_iris()	calls the "load_iris()" function to load the Iris dataset
X = iris.data	X is a variable and assigned as feature vector.The feature vectors contain the input data for the machine learning model
y= iris.target	Y is a variable and assigned as target variable.The target variable contains the output or the variable we want to predict with the model.

Sample output – First 10 rows of X



```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']

First 10 rows of X:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

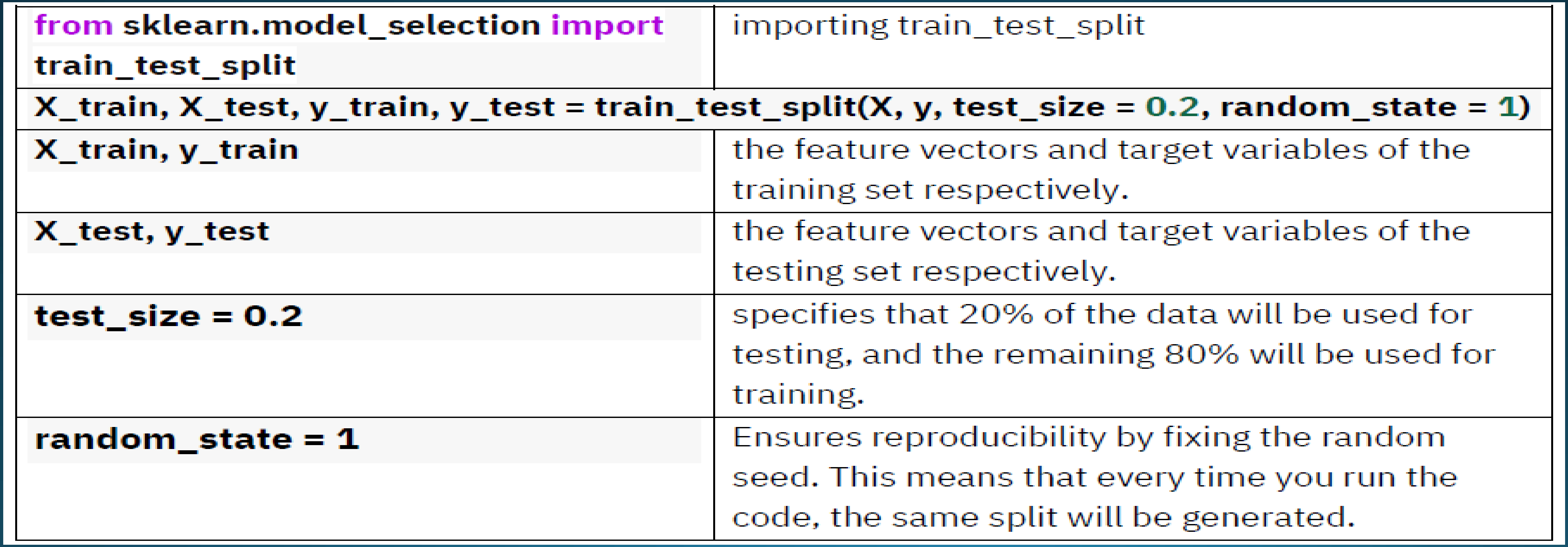
Here, each row represents a sample (i.e., an iris flower), and each column represents a feature (i.e., a measurement of the flower).

For example, the first row [5.1 3.5 1.4 0.2] corresponds to an iris flower with the following measurements:

- Sepal length: 5.1 cm
- Sepal width: 3.5 cm
- Petal length: 1.4 cm
- Petal width: 0.2 cm

train_test_split (In sklearn.model_selection)

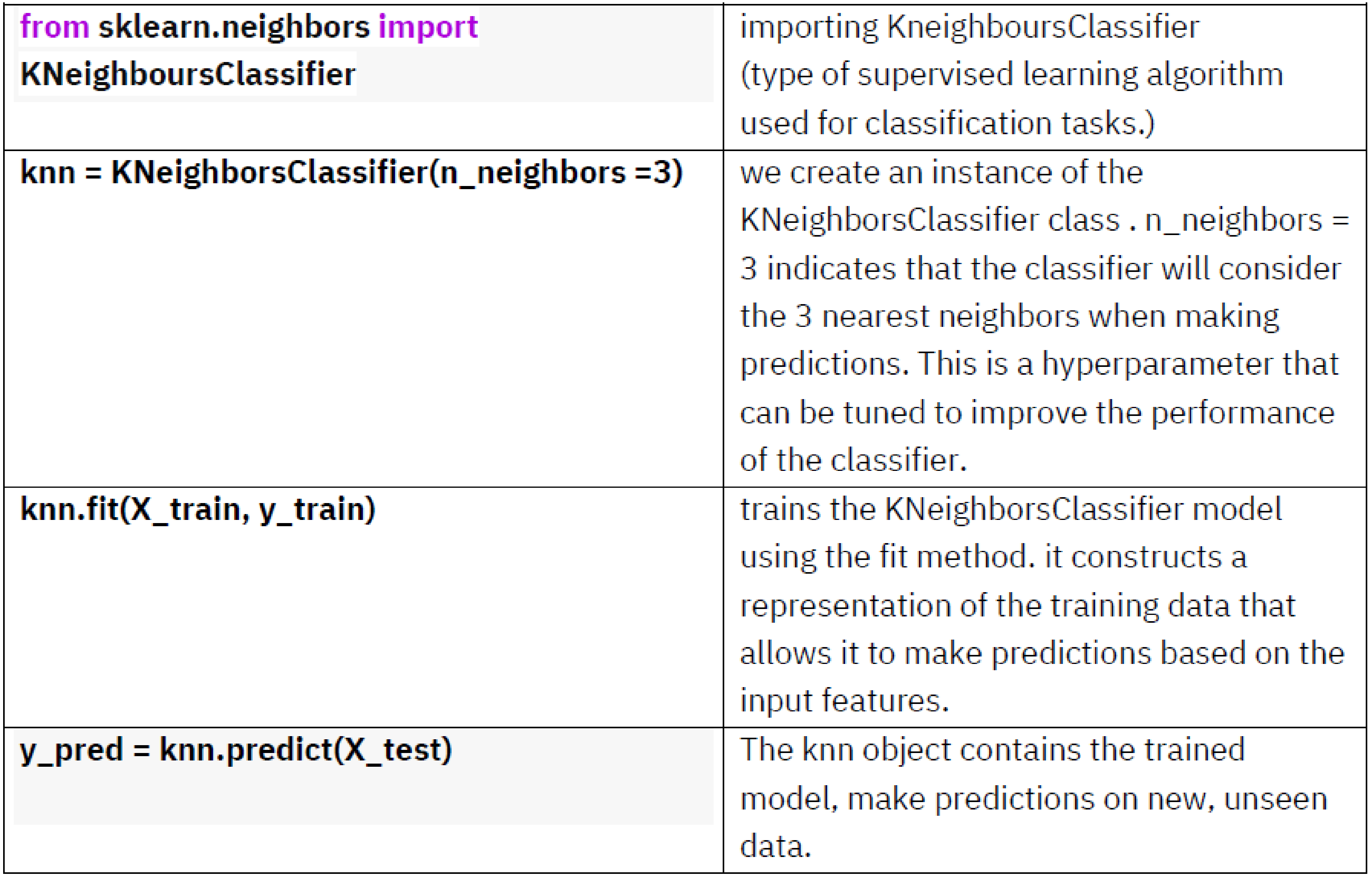
- Datasets are usually split into training set and testing set.
- The training set is used to train the model and testing set is used to test the model.
- Most common splitting ratio is 80 : 20 . (Training -80%, Testing-20%)



from sklearn.model_selection import train_test_split	importing train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)	
X_train, y_train	the feature vectors and target variables of the training set respectively.
X_test, y_test	the feature vectors and target variables of the testing set respectively.
test_size = 0.2	specifies that 20% of the data will be used for testing, and the remaining 80% will be used for training.
random_state = 1	Ensures reproducibility by fixing the random seed. This means that every time you run the code, the same split will be generated.

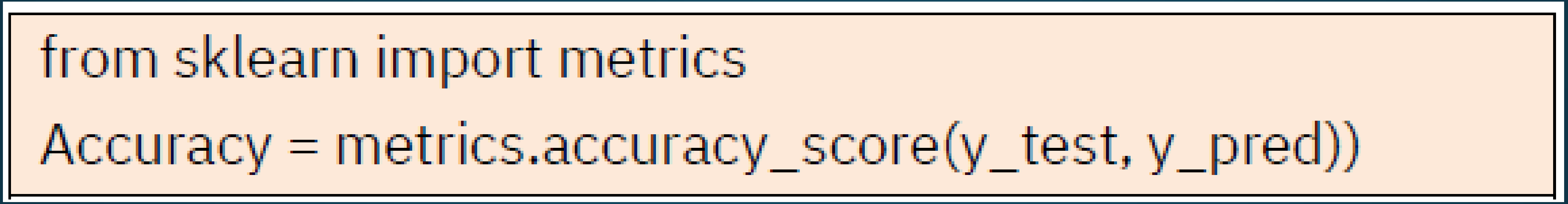
KNeighborsClassifier (In sklearn.neighbors)

- Scikit-learn has wide range of Machine Learning (ML) algorithms which have a consistent interface for fitting, predicting accuracy, recall etc.
- Here we are going to use KNN (K nearest neighbors) classifier.



from sklearn.neighbors import KNeighboursClassifier	importing KneighboursClassifier (type of supervised learning algorithm used for classification tasks.)
knn = KNeighborsClassifier(n_neighbors =3)	we create an instance of the KNeighborsClassifier class . n_neighbors = 3 indicates that the classifier will consider the 3 nearest neighbors when making predictions. This is a hyperparameter that can be tuned to improve the performance of the classifier.
knn.fit(X_train, y_train)	trains the KNeighborsClassifier model using the fit method. it constructs a representation of the training data that allows it to make predictions based on the input features.
y_pred = knn.predict(X_test)	The knn object contains the trained model, make predictions on new, unseen data.

metrics



```
from sklearn import metrics
Accuracy = metrics.accuracy_score(y_test, y_pred))
```

- This calculates the accuracy of the model by comparing the predicted target values (y_pred) with the actual target values (y_test).
- The accuracy_score represents the proportion of correctly predicted instances out of all instances in the testing set.

- Scikit-learn offers a variety of modules that simplify the process of building, training, and evaluating machine learning models, making it a popular choice for various tasks in this domain.

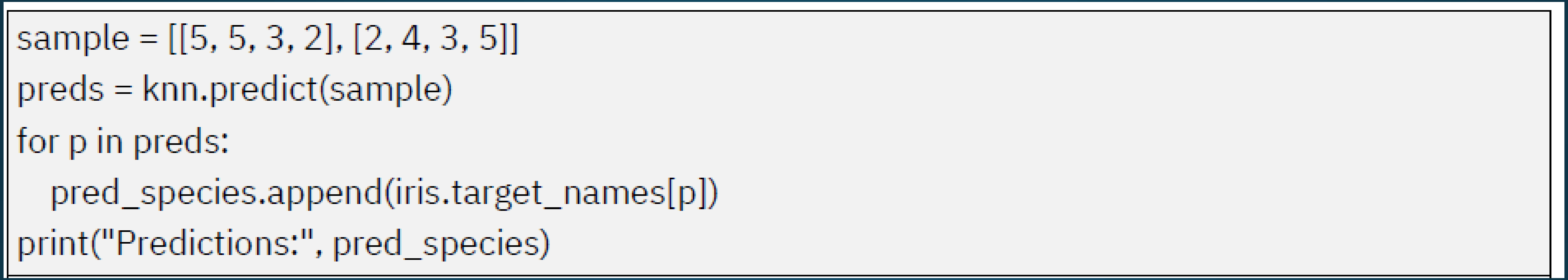
- In our session, we utilized the 'load_iris()' function to load the Iris dataset.

- Upon loading, we split the dataset into training and test sets using the 'train_test_split' function.

- Subsequently, we trained our model using the K-Nearest Neighbors Classifier ('KNeighborsClassifier') and evaluated its performance using appropriate metrics.

- This workflow represents a typical data analysis pipeline in AI project development.

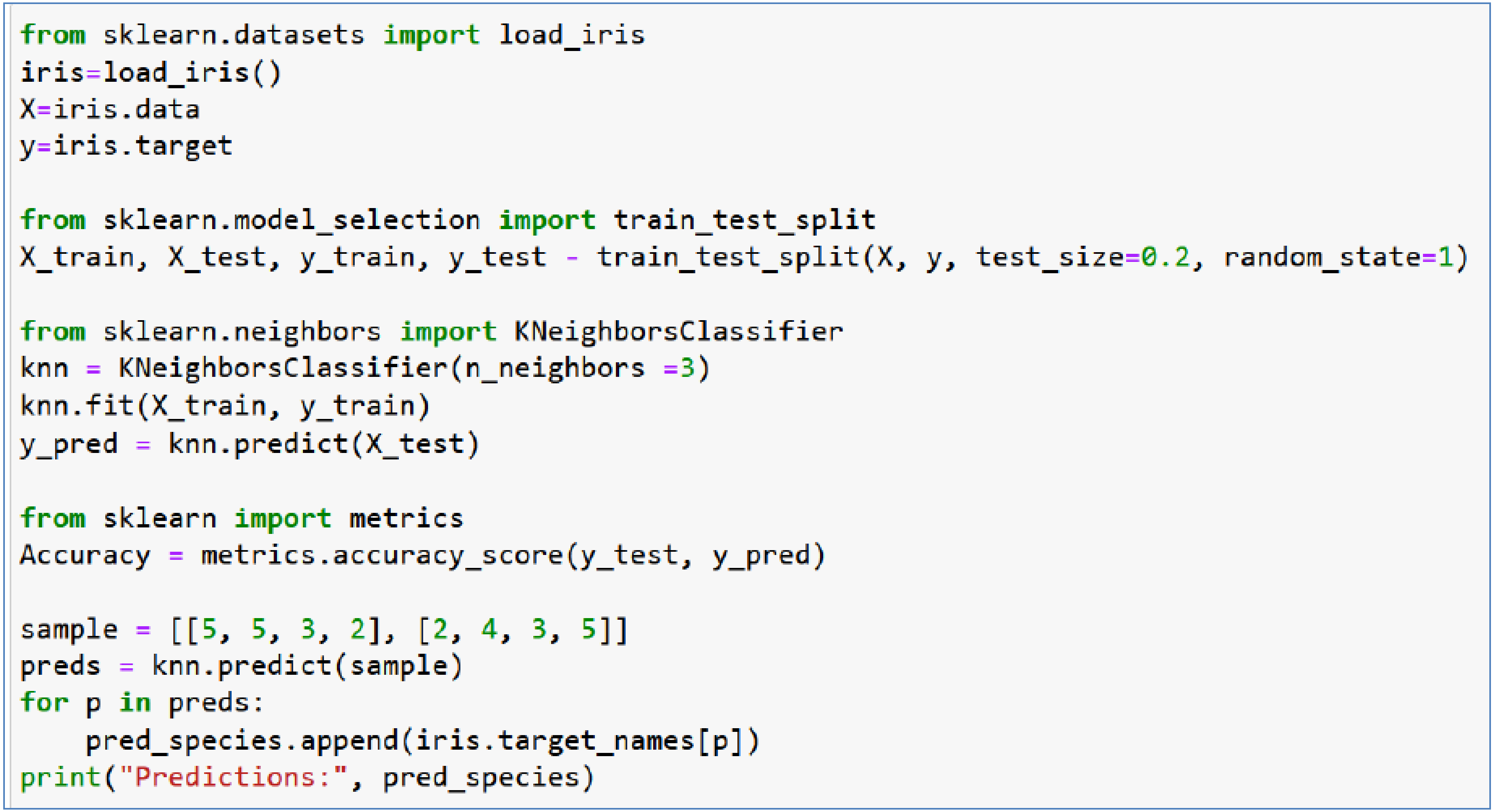
Now, to validate the model's predictive accuracy, we can use some sample data.



```
sample = [[5, 5, 3, 2], [2, 4, 3, 5]]
preds = knn.predict(sample)
for p in preds:
   pred_species.append(iris.target_names[p])
print("Predictions:", pred_species)
```

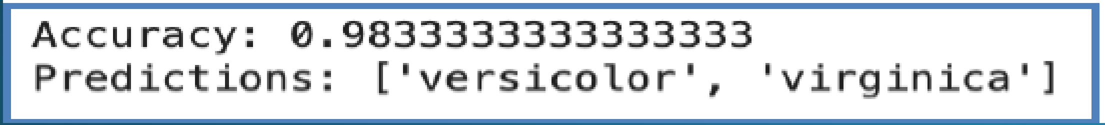
- The provided code snippet demonstrates how to use the trained classifier to make predictions on sample data.
- After initializing the sample data as [5, 5, 3, 2], the classifier predicts the species of iris flowers based on these measurements.
- Finally, the predicted species are printed to the console.

This is a program that combines different parts of our project to make it complete and understandable.



```
from sklearn.datasets import load_iris
iris=load_iris()
X=iris.data
y=iris.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test - train_test_split(X, y, test_size=0.2, random_state=1)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors =3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

from sklearn import metrics
Accuracy = metrics.accuracy_score(y_test, y_pred)

sample = [[5, 5, 3, 2], [2, 4, 3, 5]]
preds = knn.predict(sample)
for p in preds:
    pred_species.append(iris.target_names[p])
print("Predictions:", pred_species)
```

Output-



```
Accuracy: 0.9833333333333333
Predictions: ['versicolor', 'virginica']
```

- Using this model, we can identify the type of flower in the iris dataset.
- By analyzing the length and width of the sepals and petals, we can compare them with the features of the setosa, versicolor, and virginica species to determine the flower's species.

THANK YOU