# Data Mining and Data Warehousing Lab

Lab Assignment 01

Course Code: CSEL-4108

**Submitted to,**

Dr. Md. Manowarul Islam
Associate Professor
Department of Computer Science and Engineering
Jagannath University, Dhaka

**Submitted by,**

Momtaj Hossain Mow
ID: B180305025
&
Md. Abu Yousuf Siam
ID: B180305012

Department of Computer Science and Engineering
Jagannath University, Dhaka

**Date of Submission: 04 March 2023**

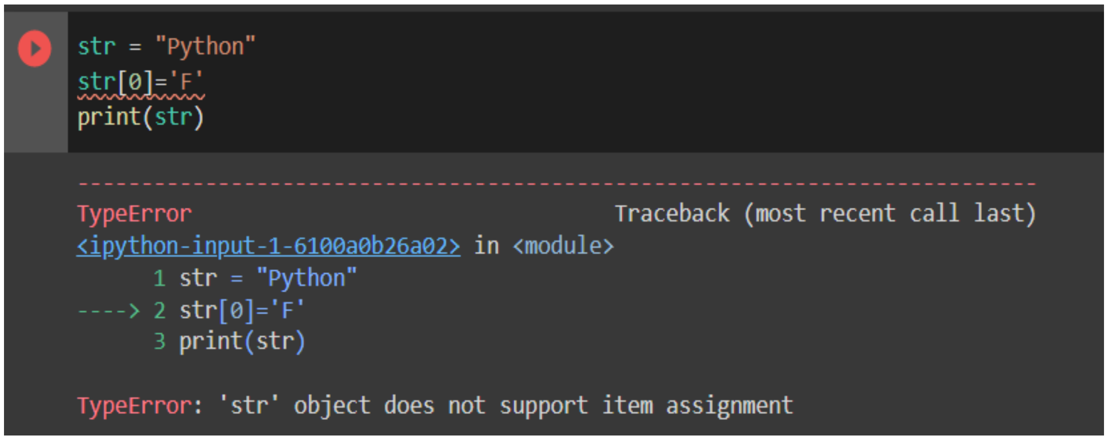
## Strings

An alphabet, word, or other character collection is referred to as a "string." As one of the most fundamental data structures, it serves as a framework for manipulating data. An in-built string class called "str" is available in Python. After they've been produced, strings are "immutable," which means that they can't be rewritten. Because of the immutability of strings, we must generate new ones each time we want to represent newly computed values.
Quotes are used to denote a string.**Strings are immutable.** Once created, the object's contents cannot be changed.

Single quotation marks, as in the following example:
‘For "double" quotations in your string, use "single" quotes.’
Use double quotation marks. Using single quotations in your string is easy with double-quotes.
"Double quotes allow you to embed 'single' quotes in your string."

Note:

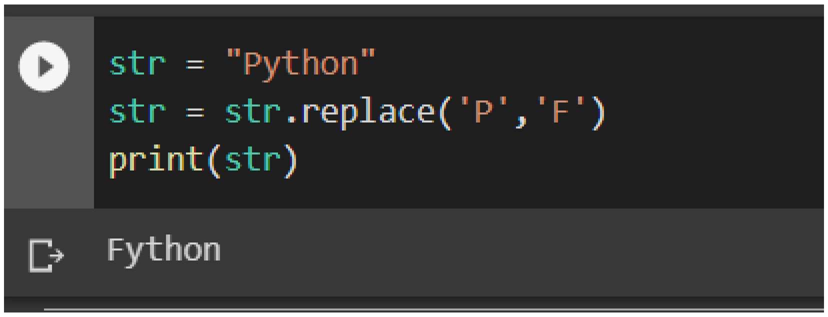
- Python does not distinguish between characters and strings.
- Characters are just Strings of length 1.



```
str = "Python"
str[0]='F'
print(str)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-6100a0b26a02> in <module>
      1 str = "Python"
----> 2 str[0]='F'
      3 print(str)

TypeError: 'str' object does not support item assignment
```

Note: We cannot change the contents of a string - we must make a new string to make any change


```
str = "Python"
str = str.replace('P','F')
print(str)
```
```
Fython
```

To access a string element, we use square brackets. The characters in a string start with index 0 and increase as index 1, index 2 etc. So, to access any letter of a string, we will write the previous number. For example, to access character 3, we will enter 2 inside the square brackets.

```
x = "Momtaj"
print(x[2])
```

The output of this python programming code will be ‘m’ like below

```
m
```

Or to access the last item in a string, we can use -1 value inside the square brackets.

```
x = "Momtaj"
print(x[-1])
```

Output:

```
j
```

## Arrays in Python

Python has a data type known as a list. Python arrays and Python lists are two components of python programming which collects several items. For our purposes, lists are arrays.
A collection allows us to put many values in a single “variable”

Python arrays and python lists both are used to store data. We can store various data with these components. Both of these components are mutable and they can be sliced, indexed and iterated.

- Both stores various data
- Both are mutable
- Both can be sliced
- Both can be indexed and iterated

**Declaration syntax:** <name> = [<value>, <value>, <value>, …, <value>]
<name> = [<default value>] * <initial array size>

Example:
```
numbers = [12, 49, -2, 26, 5, 17, -6]
zeros = [0] * 10
```

Indexing: Lists have zero based indexing from front
Negative Indexing: You can also refer to an element by a negative index representing how far it is from the end.

Example:

| | | | | | | |
|---|---|---|---|---|---|---|
| *index from front* | *0* | *1* | *2* | *3* | *4* | *5* |
| *numbers* | *= [ 13,* | *25,* | *39,* | *46,* | *54,* | *68]* |
| *index from back* | *-6* | *-5* | *-4* | *-3* | *-2* | *-1* |

Lists use more memory than arrays. But we can easily modify a list. They are flexible components. This is not easy with arrays because arrays are not flexible.
In a list we do not need to have a specific loop to access the list components. But in an array we need a loop to access the components of the array.
Lists are good for shorter sequences of data. But arrays are better for longer sequences of data.

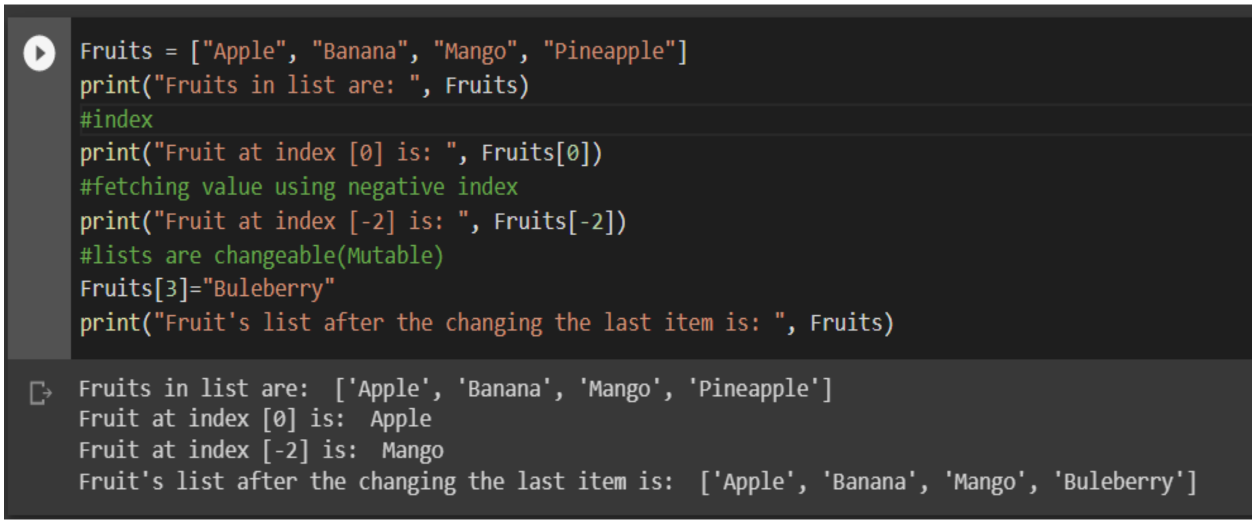
**Input:**

```
numbers=[12, 49, -2, 26, 5, 17, -6]

print(numbers)

print("The element in -3 is", numbers[-3])
```

**Output:**

```
[12, 49, -2, 26, 5, 17, -6]
The element in -3 is 5
```

Another example:



```
Fruits = ["Apple", "Banana", "Mango", "Pineapple"]
print("Fruits in list are: ", Fruits)
#index
print("Fruit at index [0] is: ", Fruits[0])
#fetching value using negative index
print("Fruit at index [-2] is: ", Fruits[-2])
#lists are changeable(Mutable)
Fruits[3]="Buleberry"
print("Fruit's list after the changing the last item is: ", Fruits)
```

```
Fruits in list are:  ['Apple', 'Banana', 'Mango', 'Pineapple']
Fruit at index [0] is:  Apple
Fruit at index [-2] is:  Mango
Fruit's list after the changing the last item is:  ['Apple', 'Banana', 'Mango', 'Buleberry']
```

## Methods for Lists

**Basic Methods** – directly modify the lists

- list.append(item) – appends the item to the end of the list
- list.insert(index, item) – inserts the item at the specified index
- list.remove(item) – removes the first occurrence of item from the list
- list.extend(second_list) – appends second_list to the list

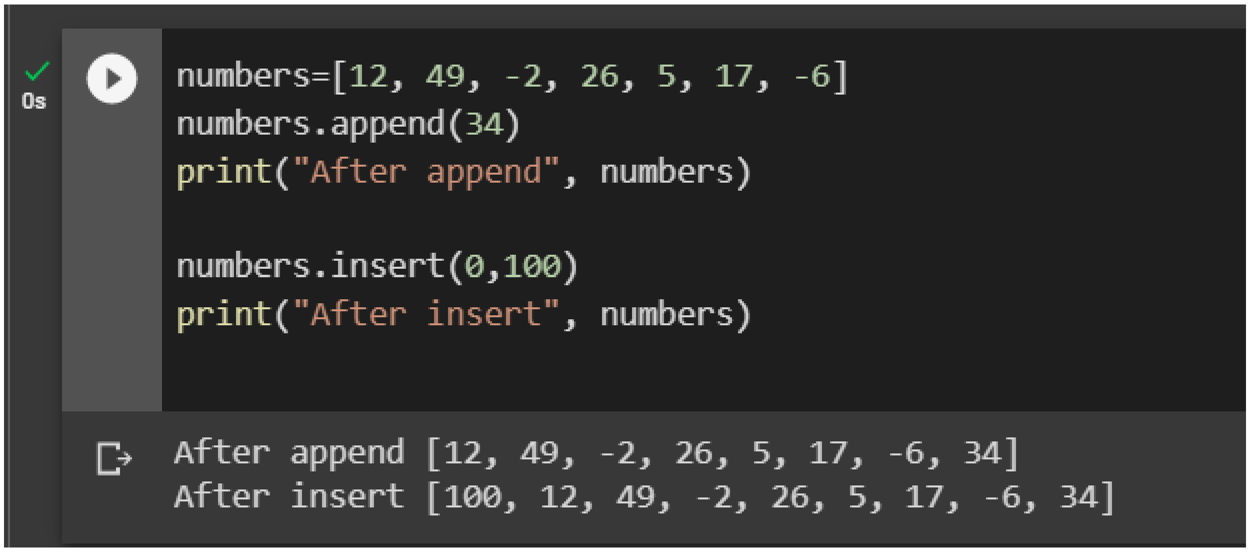
**Example:**

```
numbers=[12, 49, -2, 26, 5, 17, -6]
numbers.append(34)
print("After append", numbers)

numbers.insert(0,100)
print("After insert", numbers)
```

```
After append [12, 49, -2, 26, 5, 17, -6, 34]
```

```
After insert [100, 12, 49, -2, 26, 5, 17, -6, 34]
```

Screenshot of the input output of the code is given below (google colab):



```
numbers=[12, 49, -2, 26, 5, 17, -6]
numbers.append(34)
print("After append", numbers)

numbers.insert(0,100)
print("After insert", numbers)
```

```
After append [12, 49, -2, 26, 5, 17, -6, 34]
After insert [100, 12, 49, -2, 26, 5, 17, -6, 34]
```

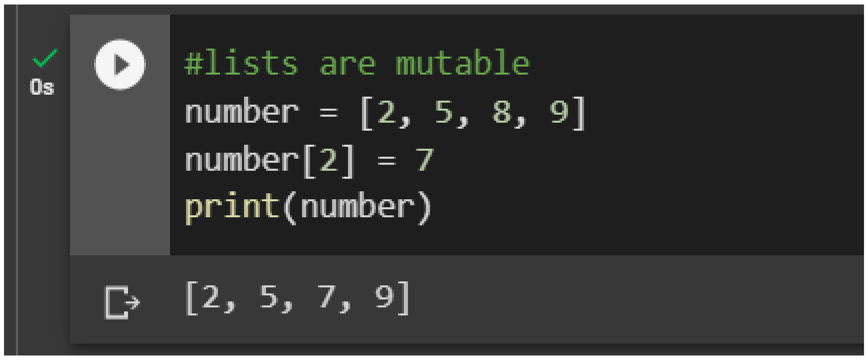
## Lists can be Manipulated

Lists are "**mutable**" - we can change an element of a list using the index operator

**Input:**
```
abc= [2, 14, 26, 41, 63]
abc[2] = 28
print (abc)
```

**Output:**
```
[2, 14, 28, 41, 63]
```

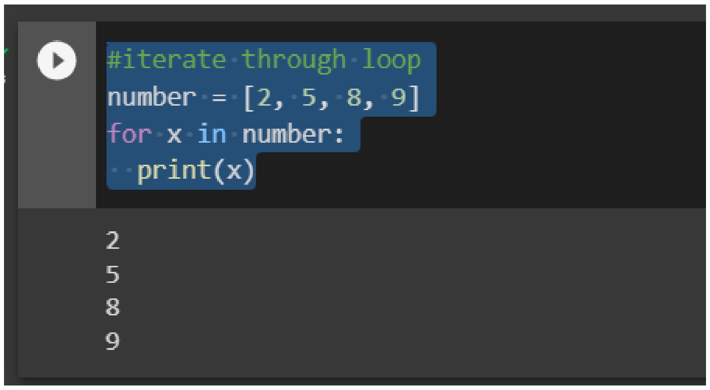
Another example of code(ss):



```
#lists are mutable
number = [2, 5, 8, 9]
number[2] = 7
print(number)
```
```
[2, 5, 7, 9]
```

You can use the for in loop to loop through all the elements of an array.
Print each item in the array:

```
#iterate through loop
number = [2, 5, 8, 9]
for x in number:
  print(x)
```

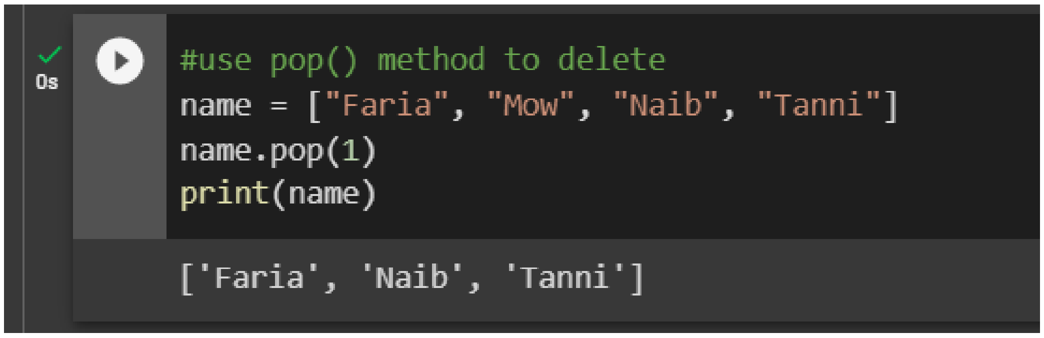


```
#iterate through loop
number = [2, 5, 8, 9]
for x in number:
  print(x)
```
```
2
5
8
9
```

You can use the pop() method to remove an element from the array. For example: Delete the second element of the array:

```
name.pop(1)
```

Screenshot of the code portion:



```
#use pop() method to delete
name = ["Faria", "Mow", "Naib", "Tanni"]
name.pop(1)
print(name)
```

```
['Faria', 'Naib', 'Tanni']
```

## More Methods

- list.count(element) – returns number of times element occurs in the list
- list.sort – sorts the element in place
- list.reverse – reverses the element in place

## Extend Lists

```
# Creating a List
List = [1,2,3,4]
print("Initial List: ")
print(List)

# Addition of multiple elements to the List at the end #(using
Extend Method)
List.extend([8, 'Siam', ‘Showdagor'])
print("\nList after performing Extend Operation: ")
print(List)
```

```
Initial List: [1, 2, 3, 4]
List after performing Extend Operation: [1, 2, 3, 4, 8, 'Siam',
'Showdagor']
```

## remove() Lists

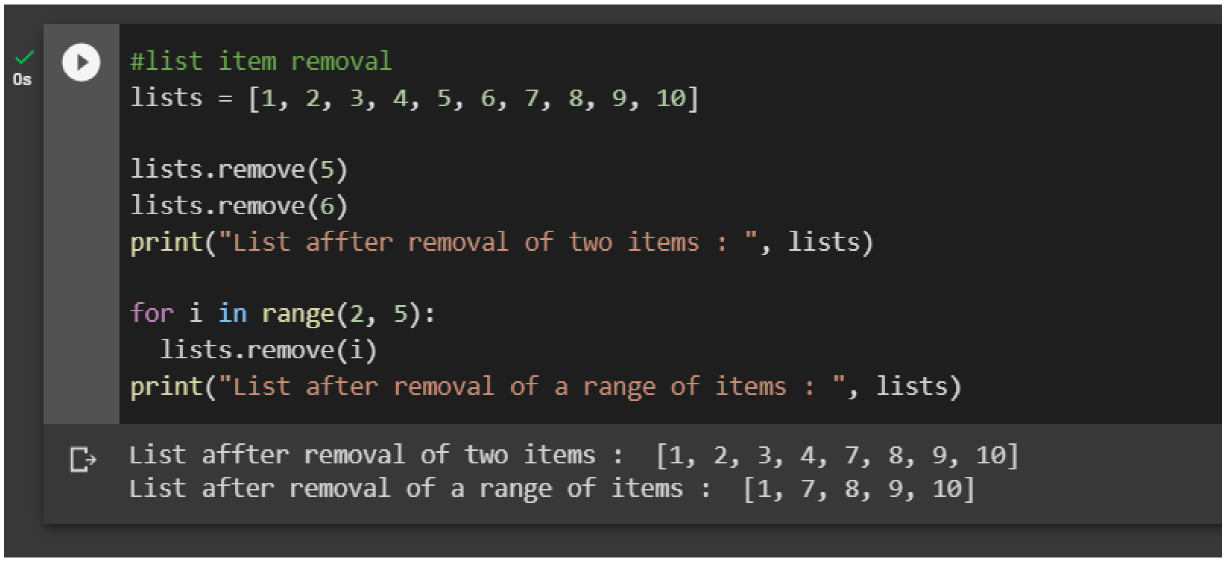
```
# Creating a List
List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
print("Initial List: ")
print(List)

# Removing elements from List using remove() method
List.remove(5)
List.remove(6)
print("\nList after Removal of two elements: ")
print(List)
# Removing elements from List using iterator method
for i in range(1, 5):
    List.remove(i) # There is a tab here, be careful
print("\nList after Removing a range of elements: ")
print(List)
```

```
Initial List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
List after Removal of two elements: [1, 2, 3, 4, 7, 8, 9, 10,
11, 12]
List after Removing a range of elements: [7, 8, 9, 10, 11, 12]
```

**Example:**

```
range(start, stop)
range(1, 8)
Like : (int i=1;i<8;i++)
```

Another example:



```
#list item removal
lists = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

lists.remove(5)
lists.remove(6)
print("List affter removal of two items : ", lists)

for i in range(2, 5):
  lists.remove(i)
print("List after removal of a range of items : ", lists)
```

```
List affter removal of two items :  [1, 2, 3, 4, 7, 8, 9, 10]
List after removal of a range of items :  [1, 7, 8, 9, 10]
```

## More Methods- Slicing

**Slicing** – can get a sublist of a list

<name>[<first index inclusive> : <second index not-inclusive>]

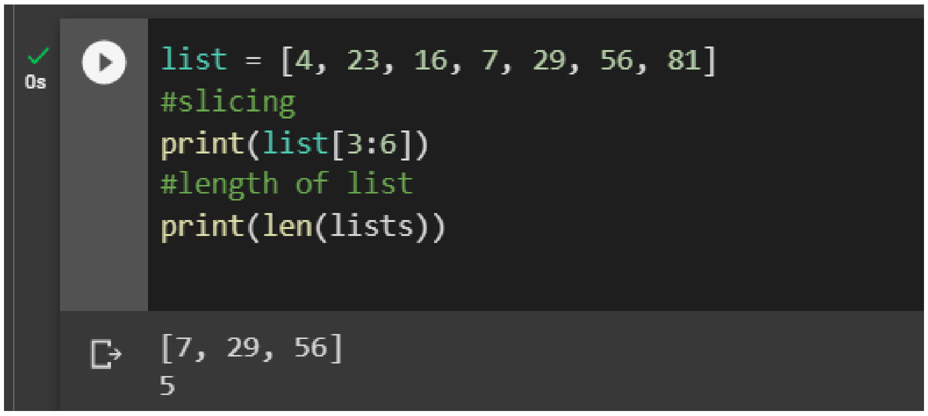
```
list = [4, 23, 16, 7, 29, 56, 81]
list[3:6] => [7, 29, 56]
```

# Note: Indices start with 0 in python

Length of lists

len(list) => 7

For example:



```
list = [4, 23, 16, 7, 29, 56, 81]
#slicing
print(list[3:6])
#length of list
print(len(lists))
```

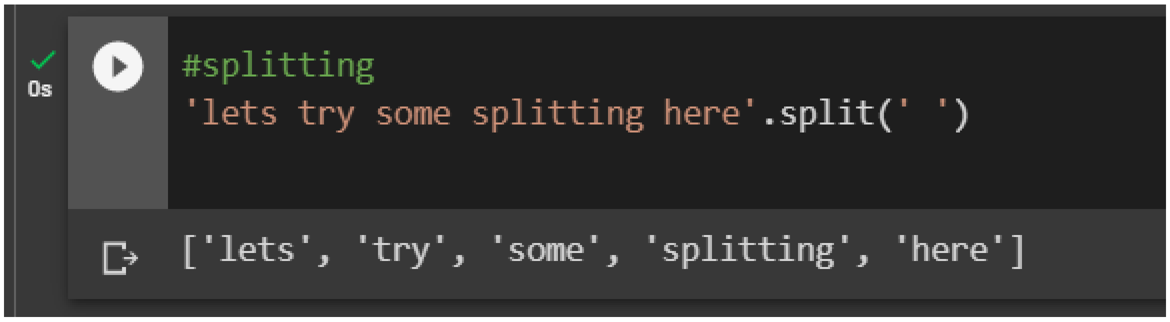
```
[7, 29, 56]
5
```

**Split** – returns a list

*Example:*

```
“lets try some splitting here”.split(“ ”)=>['lets', 'try', 'some', 'splitting', 'here’]
```

Screenshot of the code:



```
#splitting
'lets try some splitting here'.split(' ')
```

```
['lets', 'try', 'some', 'splitting', 'here']
```

Note: If you directly copy this into the IDE, there are chances to get errors. Just remove the quotation marks and then type from the keyboard, the errors will vanish.

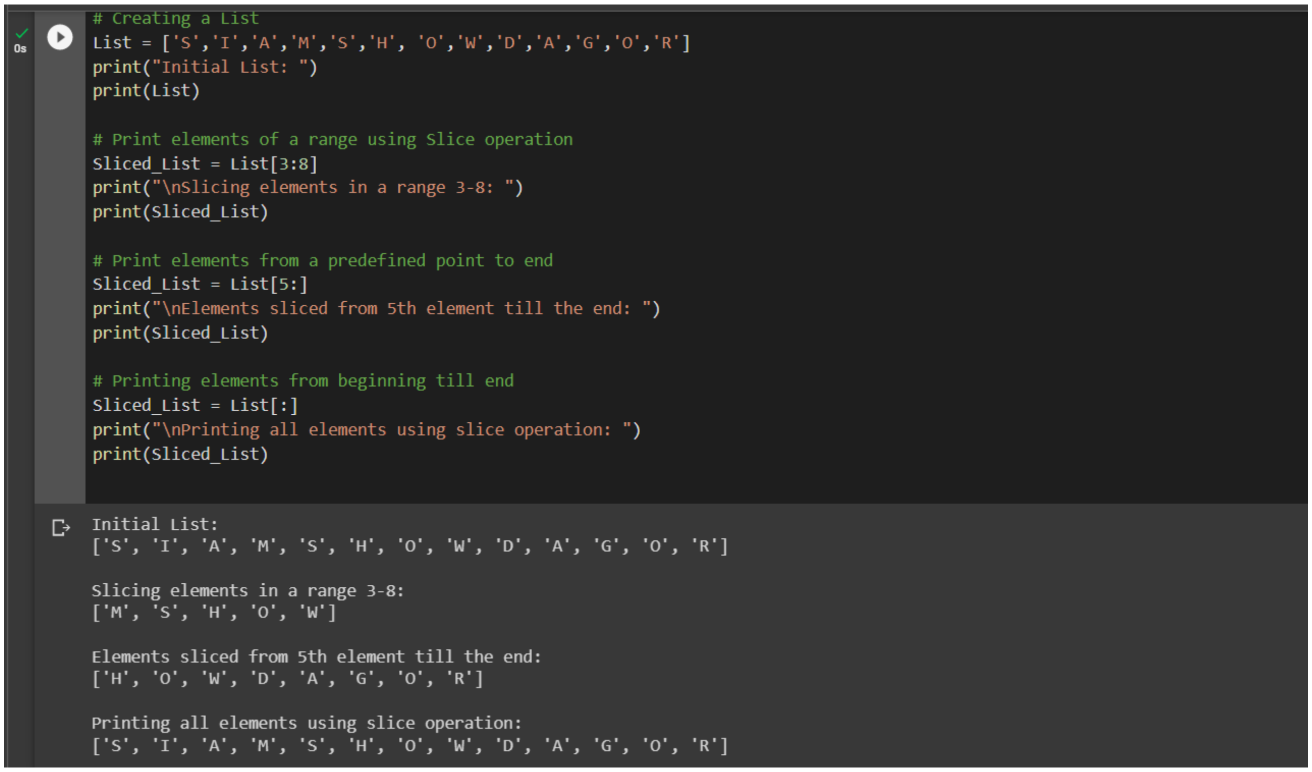
**Input:**

```
# Creating a List
List = ['S','I','A','M','S','H', 'O','W','D','A','G','O','R']
print("Initial List: ")
print(List)

# Print elements of a range using Slice operation
Sliced_List = List[3:8]
print("\nSlicing elements in a range 3-8: ")
print(Sliced_List)

# Print elements from a predefined point to end
Sliced_List = List[5:]
print("\nElements sliced from 5th element till the end: ")
print(Sliced_List)

# Printing elements from beginning till end
Sliced_List = List[:]
print("\nPrinting all elements using slice operation: ")
print(Sliced_List)
```

**Output:**

```
InItial List:
['S','I','A','M','S','H', 'O','W','D','A','G','O','R']

Slicing elements in a range 3-8:
['M','S','H', 'O','W','D']

Elements sliced from 5th element till the end:
['H', 'O','W','D','A','G','O','R']

Printing all elements using slice operation:
['S', 'I', 'A', 'M', 'S', 'H', 'O', 'W', 'D', 'A', 'G', 'O', 'R']
```

Screenshot of the code in colab:



```
# Creating a List
List = ['S','I','A','M','S','H', 'O','W','D','A','G','O','R']
print("Initial List: ")
print(List)

# Print elements of a range using Slice operation
Sliced_List = List[3:8]
print("\nSlicing elements in a range 3-8: ")
print(Sliced_List)

# Print elements from a predefined point to end
Sliced_List = List[5:]
print("\nElements sliced from 5th element till the end: ")
print(Sliced_List)

# Printing elements from beginning till end
Sliced_List = List[:]
print("\nPrinting all elements using slice operation: ")
print(Sliced_List)
```

```
Initial List:
['S', 'I', 'A', 'M', 'S', 'H', 'O', 'W', 'D', 'A', 'G', 'O', 'R']

Slicing elements in a range 3-8:
['M', 'S', 'H', 'O', 'W']

Elements sliced from 5th element till the end:
['H', 'O', 'W', 'D', 'A', 'G', 'O', 'R']

Printing all elements using slice operation:
['S', 'I', 'A', 'M', 'S', 'H', 'O', 'W', 'D', 'A', 'G', 'O', 'R']
```

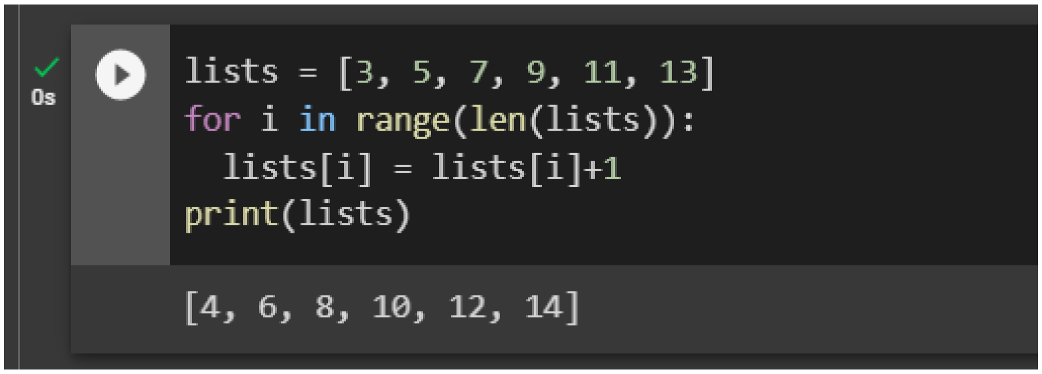
## Methods

| | |
|---|---|
| append() | Adds an element at the end of the list |
| clear() | Removes all the elements from the list |
| copy() | Returns a copy of the list |
| count() | Returns the number of elements with the specified value |
| extend() | Add the elements of a list (or any iterable), to the end of the current list |
| index() | Returns the index of the first element with the specified value |

| | |
|---|---|
| insert() | Adds an element at the specified position |
| pop() | Removes the element at the specified position |
| remove() | Removes the first item with the specified value |
| reverse() | Reverses the order of the list |
| sort() | Sorts the list |

## Ranges are Lists (used in ‘for’ loop)

Python For Loop is basically used to iterate over a sequence. In other words, it repeats a job for a given sequence. This sequence can be a python list, a python tuple, a string, a dictionary or a set. The other loop method used in python is Python While Loop.
Recall how we used the method range() in for loops.
Calling range returns a list with the patterns specified by range().
The len() function takes a list as a parameter and returns the number of elements in the list.
For example:



```
lists = [3, 5, 7, 9, 11, 13]
for i in range(len(lists)):
  lists[i] = lists[i]+1
print(lists)
```

```
[4, 6, 8, 10, 12, 14]
```

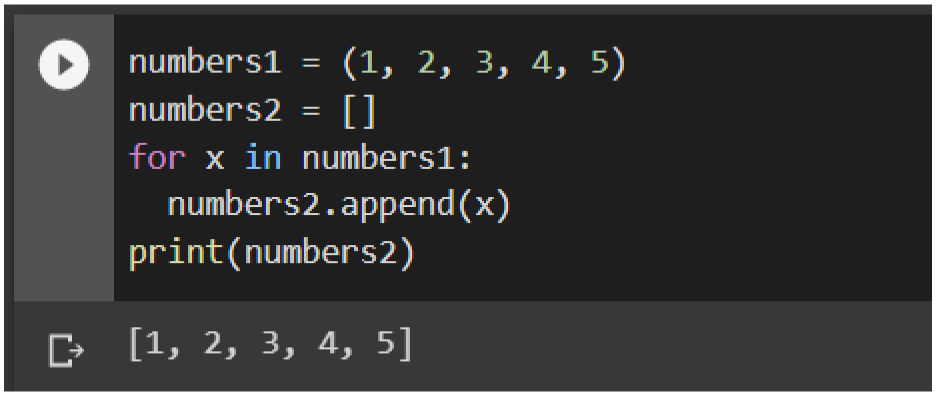
Let’s do some more code:

```
numbers1 = (1, 2, 3, 4, 5)
numbers2 = []
for x in numbers1:
  numbers2.append(x)
print(numbers2)
```

Output:

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

Screenshot of the code deployed in google colab:



```
numbers1 = (1, 2, 3, 4, 5)
numbers2 = []
for x in numbers1:
  numbers2.append(x)
print(numbers2)
```

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

## Methods for Lists

**Basic Methods – directly modify the lists**

`list.sort()` : used to sort the elements of the list in ascending order.

`list.reverse()` : reverses the order of the elements in the list– appends the item to the end of the list.

`list.clear()`: Removes all the items from list.

`list.pop(x)`: deletes the element at the position x.

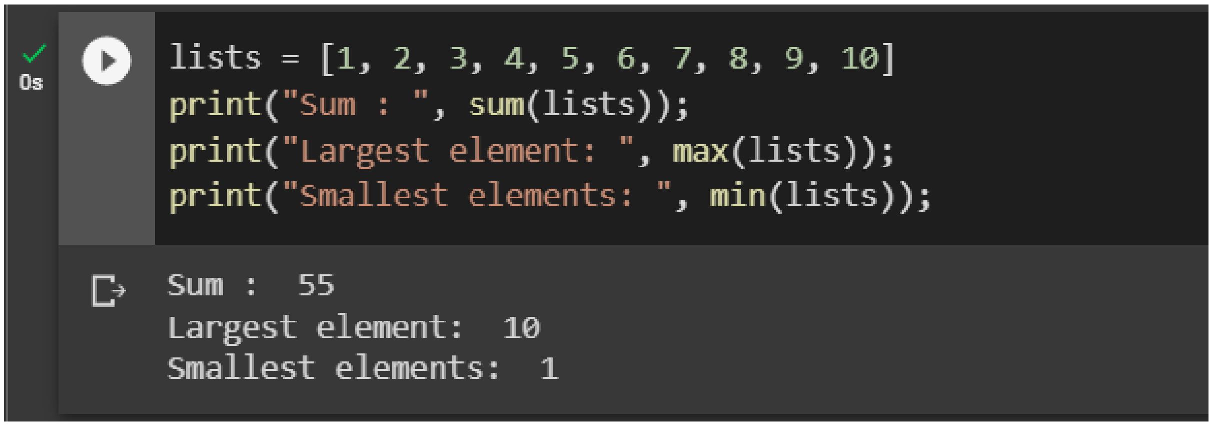
**Mathematical Methods –**

`sum(list)` : sums up the numbers in the list.

`max(list)` : Finds the maximum value.

`min(list)` : Finds the minimum value..

For example code deployed in google colab:



```
lists = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Sum : ", sum(lists));
print("Largest element: ", max(lists));
print("Smallest elements: ", min(lists));
```

```
Sum :  55
Largest element:  10
Smallest elements:  1
```

## Dictionary

In Python programming, there are different data types that store data collections. Python dictionary is one of these methods. The others are python lists, python tuples and python sets.
Dictionaries are ordered data types and they can be changed. And duplicate items are not allowed in a python dictionary.
A dictionary associates a simple data value called a key (most often a string) with a value.

- Key-value pairs
- Unordered collection of references
- Each value is referenced through key in the pair
- Curly braces {} are used to create a dictionary
- When entering values
- Use { key1:value1, … }

**Notes:**
Values can be of any Python data type.
As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.
When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.
Unordered means that the items do not have a defined order, you cannot refer to an item by using an index.

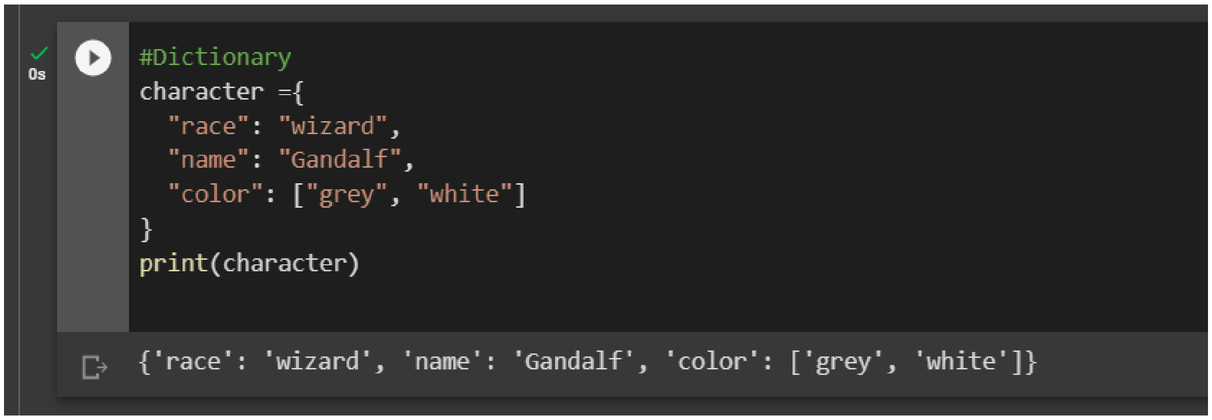
Below, let's give an **example code** for dictionaries:

```
character ={
  "race": "wizard",
  "name": "Gandalf",
  "color": ["grey", "white"]
}
print(character)
```

**Output:**

```
{'race': 'wizard', 'name': 'Gandalf', 'color': ['grey', 'white']}
```

The very same code deployed in google colab:



```
#Dictionary
character ={
  "race": "wizard",
  "name": "Gandalf",
  "color": ["grey", "white"]
}
print(character)
```

```
{'race': 'wizard', 'name': 'Gandalf', 'color': ['grey', 'white']}
```

There are different functions used with python dictionaries. One of them is the len function. We can find the length of a dictionary with this function.

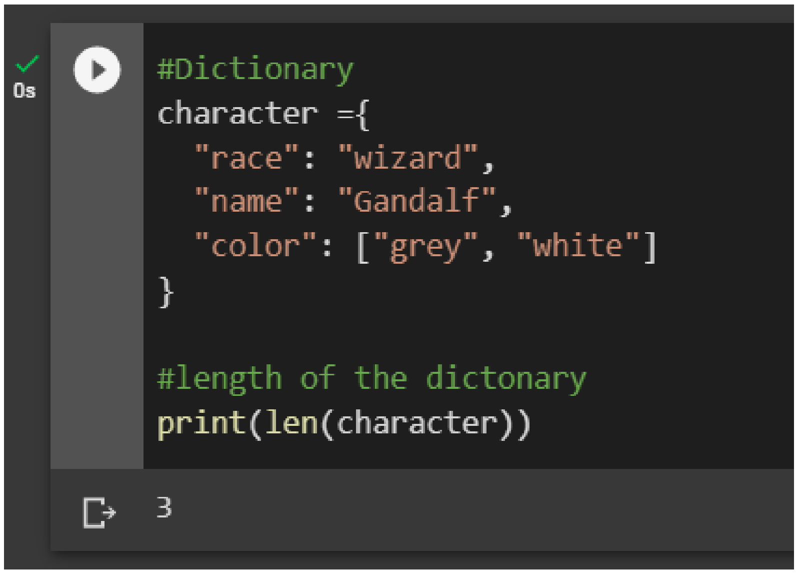
**Code:**

```
character ={
  "race": "wizard",
  "name": "Gandalf",
  "color": ["grey", "white"]
}
print(len(character))
```

**Output:**

```
3
```

The very same code deployed in google colab:



```
#Dictionary
character ={
  "race": "wizard",
  "name": "Gandalf",
  "color": ["grey", "white"]
}

#length of the dictonary
print(len(character))
```

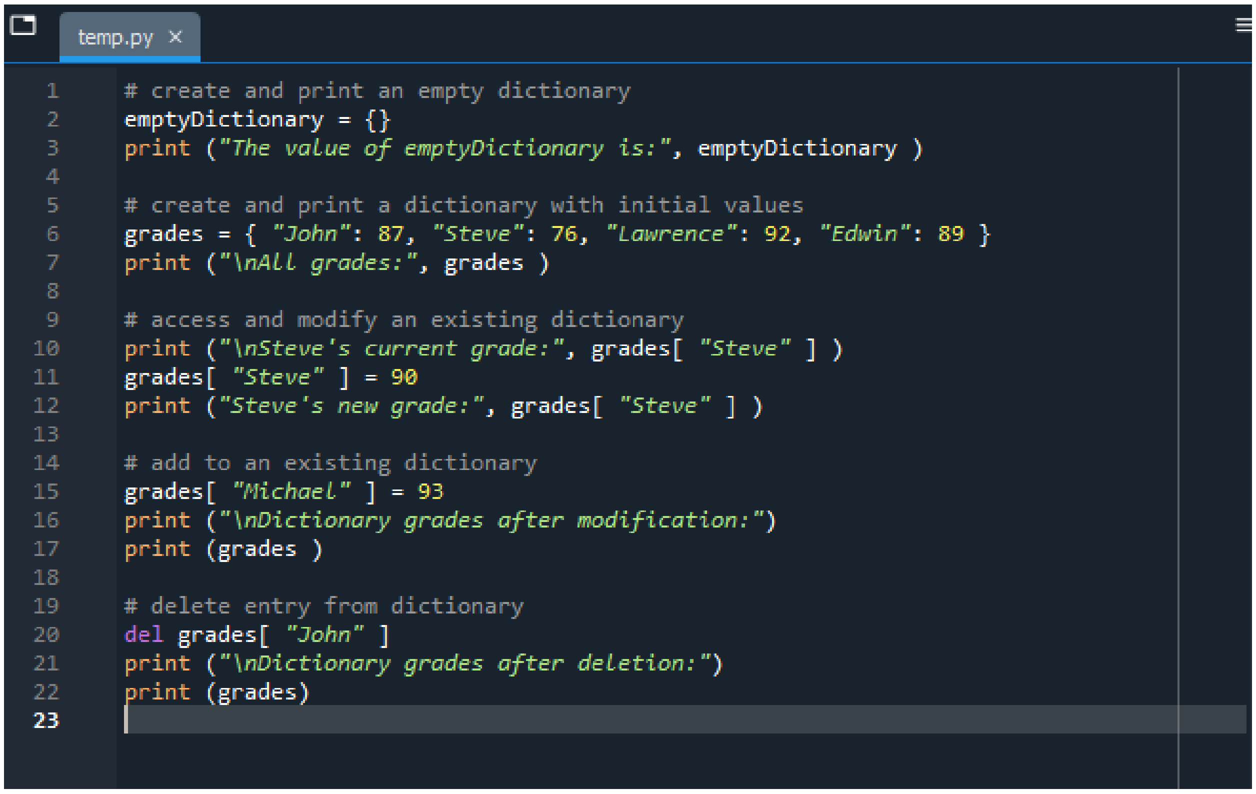
```
3
```

More examples:

Examples of dictionaries in real life include:

- A phone book:
  - Key – The phone number
  - Value – The persons name
- A physical dictionary (hence where the name of this data structure comes from)
  - Key – The word
  - Value – The description of the word (i.e. the definition).
- A Student identification number at a university
  - Key – the number
  - Value – The persons name
- Your Subway Loyalty Card:
  - Key – Your card number
  - Value – Your points you've amassed!

Code Snippet (example):

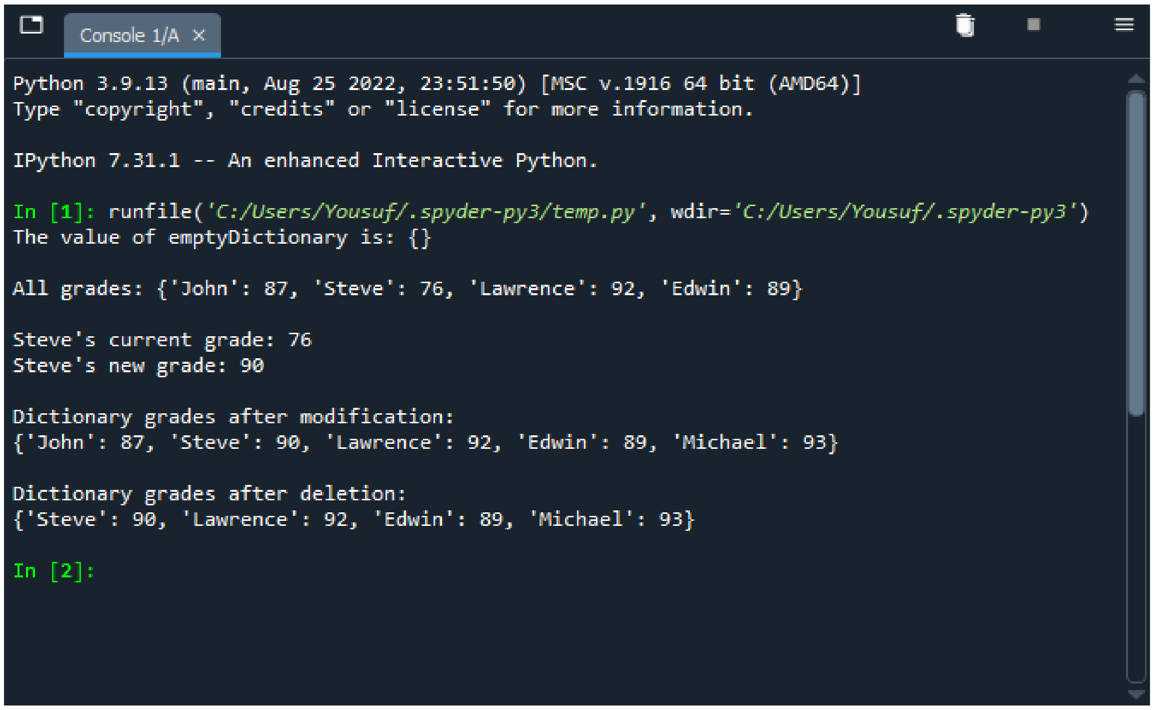


```python
# create and print an empty dictionary
emptyDictionary = {}
print ("The value of emptyDictionary is:", emptyDictionary )

# create and print a dictionary with initial values
grades = { "John": 87, "Steve": 76, "Lawrence": 92, "Edwin": 89 }
print ("\nAll grades:", grades )

# access and modify an existing dictionary
print ("\nSteve's current grade:", grades[ "Steve" ] )
grades[ "Steve" ] = 90
print ("Steve's new grade:", grades[ "Steve" ] )

# add to an existing dictionary
grades[ "Michael" ] = 93
print ("\nDictionary grades after modification:")
print (grades )

# delete entry from dictionary
del grades[ "John" ]
print ("\nDictionary grades after deletion:")
print (grades)
```

The very same **code** is here:

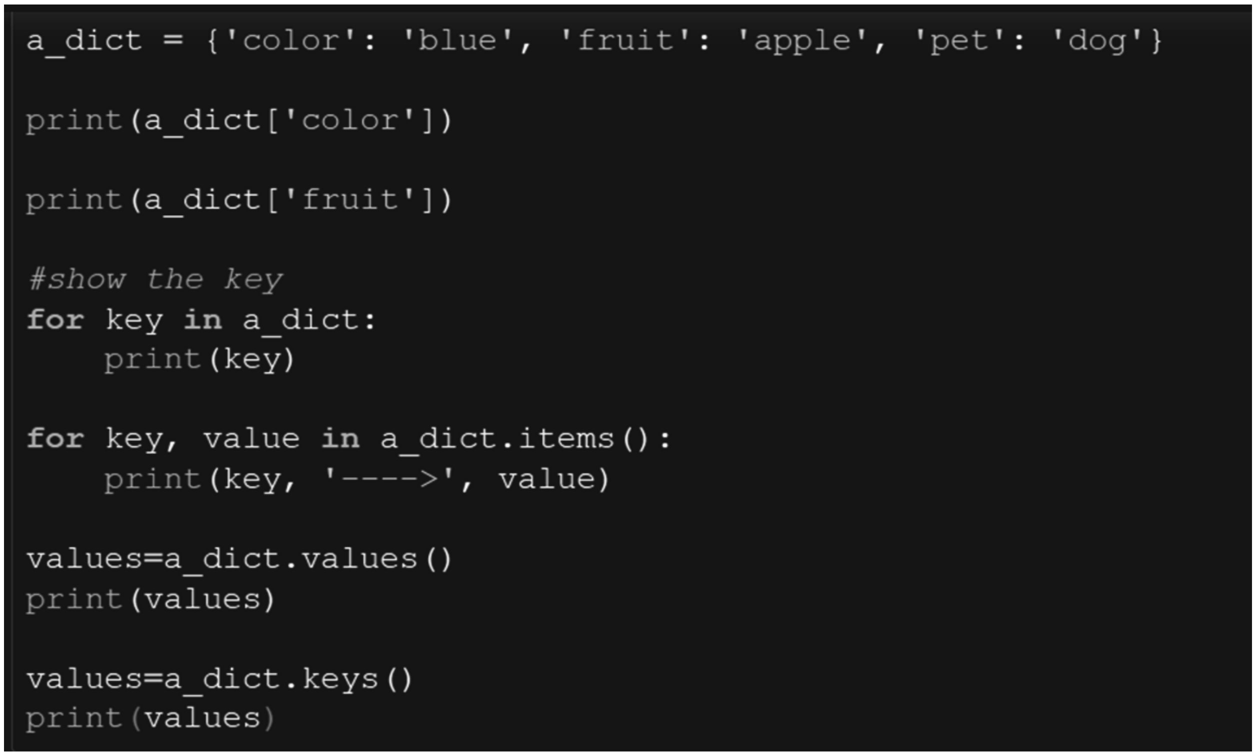
```
# create and print an empty dictionary
emptyDictionary = {}
print ("The value of emptyDictionary is:", emptyDictionary )

# create and print a dictionary with initial values
grades = { "John": 87, "Steve": 76, "Lawrence": 92, "Edwin": 89 }
print ("\nAll grades:", grades )

# access and modify an existing dictionary
print ("\nSteve's current grade:", grades[ "Steve" ] )
grades[ "Steve" ] = 90
print ("Steve's new grade:", grades[ "Steve" ] )

# add to an existing dictionary
grades[ "Michael" ] = 93
print ("\nDictionary grades after modification:")
print (grades )

# delete entry from dictionary
del grades[ "John" ]
print ("\nDictionary grades after deletion:")
print (grades)
```

**Output:**



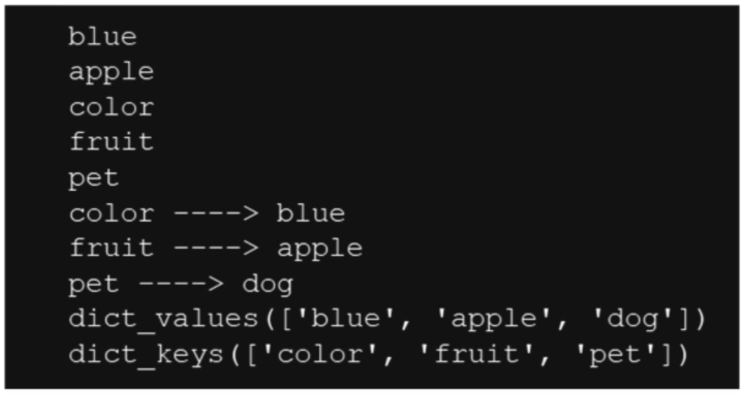
```
Python 3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.31.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-py3')
The value of emptyDictionary is: {}

All grades: {'John': 87, 'Steve': 76, 'Lawrence': 92, 'Edwin': 89}

Steve's current grade: 76
Steve's new grade: 90

Dictionary grades after modification:
{'John': 87, 'Steve': 90, 'Lawrence': 92, 'Edwin': 89, 'Michael': 93}

Dictionary grades after deletion:
{'Steve': 90, 'Lawrence': 92, 'Edwin': 89, 'Michael': 93}

In [2]:
```

Another example **code** snippet:



```
a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}

print(a_dict['color'])

print(a_dict['fruit'])

#show the key
for key in a_dict:
    print(key)

for key, value in a_dict.items():
    print(key, '---->', value)

values=a_dict.values()
print(values)

values=a_dict.keys()
print(values)
```

**Output:**



```
blue
apple
color
fruit
pet
color ----> blue
fruit ----> apple
pet ----> dog
dict_values(['blue', 'apple', 'dog'])
dict_keys(['color', 'fruit', 'pet'])
```

## What is Numpy?

Numpy, Scipy, and Matplotlib provide MATLAB-like functionalities in python.
**Numpy Features:**

- Typed multidimensional arrays (matrices)
- Fast numerical computations (matrix math)
- High-level math functions

## Python Libraries for Data Science

**Many popular Python toolboxes/libraries:**

- NumPy
- SciPy
- Pandas
- SciKit-Learn

**Visualization libraries**

- matplotlib
- Seaborn

  and many more …

***NumPy*** is the fundamental package needed for scientific computing with Python.
**It contains:**

- A powerful N-dimensional array object
- basic linear algebra functions
- basic Fourier transforms

## Why do we need NumPy

Python does numerical computations slowly.
e.g. 1000 x 1000 matrix multiply
Python triple loop takes > 10 min.
Numpy takes ~0.03 seconds

## NumPy Overview

- Arrays
- Shaping and transposition
- Mathematical Operations
- Indexing and slicing
- Broadcasting

## Arrays

**Structured lists of numbers:**

- ☐ Vectors
- ☐ Matrices

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

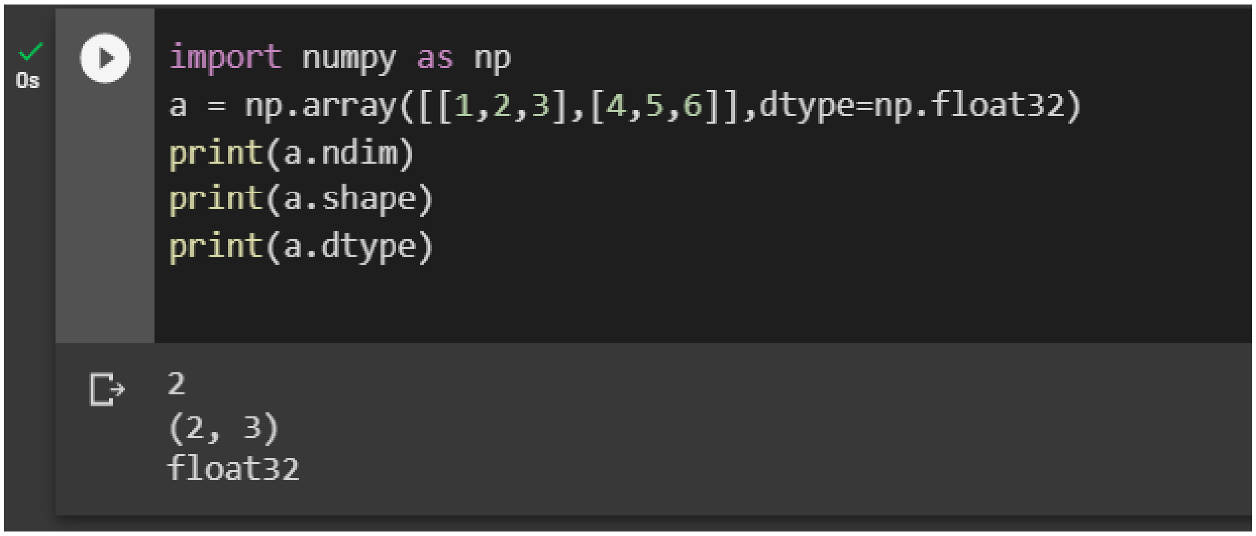
$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

## Arrays, Basic Properties

**Code example:**

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
print(a.ndim)
print(a.shape)
print(a.dtype)
```

**Screenshot for the code deployed in google colab:**



```
import numpy as np
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
print(a.ndim)
print(a.shape)
print(a.dtype)
```

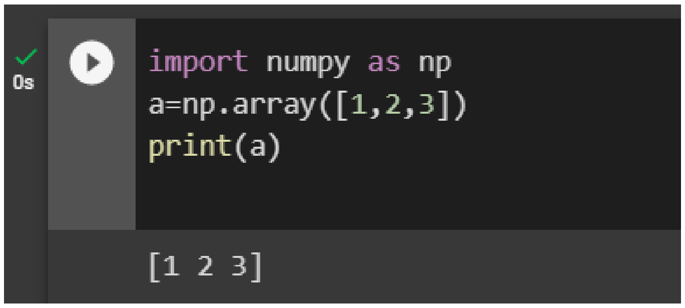
```
2
(2, 3)
float32
```

Note that:

1. Arrays can have any number of dimensions, including zero (a scalar).
2. Arrays are typed: np.uint8, np.int64, np.float32, np.float64
3. Arrays are dense. Each element of the array exists and has the same type.

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

Deployed code in google colab:



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

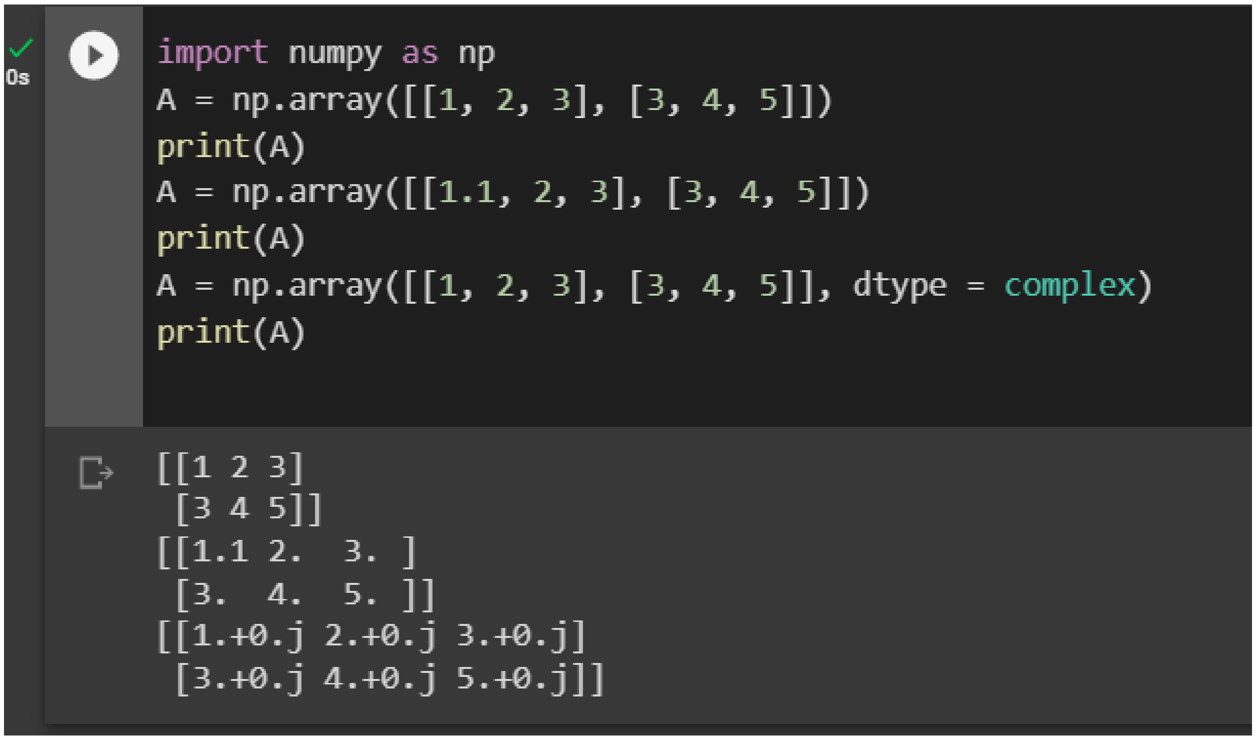
```
[1 2 3]
```

Single-dimensional Numpy Array
**Output** – [1 2 3]

```
nums = np.array([[2,4,6], [8,10,12], [14,16,18]])
```

```
import numpy as np
A = np.array([[1, 2, 3], [3, 4, 5]])
print(A)
A = np.array([[1.1, 2, 3], [3, 4, 5]])
print(A)
A = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex)
print(A)
```

Screenshot of the code portion in google colab:



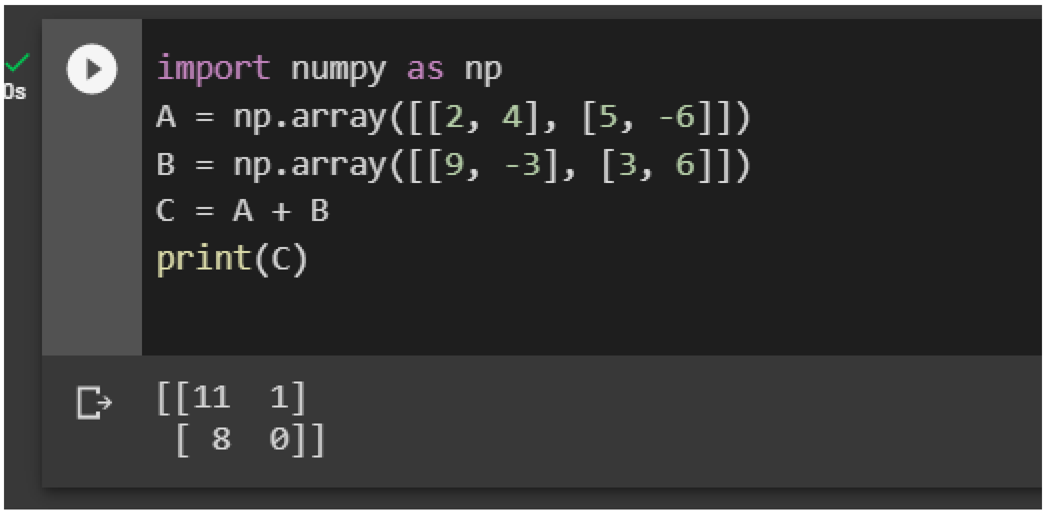
```
import numpy as np
A = np.array([[1, 2, 3], [3, 4, 5]])
print(A)
A = np.array([[1.1, 2, 3], [3, 4, 5]])
print(A)
A = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex)
print(A)
```

```
[[1 2 3]
 [3 4 5]]
[[1.1 2.  3. ]
 [3.  4.  5. ]]
[[1.+0.j 2.+0.j 3.+0.j]
 [3.+0.j 4.+0.j 5.+0.j]]
```

## Arrays, creation

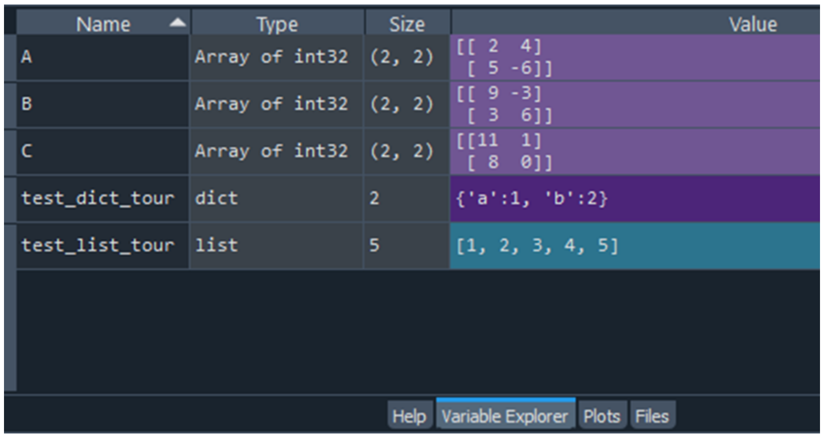
```
import numpy as np
A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A + B
print(C)
```

**Output:**



```
import numpy as np
A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A + B
print(C)
```

```
[[11  1]
 [ 8  0]]
```

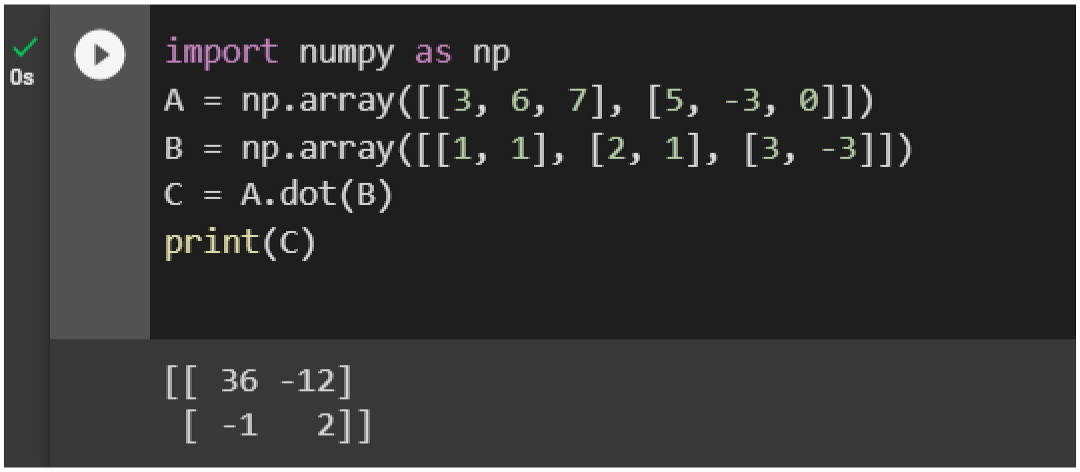


| Name | Type | Size | Value |
|---|---|---|---|
| A | Array of int32 | (2, 2) | [[ 2  4] [ 5 -6]] |
| B | Array of int32 | (2, 2) | [[ 9 -3] [ 3  6]] |
| C | Array of int32 | (2, 2) | [[11  1] [ 8  0]] |
| test_dict_tour | dict | 2 | {'a':1, 'b':2} |
| test_list_tour | list | 5 | [1, 2, 3, 4, 5] |

Another example **input code**:

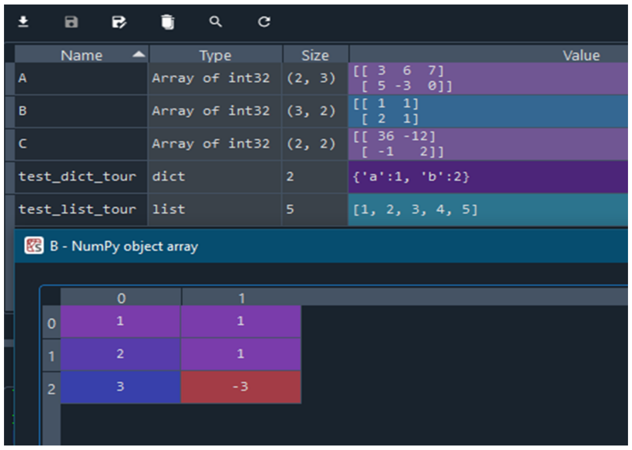
```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
print(C)
```

**Output:**



```
import numpy as np
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
print(C)
```

```
[[ 36 -12]
 [ -1   2]]
```



| Name | Type | Size | Value |
|---|---|---|---|
| A | Array of int32 | (2, 3) | [[ 3  6  7]<br> [ 5 -3  0]] |
| B | Array of int32 | (3, 2) | [[ 1  1]<br> [ 2  1]] |
| C | Array of int32 | (2, 2) | [[ 36 -12]<br> [ -1   2]] |
| test_dict_tour | dict | 2 | {'a':1, 'b':2} |
| test_list_tour | list | 5 | [1, 2, 3, 4, 5] |

B - NumPy object array

|  | 0 | 1 |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 2 | 1 |
| 2 | 3 | -3 |

## The zeros Method

We can use the **zeros** method to create an array of all zeros as shown below:

```
zeros = np.zeros(5)
```

```
zeros = np.zeros((5, 4))
```

```
zeros = np.ones((3,5), dtype=float)
```

## The ones Method

We can create one-dimensional and two-dimensional arrays of all ones using the ones method

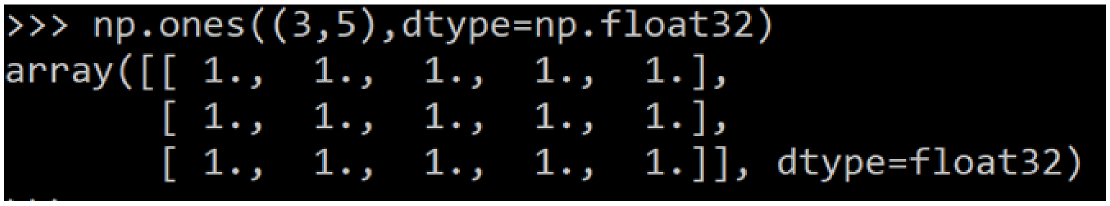
```
ones = np.ones(5)
```

```
ones = np.ones((5, 4))
```

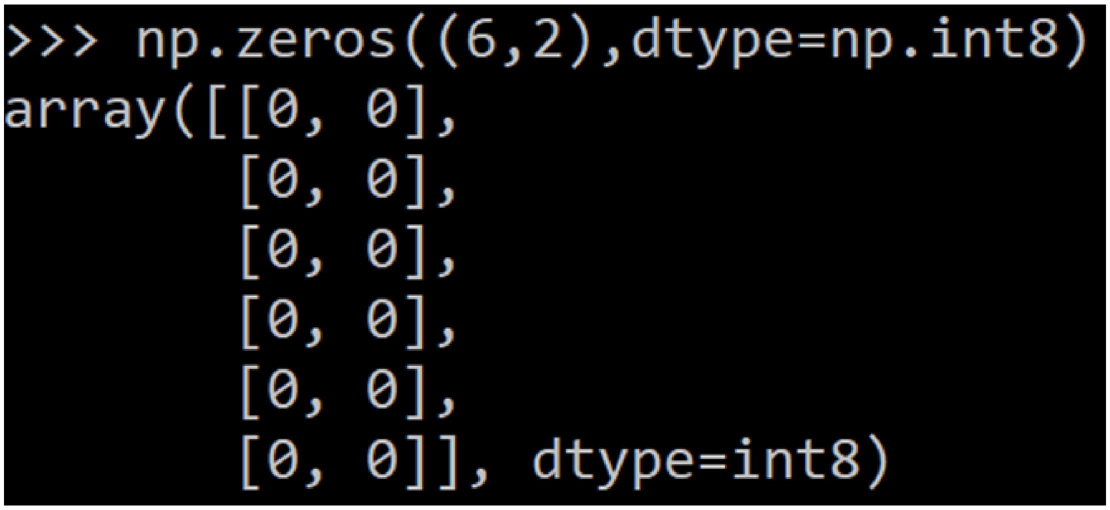
```
zeros = np.ones((3,5), dtype=float)
```

**Functions:**

- *np.ones, np.zeros*
- np.arange
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

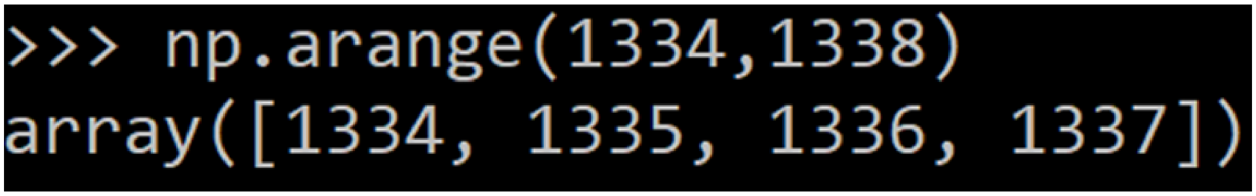


```
>>> np.ones((3,5),dtype=np.float32)
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]], dtype=float32)
```



```
>>> np.zeros((6,2),dtype=np.int8)
array([[0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0]], dtype=int8)
```

## np.arange



```
>>> np.arange(1334,1338)
array([1334, 1335, 1336, 1337])
```

**The arange method**
This method takes the start index of the array, the end index, and the step size

**Syntax:**
*numpy.arange(start, stop, step, dtype)*

**Python NumPy arange Parameters:**
*Start: Start of interval for np.arange in Python function.*
*Stop: End of interval.*
*Step: Spacing between values. Default step is 1.*
*Dtype: Is a type of array output for NumPy arange in Python.*

## The arange method

This method takes the start index of the array, the end index, and the step size

```
nums = np.arange(2, 7)
```

```
nums = np.arange(2, 7, 2)
```

## The linspace Method

*numpy.linspace(start, stop, num, endpoint)*

- Start: Starting value of the sequence
- Stop: End value of the sequence
- Num: Number of samples to generate. Default is 50
- Endpoint: If True (default), stop is the last value. If False, stop value is not included.

```
nums = np.linspace(1.0, 5.0, num=10)
```

```
nums = np.linspace(1.0, 5.0, num=5, endpoint=False)
```

```
lin = np.linspace(1, 10, 20)
```

## The random Method

We can use the randint.

**Example:**
Generate a random integer from 0 to 100:

```
x = np.random.randint(100)
```

```
x=np.random.randint(100, size=(5))
```

```
x=np. random.randint(100, size=(3, 5))
```

```
random = np.random.randint(50, 100, 5)
```

The randint method takes the lower bound, upper bound, and the number of integers to return.

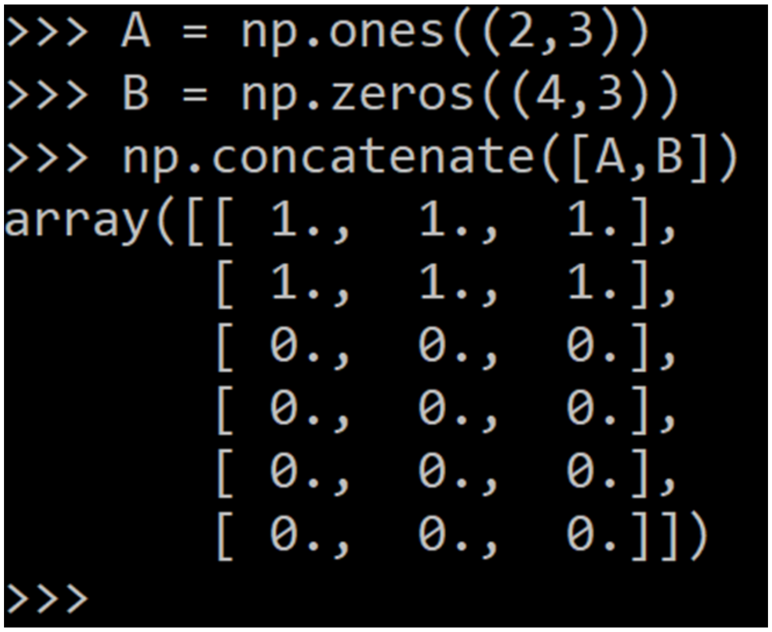
We can use the rand function for the floating point numbers.

```
random = np. random.rand(5)
```

```
random = np.random.rand(2, 3)
```

returns a matrix of 2 rows and 3 columns.

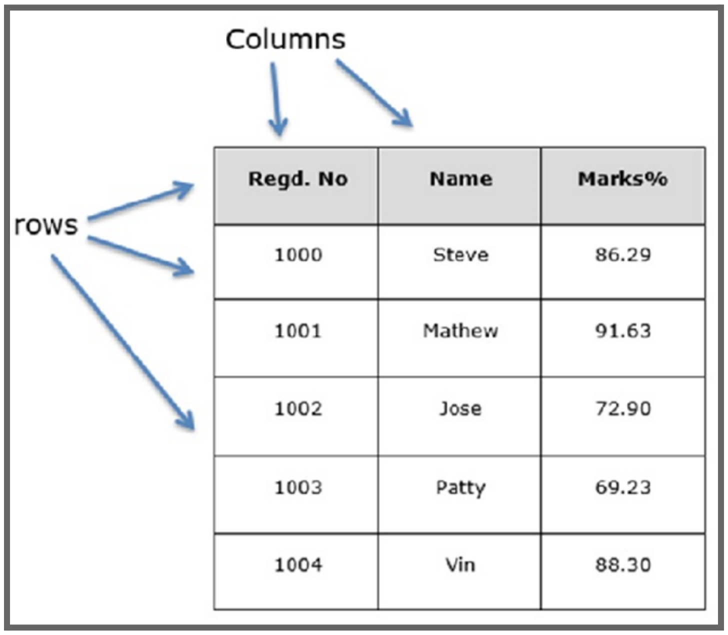
## np.concatenate



```
>>> A = np.ones((2,3))
>>> B = np.zeros((4,3))
>>> np.concatenate([A,B])
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>>
```

## DataFrame

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.



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

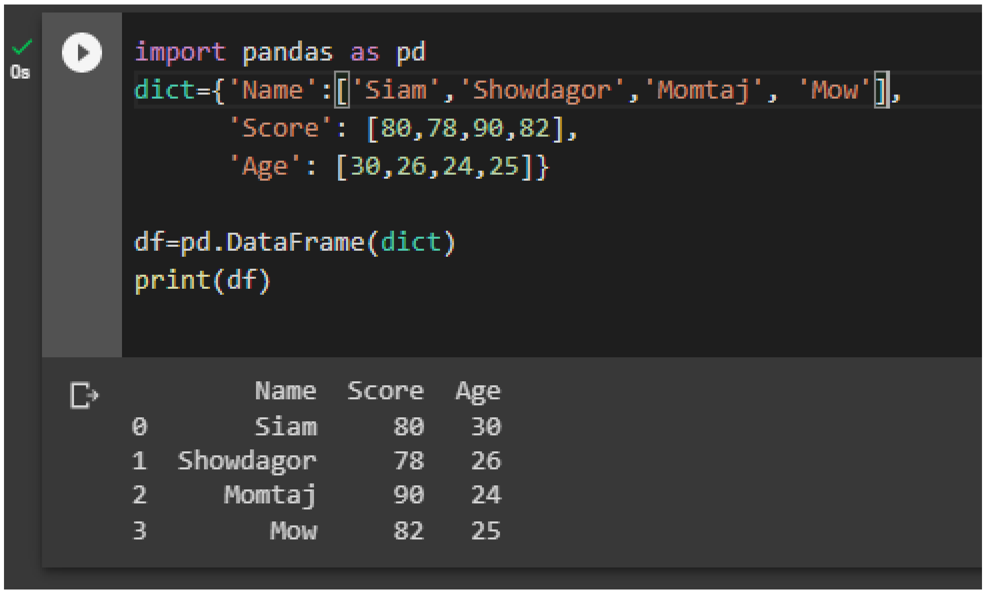
**Includes:**

- Features of DataFrame
- Potentially columns are of different types
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

**Example:**

```
import pandas as pd
dict={'Name':['Siam','Showdagor','Momtaj
', 'Mow'],
      'Score': [80,78,90,82],
      'Age': [30,26,24,25]}

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

**Code and execution in IDE (colab- google):**



```
import pandas as pd
dict={'Name':['Siam','Showdagor','Momtaj', 'Mow'],
      'Score': [80,78,90,82],
      'Age': [30,26,24,25]}

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

```
        Name  Score  Age
0       Siam     80   30
1  Showdagor     78   26
2     Momtaj     90   24
3        Mow     82   25
```

## Pandas

**pandas** is an open source library in Python. It provides ready to use high-performance data structures and data analysis tools. Pandas module runs on top of NumPy and it is popularly used for data science and data analytics.

NumPy is a low-level data structure that supports multi-dimensional arrays and a wide range of mathematical array operations. Pandas has a higher-level interface. It also provides streamlined alignment of tabular data and powerful time series functionality.

DataFrame is the key data structure in Pandas. It allows us to store and manipulate tabular data as a 2-D data structure. Pandas provides a rich feature-set on the DataFrame. For example, data alignment, data statistics, slicing, grouping, merging, concatenating data, etc.

There are 3 data structures provided by the Pandas module, which are as follows:

- Series: It is a 1-D size-immutable array-like structure having homogeneous data.
- DataFrames: It is a 2-D size-mutable tabular structure with heterogeneously typed columns.
- Panel: It is a 3-D, size-mutable array.

DataFrame is the most important and widely used data structure and is a standard way to store data. DataFrame has data aligned in rows and columns like the SQL table or a spreadsheet database. We can either hard code data into a DataFrame or import a CSV file, tsv file, Excel file, SQL table, etc. We can use the below constructor for creating a DataFrame object.
For example:

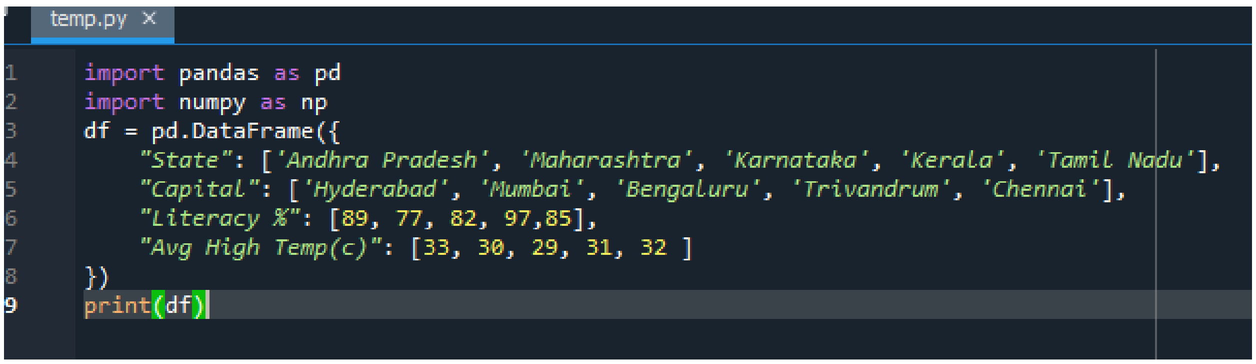
```
pandas.DataFrame(data, index, columns, dtype, copy)
```

Below is a short description of the parameters:

- data - create a DataFrame object from the input data. It can be list, dict, series, Numpy ndarrays or even, any other DataFrame.
- index - has the row labels
- columns - used to create column labels
- dtype - used to specify the data type of each column, optional parameter
- copy - used for copying data, if any

There are many ways to create a DataFrame. We can create a DataFrame object from Dictionaries or a list of dictionaries. We can also create it from a list of tuples, CSV, Excel files, etc. Let’s run a simple code to create a DataFrame from the list of dictionaries.

For example the below code snippet:

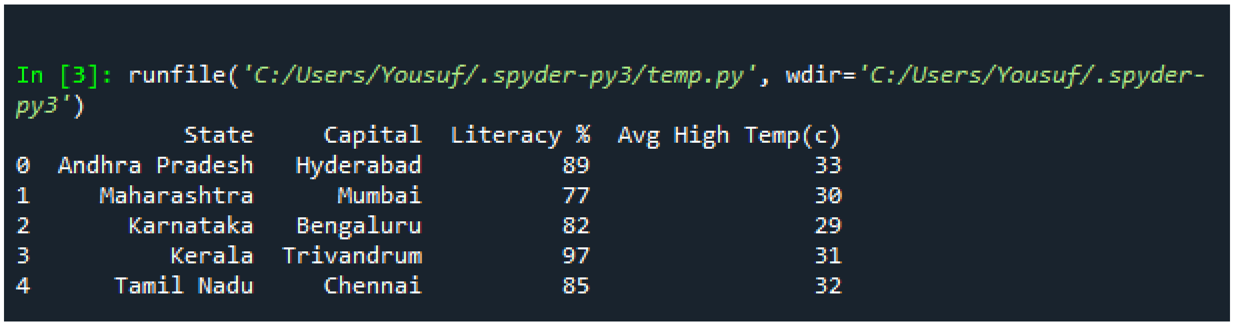


```
temp.py ×
import pandas as pd
import numpy as np
df = pd.DataFrame({
    "State": ['Andhra Pradesh', 'Maharashtra', 'Karnataka', 'Kerala', 'Tamil Nadu'],
    "Capital": ['Hyderabad', 'Mumbai', 'Bengaluru', 'Trivandrum', 'Chennai'],
    "Literacy %": [89, 77, 82, 97,85],
    "Avg High Temp(c)": [33, 30, 29, 31, 32 ]
})
print(df)
```

The original **code**:

```
import pandas as pd
import numpy as np
df = pd.DataFrame({
    "State": ['Andhra Pradesh', 'Maharashtra', 'Karnataka',
'Kerala', 'Tamil Nadu'],
    "Capital": ['Hyderabad', 'Mumbai', 'Bengaluru', 'Trivandrum',
'Chennai'],
    "Literacy %": [89, 77, 82, 97,85],
    "Avg High Temp(c)": [33, 30, 29, 31, 32 ]
})
print(df)
```

**Output:**

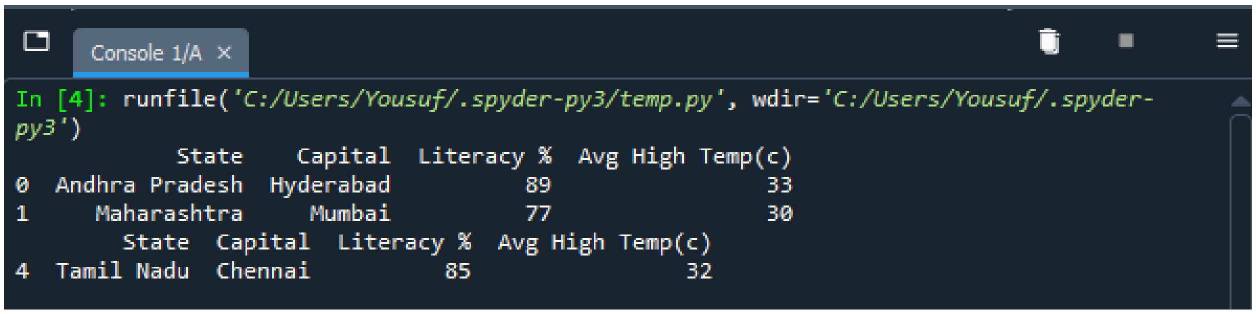


```
In [3]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
            State     Capital  Literacy %  Avg High Temp(c)
0  Andhra Pradesh   Hyderabad          89                33
1     Maharashtra      Mumbai          77                30
2       Karnataka   Bengaluru          82                29
3          Kerala  Trivandrum          97                31
4      Tamil Nadu     Chennai          85                32
```

**Input:**

```
print(df.head(2))
```

**Output:**

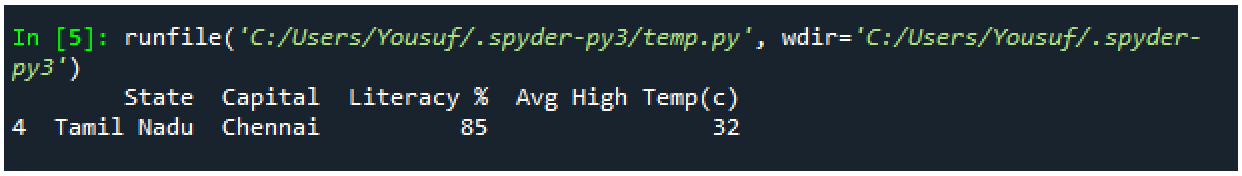


```
Console 1/A ×
In [4]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
            State    Capital  Literacy %  Avg High Temp(c)
0  Andhra Pradesh  Hyderabad          89                33
1     Maharashtra     Mumbai          77                30
        State  Capital  Literacy %  Avg High Temp(c)
4  Tamil Nadu  Chennai          85                32
```

**Input:**

```
print(df.tail(1))
```

**Output:**



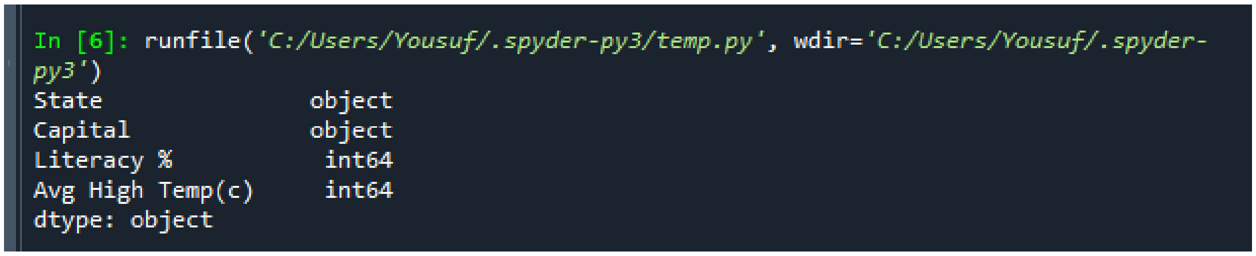
```
In [5]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
        State  Capital  Literacy %  Avg High Temp(c)
4  Tamil Nadu  Chennai          85                32
```

Similarly, *print(df.dtypes)* prints the data types.

**Code:**

```
print(df.dtypes)
```

**Output:**

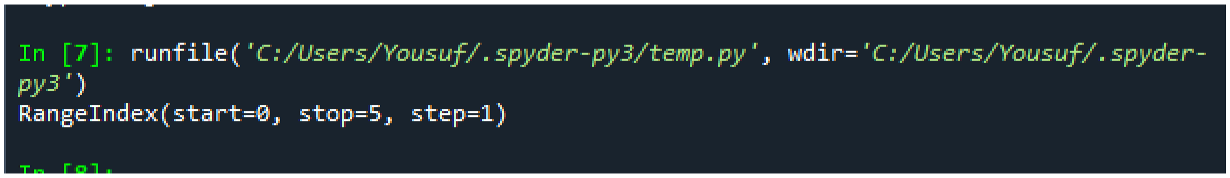


```
In [6]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
State               object
Capital             object
Literacy %           int64
Avg High Temp(c)     int64
dtype: object
```

Again, another **code** snippet:

```
print(df.index)
```

**Output:**

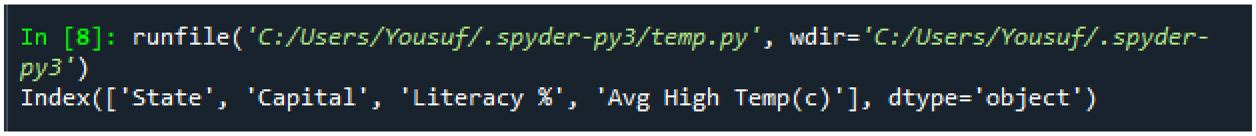


```
In [7]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
RangeIndex(start=0, stop=5, step=1)
```

**Code:**

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

**Output:**

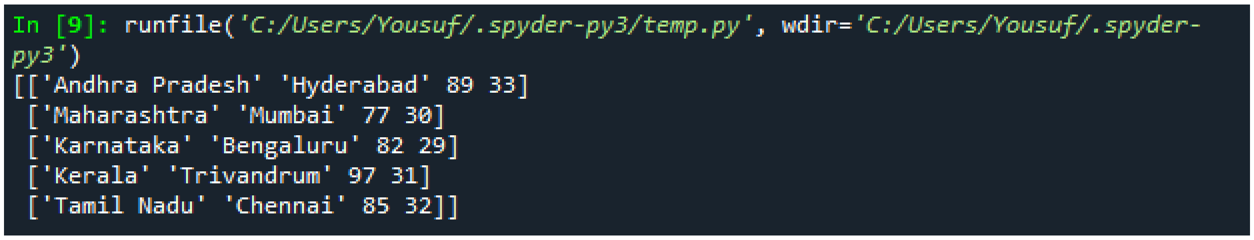


```
In [8]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
Index(['State', 'Capital', 'Literacy %', 'Avg High Temp(c)'], dtype='object')
```

**Code:**

```
print(df.values)
```

**Output:**



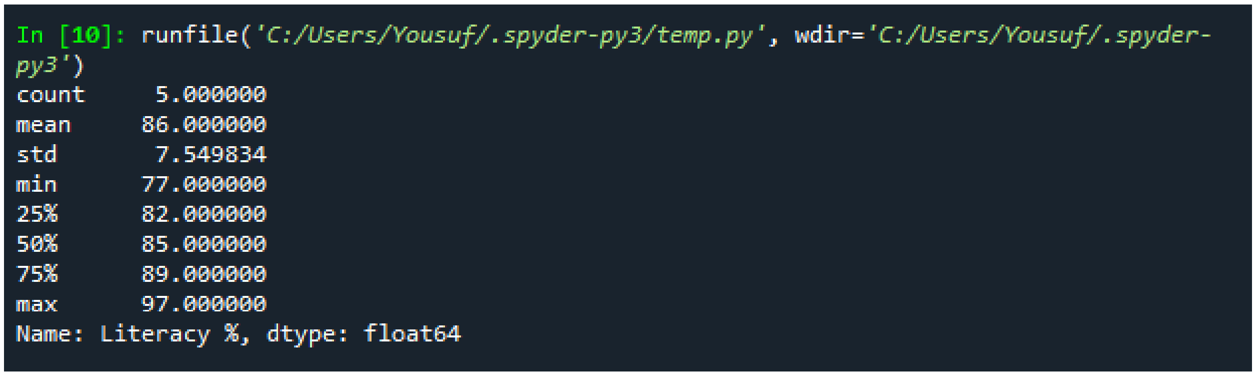
```
In [9]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
[['Andhra Pradesh' 'Hyderabad' 89 33]
 ['Maharashtra' 'Mumbai' 77 30]
 ['Karnataka' 'Bengaluru' 82 29]
 ['Kerala' 'Trivandrum' 97 31]
 ['Tamil Nadu' 'Chennai' 85 32]]
```

## Getting Statistical Summary of Record & Values

We can get a statistical summary (count, mean, standard deviation, min, max etc.) of the data using **df.describe()** function. Now, let’s use this function to display the statistical summary of the “Literacy %” column. To do this, we may add the below piece of code:

```
print(df['Literacy %'].describe())
```

**Output:**



```
In [10]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
count     5.000000
mean     86.000000
std       7.549834
min      77.000000
25%      82.000000
50%      85.000000
75%      89.000000
max      97.000000
Name: Literacy %, dtype: float64
```

The df.describe() function displays the statistical summary, along with the data type.

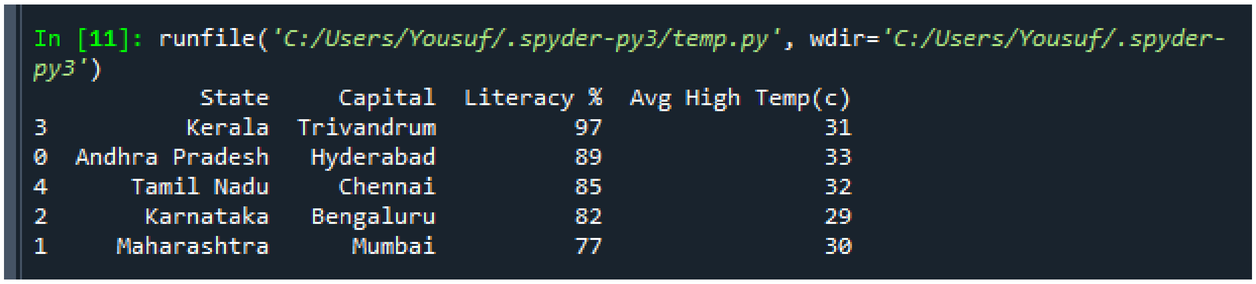
## Sorting records

We can sort records by any column using df.sort_values() function. For example, let’s sort the “Literacy %” column in descending order.

**Input Code Portion:**

```
print(df.sort_values('Literacy %', ascending=False))
```

**Output:**



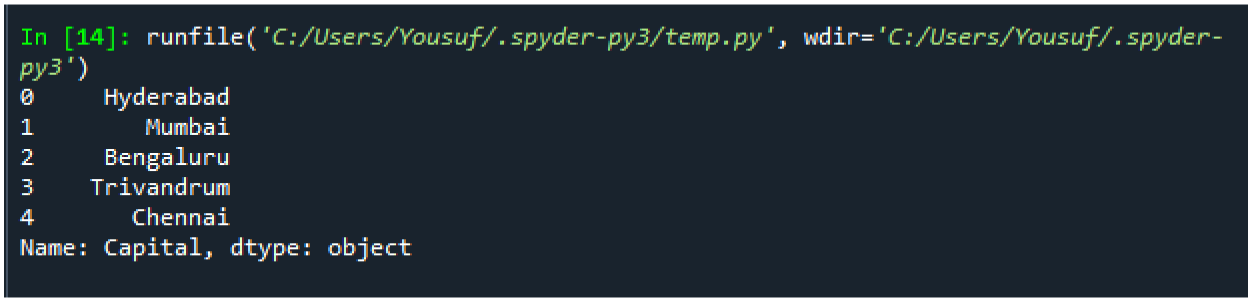
```
In [11]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
            State     Capital  Literacy %  Avg High Temp(c)
3          Kerala  Trivandrum          97                31
0  Andhra Pradesh   Hyderabad          89                33
4      Tamil Nadu     Chennai          85                32
2       Karnataka   Bengaluru          82                29
1     Maharashtra      Mumbai          77                30
```

## Slicing Records

It is possible to extract data of a particular column, by using the column name. For example, to extract the ‘Capital’ column, we use:

```
print(df.Capital)
```

**Output:**

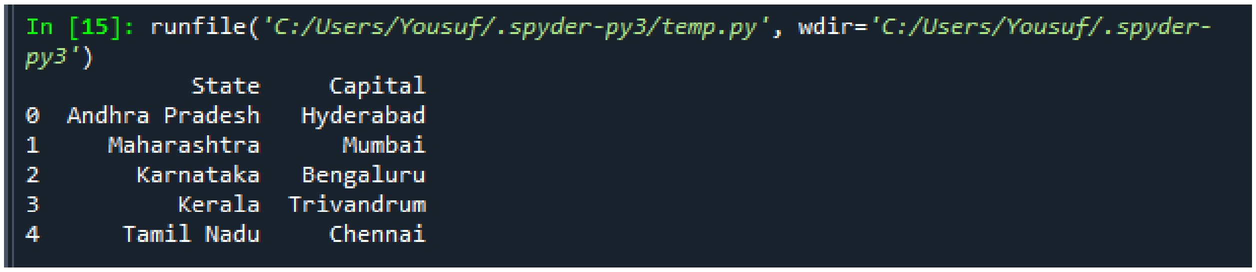


```
In [14]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
0     Hyderabad
1        Mumbai
2     Bengaluru
3    Trivandrum
4       Chennai
Name: Capital, dtype: object
```

It is also possible to slice multiple columns. This is done by enclosing multiple column names enclosed in 2 square brackets, with the column names separated using commas. The following code slices the ‘State’ and ‘Capital’ columns of the DataFrame.

```
print(df[['State', 'Capital']])
```

**Output:**



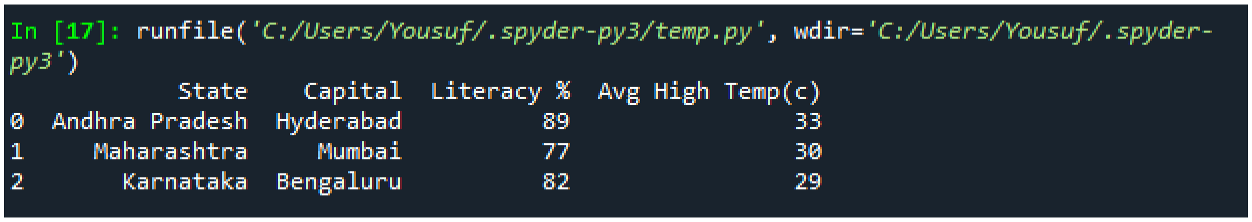
```
In [15]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
            State    Capital
0  Andhra Pradesh  Hyderabad
1     Maharashtra     Mumbai
2       Karnataka  Bengaluru
3          Kerala Trivandrum
4      Tamil Nadu    Chennai
```

It is also possible to slice rows. Multiple rows can be selected using ":" operator. The below code returns the first 3 rows.

**Code:**

```
print(df[0:3])
```

**Output:**



```
In [17]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
            State    Capital  Literacy %  Avg High Temp(c)
0  Andhra Pradesh  Hyderabad          89                33
1     Maharashtra     Mumbai          77                30
2       Karnataka  Bengaluru          82                29
```

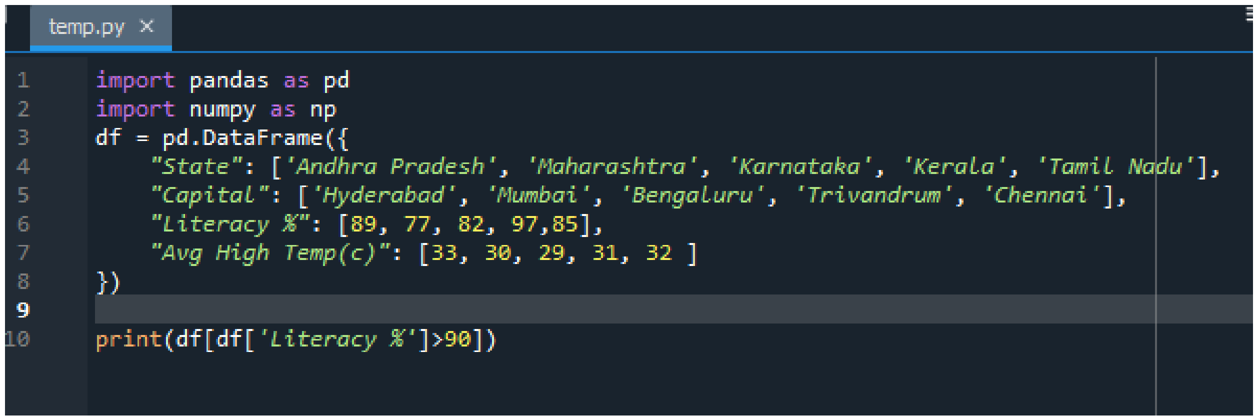
An interesting feature of Pandas library is to select data based on its row and column labels using the iloc[0] function. Many times, we may need only a few columns to analyze. We can also select by index using loc['index_one']). For example, to select the second row, we can use df.iloc[1,:] . Let's say, we need to select the second element of the second column. This can be done by using the df.iloc[1,1] function. In this example, the function df.iloc[1,1] displays "Mumbai" as output.

## Filtering data

It is also possible to filter on column values. For example, below code filters the columns having Literacy% above 90%.
Code portion:

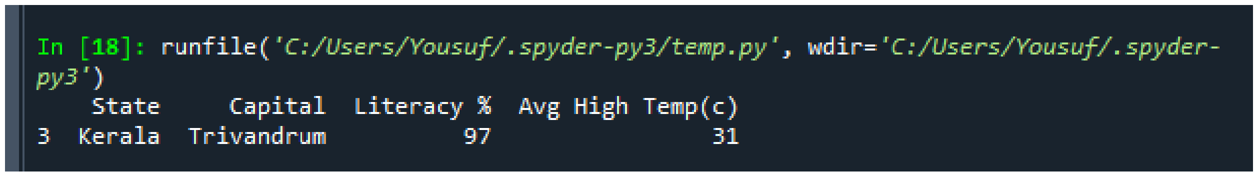
```
print(df[df['Literacy %']>90])
```

The screenshot of this **code** in Spyder:



```
import pandas as pd
import numpy as np
df = pd.DataFrame({
    "State": ['Andhra Pradesh', 'Maharashtra', 'Karnataka', 'Kerala', 'Tamil Nadu'],
    "Capital": ['Hyderabad', 'Mumbai', 'Bengaluru', 'Trivandrum', 'Chennai'],
    "Literacy %": [89, 77, 82, 97,85],
    "Avg High Temp(c)": [33, 30, 29, 31, 32 ]
})

print(df[df['Literacy %']>90])
```

**Output**:



```
In [18]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
    State     Capital  Literacy %  Avg High Temp(c)
3  Kerala  Trivandrum          97                31
```

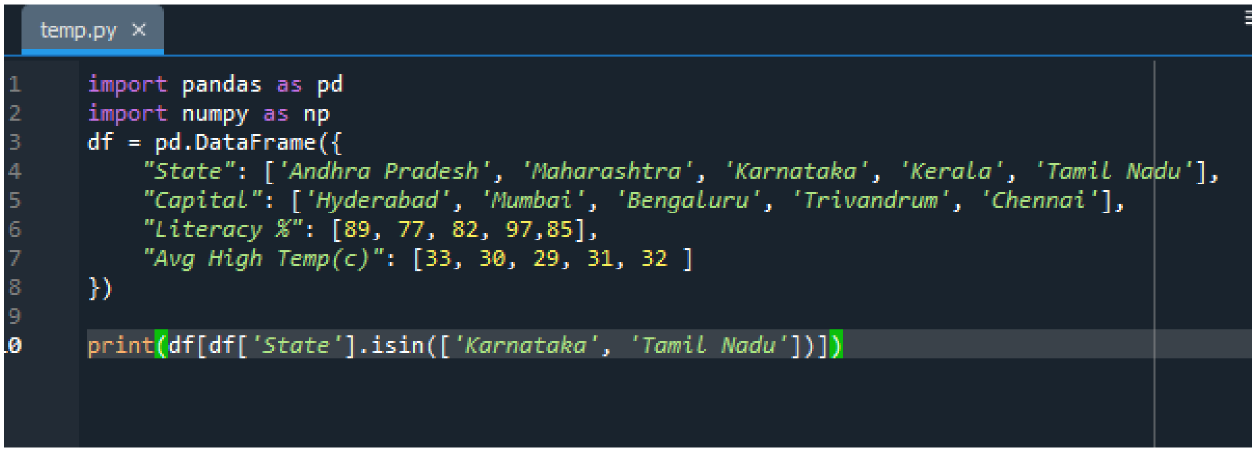
Another way to filter data is using the isin. Following is the code to filter only 2 states 'Karnataka' and 'Tamil Nadu'.

```
print(df[df['State'].isin(['Karnataka', 'Tamil Nadu'])])
```

**Code** (Original):

```
import pandas as pd
import numpy as np
df = pd.DataFrame({
    "State": ['Andhra Pradesh', 'Maharashtra', 'Karnataka',
'Kerala', 'Tamil Nadu'],
    "Capital": ['Hyderabad', 'Mumbai', 'Bengaluru', 'Trivandrum',
'Chennai'],
    "Literacy %": [89, 77, 82, 97,85],
    "Avg High Temp(c)": [33, 30, 29, 31, 32 ]
})

print(df[df['State'].isin(['Karnataka', 'Tamil Nadu'])])
```

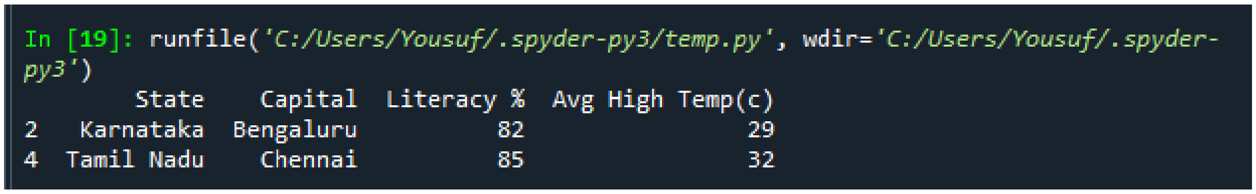
**SS of code in Spyder:**



temp.py

```
import pandas as pd
import numpy as np
df = pd.DataFrame({
    "State": ['Andhra Pradesh', 'Maharashtra', 'Karnataka', 'Kerala', 'Tamil Nadu'],
    "Capital": ['Hyderabad', 'Mumbai', 'Bengaluru', 'Trivandrum', 'Chennai'],
    "Literacy %": [89, 77, 82, 97,85],
    "Avg High Temp(c)": [33, 30, 29, 31, 32 ]
})

print(df[df['State'].isin(['Karnataka', 'Tamil Nadu'])])
```

**Output**:



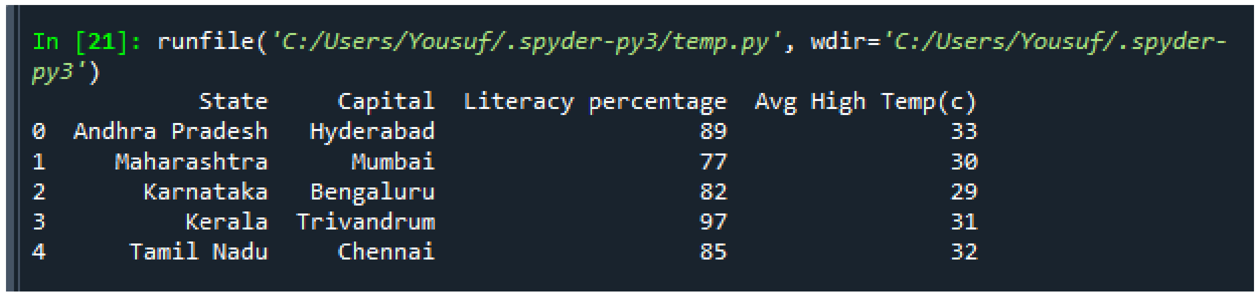
```
In [19]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
        State    Capital  Literacy %  Avg High Temp(c)
2   Karnataka  Bengaluru          82                29
4  Tamil Nadu    Chennai          85                32
```

## Rename column

It is possible to use the df.rename() function to rename a column. The function takes the old column name and new column name as arguments. For example, let's rename the column 'Literacy %' to 'Literacy percentage'.

```
df.rename(columns = {'Literacy %':'Literacy percentage'},
inplace=True)
print(df.head())
```

**Output**:



```
In [21]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
             State     Capital  Literacy percentage  Avg High Temp(c)
0   Andhra Pradesh   Hyderabad                   89                33
1      Maharashtra      Mumbai                   77                30
2        Karnataka   Bengaluru                   82                29
3           Kerala  Trivandrum                   97                31
4       Tamil Nadu     Chennai                   85                32
```

The argument `inplace=True` makes the changes to the DataFrame.

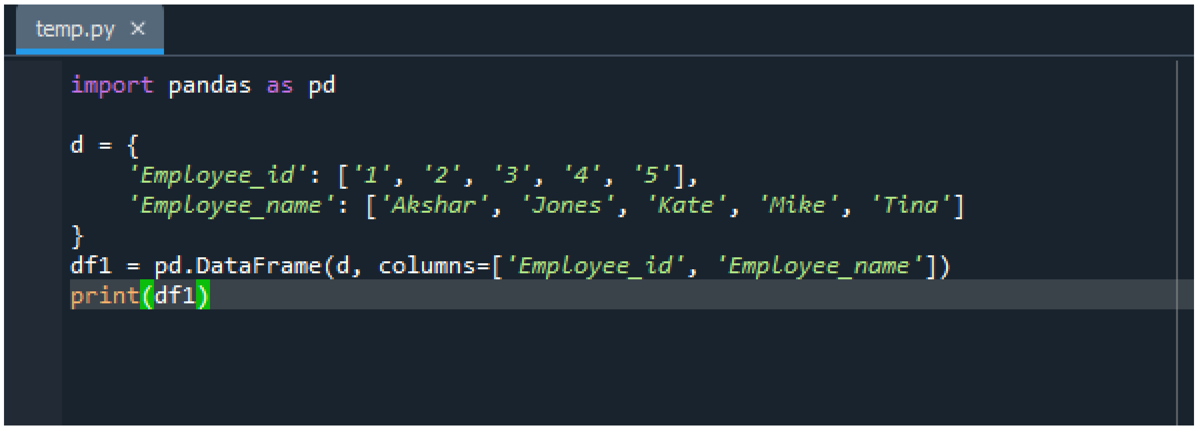
## Data Wrangling

Data Science involves the processing of data so that the data can work well with the data algorithms. Data Wrangling is the process of processing data, like merging, grouping and concatenating. The Pandas library provides useful functions like merge(), groupby() and concat() to support Data Wrangling tasks. Let's create 2 DataFrames and show the Data Wrangling functions to understand it better.

**Code:**

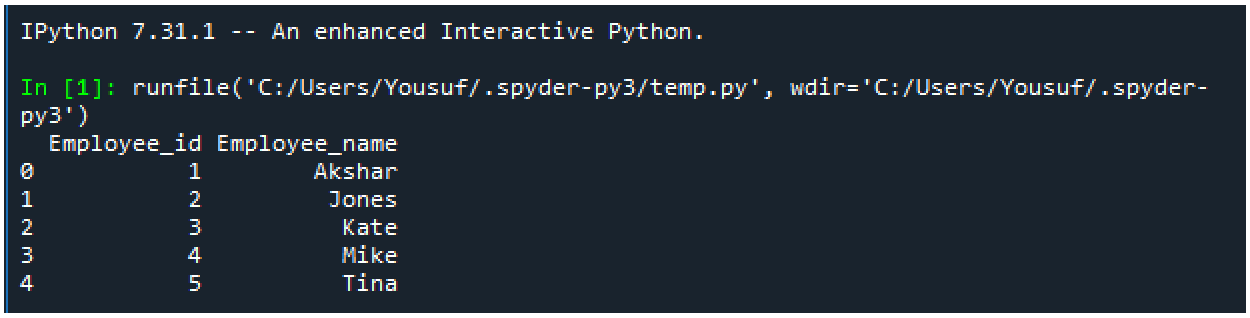
```
import pandas as pd

d = {
    'Employee_id': ['1', '2', '3', '4', '5'],
    'Employee_name': ['Akshar', 'Jones', 'Kate', 'Mike', 'Tina']
}
df1 = pd.DataFrame(d, columns=['Employee_id', 'Employee_name'])
print(df1)
```

**Screenshot** of Code in Spyder:



```
import pandas as pd

d = {
    'Employee_id': ['1', '2', '3', '4', '5'],
    'Employee_name': ['Akshar', 'Jones', 'Kate', 'Mike', 'Tina']
}
df1 = pd.DataFrame(d, columns=['Employee_id', 'Employee_name'])
print(df1)
```

**Output**:

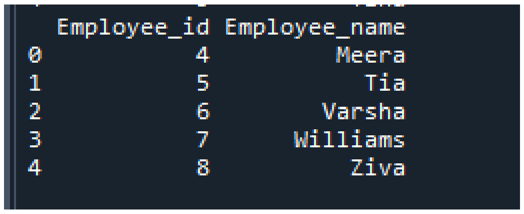


```
IPython 7.31.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
  Employee_id Employee_name
0           1        Akshar
1           2         Jones
2           3          Kate
3           4          Mike
4           5          Tina
```

Let’s create the second DataFrame using the below code:

```
import pandas as pd

data = {
    'Employee_id': ['4', '5', '6', '7', '8'],
    'Employee_name': ['Meera', 'Tia', 'Varsha', 'Williams', 'Ziva']
}
df2 = pd.DataFrame(data, columns=['Employee_id', 'Employee_name'])
print(df2)
```

**Output**:



```
  Employee_id Employee_name
0           4         Meera
1           5           Tia
2           6        Varsha
3           7      Williams
4           8          Ziva
```

## a. Merging

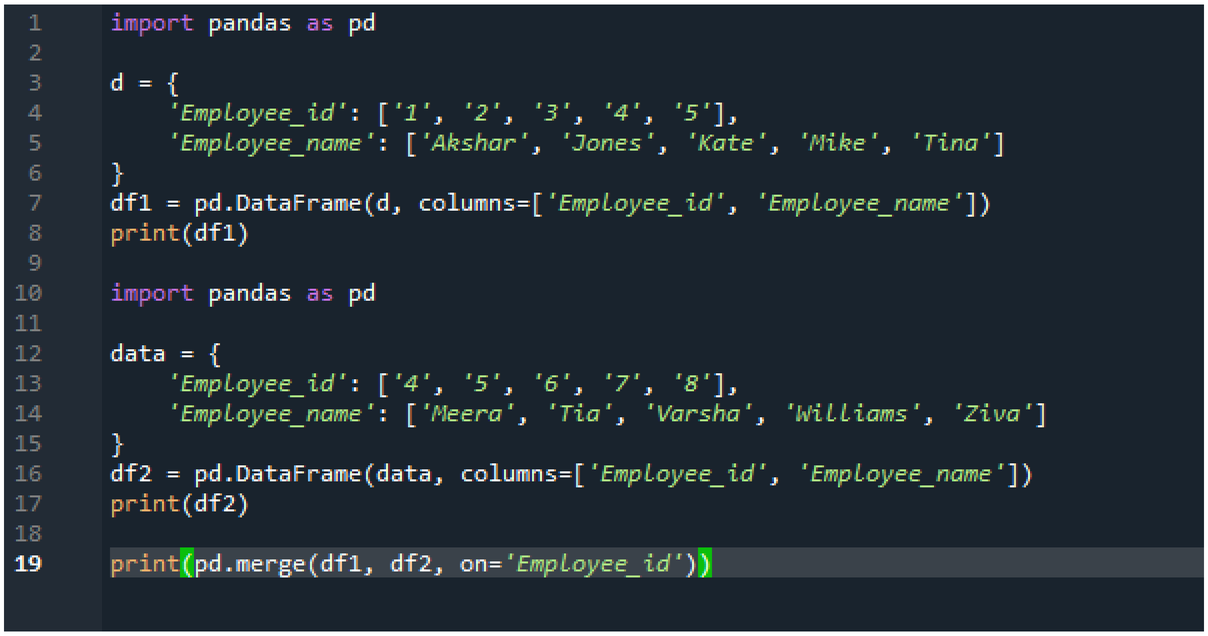
Now, let’s merge the 2 DataFrames we created, along the values of ‘Employee_id’ using the merge() function:

```
print(pd.merge(df1, df2, on='Employee_id'))
```

Whole **code**:

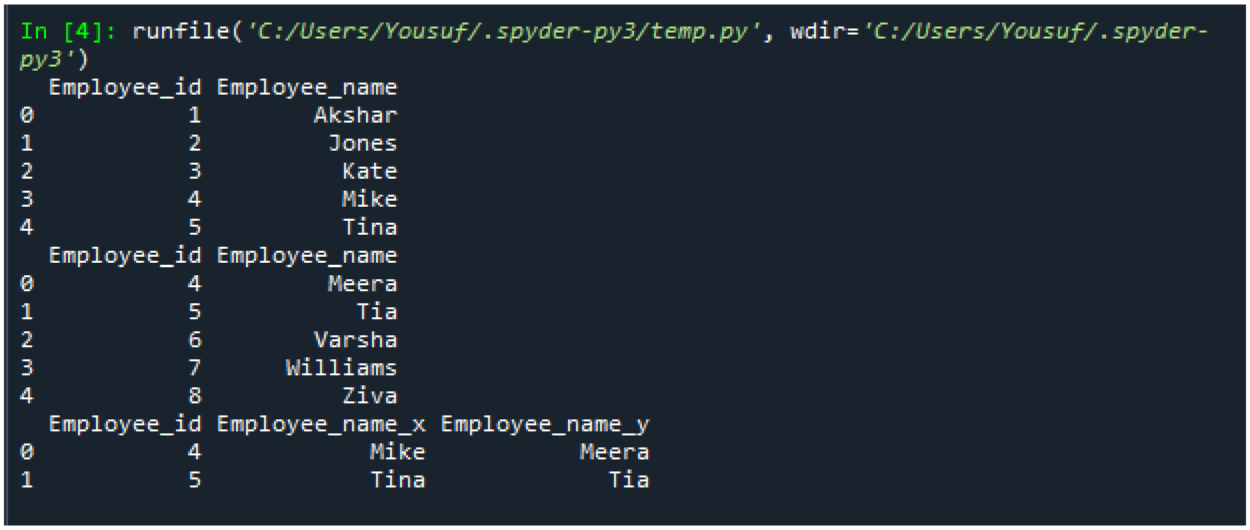
```
import pandas as pd

d = {
    'Employee_id': ['1', '2', '3', '4', '5'],
    'Employee_name': ['Akshar', 'Jones', 'Kate', 'Mike', 'Tina']
}
df1 = pd.DataFrame(d, columns=['Employee_id', 'Employee_name'])
print(df1)

import pandas as pd

data = {
    'Employee_id': ['4', '5', '6', '7', '8'],
    'Employee_name': ['Meera', 'Tia', 'Varsha', 'Williams', 'Ziva']
}
df2 = pd.DataFrame(data, columns=['Employee_id', 'Employee_name'])
print(df2)

print(pd.merge(df1, df2, on='Employee_id'))
```

Whole **code** screenshot:



```
import pandas as pd

d = {
    'Employee_id': ['1', '2', '3', '4', '5'],
    'Employee_name': ['Akshar', 'Jones', 'Kate', 'Mike', 'Tina']
}
df1 = pd.DataFrame(d, columns=['Employee_id', 'Employee_name'])
print(df1)

import pandas as pd

data = {
    'Employee_id': ['4', '5', '6', '7', '8'],
    'Employee_name': ['Meera', 'Tia', 'Varsha', 'Williams', 'Ziva']
}
df2 = pd.DataFrame(data, columns=['Employee_id', 'Employee_name'])
print(df2)

print(pd.merge(df1, df2, on='Employee_id'))
```

**Output**:



```
In [4]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
  Employee_id Employee_name
0           1        Akshar
1           2         Jones
2           3          Kate
3           4          Mike
4           5          Tina
  Employee_id Employee_name
0           4         Meera
1           5           Tia
2           6        Varsha
3           7      Williams
4           8          Ziva
  Employee_id Employee_name_x Employee_name_y
0           4            Mike           Meera
1           5            Tina             Tia
```

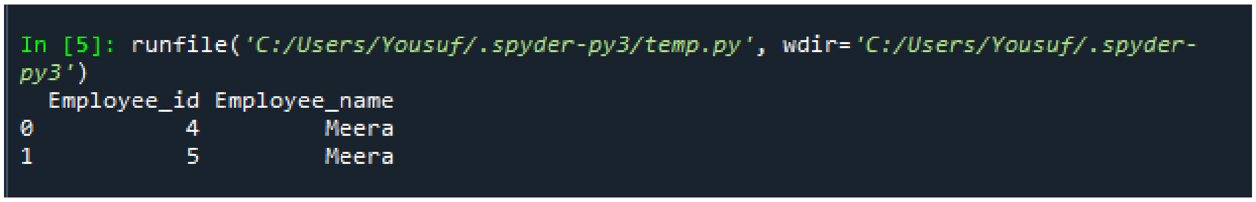
We can see that the merge() function returns the rows from both the DataFrames having the same column value that was used while merging.

## b. Grouping

Grouping is a process of collecting data into different categories. For example, in the below example, the “Employee_Name” field has the name “Meera” two times. So, let’s group it by “Employee_name” column.

```
import pandas as pd
import numpy as np

data = {
    'Employee_id': ['4', '5', '6', '7', '8'],
    'Employee_name': ['Meera', 'Meera', 'Varsha', 'Williams',
'Ziva']
}
df2 = pd.DataFrame(data)

group = df2.groupby('Employee_name')
print(group.get_group('Meera'))
```

Output:



```
In [5]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
  Employee_id Employee_name
0           4         Meera
1           5         Meera
```

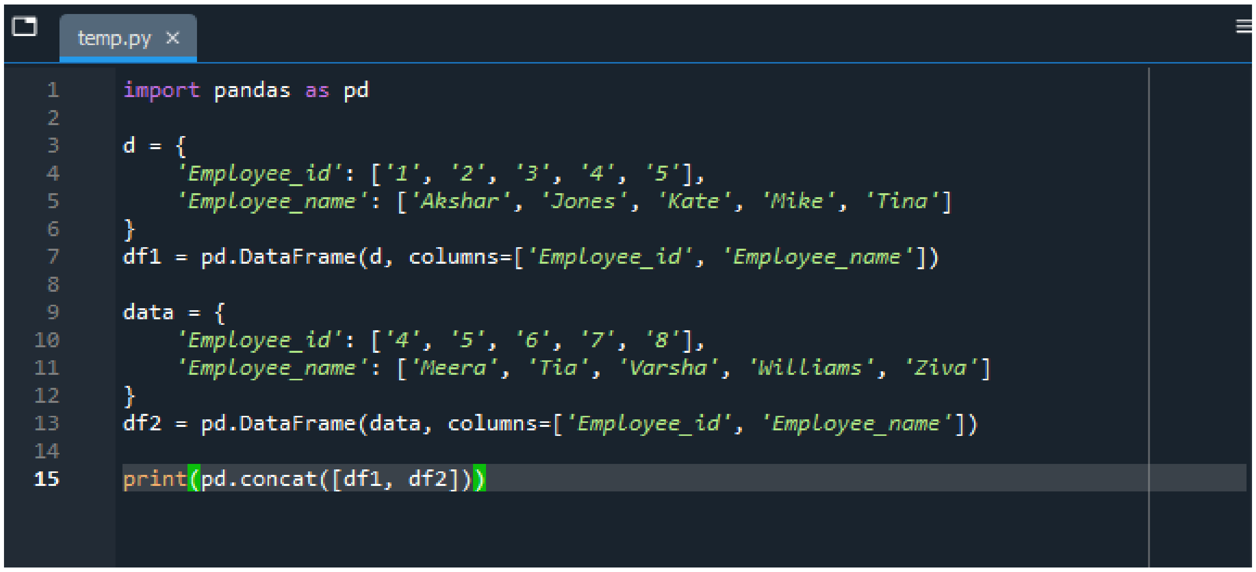
The ‘Employee_name’ field having value ‘Meera’ is grouped by the column “Employee_name”.

## c. Concatenating

Concatenating data involves adding one set of data to another. Pandas provides a function named concat() to concatenate DataFrames. For example, let’s concatenate the DataFrames df1 and df2 , using :

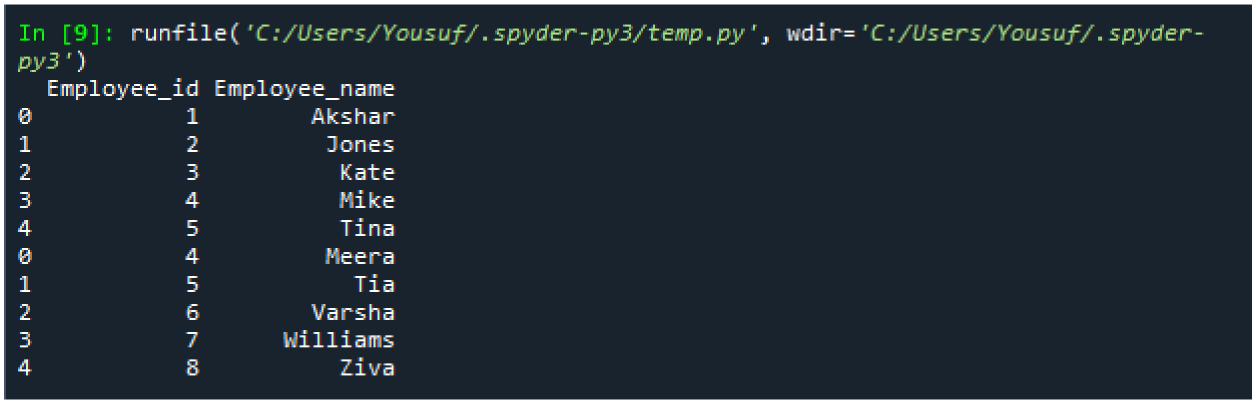
```
import pandas as pd

d = {
    'Employee_id': ['1', '2', '3', '4', '5'],
    'Employee_name': ['Akshar', 'Jones', 'Kate', 'Mike', 'Tina']
}
df1 = pd.DataFrame(d, columns=['Employee_id', 'Employee_name'])

data = {
    'Employee_id': ['4', '5', '6', '7', '8'],
    'Employee_name': ['Meera', 'Tia', 'Varsha', 'Williams', 'Ziva']
}
df2 = pd.DataFrame(data, columns=['Employee_id', 'Employee_name'])

print(pd.concat([df1, df2]))
```

**Screenshot** of the code in Spyder:



```
import pandas as pd

d = {
    'Employee_id': ['1', '2', '3', '4', '5'],
    'Employee_name': ['Akshar', 'Jones', 'Kate', 'Mike', 'Tina']
}
df1 = pd.DataFrame(d, columns=['Employee_id', 'Employee_name'])

data = {
    'Employee_id': ['4', '5', '6', '7', '8'],
    'Employee_name': ['Meera', 'Tia', 'Varsha', 'Williams', 'Ziva']
}
df2 = pd.DataFrame(data, columns=['Employee_id', 'Employee_name'])

print(pd.concat([df1, df2]))
```

**Output**:



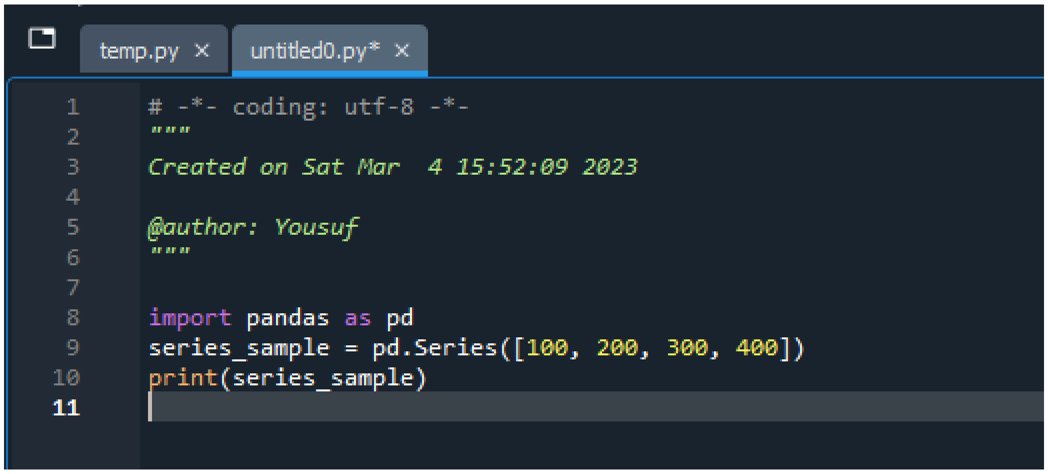
```
In [9]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/Yousuf/.spyder-
py3')
   Employee_id Employee_name
0            1        Akshar
1            2         Jones
2            3          Kate
3            4          Mike
4            5          Tina
0            4         Meera
1            5           Tia
2            6        Varsha
3            7      Williams
4            8          Ziva
```

## Create a DataFrame by passing Dict of Series

To create a Series, we can use the pd.Series() method and pass an array to it. Let's create a simple Series as follows:

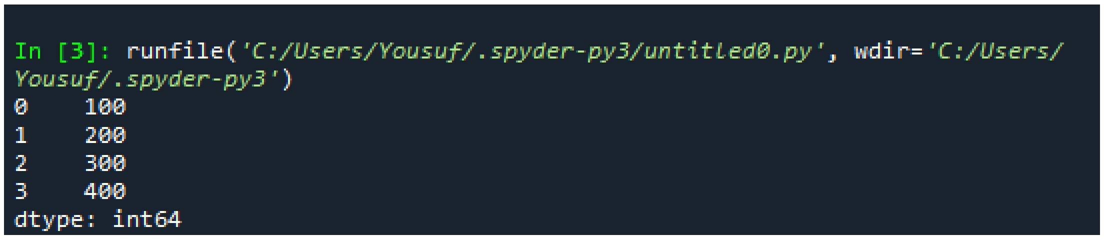
```
import pandas as pd
series_sample = pd.Series([100, 200, 300, 400])
print(series_sample)
```

**Code** SS in Spyder:



```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar  4 15:52:09 2023

@author: Yousuf
"""

import pandas as pd
series_sample = pd.Series([100, 200, 300, 400])
print(series_sample)

```

**Output**:

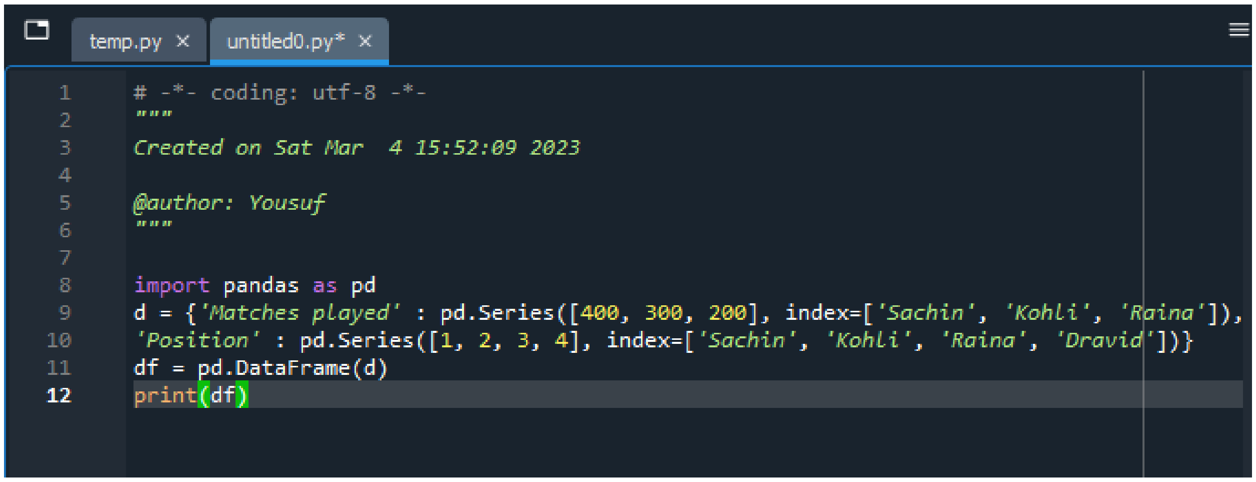


```
In [3]: runfile('C:/Users/Yousuf/.spyder-py3/untitled0.py', wdir='C:/Users/
Yousuf/.spyder-py3')
0    100
1    200
2    300
3    400
dtype: int64
```

We have created a Series. You can see that 2 columns are displayed. The first column contains the index values starting from 0. The second column contains the elements passed as series. It is possible to create a DataFrame by passing a dictionary of `Series`. Let's create a DataFrame that is formed by uniting and passing the indexes of the series.
For example:

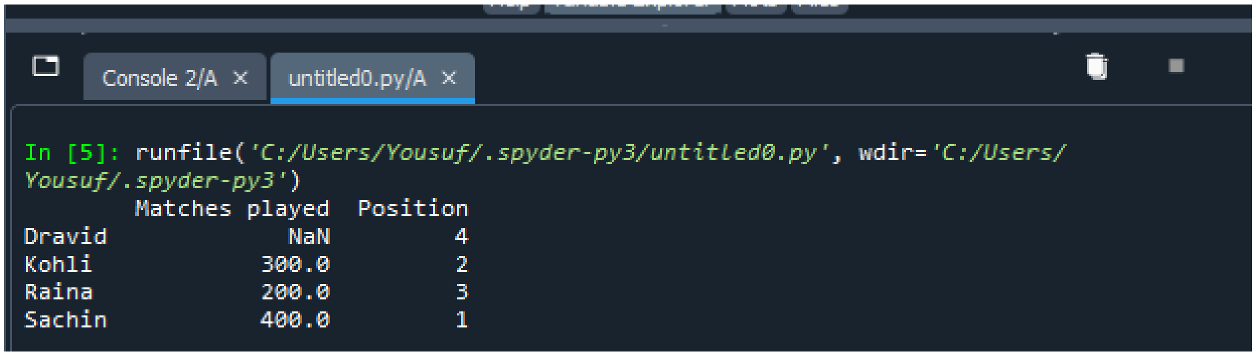
```
d = {'Matches played' : pd.Series([400, 300, 200], index=['Sachin',
'Kohli', 'Raina']),
'Position' : pd.Series([1, 2, 3, 4], index=['Sachin', 'Kohli',
'Raina', 'Dravid'])}
df = pd.DataFrame(d)
print(df)
```

**Screenshot** in Spyder:



```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar  4 15:52:09 2023

@author: Yousuf
"""

import pandas as pd
d = {'Matches played' : pd.Series([400, 300, 200], index=['Sachin', 'Kohli', 'Raina']),
'Position' : pd.Series([1, 2, 3, 4], index=['Sachin', 'Kohli', 'Raina', 'Dravid'])}
df = pd.DataFrame(d)
print(df)
```

**Output** in console:



```
In [5]: runfile('C:/Users/Yousuf/.spyder-py3/untitled0.py', wdir='C:/Users/
Yousuf/.spyder-py3')
        Matches played  Position
Dravid             NaN         4
Kohli            300.0         2
Raina            200.0         3
Sachin           400.0         1
```

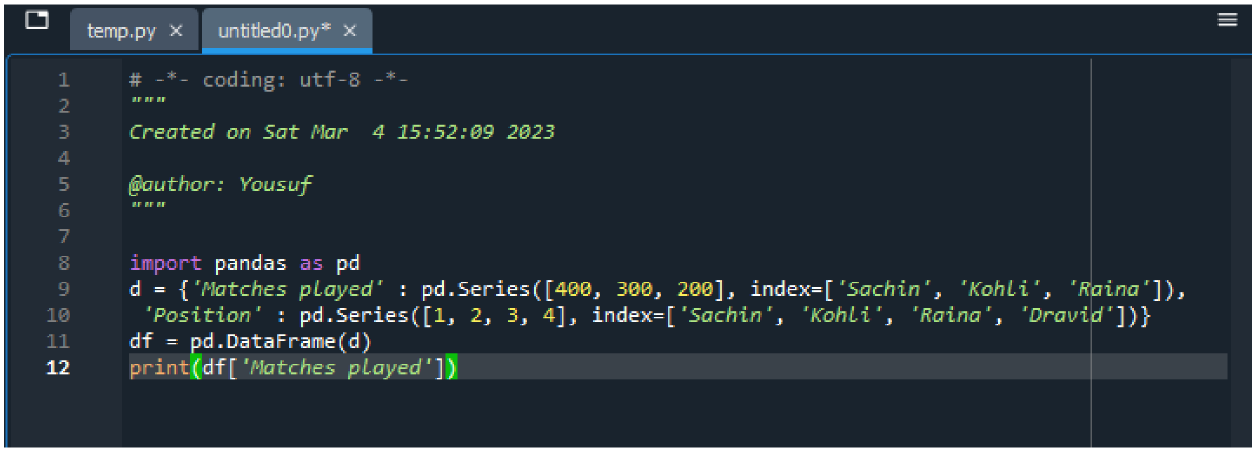
For series one, as we have not specified label 'd', NaN is returned.

## Column Selection, Addition, Deletion

It is possible to select a specific column from the DataFrame. For example, to display only the first column, we can rewrite the above **code** as:

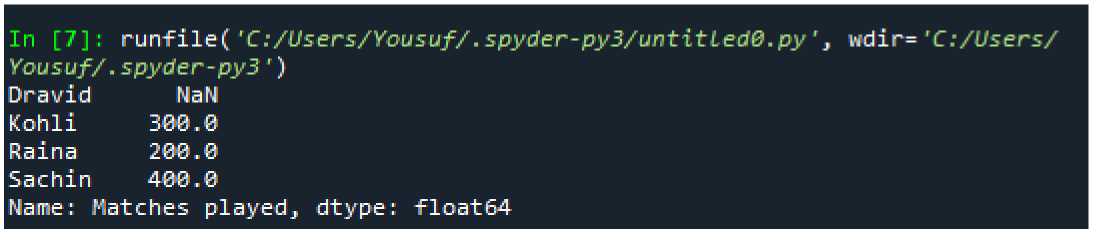
```
import pandas as pd
d = {'Matches played' : pd.Series([400, 300, 200], index=['Sachin',
'Kohli', 'Raina']),
 'Position' : pd.Series([1, 2, 3, 4], index=['Sachin', 'Kohli',
'Raina', 'Dravid'])}
df = pd.DataFrame(d)
print(df['Matches played'])
```

**Screenshot** in Spyder:



```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar  4 15:52:09 2023

@author: Yousuf
"""

import pandas as pd
d = {'Matches played' : pd.Series([400, 300, 200], index=['Sachin', 'Kohli', 'Raina']),
 'Position' : pd.Series([1, 2, 3, 4], index=['Sachin', 'Kohli', 'Raina', 'Dravid'])}
df = pd.DataFrame(d)
print(df['Matches played'])
```

**Output** in console:



```
In [7]: runfile('C:/Users/Yousuf/.spyder-py3/untitled0.py', wdir='C:/Users/
Yousuf/.spyder-py3')
Dravid      NaN
Kohli     300.0
Raina     200.0
Sachin    400.0
Name: Matches played, dtype: float64
```

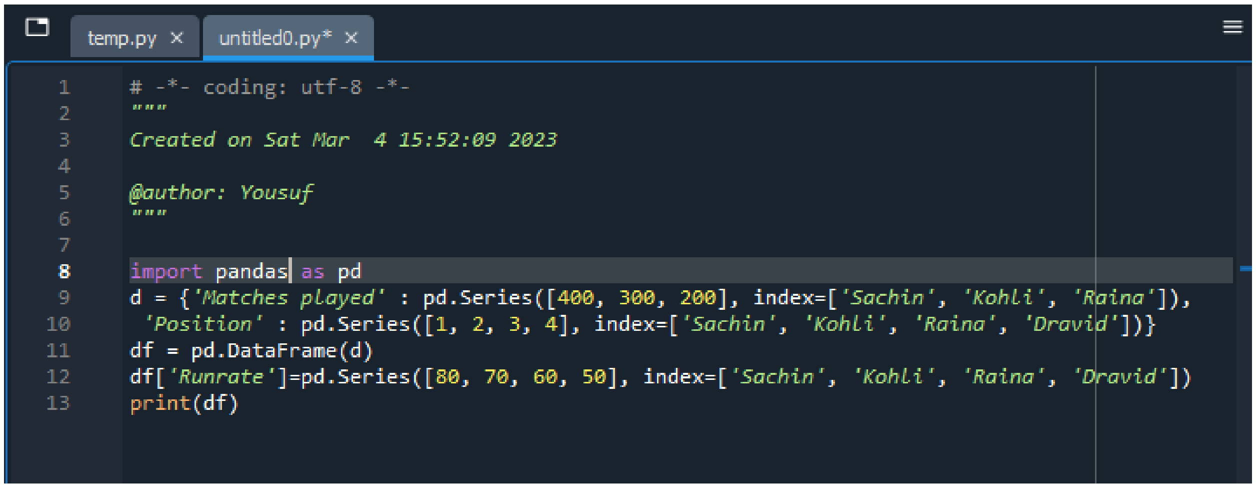
The above code prints only the “Matches played” column of the DataFrame.

It is also possible to add columns to an existing DataFrame. For example, the below code adds a new column named “Runrate” to the above DataFrame.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar  4 15:52:09 2023

@author: Yousuf
"""
```

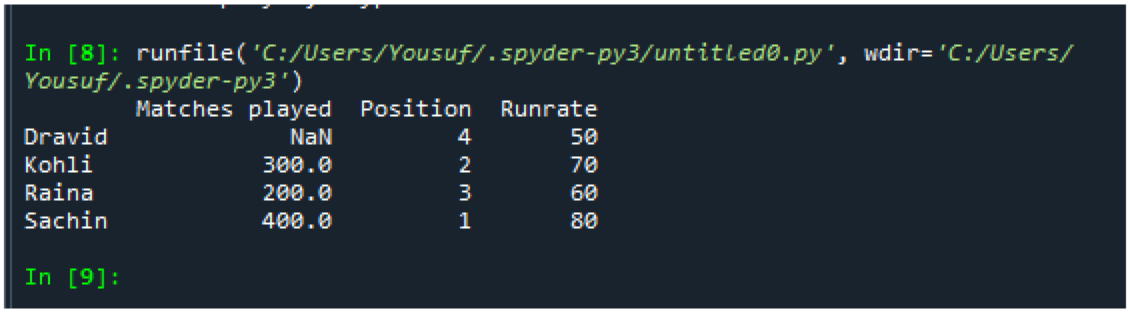
```
import pandas as pd
d = {'Matches played' : pd.Series([400, 300, 200], index=['Sachin',
'Kohli', 'Raina']),
 'Position' : pd.Series([1, 2, 3, 4], index=['Sachin', 'Kohli',
'Raina', 'Dravid'])}
df = pd.DataFrame(d)
df['Runrate']=pd.Series([80, 70, 60, 50], index=['Sachin', 'Kohli',
'Raina', 'Dravid'])
print(df)
```

**Screenshot**:



```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar  4 15:52:09 2023

@author: Yousuf
"""

import pandas as pd
d = {'Matches played' : pd.Series([400, 300, 200], index=['Sachin', 'Kohli', 'Raina']),
 'Position' : pd.Series([1, 2, 3, 4], index=['Sachin', 'Kohli', 'Raina', 'Dravid'])}
df = pd.DataFrame(d)
df['Runrate']=pd.Series([80, 70, 60, 50], index=['Sachin', 'Kohli', 'Raina', 'Dravid'])
print(df)
```

**Output** in console:



```
In [8]: runfile('C:/Users/Yousuf/.spyder-py3/untitled0.py', wdir='C:/Users/
Yousuf/.spyder-py3')
        Matches played  Position  Runrate
Dravid             NaN         4       50
Kohli            300.0         2       70
Raina            200.0         3       60
Sachin           400.0         1       80

In [9]:
```

We can delete columns using the `delete` and `pop` functions. For example to delete the 'Matches played' column in the above example, we can do it by either of the below 2 ways:

```
del df['Matches played']
```

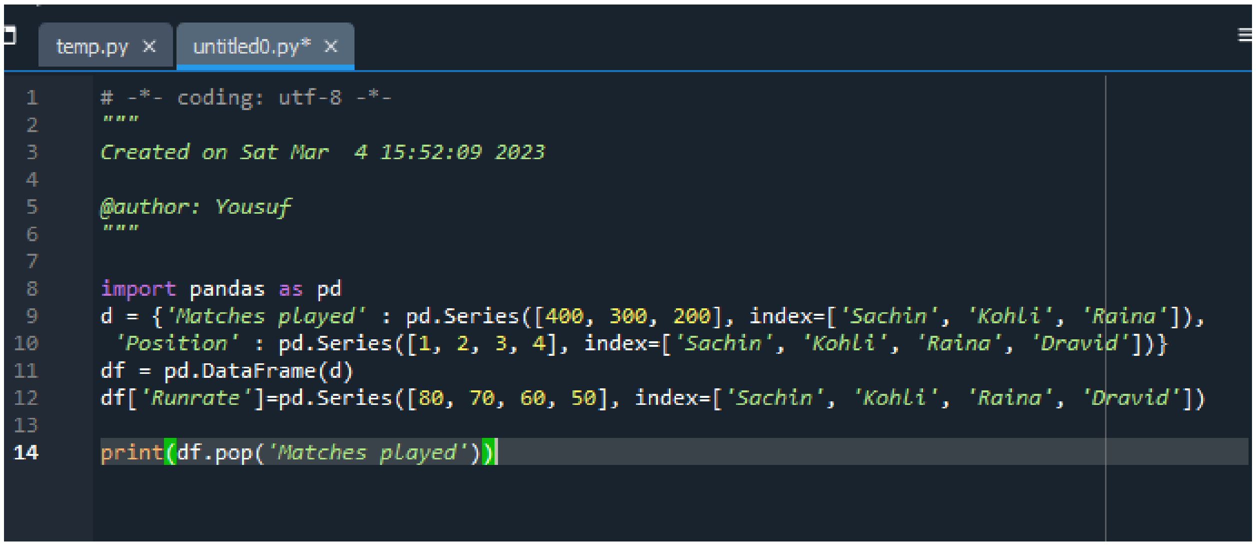
Or,

```
df.pop('Matches played')
```

Whole **code** is here:

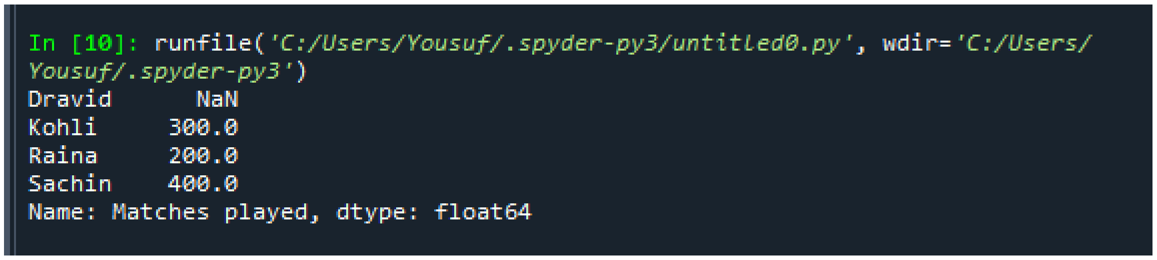
```
import pandas as pd
d = {'Matches played' : pd.Series([400, 300, 200], index=['Sachin',
'Kohli', 'Raina']),
 'Position' : pd.Series([1, 2, 3, 4], index=['Sachin', 'Kohli',
'Raina', 'Dravid'])}
df = pd.DataFrame(d)
df['Runrate']=pd.Series([80, 70, 60, 50], index=['Sachin', 'Kohli',
'Raina', 'Dravid'])

print(df.pop('Matches played'))
```

**Screenshot**:



```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar  4 15:52:09 2023

@author: Yousuf
"""

import pandas as pd
d = {'Matches played' : pd.Series([400, 300, 200], index=['Sachin', 'Kohli', 'Raina']),
 'Position' : pd.Series([1, 2, 3, 4], index=['Sachin', 'Kohli', 'Raina', 'Dravid'])}
df = pd.DataFrame(d)
df['Runrate']=pd.Series([80, 70, 60, 50], index=['Sachin', 'Kohli', 'Raina', 'Dravid'])

print(df.pop('Matches played'))
```

**Output** of this code:



```
In [10]: runfile('C:/Users/Yousuf/.spyder-py3/untitled0.py', wdir='C:/Users/
Yousuf/.spyder-py3')
Dravid      NaN
Kohli     300.0
Raina     200.0
Sachin    400.0
Name: Matches played, dtype: float64
```

## More Statistical Functionalities using Pandas

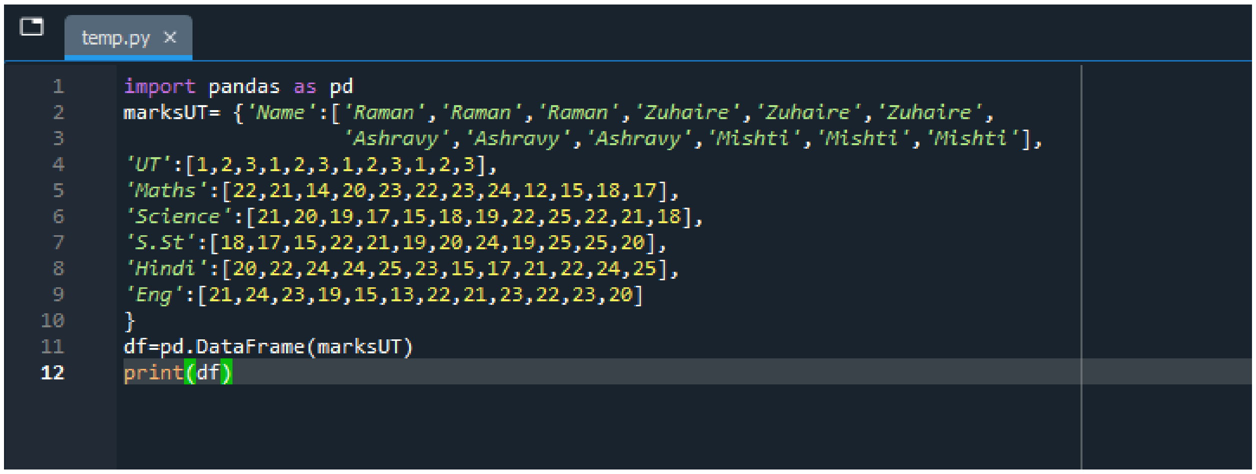
The statistical calculations like mean, median, mode, standard deviation etc can be solved using pandas.

For example we will store some data and then from this, we will manipulate, calculate and retrieve data results.

**Code**:

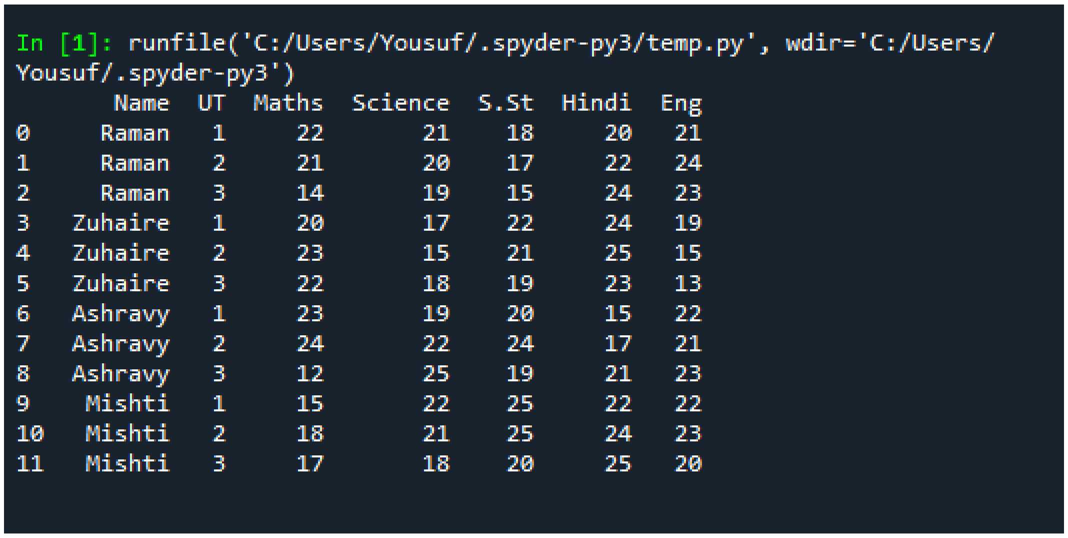
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)
print(df)
```

**Code** in spyder:



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                 'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)
print(df)
```

**Output**:



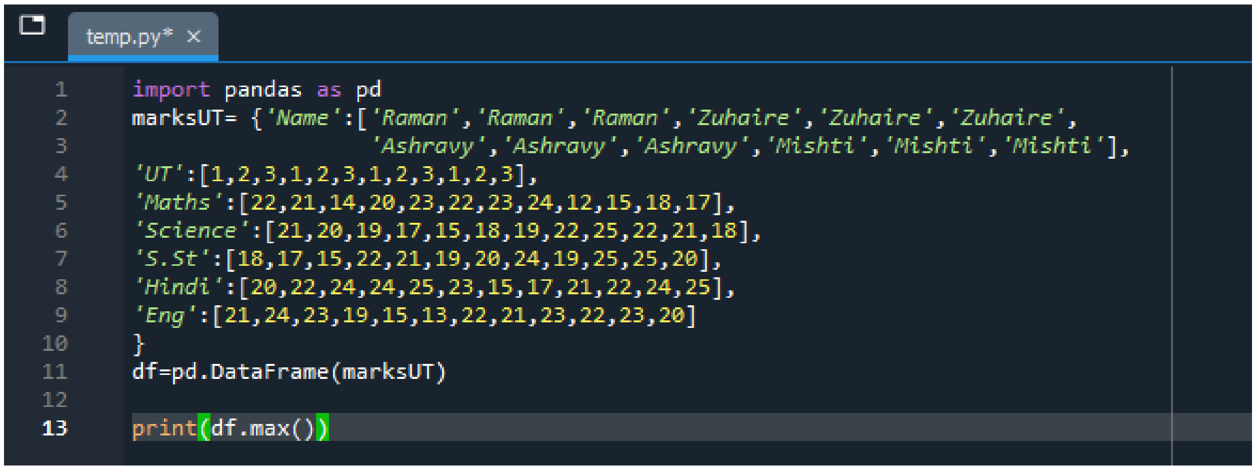
```
In [1]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
       Name  UT  Maths  Science  S.St  Hindi  Eng
0     Raman   1     22       21    18     20   21
1     Raman   2     21       20    17     22   24
2     Raman   3     14       19    15     24   23
3   Zuhaire   1     20       17    22     24   19
4   Zuhaire   2     23       15    21     25   15
5   Zuhaire   3     22       18    19     23   13
6   Ashravy   1     23       19    20     15   22
7   Ashravy   2     24       22    24     17   21
8   Ashravy   3     12       25    19     21   23
9    Mishti   1     15       22    25     22   22
10   Mishti   2     18       21    25     24   23
11   Mishti   3     17       18    20     25   20
```

## Calculating Maximum Values

Add the code portion :

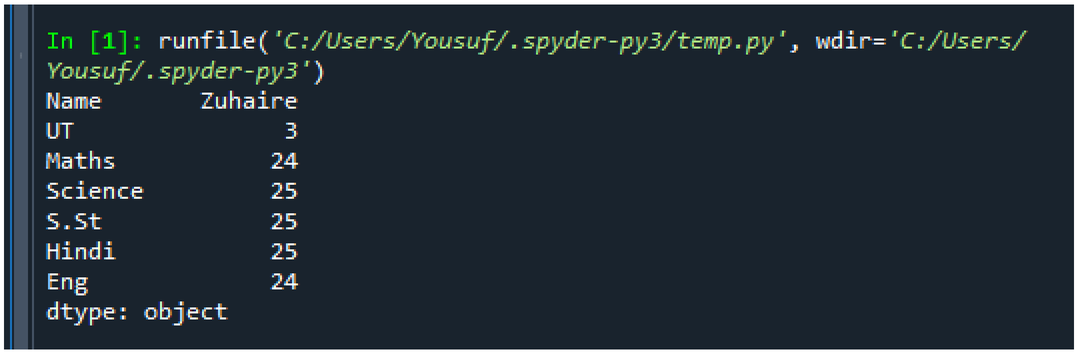
```
print(df.max())
```

**Input Code:**



```python
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

print(df.max())
```

**Output:**

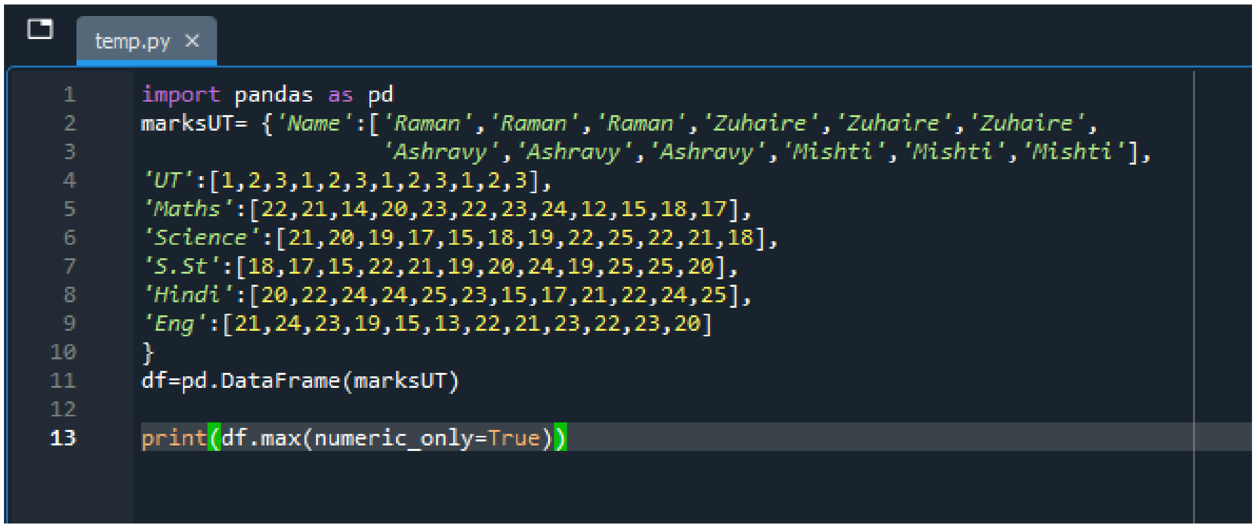


```
In [1]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
Name       Zuhaire
UT               3
Maths           24
Science         25
S.St            25
Hindi           25
Eng             24
dtype: object
```

If we want to output maximum value for the columns having only numeric values, then we can set the parameter numeric_only=True in the max() method, as shown below:

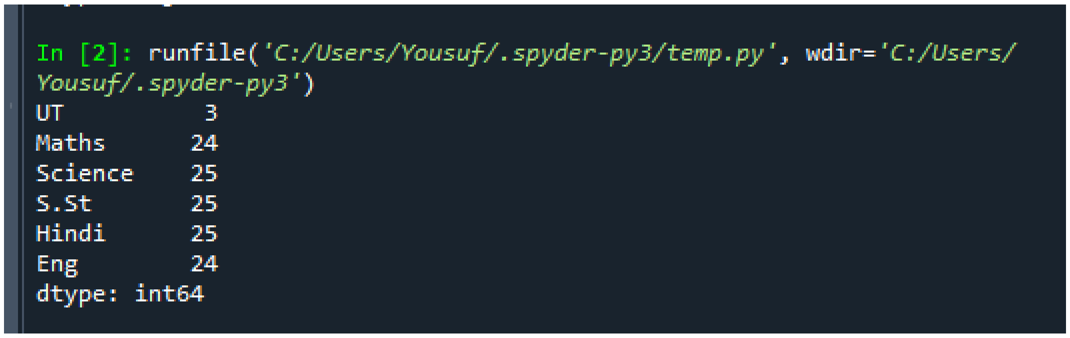
```
print(df.max(numeric_only=True))
```

**Code Screenshot:**



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

print(df.max(numeric_only=True))
```

**Output:**

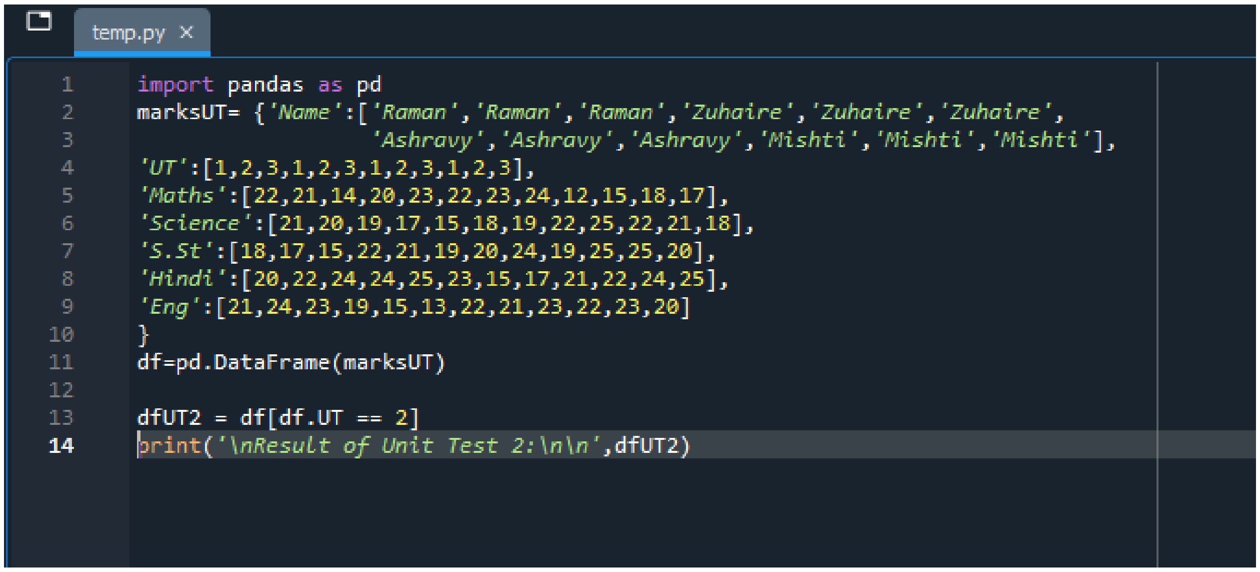


```
In [2]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
UT          3
Maths      24
Science    25
S.St       25
Hindi      25
Eng        24
dtype: int64
```

This code is the necessary code portion to show the highest marks obtained in each subject in Unit Test 2:

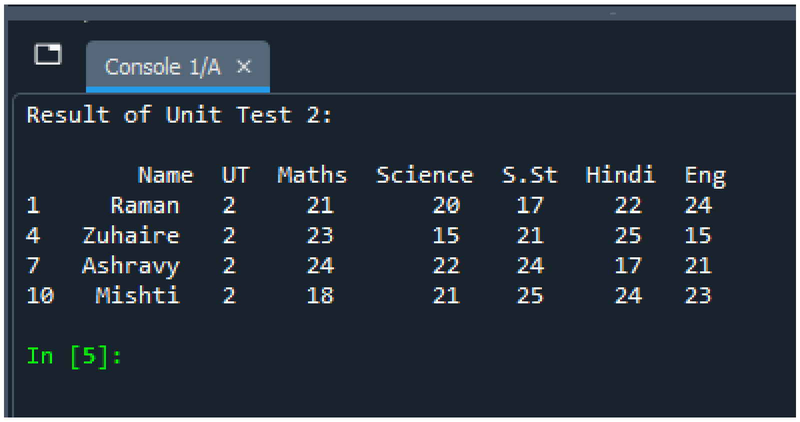
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]
print('\nResult of Unit Test 2:\n\n',dfUT2)
```

**Screenshot of the code in Spyder:**



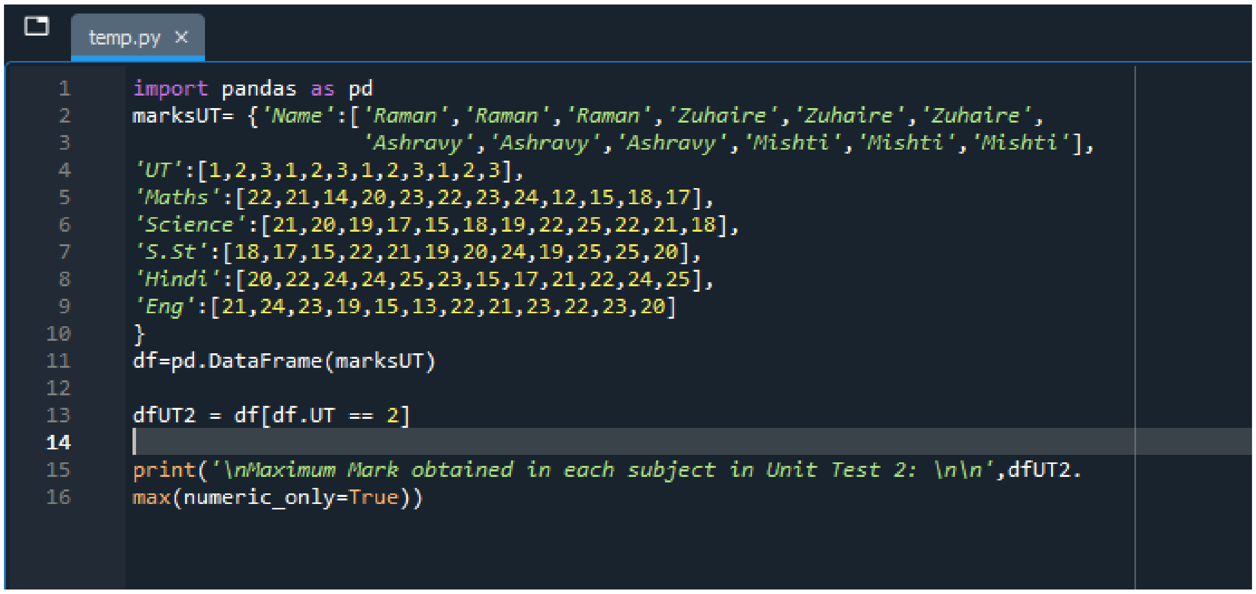
```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]
print('\nResult of Unit Test 2:\n\n',dfUT2)
```

**Output:**



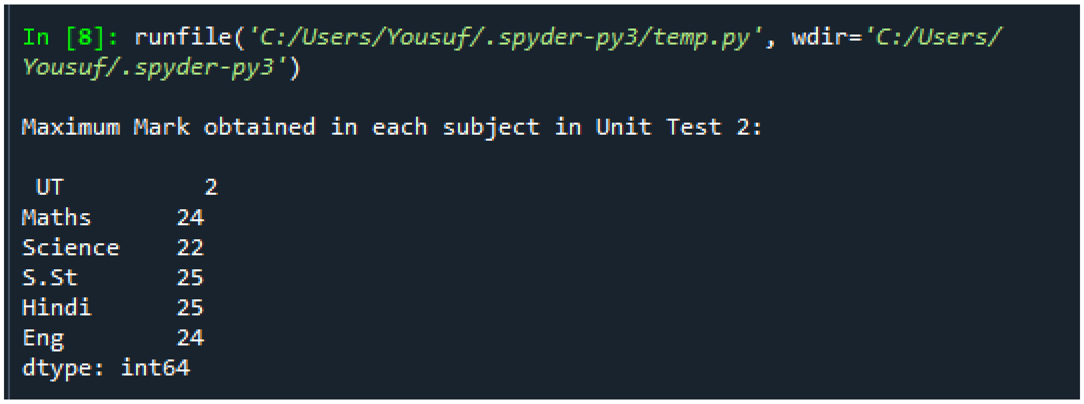
```
Result of Unit Test 2:

       Name  UT  Maths  Science  S.St  Hindi  Eng
1     Raman   2     21       20    17     22   24
4   Zuhaire   2     23       15    21     25   15
7   Ashravy   2     24       22    24     17   21
10   Mishti   2     18       21    25     24   23

In [5]:
```

Furthermore specifically taken output by the following **code**:



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

print('\nMaximum Mark obtained in each subject in Unit Test 2: \n\n',dfUT2.
max(numeric_only=True))
```

**Output:**



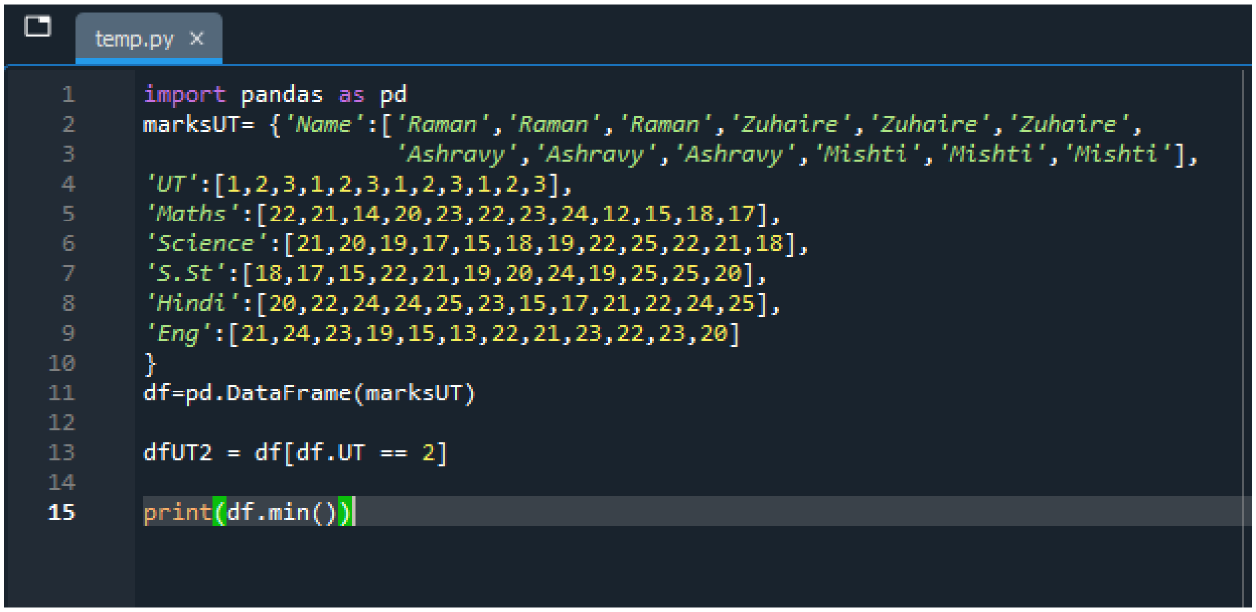
```
In [8]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')

Maximum Mark obtained in each subject in Unit Test 2:

 UT          2
Maths      24
Science    22
S.St       25
Hindi      25
Eng        24
dtype: int64
```

## Calculating Minimum Values

DataFrame.min() is used to display the minimum values from the DataFrame, regardless of the data types. That is, it shows the minimum value of each column or row.
The following line of code output the minimum value of each column of the DataFrame:

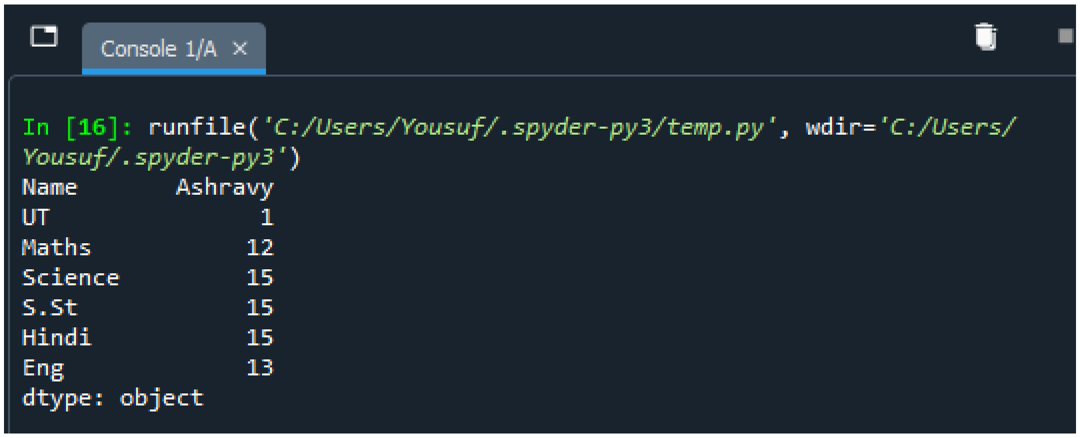
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

print(df.min())
```

The **screenshot** of this code in **Spyder**(Anaconda):


temp.py ×

```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

print(df.min())
```

**Output:**

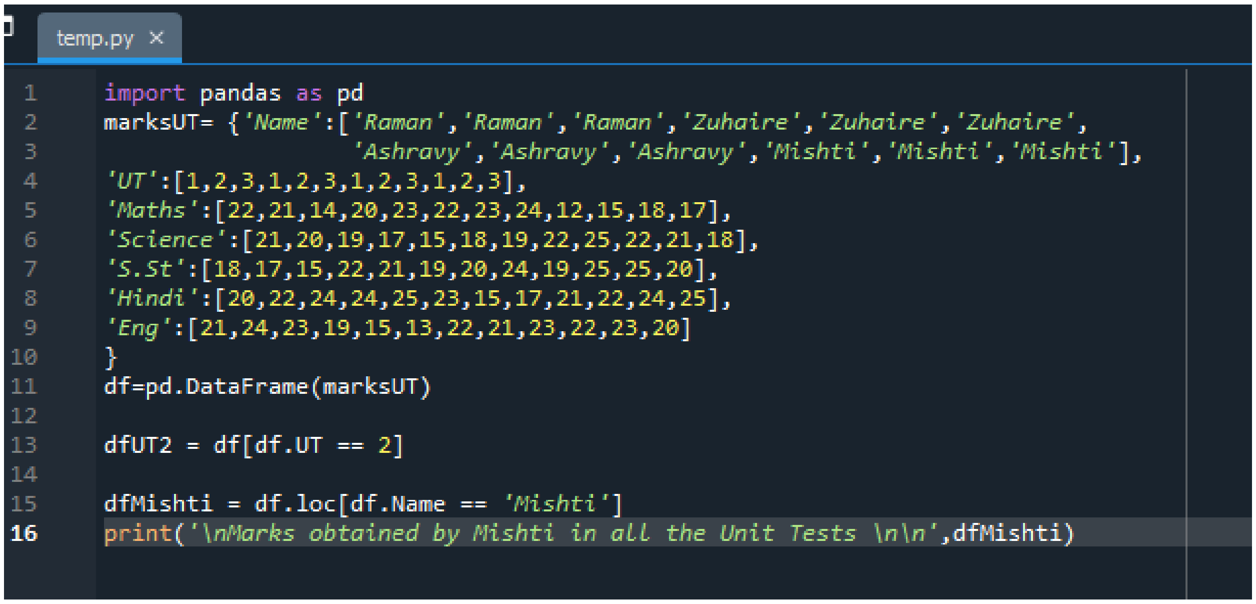


```
In [16]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
Name        Ashravy
UT                1
Maths            12
Science          15
S.St             15
Hindi            15
Eng              13
dtype: object
```

If we want to display the minimum marks obtained by a particular student ‘Mishti’ in all the unit tests for each subject then the code portion is:

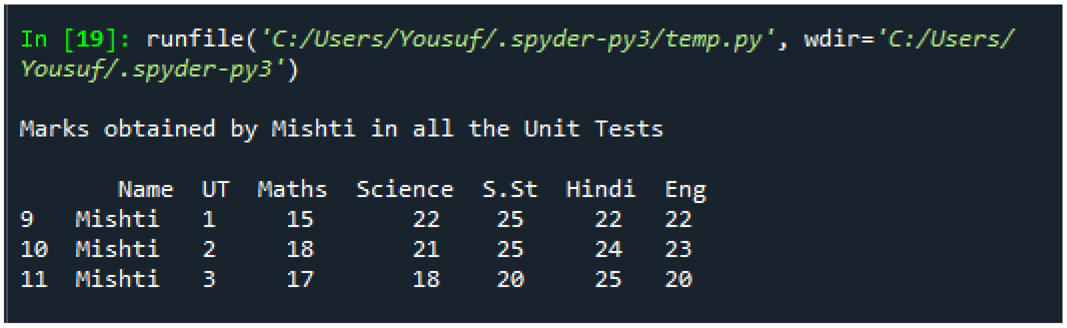
```
dfMishti = df.loc[df.Name == 'Mishti']
print('\nMarks obtained by Mishti in all
the Unit Tests \n\n',dfMishti)
```

The **screenshot** of this **code**:



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfMishti = df.loc[df.Name == 'Mishti']
print('\nMarks obtained by Mishti in all the Unit Tests \n\n',dfMishti)
```

**Output:**

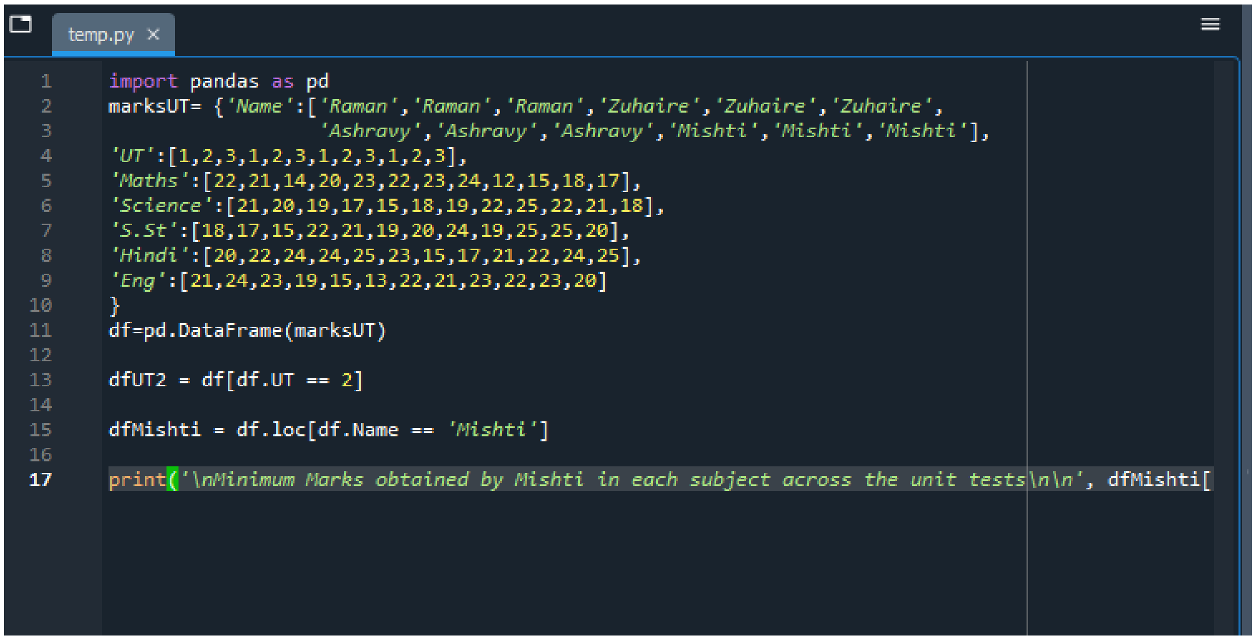


```
In [19]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')

Marks obtained by Mishti in all the Unit Tests

      Name  UT  Maths  Science  S.St  Hindi  Eng
9   Mishti   1     15       22    25     22   22
10  Mishti   2     18       21    25     24   23
11  Mishti   3     17       18    20     25   20
```

For the minimum marks in each subjects the code is:

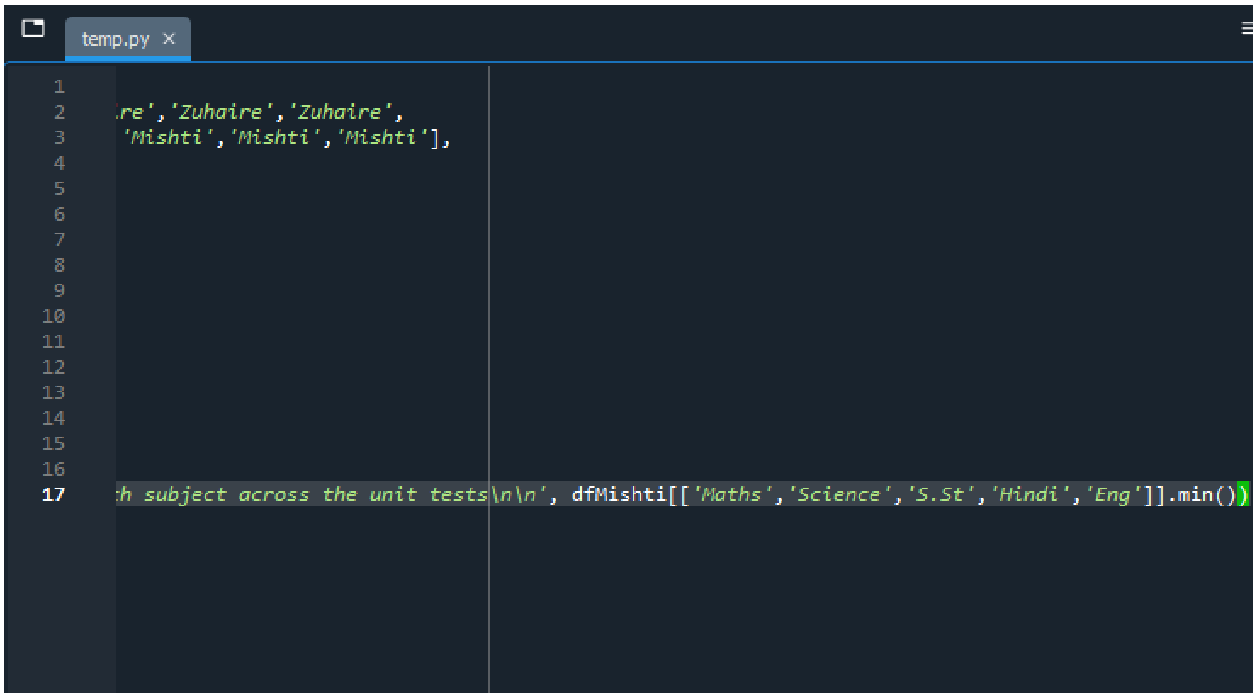
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfMishti = df.loc[df.Name == 'Mishti']

print('\nMinimum Marks obtained by Mishti in each subject across
the unit tests\n\n',
dfMishti[['Maths','Science','S.St','Hindi','Eng']].min())
```

Code **screenshot** in spyder:



temp.py

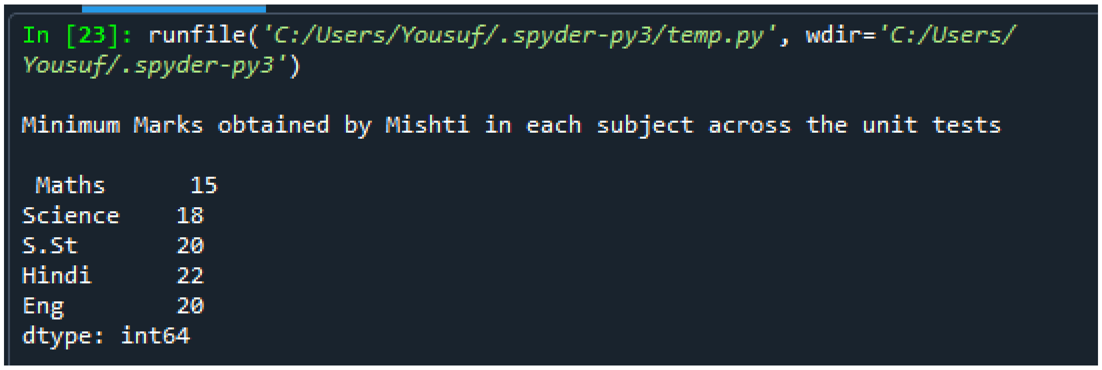
```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfMishti = df.loc[df.Name == 'Mishti']

print('\nMinimum Marks obtained by Mishti in each subject across the unit tests\n\n', dfMishti[
```



temp.py

```
ire','Zuhaire','Zuhaire',
 'Mishti','Mishti','Mishti'],














h subject across the unit tests\n\n', dfMishti[['Maths','Science','S.St','Hindi','Eng']].min())
```

**Output:**



```
In [23]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')

Minimum Marks obtained by Mishti in each subject across the unit tests

 Maths      15
Science    18
S.St       20
Hindi      22
Eng        20
dtype: int64
```

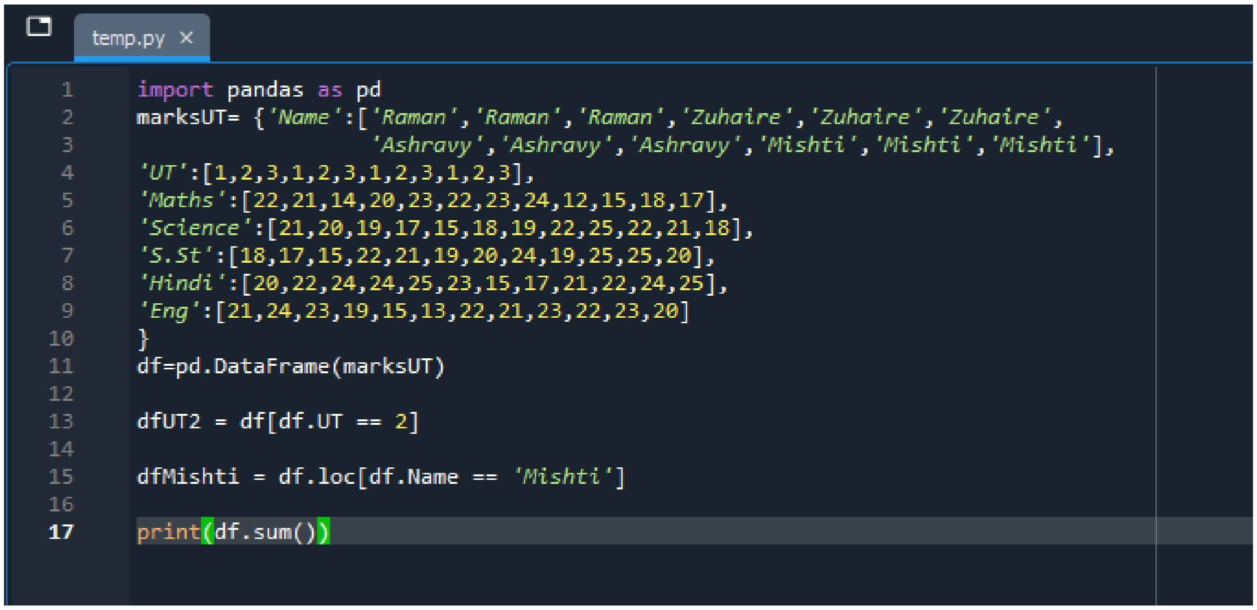
Note: Since we did not want to output the min value of column UT, we mentioned all the other column names for which minimum is to be calculated.

## Calculating Sum of Values

To find summation of values we use the sum() function or method. For example:

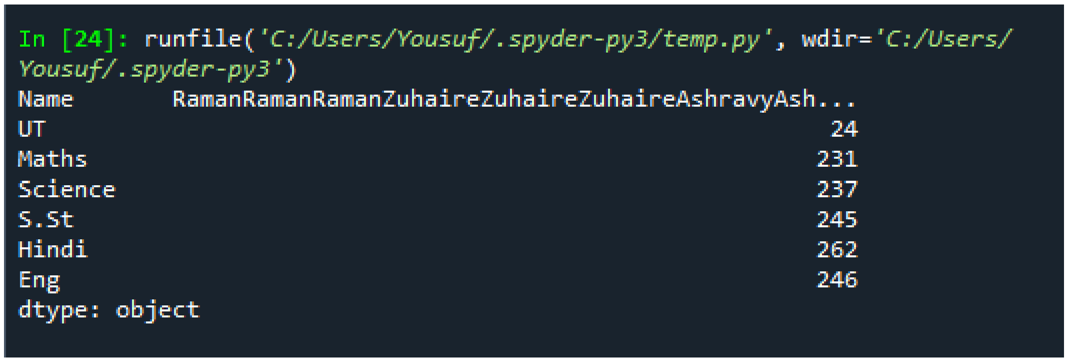
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfMishti = df.loc[df.Name == 'Mishti']

print(df.sum())
```

**Screenshot of this code is:**



```python
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfMishti = df.loc[df.Name == 'Mishti']

print(df.sum())
```

**Output:**

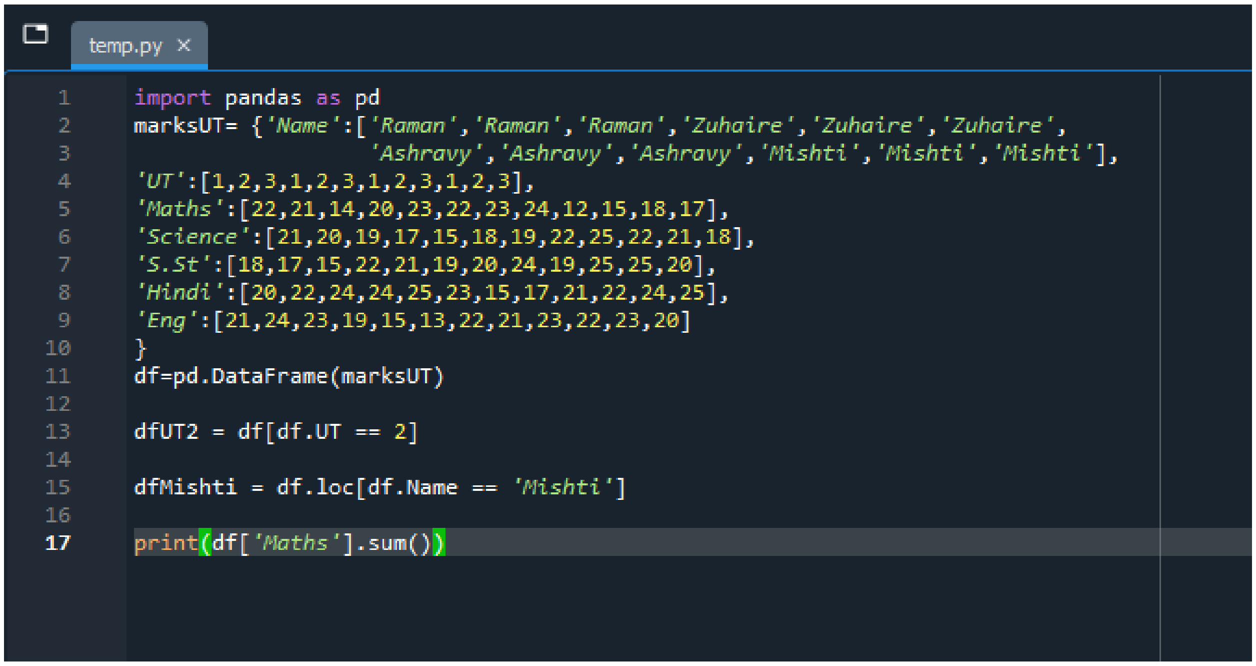


```
In [24]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
Name       RamanRamanRamanZuhaireZuhaireZuhaireAshravyAsh...
UT                                                       24
Maths                                                   231
Science                                                 237
S.St                                                    245
Hindi                                                   262
Eng                                                     246
dtype: object
```

We may not be interested in sum text values. So, to print the sum of a particular column, we need to specify the column name in the call to function sum.
The following statement prints the total marks of subject mathematics:

```
print(df['Maths'].sum())
```

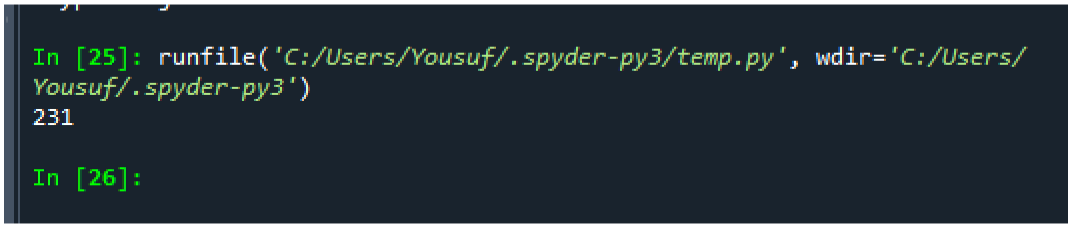
**Code Screenshot:**



temp.py

```python
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfMishti = df.loc[df.Name == 'Mishti']

print(df['Maths'].sum())
```

**Output:**

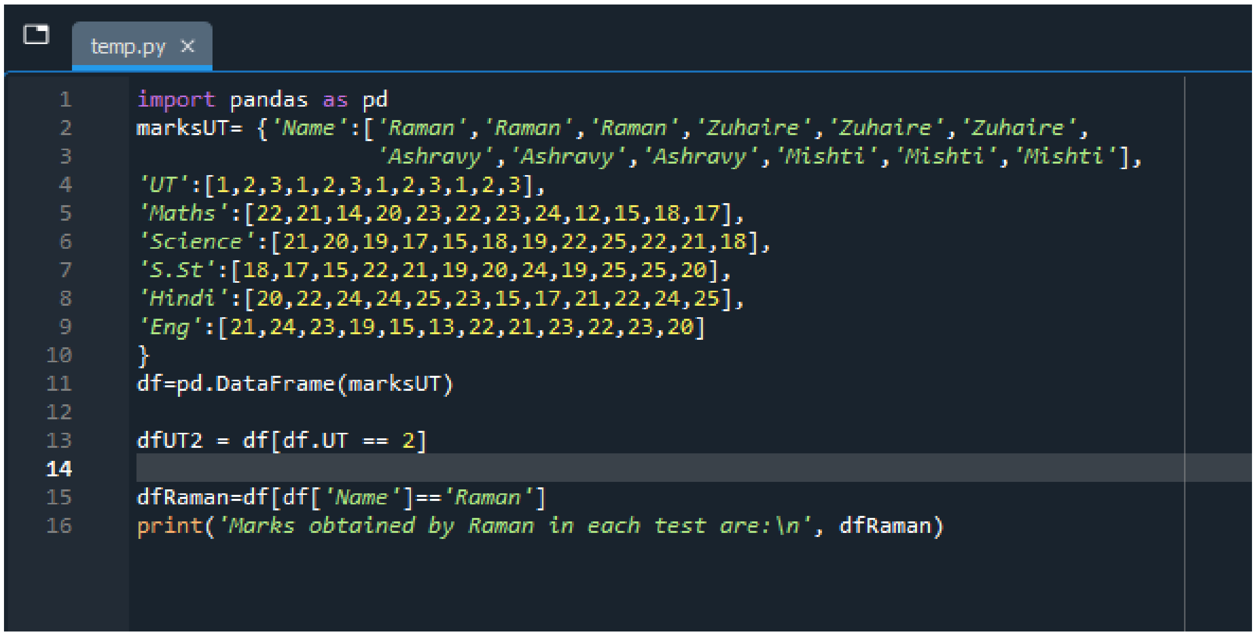


```
In [25]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
231

In [26]:
```

Now the python statement to print the total marks secured by raman in each subject:

```
dfRaman=df[df['Name']=='Raman']
print("Marks obtained by Raman in each test
are:\n", dfRaman)
```

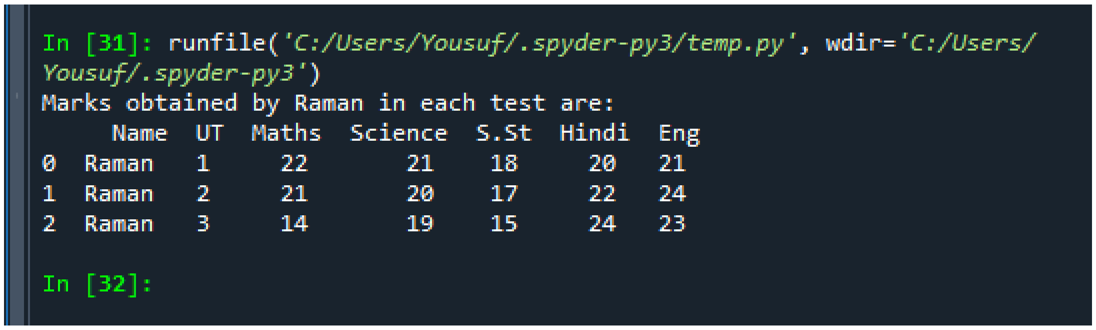
**Screenshot of the code input:**



temp.py

```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfRaman=df[df['Name']=='Raman']
print('Marks obtained by Raman in each test are:\n', dfRaman)
```

**Output:**



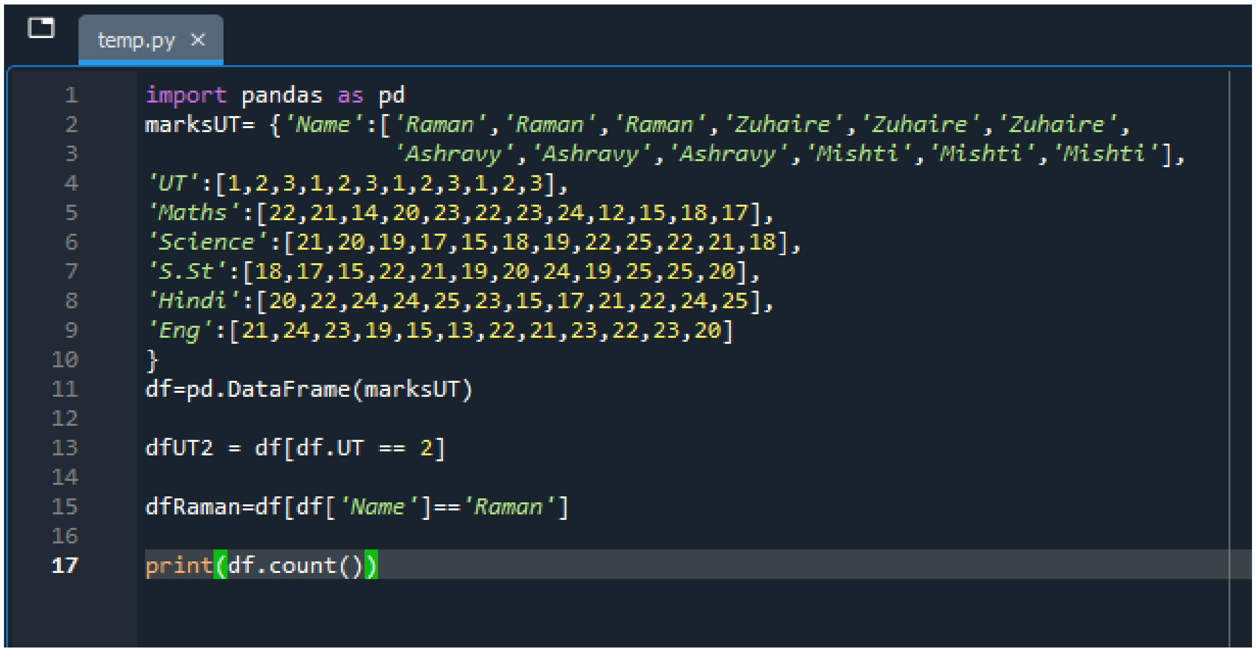
```
In [31]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
Marks obtained by Raman in each test are:
     Name  UT  Maths  Science  S.St  Hindi  Eng
0  Raman   1     22       21    18     20   21
1  Raman   2     21       20    17     22   24
2  Raman   3     14       19    15     24   23

In [32]:
```

## Calculating Number of Values

DataFrame.count() will display the total number of values for each column or row of a DataFrame. To count the rows we need to use the argument axis=1.
Code portion:

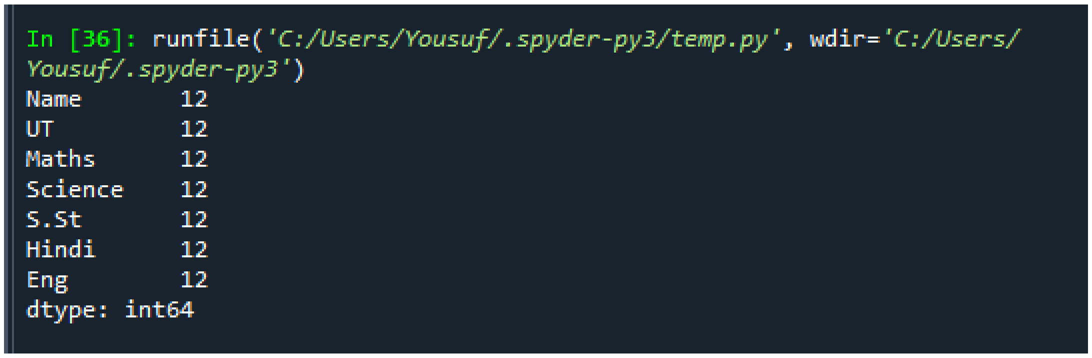
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfRaman=df[df['Name']=='Raman']

print(df.count())
```

**Screenshot in Spyder:**



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
               'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfRaman=df[df['Name']=='Raman']

print(df.count())
```

**Output:**



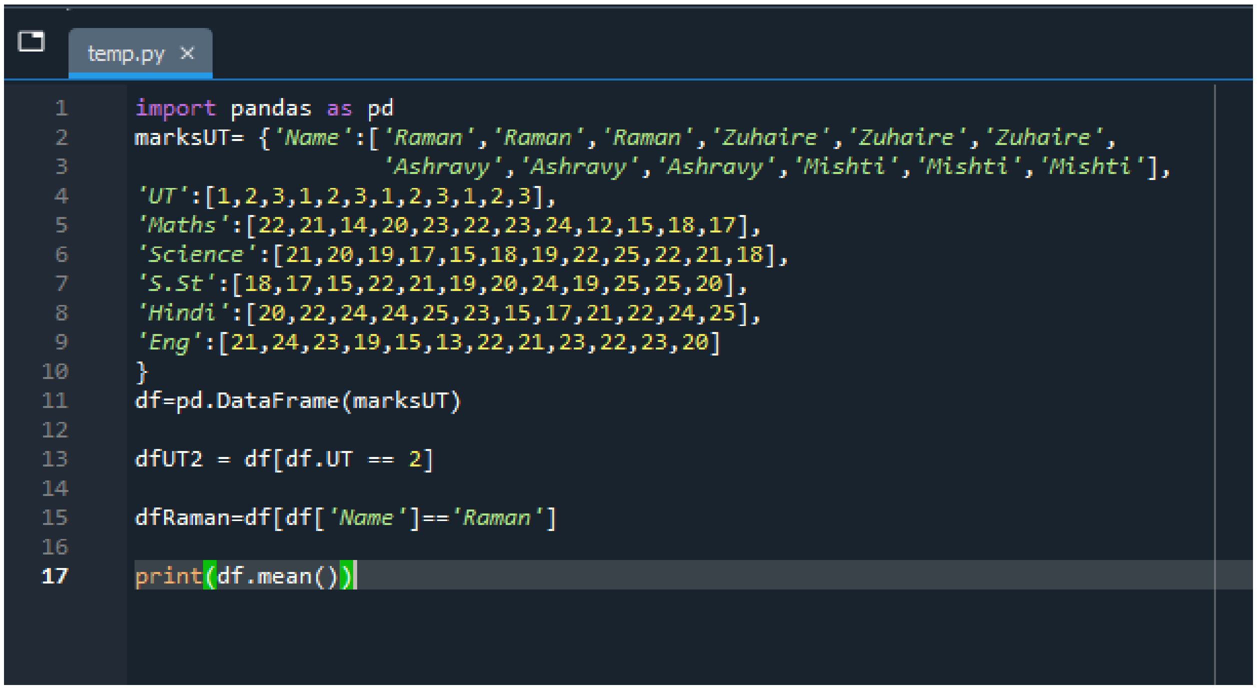
```
In [36]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
Name       12
UT         12
Maths      12
Science    12
S.St       12
Hindi      12
Eng        12
dtype: int64
```

## Calculating Mean

DataFrame.mean() can display the mean (average) of the values of each column of a DataFrame, obviously only for the numeric values.

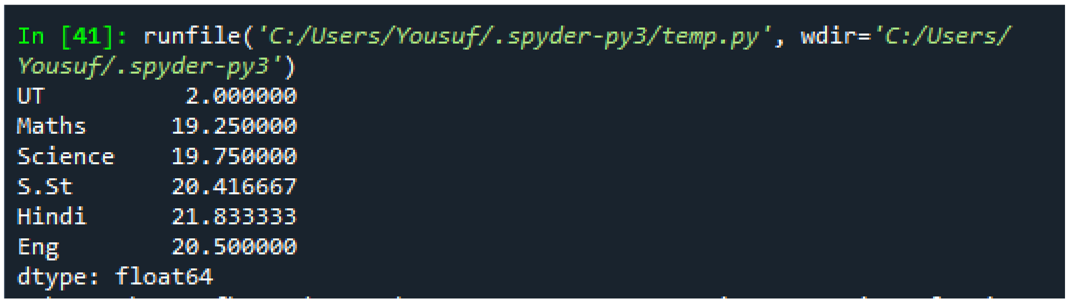
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfRaman=df[df['Name']=='Raman']

print(df.mean())
```

**Screenshot of the code snippet:**



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                   'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfRaman=df[df['Name']=='Raman']

print(df.mean())
```

**Output:**

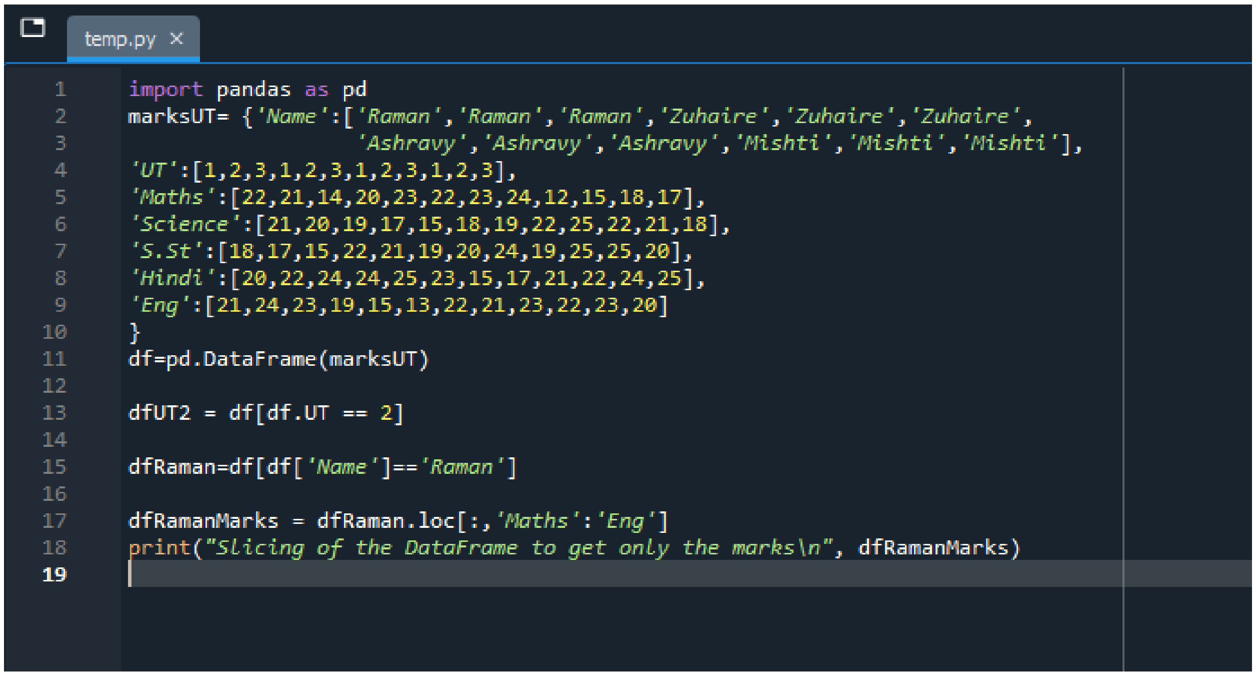


```
In [41]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
UT           2.000000
Maths       19.250000
Science     19.750000
S.St        20.416667
Hindi       21.833333
Eng         20.500000
dtype: float64
```

The statements to get an average of marks obtained by Raman in all the Unit Tests are given below:

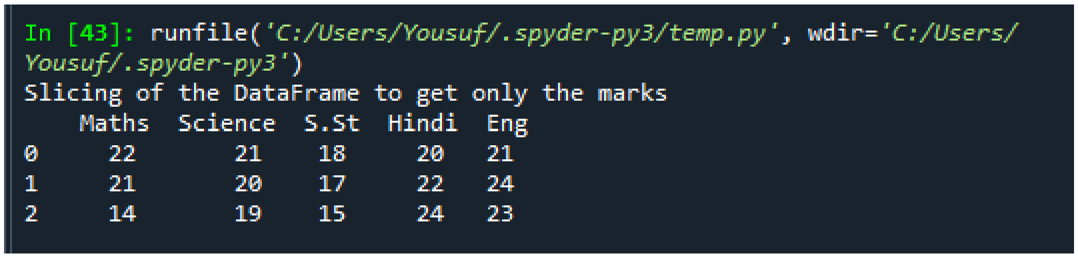
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfRaman=df[df['Name']=='Raman']

dfRamanMarks = dfRaman.loc[:,'Maths':'Eng']
print("Slicing of the DataFrame to get only the marks\n",
dfRamanMarks)
```

**Screenshot for the code is:**



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfRaman=df[df['Name']=='Raman']

dfRamanMarks = dfRaman.loc[:,'Maths':'Eng']
print("Slicing of the DataFrame to get only the marks\n", dfRamanMarks)

```

**Output:**



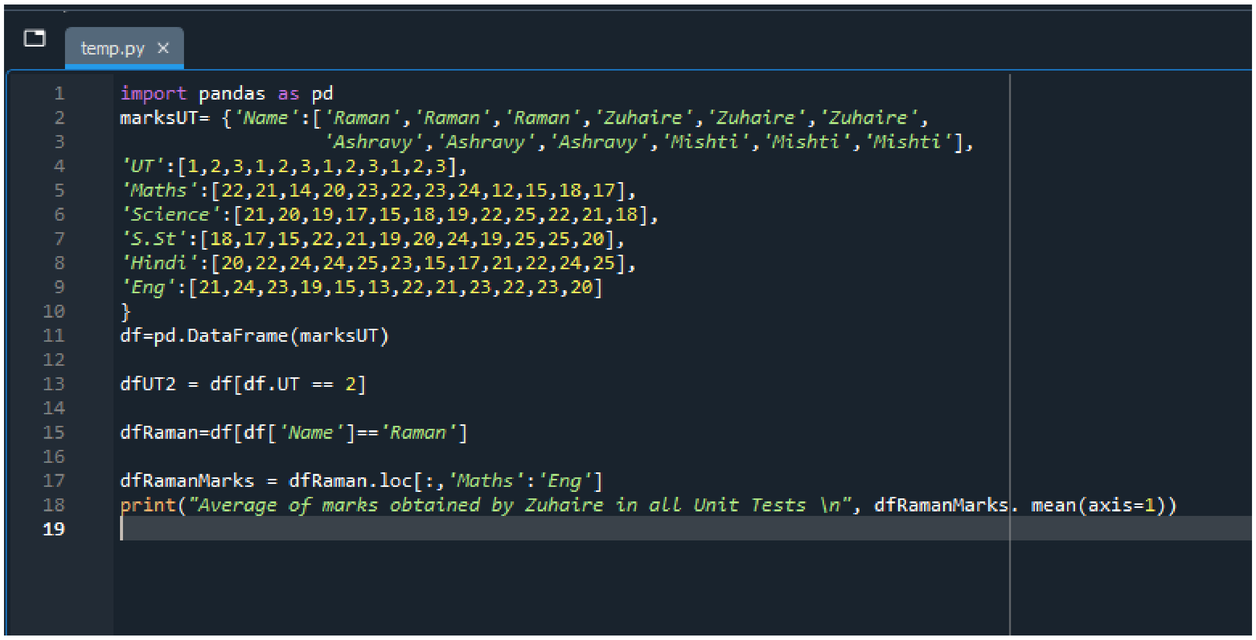
```
In [43]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/Users/
Yousuf/.spyder-py3')
Slicing of the DataFrame to get only the marks
    Maths  Science  S.St  Hindi  Eng
0     22       21    18     20   21
1     21       20    17     22   24
2     14       19    15     24   23
```

Again, average of the marks are retrieved by the **code** below:

```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]
dfRaman=df[df['Name']=='Raman']
```

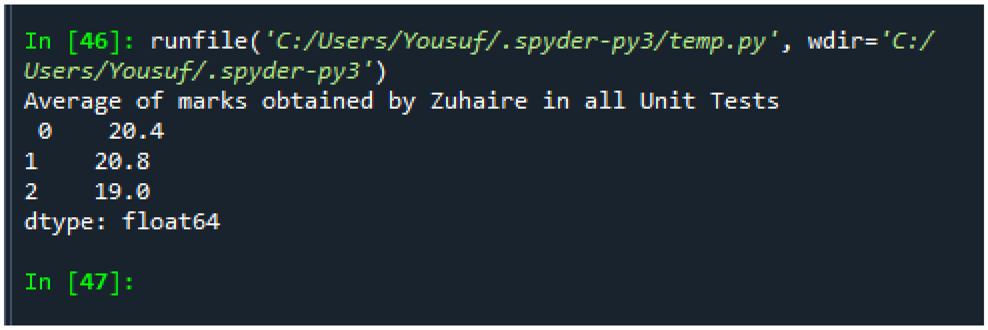
```
dfRamanMarks = dfRaman.loc[:,'Maths':'Eng']
print("Average of marks obtained by Zuhaire in all Unit Tests \n",
dfRamanMarks. mean(axis=1))
```

**Screenshot in Spyder IDE:**



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

dfRaman=df[df['Name']=='Raman']

dfRamanMarks = dfRaman.loc[:,'Maths':'Eng']
print("Average of marks obtained by Zuhaire in all Unit Tests \n", dfRamanMarks. mean(axis=1))

```

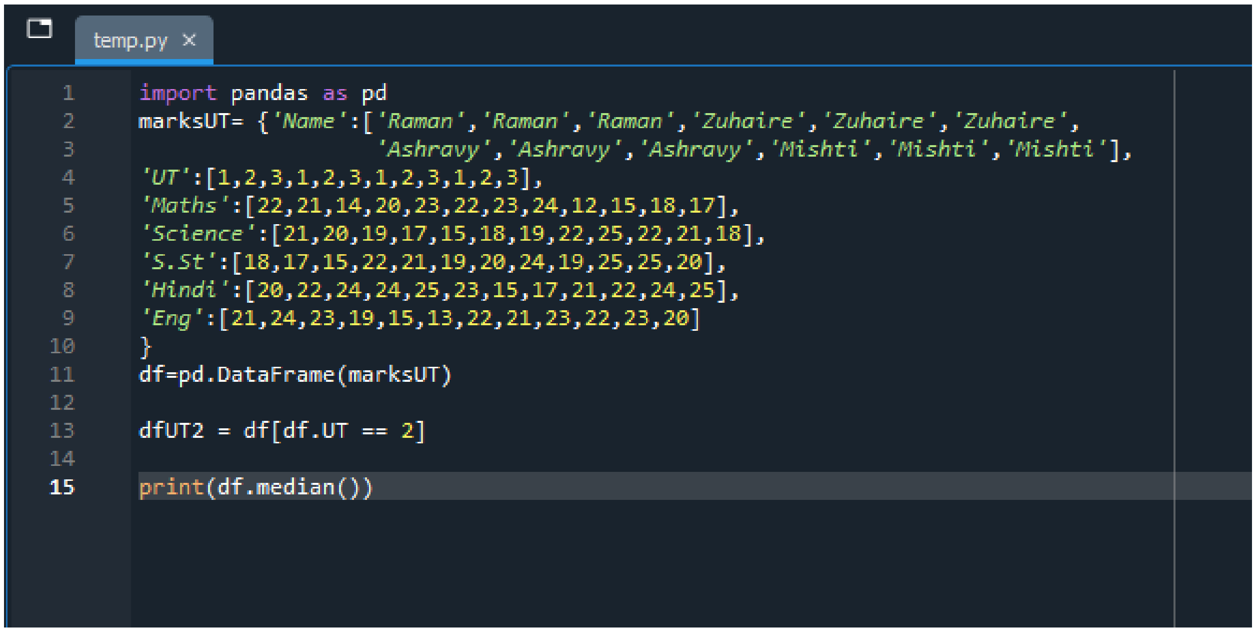
**Output:**



```
In [46]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/
Users/Yousuf/.spyder-py3')
Average of marks obtained by Zuhaire in all Unit Tests
 0    20.4
1    20.8
2    19.0
dtype: float64

In [47]:
```

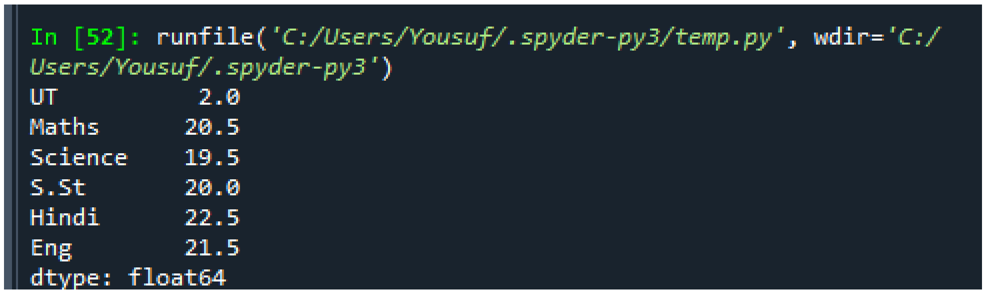
## Calculating Median

DataFrame.Median() will display the middle value of the data. This function will display the median of the values of each column of a DataFrame. It is only applicable for numeric values. For example:



```python
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfUT2 = df[df.UT == 2]

print(df.median())
```

**Output:**



```
In [52]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/
Users/Yousuf/.spyder-py3')
UT          2.0
Maths      20.5
Science    19.5
S.St       20.0
Hindi      22.5
Eng        21.5
dtype: float64
```

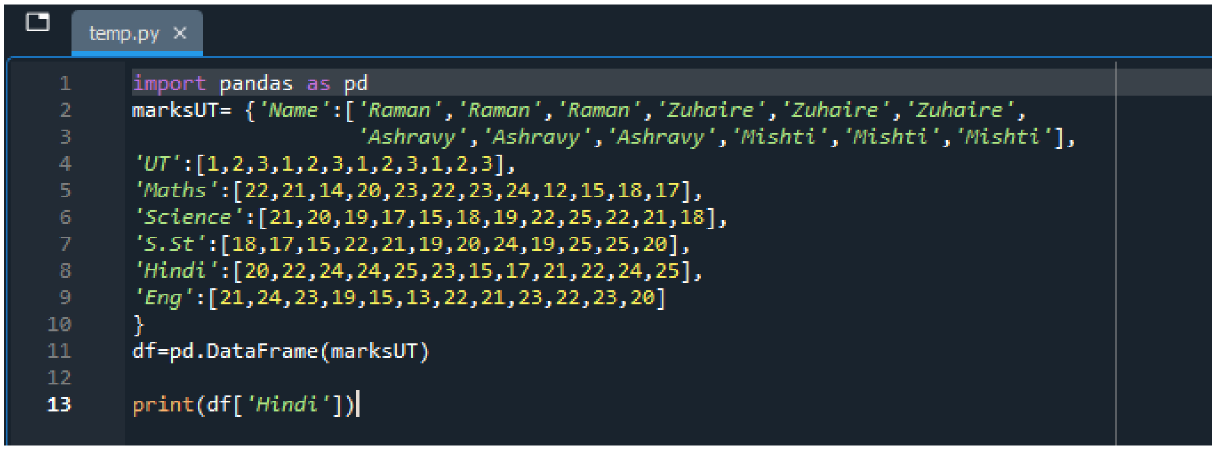
## Calculating Mode

DateFrame.mode() will display the mode. The mode is defined as the value that appears the most number of times in a data. This function will display the mode of each column or row of the DataFrame. To get the mode of Hindi marks, the following statement can be used.

**Code:**

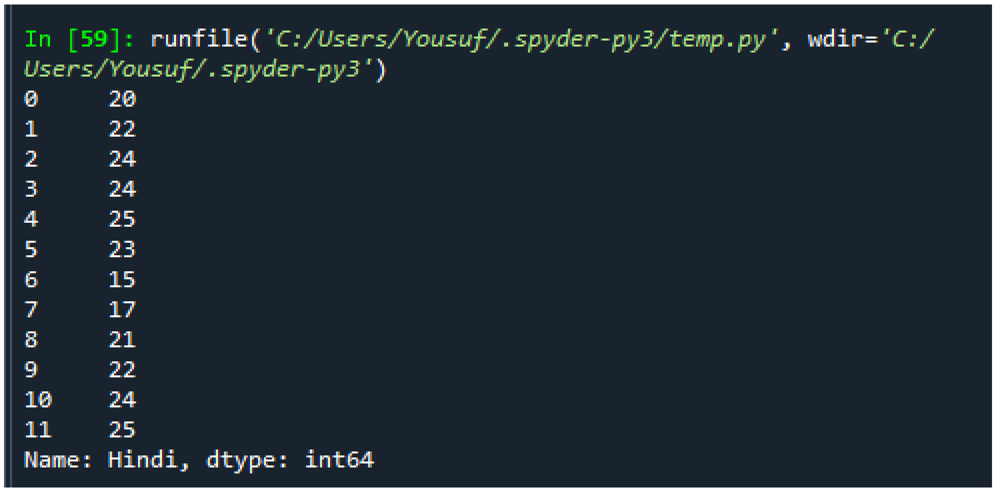
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

print(df['Hindi'])
```

**Screenshot of the code portion:**



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
             'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

print(df['Hindi'])
```

**Output:**



```
In [59]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/
Users/Yousuf/.spyder-py3')
0     20
1     22
2     24
3     24
4     25
5     23
6     15
7     17
8     21
9     22
10    24
11    25
Name: Hindi, dtype: int64
```

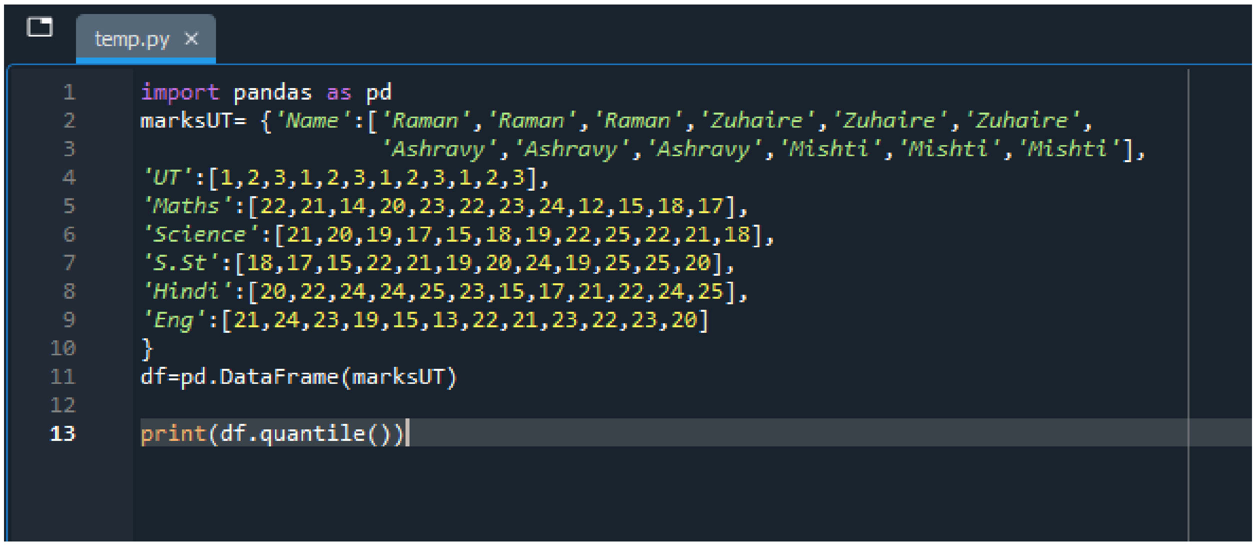
Note that three students have got 24 marks in Hindi subject while two students got 25 marks, one student got 23 marks, two students got 22 marks, one student each got 21, 20, 15, 17 marks.

## Calculating Quartile

Dataframe.quantile() is used to get the quartiles. It will output the quartile of each column or row of the DataFrame in four parts i.e. the first quartile is 25% (parameter q = .25), the second quartile is 50% (Median), the third quartile is 75% (parameter q = .75). By default, it will display the second quantile (median) of all numeric values.
Code:

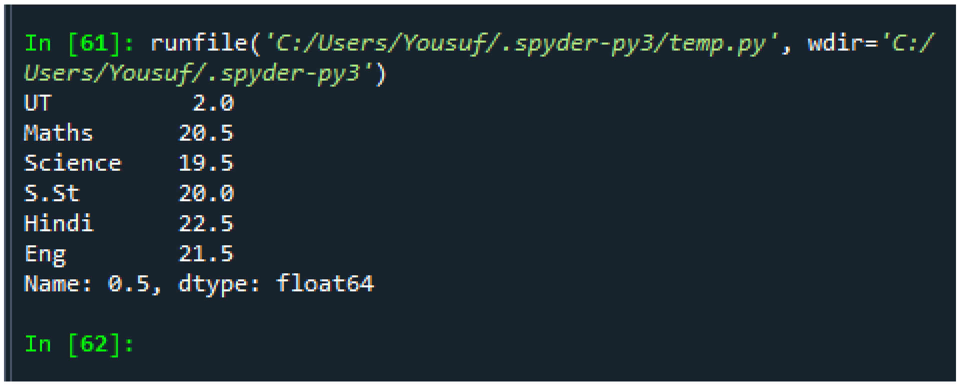
```
import pandas as pd
marksUT=
{'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',

'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

print(df.quantile())
```

Screenshot of the code:



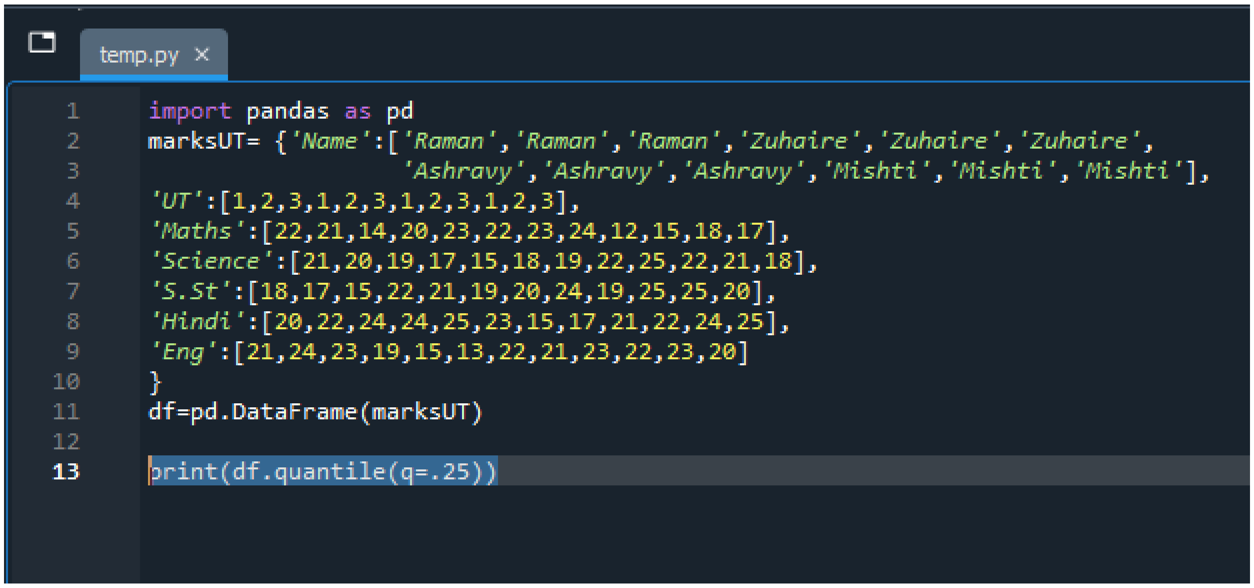
```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

print(df.quantile())
```

Output:



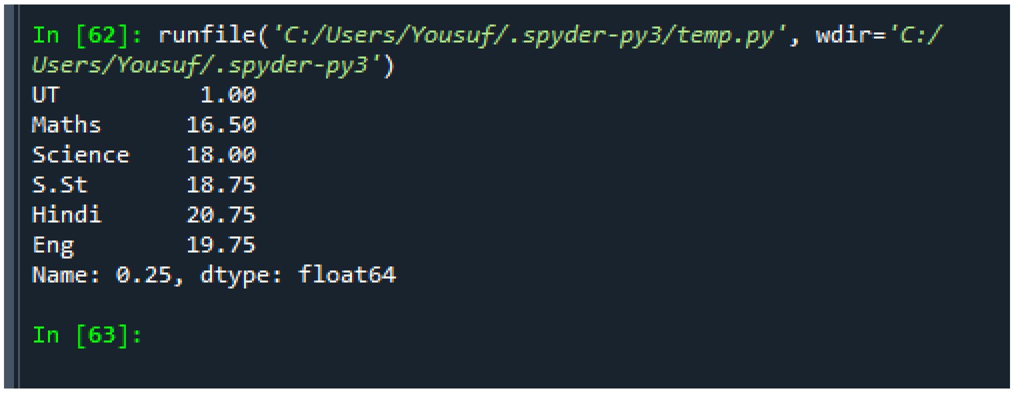
```
In [61]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/
Users/Yousuf/.spyder-py3')
UT          2.0
Maths      20.5
Science    19.5
S.St       20.0
Hindi      22.5
Eng        21.5
Name: 0.5, dtype: float64

In [62]:
```

If it is for specific numbers then the code portion is:



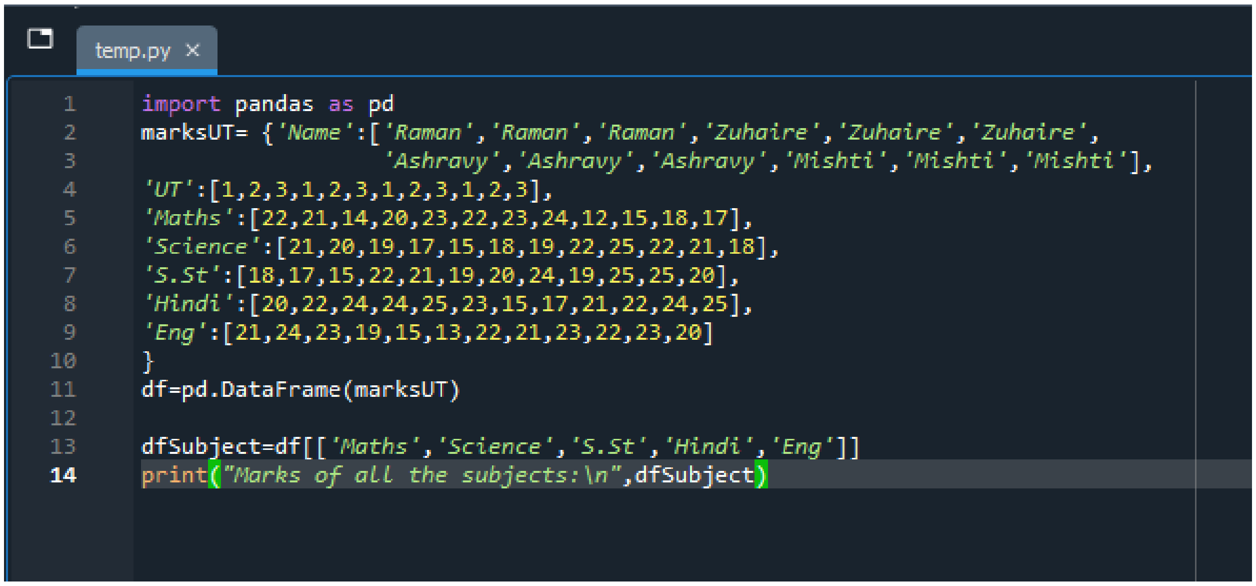
```python
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

print(df.quantile(q=.25))
```

Output:



```
In [62]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/
Users/Yousuf/.spyder-py3')
UT          1.00
Maths      16.50
Science    18.00
S.St       18.75
Hindi      20.75
Eng        19.75
Name: 0.25, dtype: float64

In [63]:
```

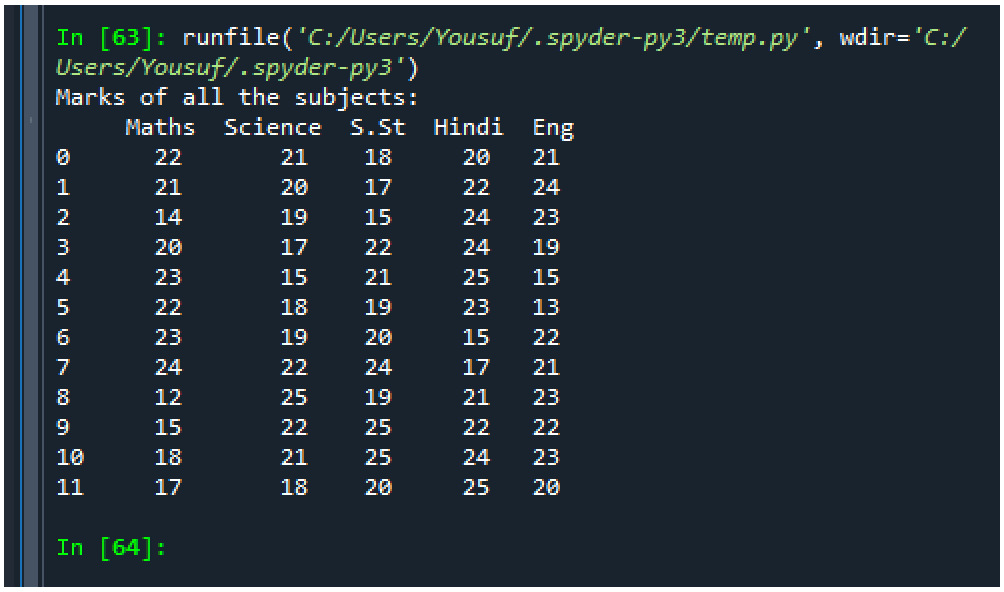
Again to summarize marks of all subjects:



temp.py

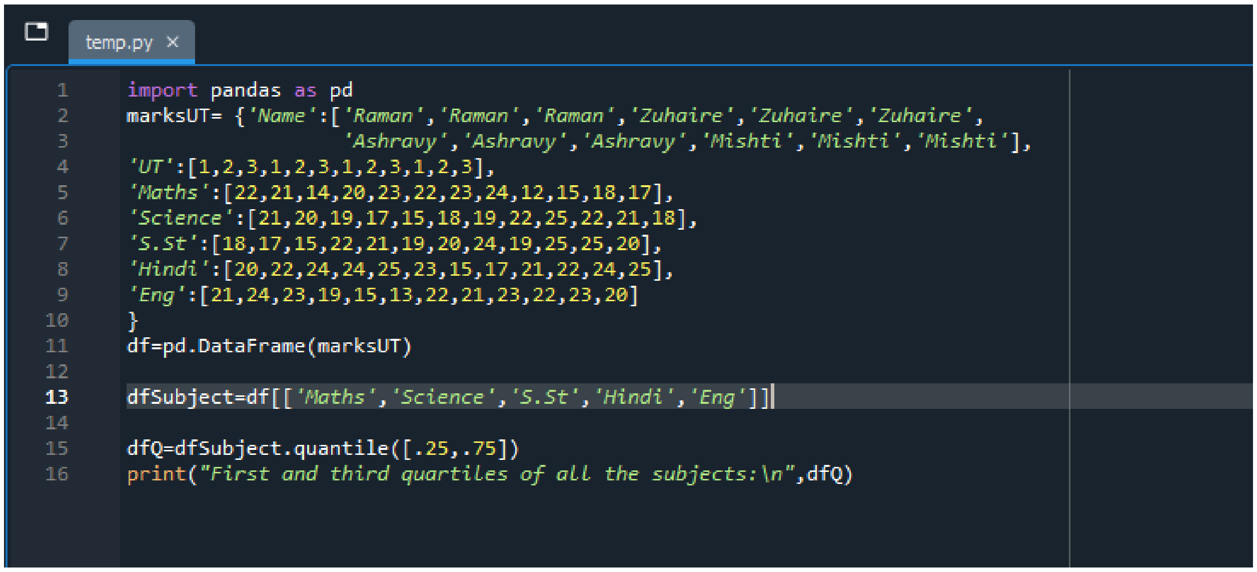
```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfSubject=df[['Maths','Science','S.St','Hindi','Eng']]
print("Marks of all the subjects:\n",dfSubject)
```

**Output:**



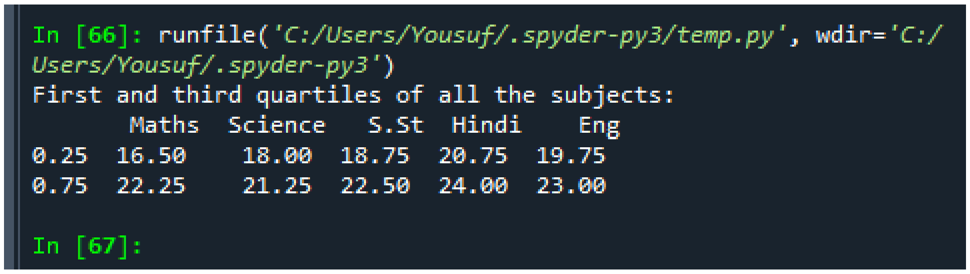
```
In [63]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/
Users/Yousuf/.spyder-py3')
Marks of all the subjects:
     Maths  Science  S.St  Hindi  Eng
0       22       21    18     20   21
1       21       20    17     22   24
2       14       19    15     24   23
3       20       17    22     24   19
4       23       15    21     25   15
5       22       18    19     23   13
6       23       19    20     15   22
7       24       22    24     17   21
8       12       25    19     21   23
9       15       22    25     22   22
10      18       21    25     24   23
11      17       18    20     25   20

In [64]:
```

Another example from the code below:



```
import pandas as pd
marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire',
                  'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
}
df=pd.DataFrame(marksUT)

dfSubject=df[['Maths','Science','S.St','Hindi','Eng']]

dfQ=dfSubject.quantile([.25,.75])
print("First and third quartiles of all the subjects:\n",dfQ)
```

**Output:**



```
In [66]: runfile('C:/Users/Yousuf/.spyder-py3/temp.py', wdir='C:/
Users/Yousuf/.spyder-py3')
First and third quartiles of all the subjects:
       Maths  Science   S.St  Hindi    Eng
0.25  16.50    18.00  18.75  20.75  19.75
0.75  22.25    21.25  22.50  24.00  23.00

In [67]:
```

## References

[1] https://www.digitalocean.com/community/tutorials/python-pandas-module-tutorial

[2] Data Mining Classroom