proschool
An ims Initiative

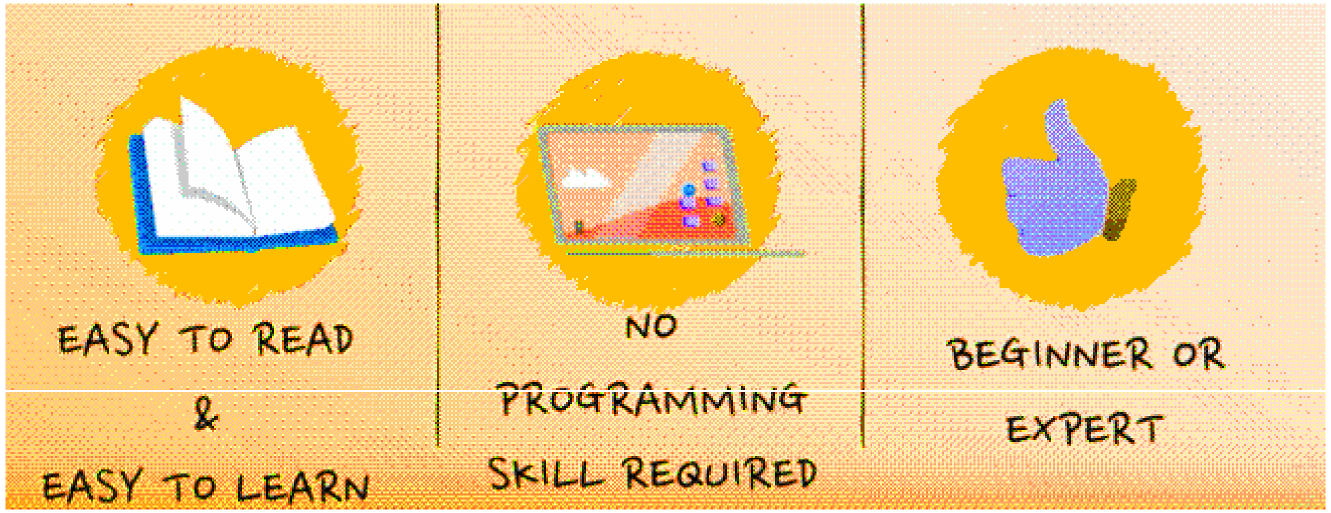
# Python

## Python

- Python is a widely used high-level programming language for general-purpose programming, created by **Guido van Rossum** and first released in 1991.
- Guido van Rossum was big fan of **Monty Python's Flying Circus** and hence had given the name of this programming language as Python.
- Python uses a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java.

## Features of Python

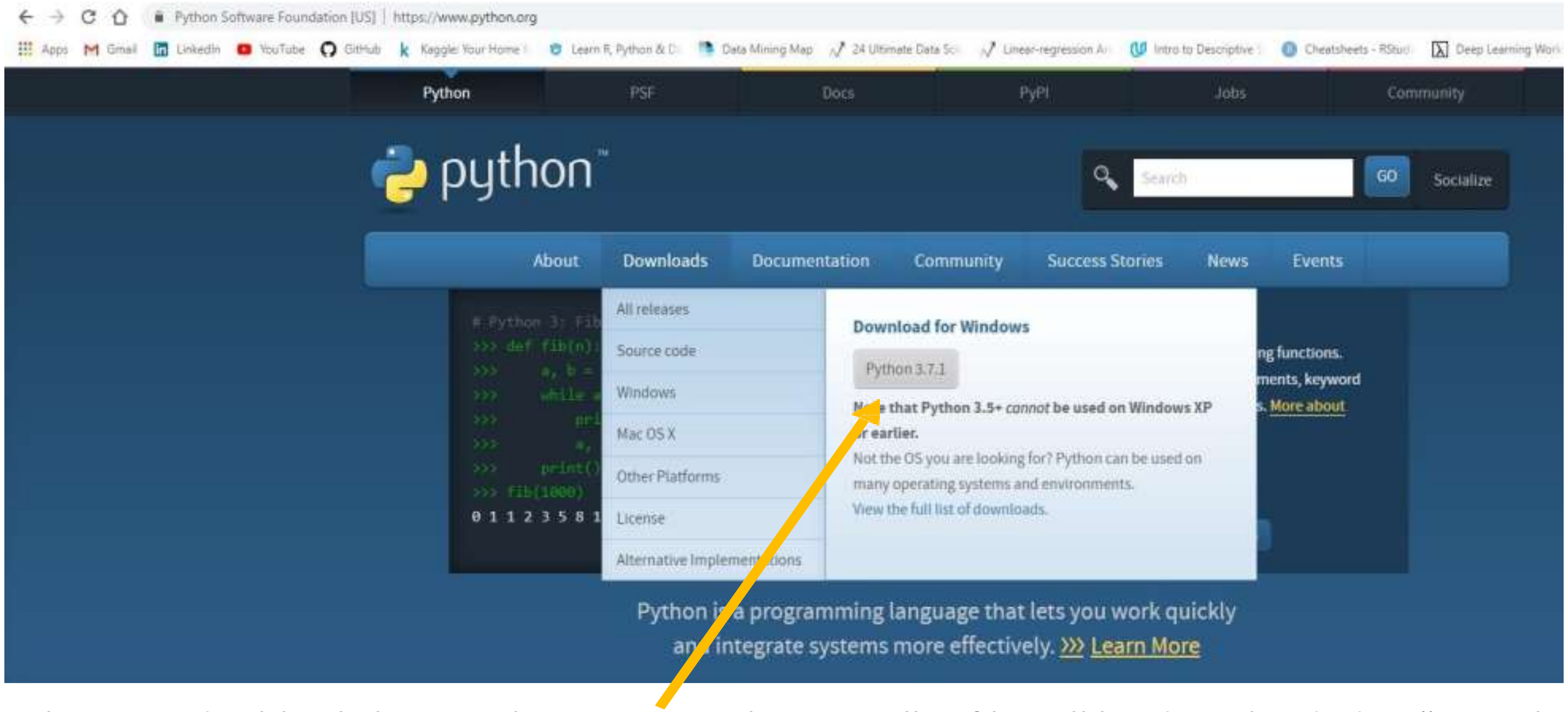
•**Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
•**Easy-to-read** − Python code is more clearly defined and visible to the eyes.
•**Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
•**Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
•**Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
•**Databases** − Python provides interfaces to all major commercial databases.
•**Scalable** − Python provides a better structure and support for large programs than shell scripting.



# Installation of Python 3.7.1

## Steps for Installing Python

1. Download the latest Python build from www.python.org as shown in Figure 1 below.



Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

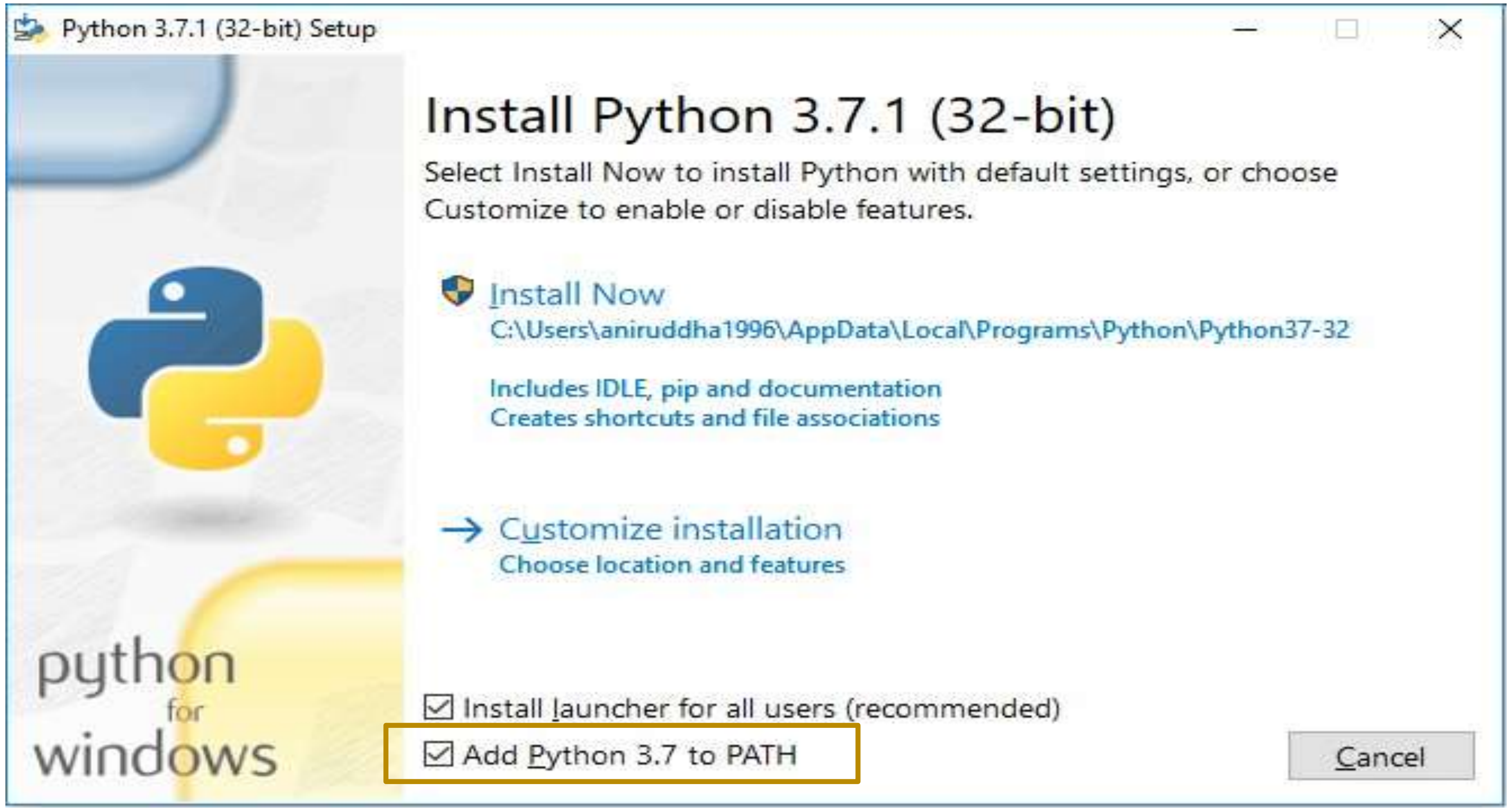
Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

When you double click on Python 3.7.1, Python installer file will be downloaded in "Download" folder.

## Installation of Python 3.7.1

proschool
An IMS Initiative

2. Double click the downloaded Python installer file to start the installation. You will be presented with the options as shown in Figure 3 below. Make sure that "Add Python 3.7 to PATH" checkbox is ticked.


Python 3.7.1 (32-bit) Setup

Install Python 3.7.1 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\aniruddha1996\AppData\Local\Programs\Python\Python37-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

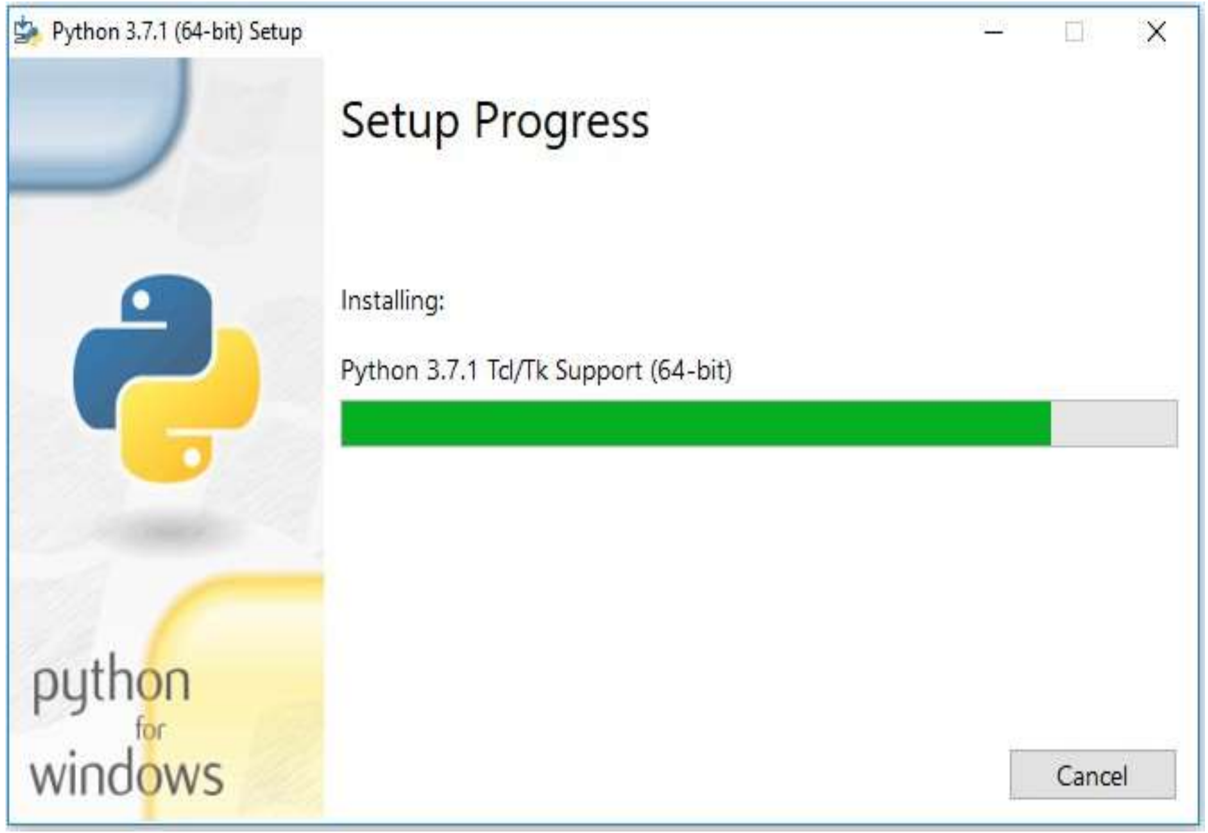
☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

Click the Install Now and Python 3.7.1 install will proceed.

# Installation of Python 3.7.1

proschool
An ims Initiative

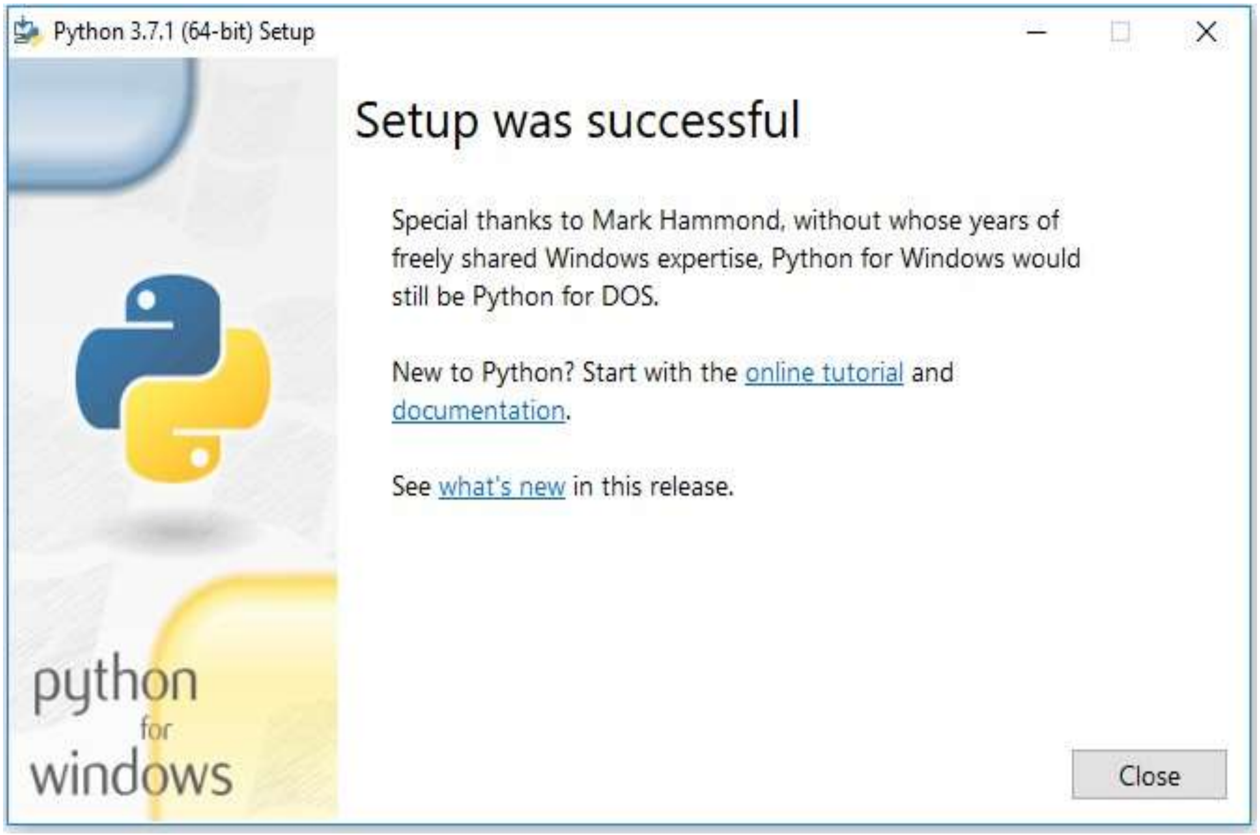
This shows the successful installation of Python 3.7.1


Python 3.7.1 (64-bit) Setup

Setup Progress

Installing:

Python 3.7.1 Tcl/Tk Support (64-bit)


Python 3.7.1 (64-bit) Setup

Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

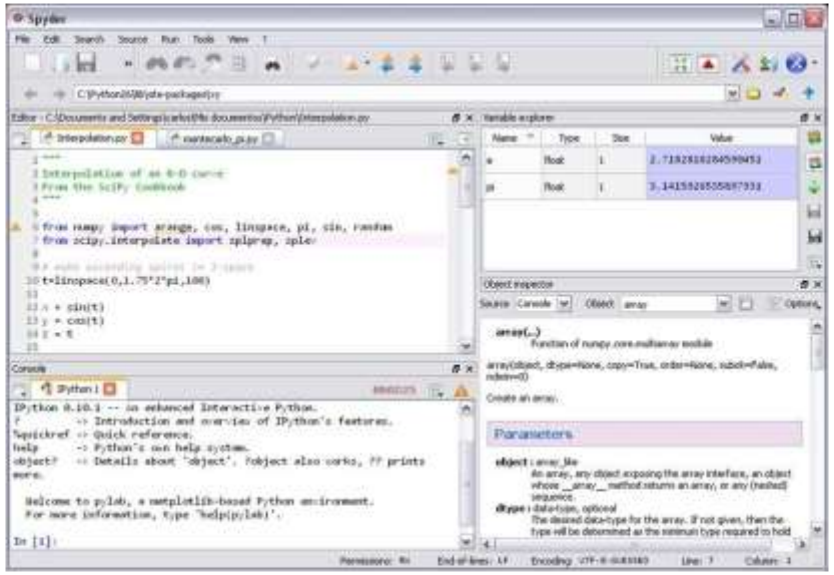
New to Python? Start with the online tutorial and documentation.

See what's new in this release.

# IDE for Python -> Integrated Development Environment

It's a coding tool which allows you to write, test and debug your code in an easier way, as they typically offer code completion or code insight by highlighting, resource management, and debugging tools.

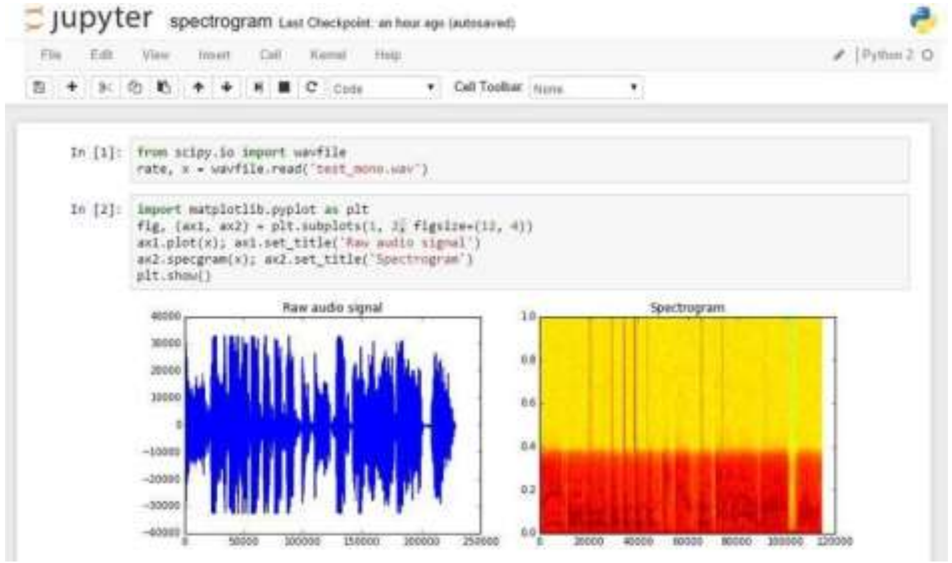
- Spyder
- Jupyter
- Atom
- Pycharm
- Rodeo



```
"""
Interpolation of an N-D curve
From the SciPy Cookbook
"""

from numpy import arange, cos, linspace, pi, sin, random
from scipy.interpolate import splprep, splev

# make ascending spiral in 3-space
t=linspace(0,1.75*2*pi,100)

x = sin(t)
y = cos(t)
z = t
```

```
IPython 0.10.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints
more.

Welcome to pylab, a matplotlib-based Python environment.
For more information, type 'help(pylab)'.

In [1]:
```

Spyder



```
In [1]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')

In [2]: import matplotlib.pyplot as plt
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram')
        plt.show()
```

Jupyter

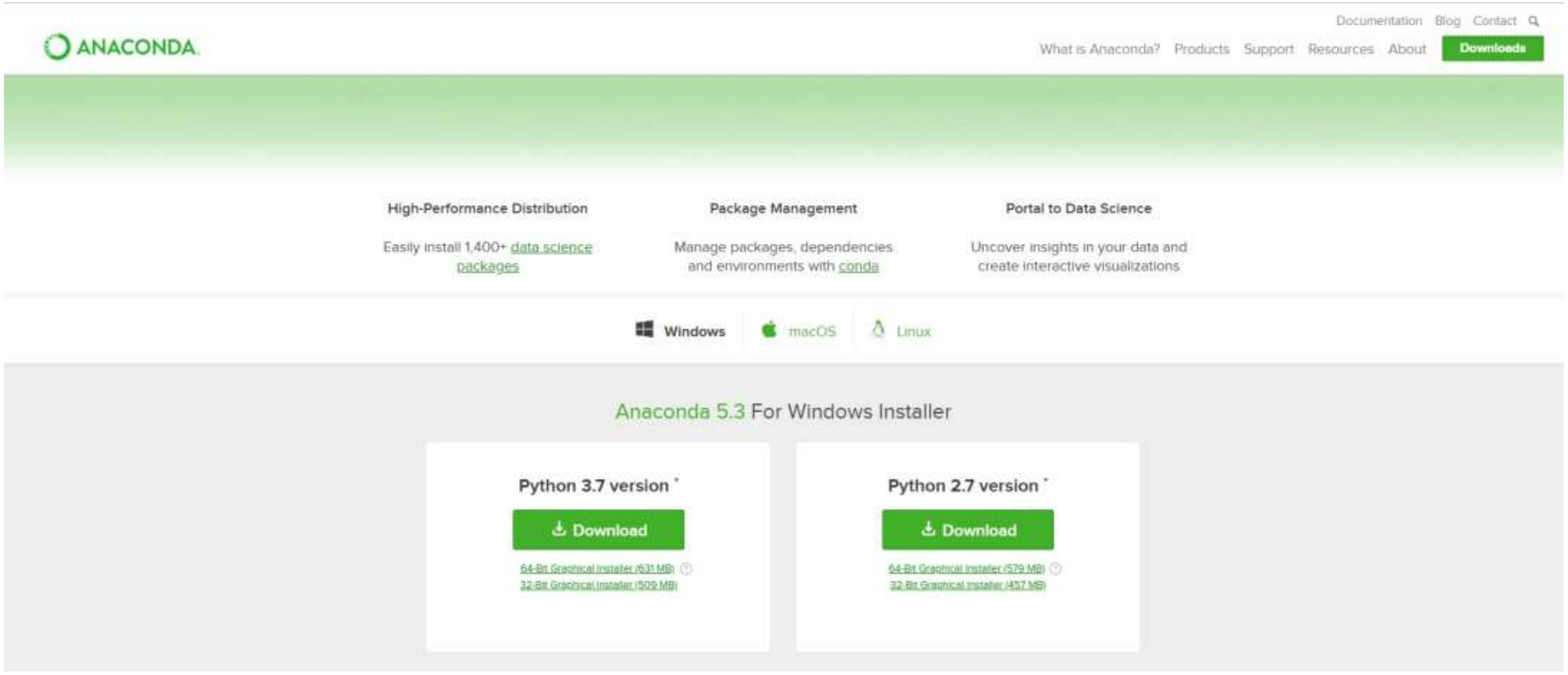
## Anaconda Installation – Many problems one solution

1. Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux.
2. Conda quickly installs, runs and updates packages and their dependencies.
3. Conda easily creates, saves, loads and switches between environments on your local computer.
4. It was created for Python programs, but it can package and distribute software for any language.
5. Conda as a package manager helps you find and install packages.
6. The conda package and environment manager is included in all versions of Anaconda.
7. If we install Anaconda – **Spyder**, **Jupyter**, and even **Python** latest version would also be installed simultaneously.
8. To install Anaconda -> Click on to the following links

   https://www.anaconda.com/download/

9. Anaconda package which provides code completion and linking for Python and takes care of the basics, there are tons of themes, lightning fast user interface, easy configuration, tons of packages for more power features

# Anaconda installation


Documentation Blog Contact
ANACONDA.
What is Anaconda? Products Support Resources About Downloads

High-Performance Distribution

Easily install 1,400+ data science packages

Package Management

Manage packages, dependencies and environments with conda

Portal to Data Science

Uncover insights in your data and create interactive visualizations

Anaconda 5.3 For Windows Installer

Python 3.7 version *

64-Bit Graphical Installer (631 MB)
32-Bit Graphical Installer (509 MB)

Python 2.7 version *

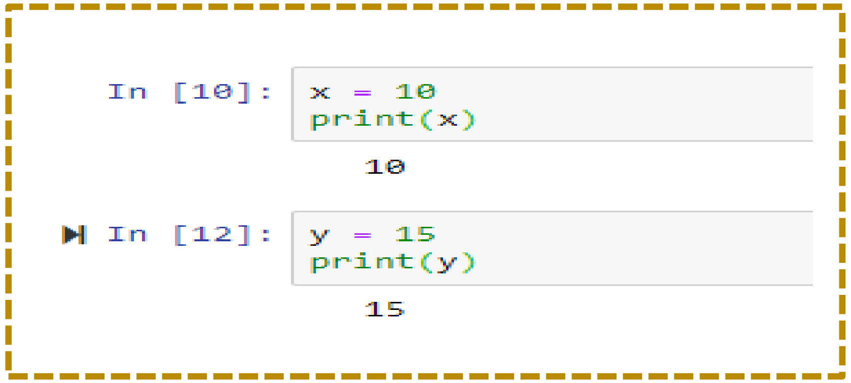
64-Bit Graphical Installer (579 MB)
32-Bit Graphical Installer (457 MB)

To download Anaconda click here and install it. Jupyter, Spyder and python will be installed simultaneously.

## Basics of Python

## Variables

- Variable is used to store data and to retrieve the data. That means, we can change its value . It is not fixed. Example :



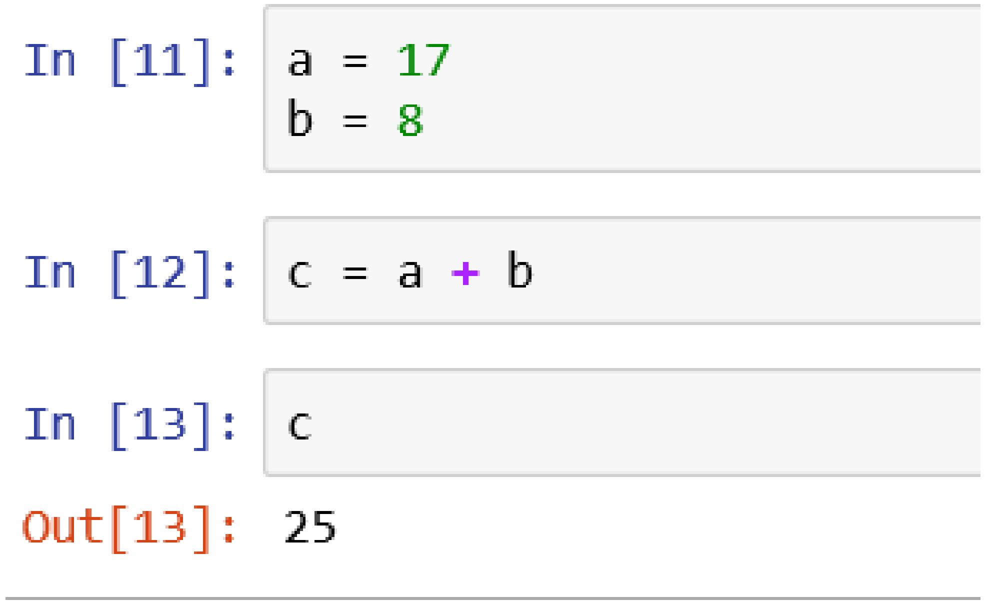
```
In [10]: x = 10
         print(x)

         10

In [12]: y = 15
         print(y)

         15
```

In python variables are case sensitive and consists of numbers, letters and underscore.

- Reserved words should not be used as a variables names.

  Example : else , return ,and print etc.

## How to assign values to variables?



```
In [11]: a = 17
         b = 8

In [12]: c = a + b

In [13]: c

Out[13]: 25
```

Practise Exercise:

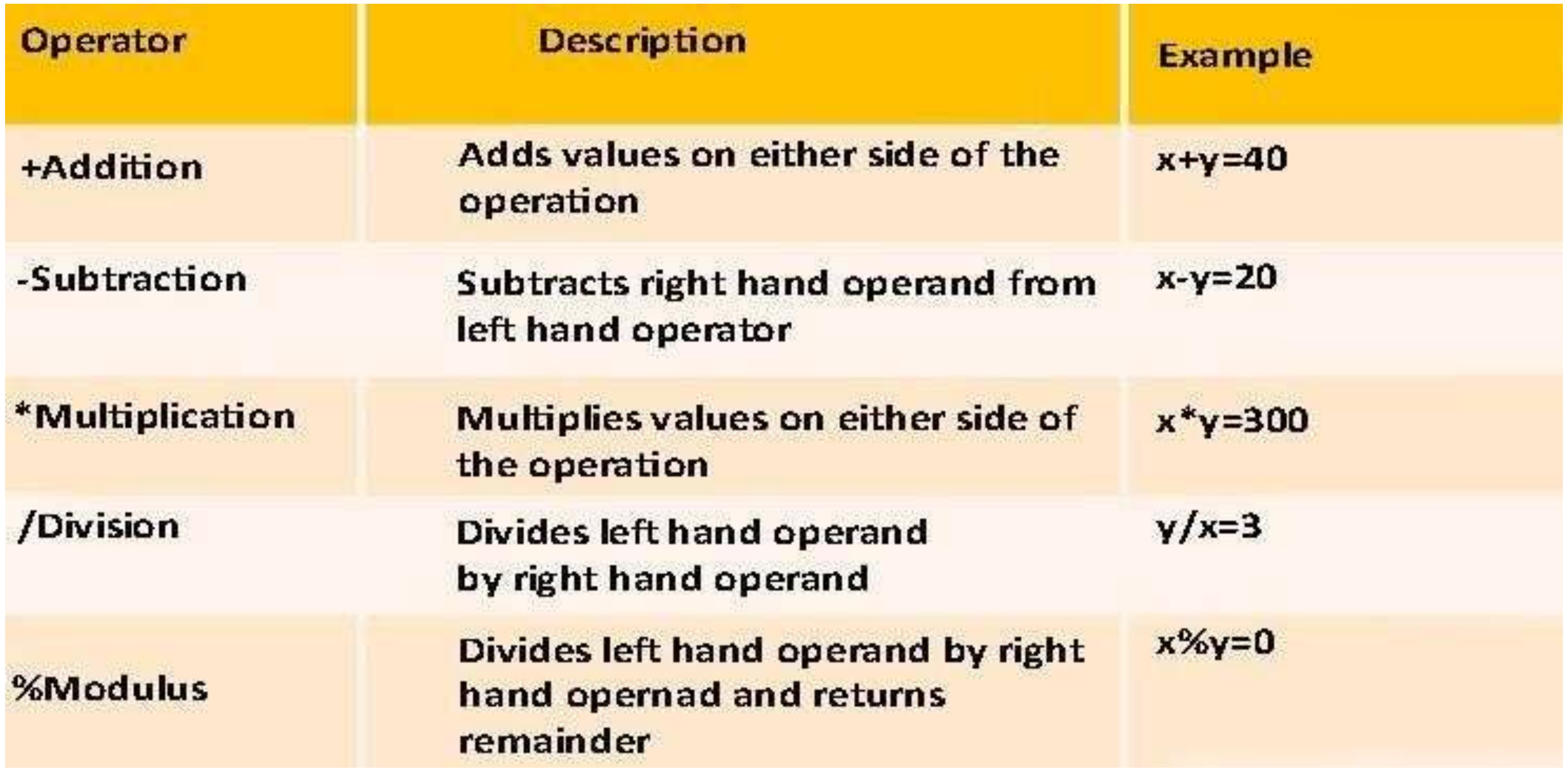
- Calculate the value of x*y, x/y where x=9 and y= 13

## Operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

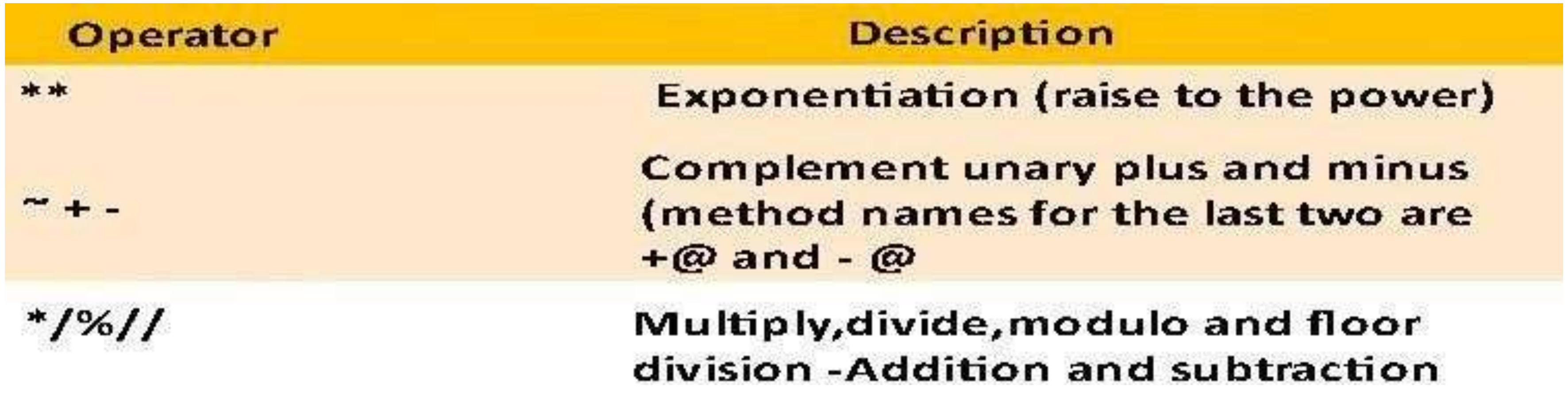
## Python Arithmetic Operators

- We use these operators for mathematical  purpose.



| Operator | Description | Example |
|---|---|---|
| +Addition | Adds values on either side of the operation | x+y=40 |
| -Subtraction | Subtracts right hand operand from left hand operator | x-y=20 |
| *Multiplication | Multiplies values on either side of the operation | x*y=300 |
| /Division | Divides left hand operand by right hand operand | y/x=3 |
| %Modulus | Divides left hand operand by right hand opernad and returns remainder | x%y=0 |

## Python Operator Precedence



| Operator | Description |
|---|---|
| ** | Exponentiation (raise to the power) |
| ~ + - | Complement unary plus and minus (method names for the last two are +@ and - @ |
| */%// | Multiply,divide,modulo and floor division -Addition and subtraction |

## Python Comparison Operators

proschool
An IMS Initiative

- We use these operators for conditional purpose.

| Operator | Description ( usage) |
|---|---|
| > | if the left operator has greater value than the right one. Ex= 2>1 |
| < | If the right operator has greater value than the left one. Ex= 6<9 |
| == | If both sides of the operators are same to same .<br>Ex= 4==4 |
| != | if both sides of the operators are not same. |
| >= | If the left operator is greater than or equal to right one. Ex = 6>=6,7>=4 |
| <= | If the right operator is greater than or equal to left one. Ex= 4<=4,6<=8 |

# Python Assignment Operators

proschool
An IMS Initiative

- These are used to assign values to variables. Few one them are as shown below.

| Operator | Description and Example |
|---|---|
| = | S =2+3 means the value 2+3=5 is assigned into S |
| += | H+=4 means 4 is added to H and final result is saved into H. So H+4=H. i.e if H value is 2 then 6+4=10 is stored in H. |
| -= | H-= 4 means 4 is subtracted to H and final result is saved into H. So H-4=H i.e 6-4=2 is stored in H. |
| *= | H*= 4 means 4 is multiply with H and final result is saved into H. So H*4=H i.e 6*4=24 is stored in H. |

- Similarly few more operators like %=,//=,**=,&=,\|= ……

## Python Logical Operators

- Logical operators are also used to check conditional statements.

| Operator | Description |
|---|---|
| AND | used to check both the operands or true<br><br>Ex a=2,b=2 if a AND b=2<br>then print(‘ a AND b is’, True)<br>i.e output is TRUE |
| OR | Used to check either one of the operator is true<br>Ex a=2,b=3 if a OR b =3<br>then print(‘ a OR b is ’, 3)<br>i.e output is b=3. |
| NOT | Used to complement the operator<br>If given X is true then it shows it is false . |

## Python Membership Operators

- These operators are used to check whether the given values are in the sequence or not.

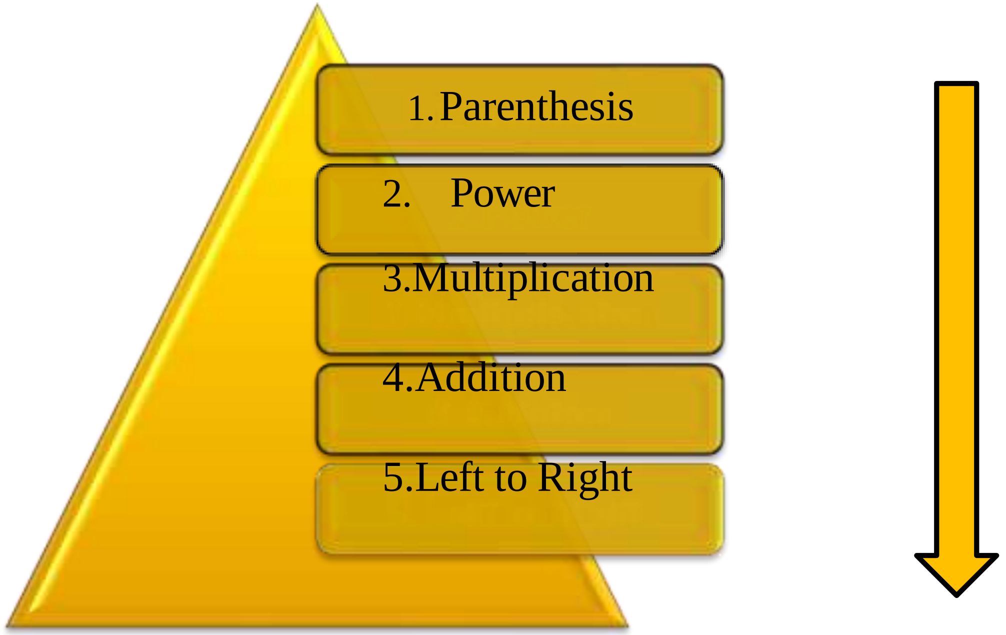
| Operator | Description |
|---|---|
| in | If the given value is within the sequence it will displays . Example as shown below<br>X = { 2,4,6,8}<br>Print(4 in X )<br>then it will display 'True' because 4 is within the x |
| Not in | Opposite to IN operation.<br>Example : 9 Not in then it will check whether 9 is in sequence or not. Regarding to the above sequence we do not have 9 so it will display 9. |

## Python Identity Operators

- These operators are used to check whether the values are in the memory or not.

| Operator | Description |
|---|---|
| is | If the operand are same then it is true.<br>EX: x=2,y=2 then print X is Y. |
| Is not | If the Operand are not same then it shows true.<br>Ex: x=2 ,y=2 print x is not Y then it shows false.<br>Ex: x=2y=3 print x is not y then it shows True. |

## Operator Precedence



- During operation where one operand is an integer and the other operand is a floating point the result is a floating point

- The integer is converted to a floating point before the operation.

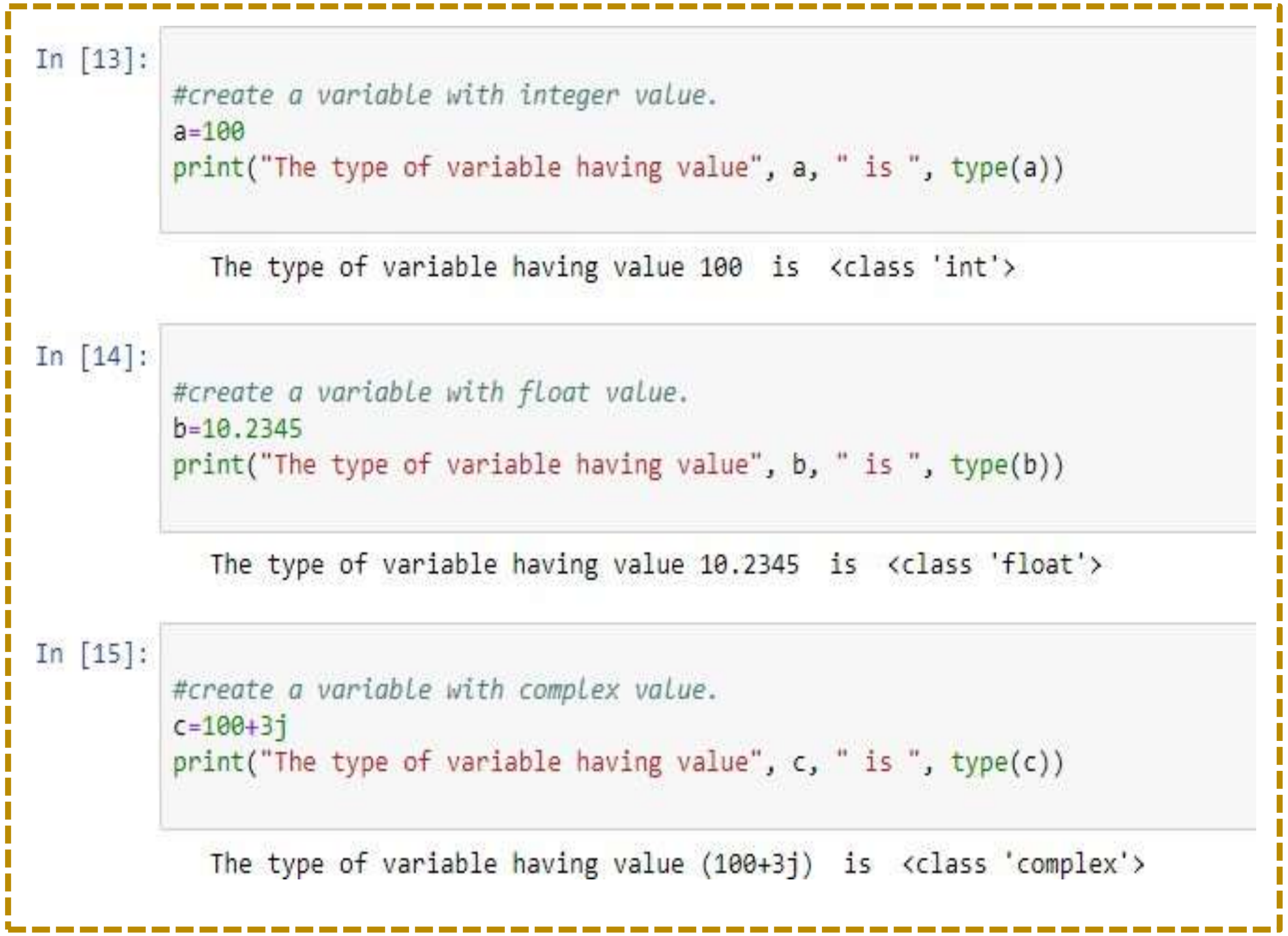
## Different Data Types

### Numeric Data Type

Python numeric data type is used to hold numeric values like;

- int – holds signed integers of non-limited length.
- float- holds floating precision numbers and it's accurate up to 15 decimal places.
- complex- holds complex numbers.

In Python we can simply assign values in a variable. But if we want to see what type of numerical value is it holding right now, we can use **type()**, like this:



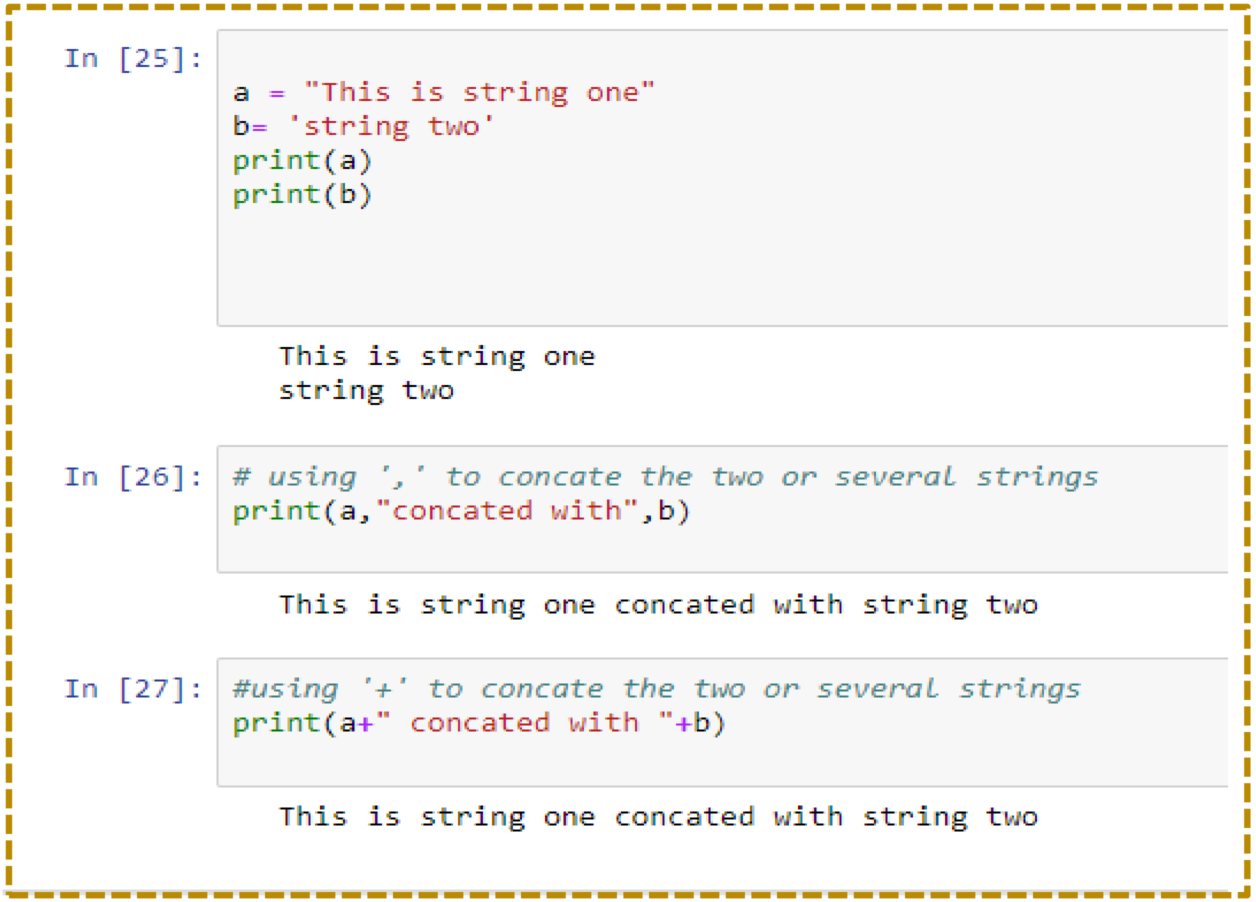
```
In [13]:
#create a variable with integer value.
a=100
print("The type of variable having value", a, " is ", type(a))

The type of variable having value 100  is  <class 'int'>

In [14]:
#create a variable with float value.
b=10.2345
print("The type of variable having value", b, " is ", type(b))

The type of variable having value 10.2345  is  <class 'float'>

In [15]:
#create a variable with complex value.
c=100+3j
print("The type of variable having value", c, " is ", type(c))

The type of variable having value (100+3j)  is  <class 'complex'>
```

# Different Data Types (Contd.)

### String

String is a sequence of characters. Python supports Unicode characters. Generally strings are represented by either single or double quotes.

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

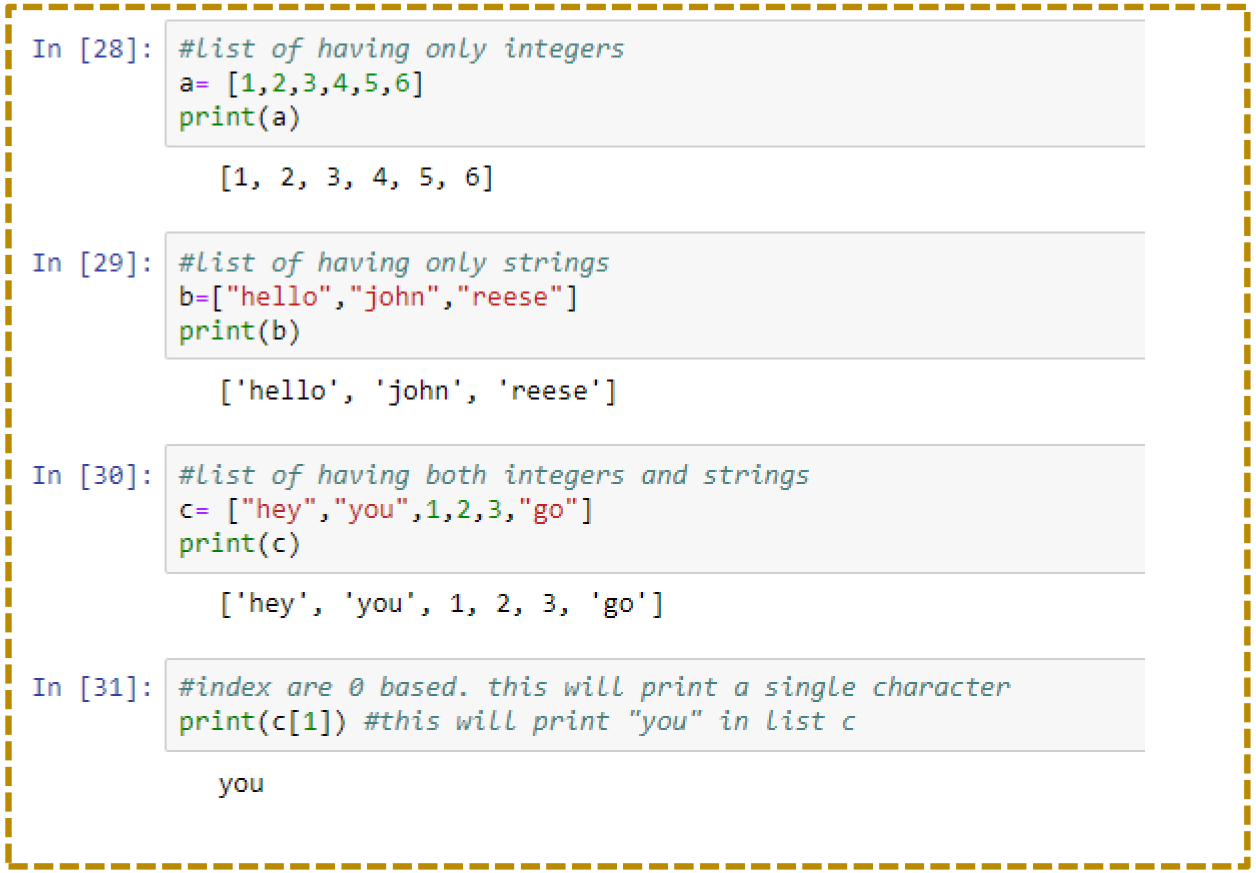


```
In [25]: a = "This is string one"
         b= 'string two'
         print(a)
         print(b)

         This is string one
         string two

In [26]: # using ',' to concate the two or several strings
         print(a,"concated with",b)

         This is string one concated with string two

In [27]: #using '+' to concate the two or several strings
         print(a+" concated with "+b)

         This is string one concated with string two
```

## Different Data Types (Contd.)

**List**

List is a versatile data type in Python. Interesting thing about list in Python is it can simultaneously hold different type of data. Formally list is a ordered sequence of some data written using square brackets([]) and commas(,).

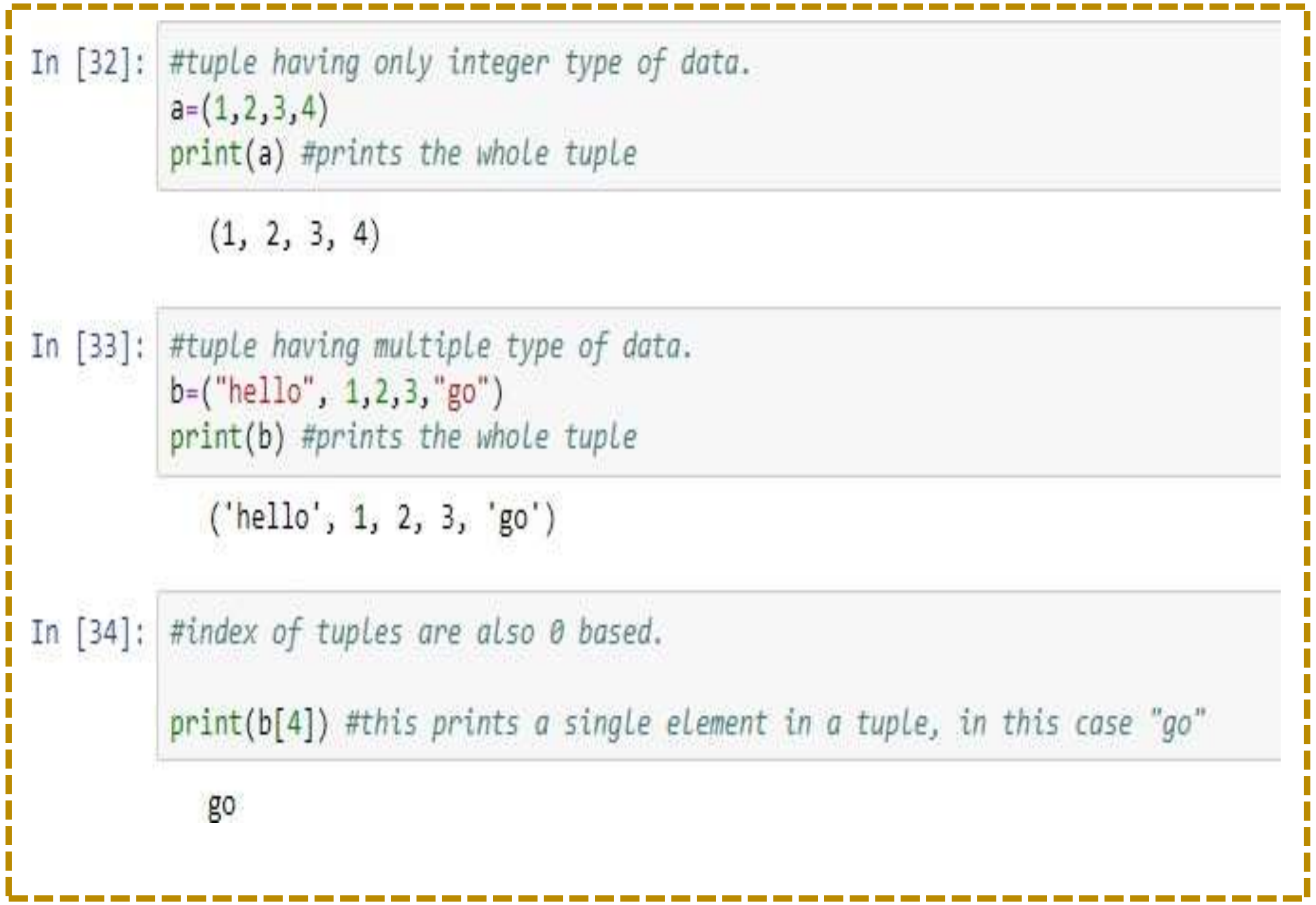


```
In [28]: #list of having only integers
         a= [1,2,3,4,5,6]
         print(a)

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

In [29]: #list of having only strings
         b=["hello","john","reese"]
         print(b)

         ['hello', 'john', 'reese']

In [30]: #list of having both integers and strings
         c= ["hey","you",1,2,3,"go"]
         print(c)

         ['hey', 'you', 1, 2, 3, 'go']

In [31]: #index are 0 based. this will print a single character
         print(c[1]) #this will print "you" in list c

         you
```

## Different Data Types (Contd.)

**Tuple**

Tuple is another data type which is a sequence of data similar to list. But it is immutable. That means data in a tuple is write protected. Data in a tuple is written using parenthesis and commas.



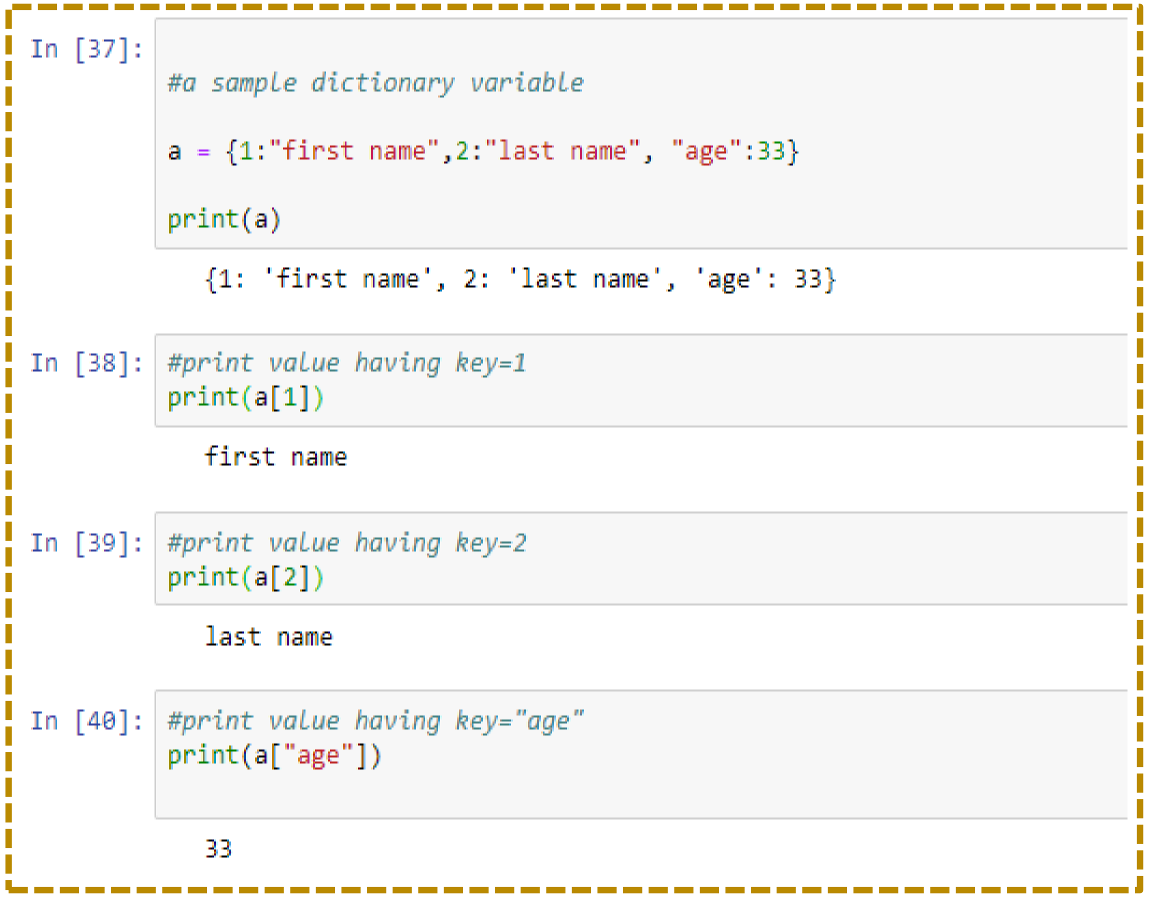
```
In [32]: #tuple having only integer type of data.
         a=(1,2,3,4)
         print(a) #prints the whole tuple

         (1, 2, 3, 4)

In [33]: #tuple having multiple type of data.
         b=("hello", 1,2,3,"go")
         print(b) #prints the whole tuple

         ('hello', 1, 2, 3, 'go')

In [34]: #index of tuples are also 0 based.

         print(b[4]) #this prints a single element in a tuple, in this case "go"

         go
```

## Different Data Types (Contd.)

**Dictionary**

Python Dictionary is an unordered sequence of data of key-value pair form. It is similar to the hash table type. Dictionaries are written in the form key : value. This means, for each key, there should be a value. All the keys are unique. It is very useful to retrieve data in an optimized way among large amount of data.

We can initialize a dictionary closed by curly braces. Key and values are separated by a semicolon and the values are separated by the comma. Dictionaries are easy to use. The following code will help you understand Python Dictionary.



```
In [37]: #a sample dictionary variable

         a = {1:"first name",2:"last name", "age":33}

         print(a)

         {1: 'first name', 2: 'last name', 'age': 33}

In [38]: #print value having key=1
         print(a[1])

         first name

In [39]: #print value having key=2
         print(a[2])

         last name

In [40]: #print value having key="age"
         print(a["age"])

         33
```

# Indexing in Python

Python follows a zero-based index where the initial element of a sequence is assigned the index 0, rather than the index 1 as is typical in everyday non-mathematical or non-programming circumstances.
These indexes are used to access element in a list, string, array etc.

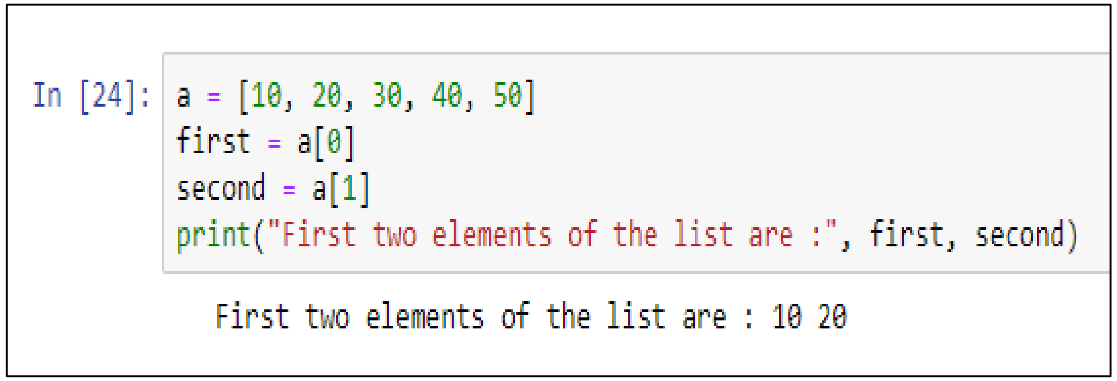
There are 2 different ways to access elements using index:

1) To start from the front

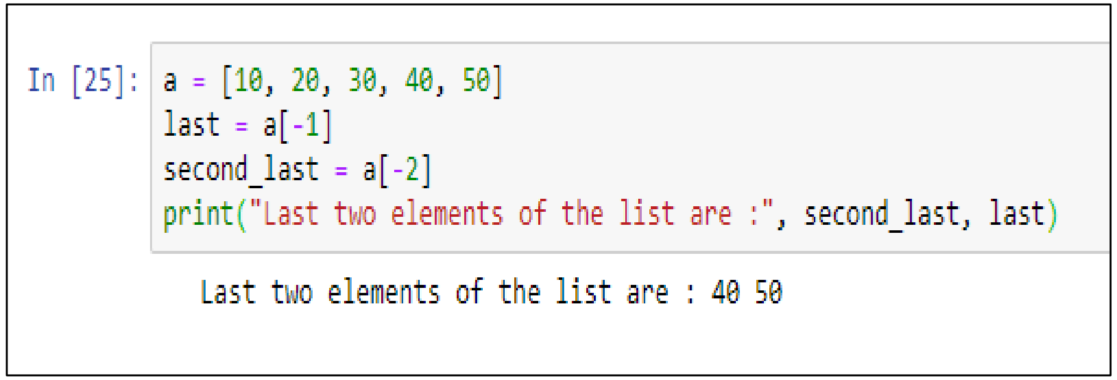
```
a = [10, 20, 30, 40, 50]
first = a[0]
second = a[1]
print(“First two elements of the list are :”, first, second)
```

2) To start from end – This is useful when you have a large set of data

```
a = [10, 20, 30, 40, 50]
last = a[-1]
second_last = a[-2]
print(“Last 2 elements of the list are :”, second_last, last)
```



```
In [24]: a = [10, 20, 30, 40, 50]
         first = a[0]
         second = a[1]
         print("First two elements of the list are :", first, second)

         First two elements of the list are : 10 20
```



```
In [25]: a = [10, 20, 30, 40, 50]
         last = a[-1]
         second_last = a[-2]
         print("Last two elements of the list are :", second_last, last)

         Last two elements of the list are : 40 50
```

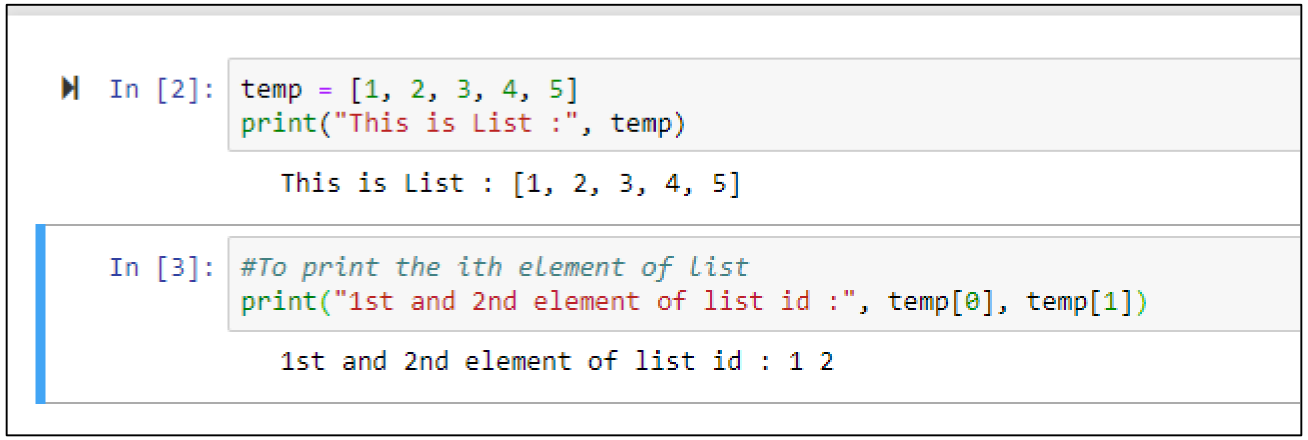
# Data Manipulation in List

proschool
An ims Initiative

- In python, **lists** are part of the standard language.
- We could store multiple values in a single object. Lets say we would like to store a sequence of numbers/Id's, this could be achieved by lists
- List and Stacks, both are same in python
  - >>> l = [1, 2, 3, 4, 5]
  - >>> l[0]
  - 1

It is initiated with brackets []

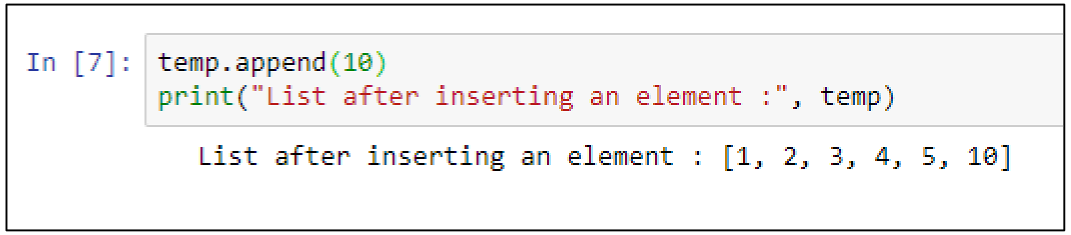
To access first element, we have to use bracket as index .



```
In [2]: temp = [1, 2, 3, 4, 5]
        print("This is List :", temp)

        This is List : [1, 2, 3, 4, 5]

In [3]: #To print the ith element of list
        print("1st and 2nd element of list id :", temp[0], temp[1])

        1st and 2nd element of list id : 1 2
```

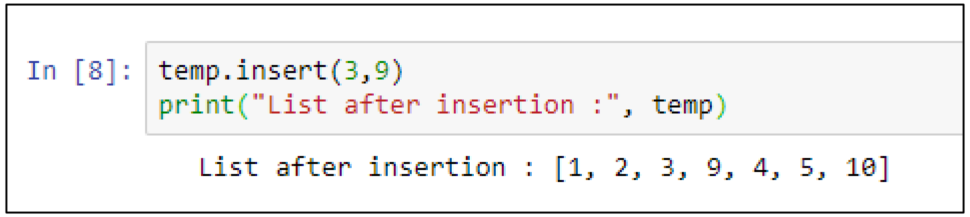
## Data Manipulation in List

- To insert element to list we use append, this corresponds to push when we use list as stack
  - temp.append(10)
  - print("List after inserting an element :", temp)



```
In [7]: temp.append(10)
        print("List after inserting an element :", temp)

        List after inserting an element : [1, 2, 3, 4, 5, 10]
```

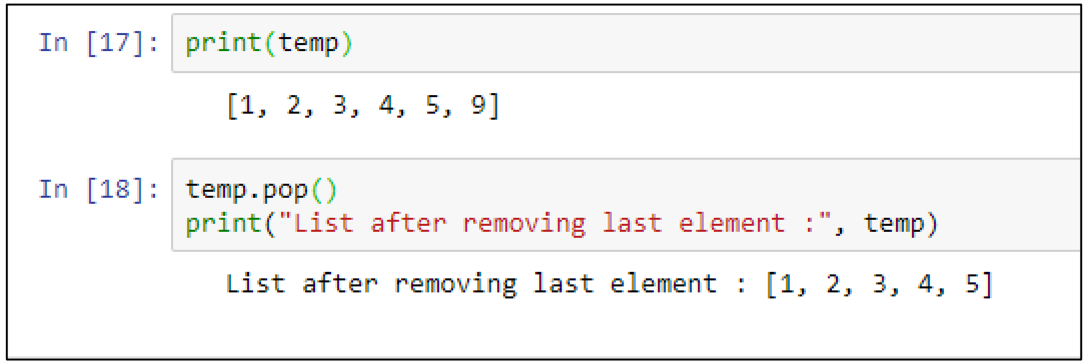
- To insert an element at ith place
  - temp.insert(3,9)
  - print("List after insertion :", temp)



```
In [8]: temp.insert(3,9)
        print("List after insertion :", temp)

        List after insertion : [1, 2, 3, 9, 4, 5, 10]
```

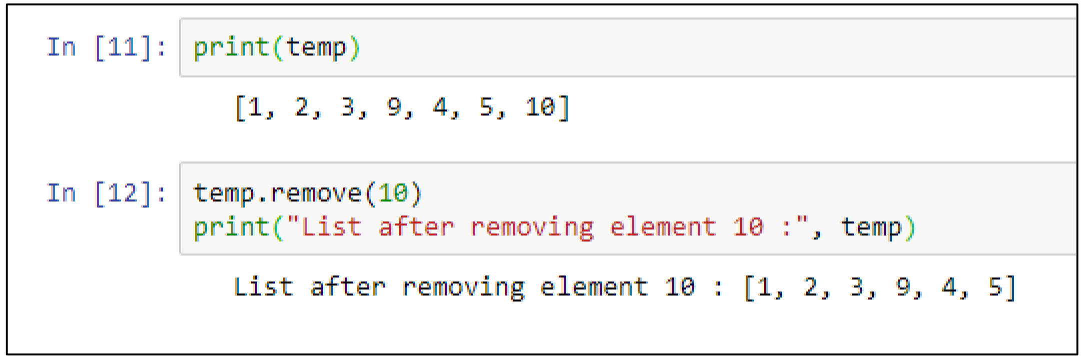
## Data Manipulation in List

- To remove last element of list or pop the element from stack
  - print(temp)
  - temp.pop()
  - print("List after removing last element",temp)



```
In [17]: print(temp)

         [1, 2, 3, 4, 5, 9]

In [18]: temp.pop()
         print("List after removing last element :", temp)

         List after removing last element : [1, 2, 3, 4, 5]
```

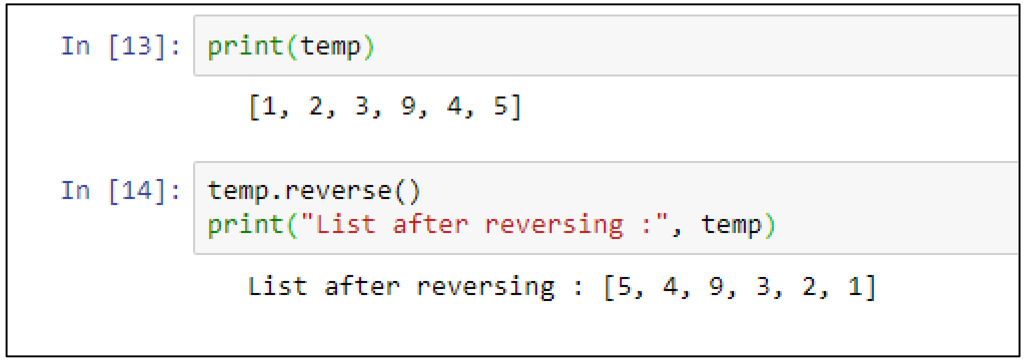
- To remove an element
  - print(temp)
  - temp.remove(10)
  - print("List after removing element 10 :", temp)



```
In [11]: print(temp)

         [1, 2, 3, 9, 4, 5, 10]

In [12]: temp.remove(10)
         print("List after removing element 10 :", temp)

         List after removing element 10 : [1, 2, 3, 9, 4, 5]
```

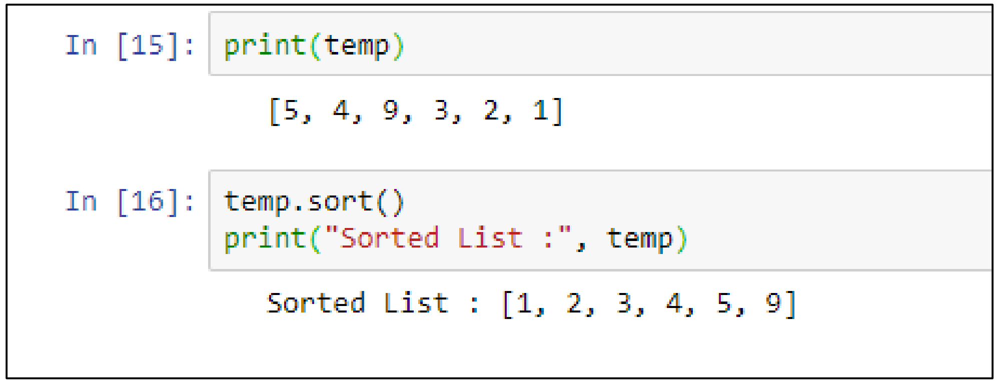
# Data Manipulation in List

- To reverse the list
  - print(temp)
  - temp.reverse()
  - print("List after reversing", temp)



```
In [13]: print(temp)

         [1, 2, 3, 9, 4, 5]

In [14]: temp.reverse()
         print("List after reversing :", temp)

         List after reversing : [5, 4, 9, 3, 2, 1]
```

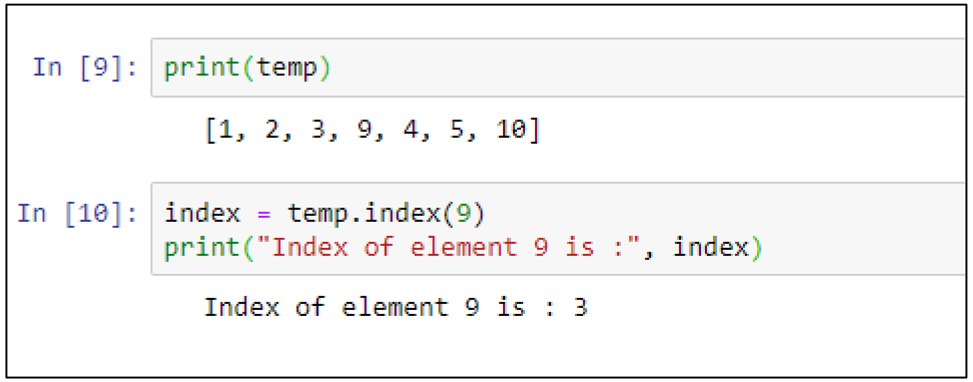
- To sort the list
  - print(temp)
  - temp.sort()
  - print("Sorted list",temp)



```
In [15]: print(temp)

         [5, 4, 9, 3, 2, 1]

In [16]: temp.sort()
         print("Sorted List :", temp)

         Sorted List : [1, 2, 3, 4, 5, 9]
```

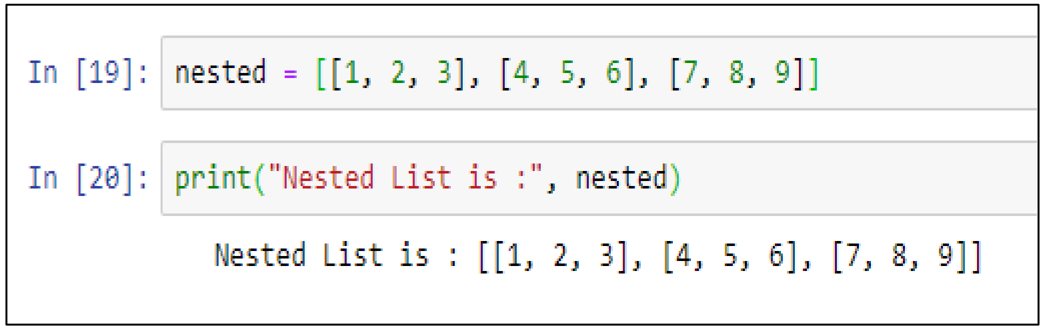
## Data Manipulation in List

- To find index of any element
  - print(temp)
  - index = temp.index(9)
  - print("Index of element 9 is :", index)



```
In [9]:  print(temp)

           [1, 2, 3, 9, 4, 5, 10]

In [10]: index = temp.index(9)
         print("Index of element 9 is :", index)

           Index of element 9 is : 3
```

- Nested lists
  - nested = [[1,2,3],[4,5,6],[7,8,9]]
  - print("Nested List is :",nested)



```
In [19]: nested = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

In [20]: print("Nested List is :", nested)

           Nested List is : [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

## Data Manipulation in List

proschool
An IMS Initiative

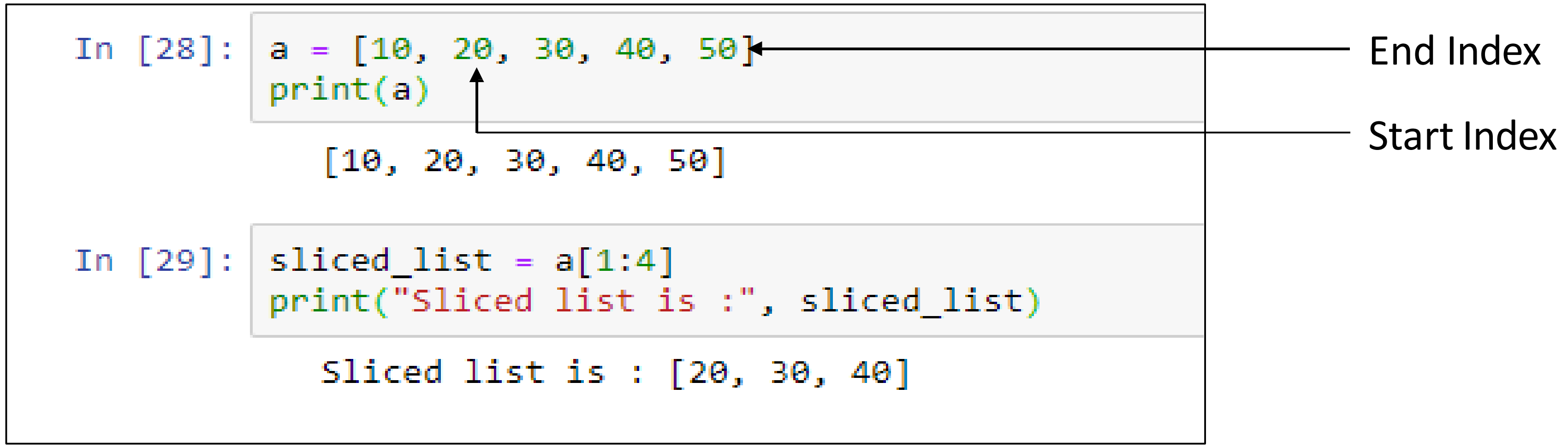
- **Slicing a List**

Slicing is used when you want to select a specific subset of a list. In Python, a slice is represented by " : ".
Syntax to slice a list is : list_name[start_index : end_index]

But hold on!!! There's a catch.

When you slice a list in Python, it includes the element on **start_index(20)** but excludes the element on **end_index(50)**. Instead, it slices the list till the element before end_index.

It works like this:- list_name[index_included : index_excluded]

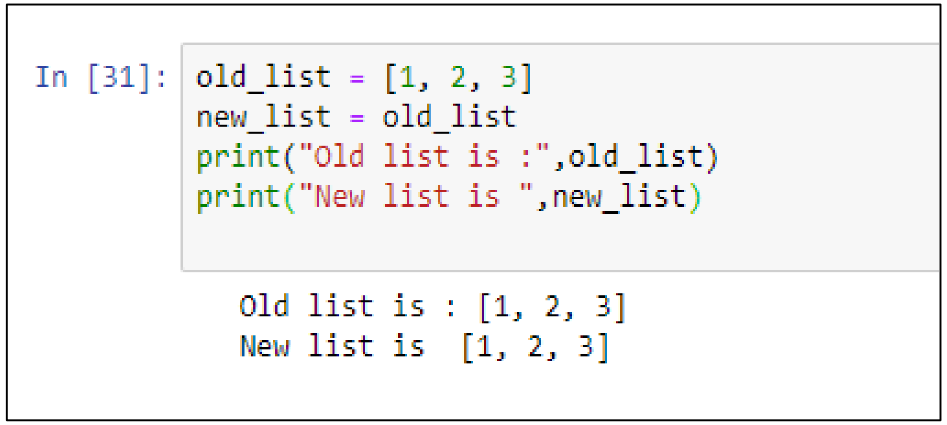


```
In [28]: a = [10, 20, 30, 40, 50]
         print(a)

         [10, 20, 30, 40, 50]

In [29]: sliced_list = a[1:4]
         print("Sliced list is :", sliced_list)

         Sliced list is : [20, 30, 40]
```

## Copying a list

A list can be copied with ”=“ operator. For example:

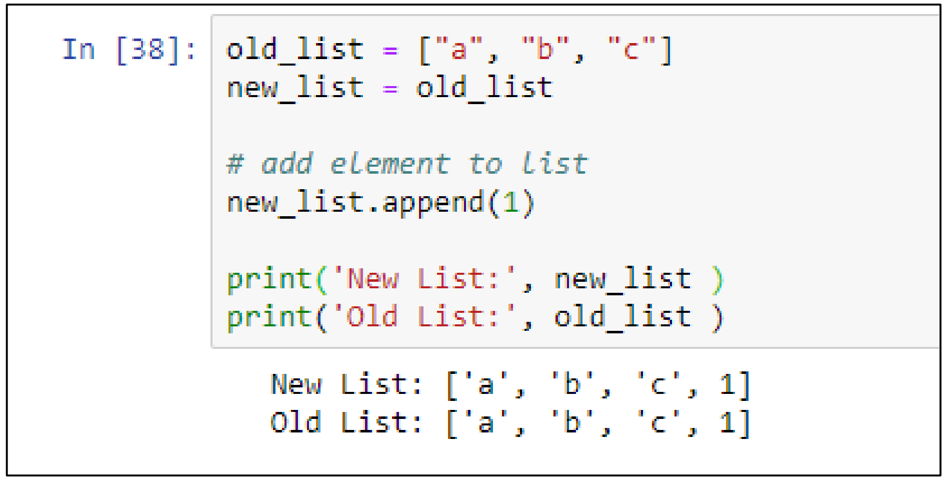
```
old_list = [1, 2, 3]
new_list = old_list
print(“Old list is :”, old_list)
print(“New List is :”, new_list)
```



```
In [31]: old_list = [1, 2, 3]
         new_list = old_list
         print("Old list is :",old_list)
         print("New list is ",new_list)

          Old list is : [1, 2, 3]
          New list is  [1, 2, 3]
```

The problem with copying the list in this way is that if you modify the new_list, the old_list is also modified.

```
old_list = [“a”, “b”, “c”]
new_list = old_list
# add element to list
new_list.append(1)
print('New List:', new_list )
print('Old List:', old_list )
```



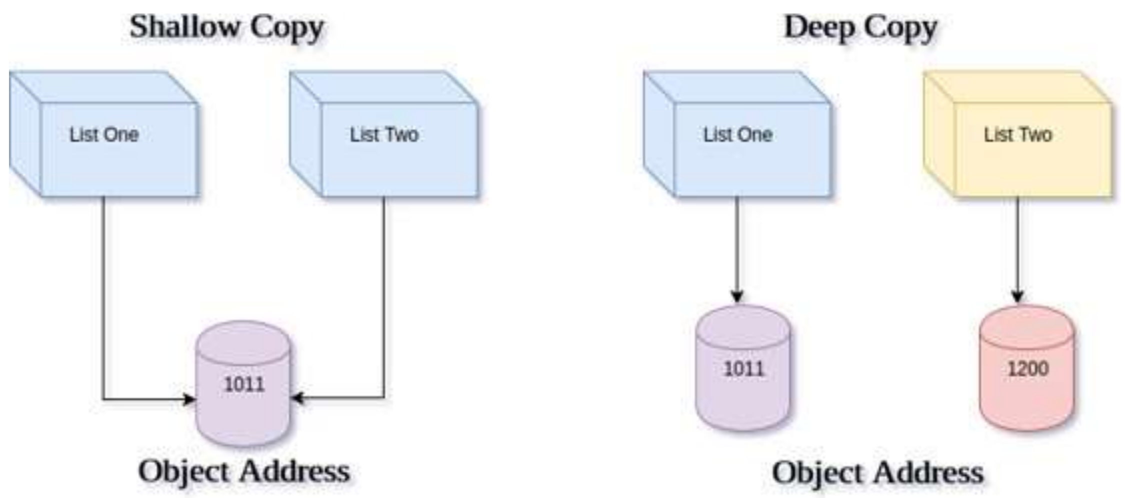
```
In [38]: old_list = ["a", "b", "c"]
         new_list = old_list

         # add element to list
         new_list.append(1)

         print('New List:', new_list )
         print('Old List:', old_list )

          New List: ['a', 'b', 'c', 1]
          Old List: ['a', 'b', 'c', 1]
```

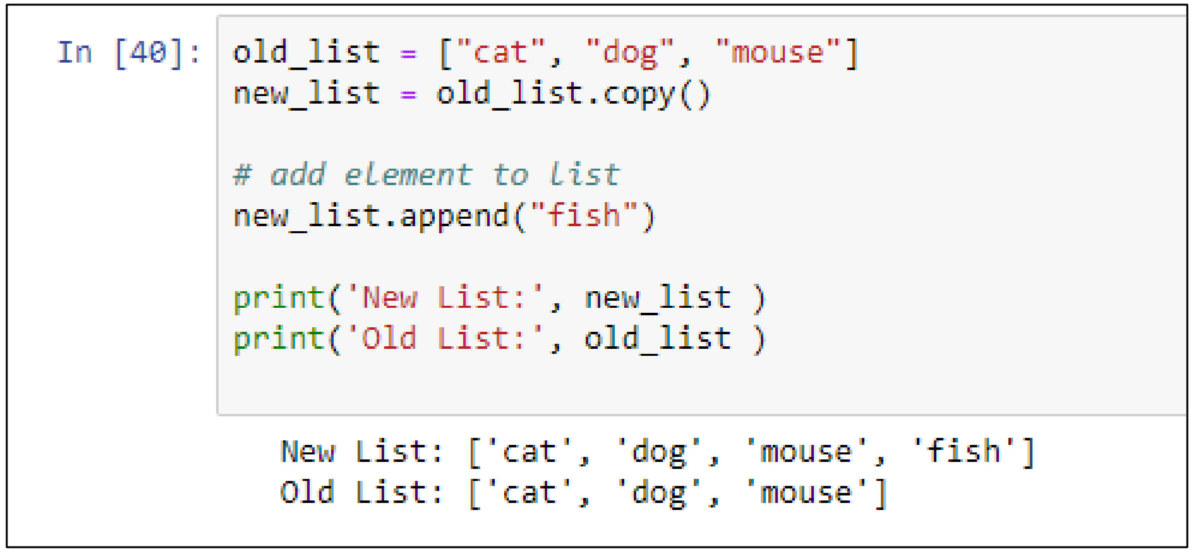
## Copying a list

This happens because when you copy old_list to new_list using “ = ” symbol, instead of allocating memory to new_list, Python creates a reference of new_list to old_list. This is called Shallow copy.

However, if you need the original list unchanged when the new list is modified, you can use copy() method. This is called deep copy.

old_list = [“cat”, “dog”, “mouse”]
new_list = old_list.copy()
# add element to list
new_list.append(“fish”)
print('New List:', new_list )
print('Old List:', old_list )



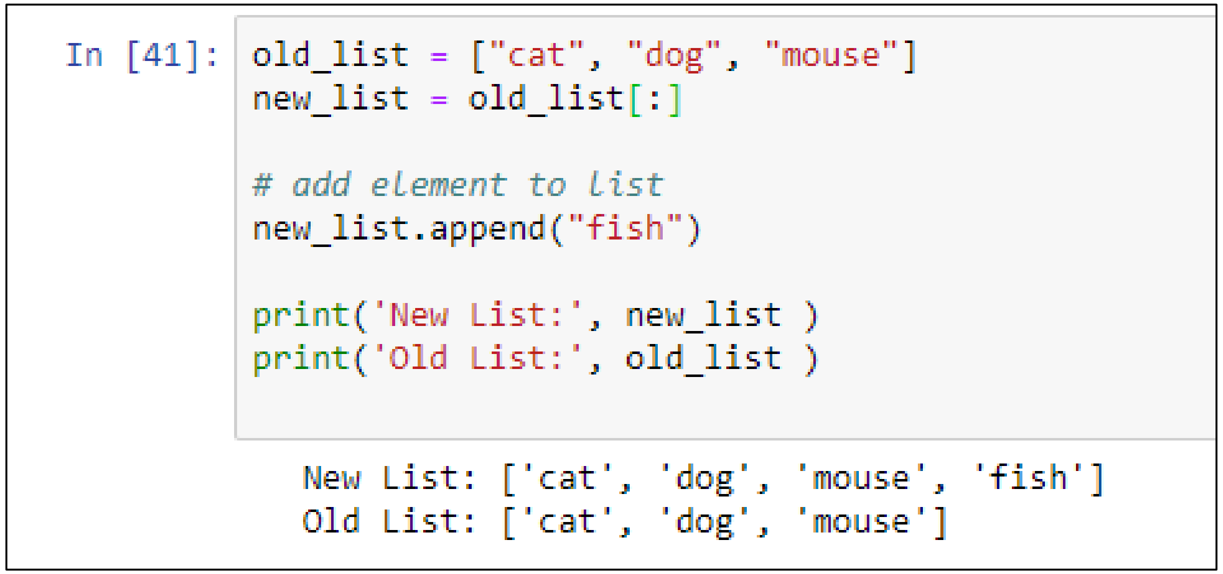


```
In [40]: old_list = ["cat", "dog", "mouse"]
         new_list = old_list.copy()

         # add element to list
         new_list.append("fish")

         print('New List:', new_list )
         print('Old List:', old_list )

    New List: ['cat', 'dog', 'mouse', 'fish']
    Old List: ['cat', 'dog', 'mouse']
```

## Copying a list

You can also achieve the same result using slicing as follows:

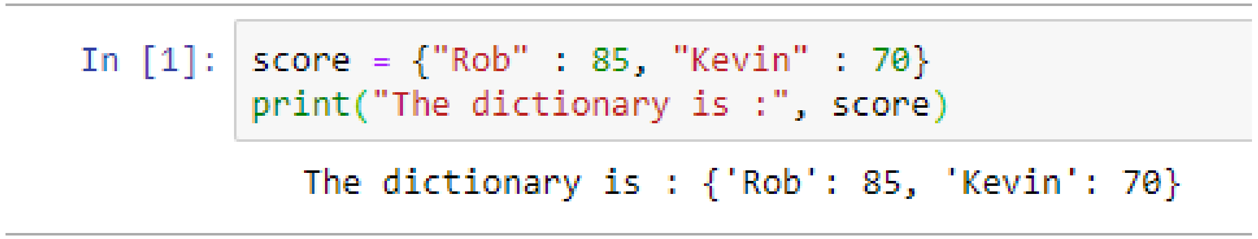
```
old_list = [“cat”, “dog”, “mouse”]
new_list = old_list[:]
# add element to list
new_list.append(“fish”)
print('New List:', new_list )
print('Old List:', old_list )
```



```
In [41]: old_list = ["cat", "dog", "mouse"]
         new_list = old_list[:]

         # add element to list
         new_list.append("fish")

         print('New List:', new_list )
         print('Old List:', old_list )

            New List: ['cat', 'dog', 'mouse', 'fish']
            Old List: ['cat', 'dog', 'mouse']
```

## Dictionary

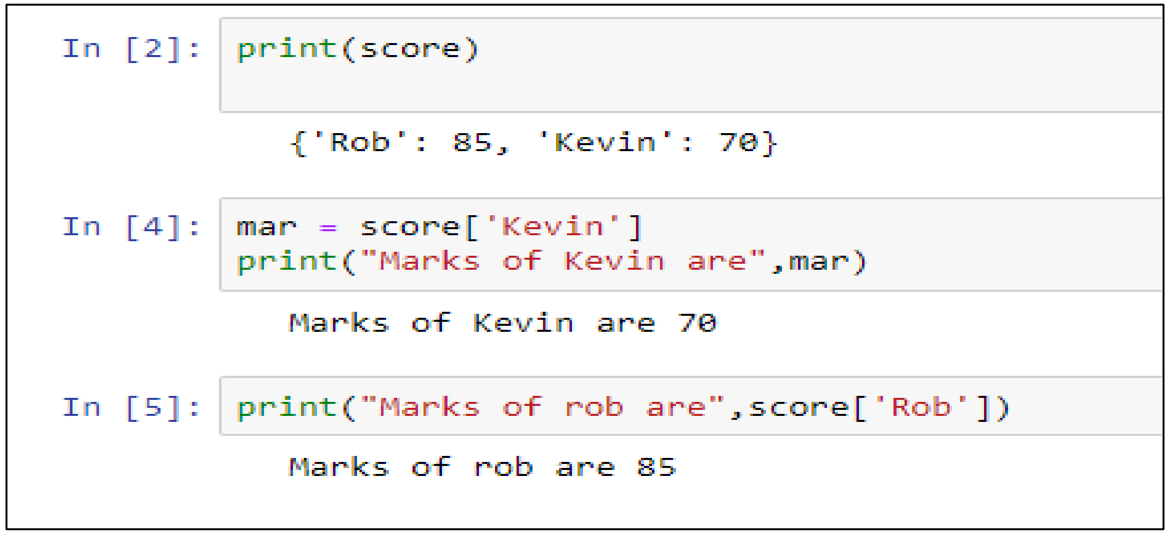
- The other main data type is the dictionary. The dictionary allows you to associate one piece of data (a "key") with another (a "value"). The analogy comes from real-life dictionaries, where we associate a word (the "key") with its meaning. It's a little harder to understand than a list, but Python makes them very easy to deal with.
- It is best to think of a dictionary as an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary).
- A pair of braces creates an empty dictionary:{}
- Placing a comma-separated list of key : value pairs within the braces adds initial key : value pairs to the dictionary; this is also the way dictionaries are written on output.
- Declaration of dictionary
  - Lets say we would like to create an object which stores the name and scores of students. This could be achieved with dictionary in **keys : values** format
- score = {“Rob" : 85, “Kevin" : 70}
- print("The dictionary is :",score)



```
In [1]: score = {"Rob" : 85, "Kevin" : 70}
        print("The dictionary is :", score)

        The dictionary is : {'Rob': 85, 'Kevin': 70}
```

## Dictionary

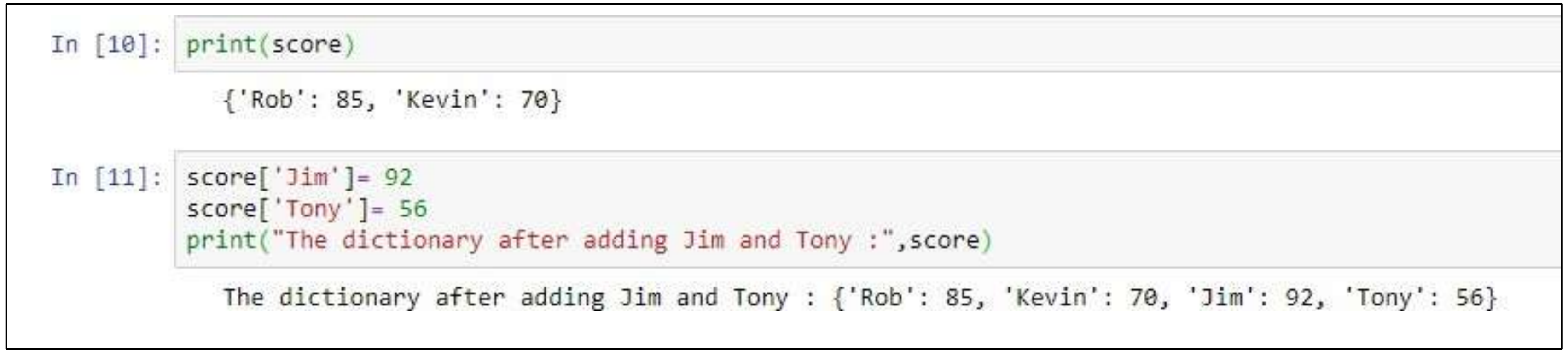
- To get the score of any student or value in dictionary
  - print(score)
  - mar = score['Kevin']
  - print("Marks of Kevin are",mar)
  - print("Marks of rob are",score['Rob'])     # print the score of rob



```
In [2]: print(score)

        {'Rob': 85, 'Kevin': 70}

In [4]: mar = score['Kevin']
        print("Marks of Kevin are",mar)

        Marks of Kevin are 70

In [5]: print("Marks of rob are",score['Rob'])

        Marks of rob are 85
```

## Dictionary

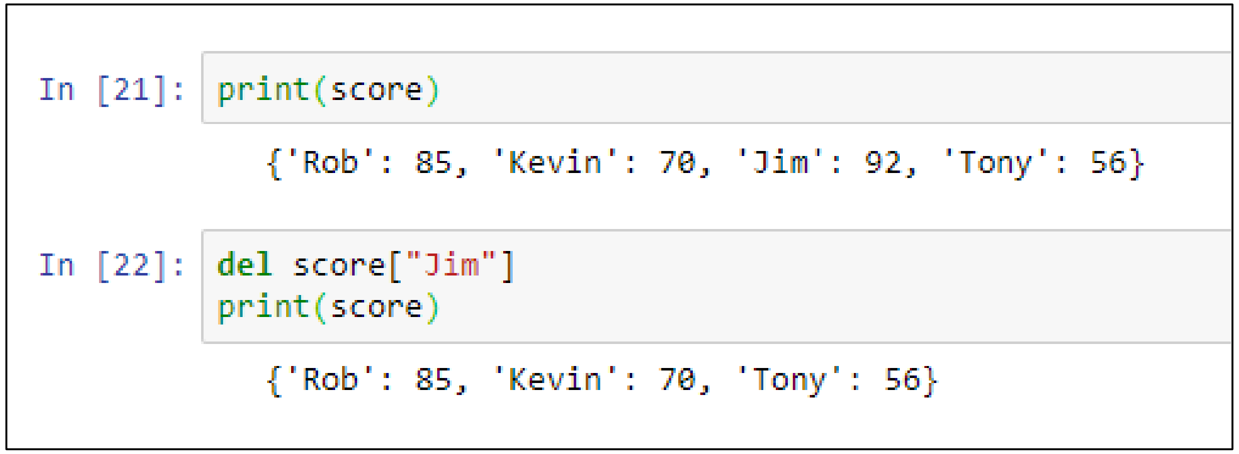
- Add new elements or students to the dictionary
  - print(score)
  - score['Jim']=92
  - score['Tony']=56
  - print("The dictionary after adding Jim and Tony -",score)



```
In [10]: print(score)

           {'Rob': 85, 'Kevin': 70}

In [11]: score['Jim']= 92
         score['Tony']= 56
         print("The dictionary after adding Jim and Tony :",score)

           The dictionary after adding Jim and Tony : {'Rob': 85, 'Kevin': 70, 'Jim': 92, 'Tony': 56}
```

## Dictionary

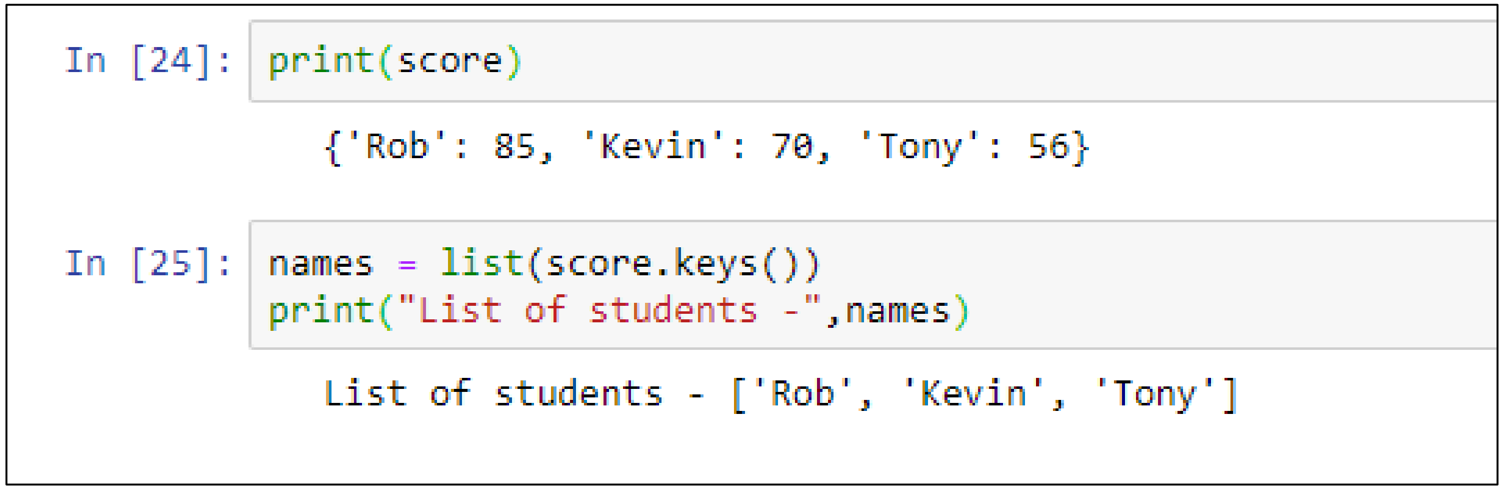
- Delete elements from a dictionary
  - print(score)
  - del score["Jim"]
  - print(score)



```
In [21]: print(score)

         {'Rob': 85, 'Kevin': 70, 'Jim': 92, 'Tony': 56}

In [22]: del score["Jim"]
         print(score)

         {'Rob': 85, 'Kevin': 70, 'Tony': 56}
```

# Dictionary

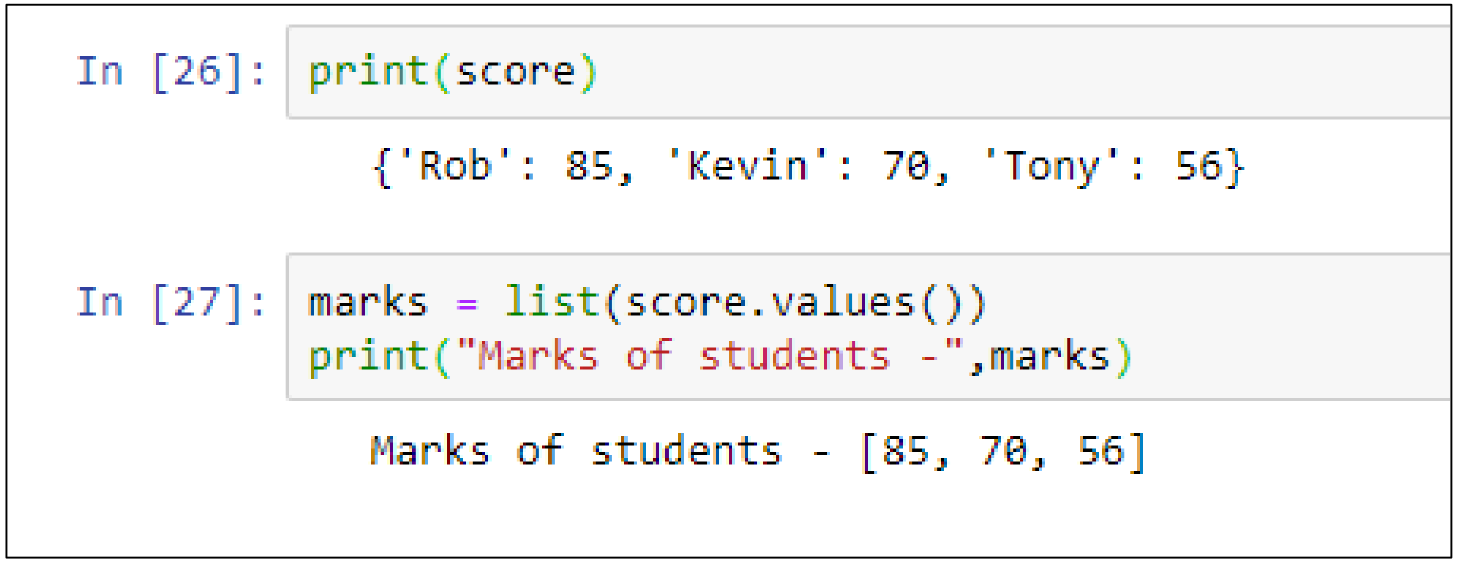
- "keys" gives you keys in dictionary, i.e. names of all students in this case
  - print(score)
  - names = list(score.keys())
  - print("List of students -",names)



```
In [24]: print(score)

    {'Rob': 85, 'Kevin': 70, 'Tony': 56}

In [25]: names = list(score.keys())
         print("List of students -",names)

    List of students - ['Rob', 'Kevin', 'Tony']
```

## Dictionary

- “values” gives you values in dictionary, i.e. marks of all students in this case
  - print(score)
  - marks = list(score.values())
  - print("List of marks -",marks)

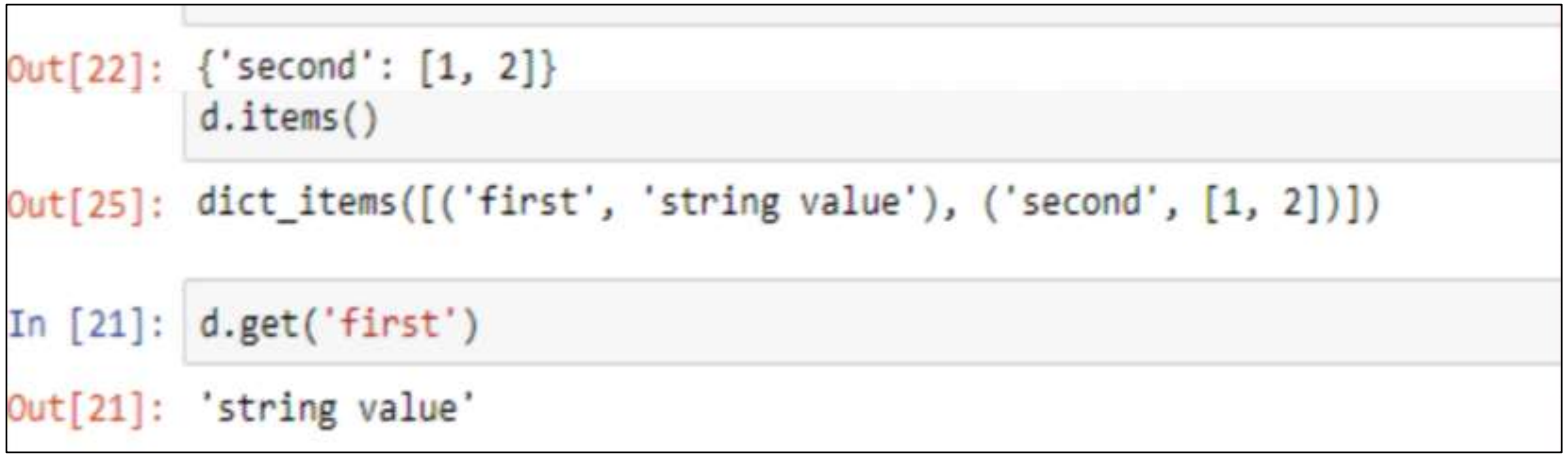


```
In [26]: print(score)

         {'Rob': 85, 'Kevin': 70, 'Tony': 56}

In [27]: marks = list(score.values())
         print("Marks of students -",marks)

         Marks of students - [85, 70, 56]
```

## Dictionary

In addition to keys and values methods, there is also the items method that returns a list of items of the form (key, value). The items are not returned in any particular order.

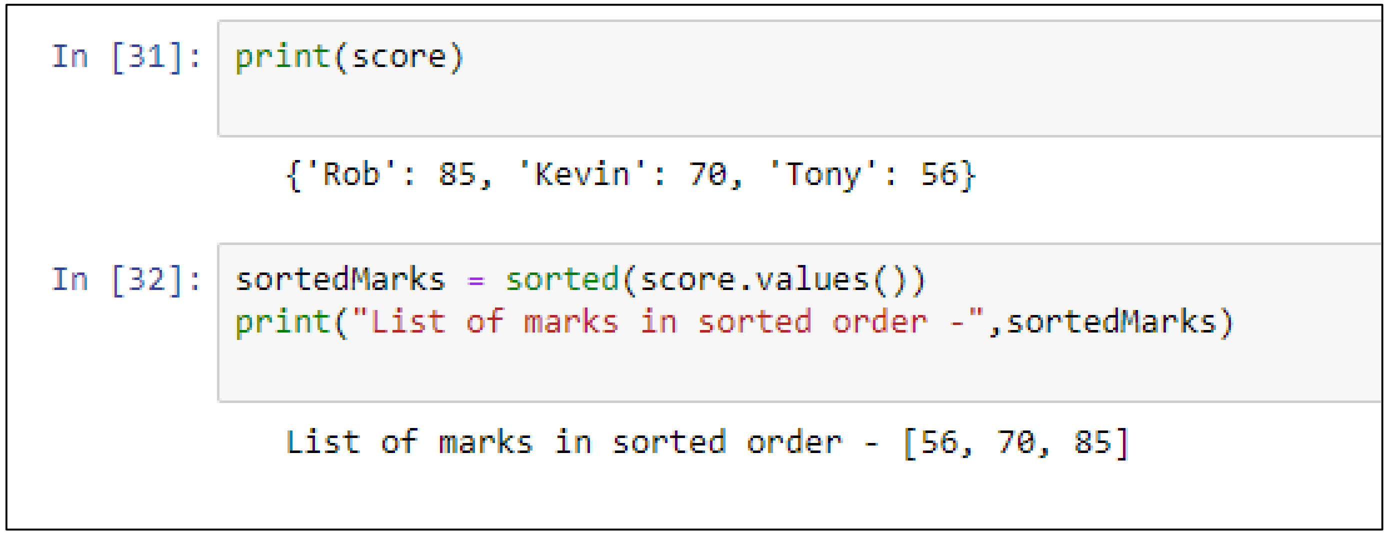
In order to get the value corresponding to a specific key, use get :



```
Out[22]: {'second': [1, 2]}
         d.items()

Out[25]: dict_items([('first', 'string value'), ('second', [1, 2])])

In [21]: d.get('first')

Out[21]: 'string value'
```

## Dictionary

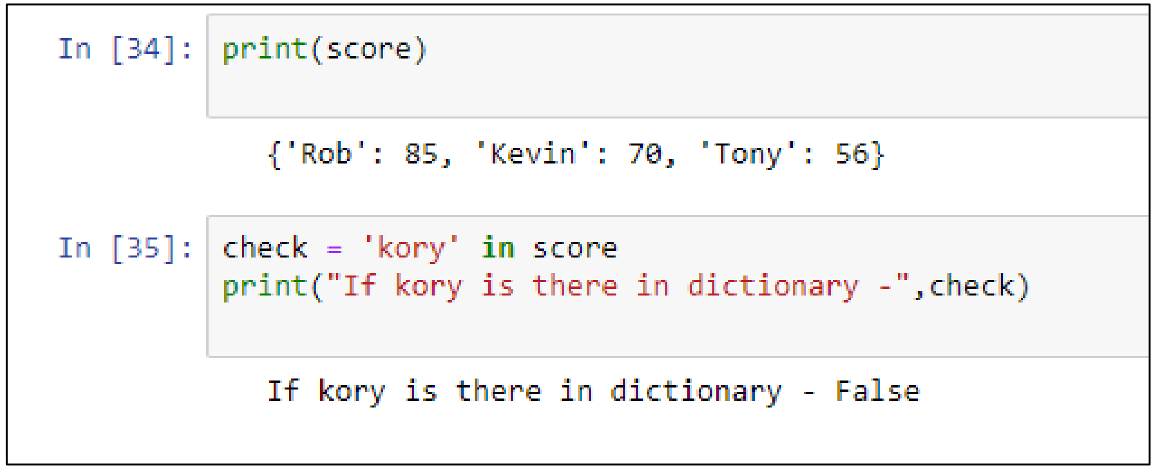
- We can also look at the marks in sorted order i.e. non descending order
  - print(score)
  - sortedMarks = sorted(score.values())
  - print("List of marks in sorted order -",sortedMarks)



```
In [31]: print(score)

         {'Rob': 85, 'Kevin': 70, 'Tony': 56}

In [32]: sortedMarks = sorted(score.values())
         print("List of marks in sorted order -",sortedMarks)

         List of marks in sorted order - [56, 70, 85]
```

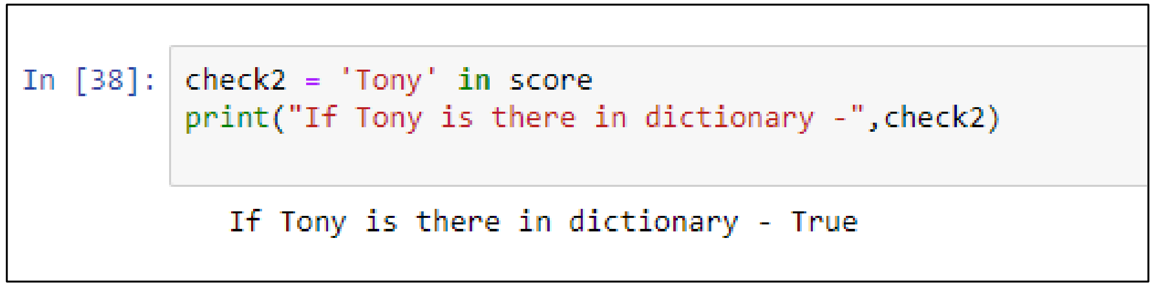
## Dictionary

- Checks if a particular student name is there in the dictionary
  - print(score)
  - check = 'kory' in score
  - print("If kory is there in dictionary -",check)



```
In [34]: print(score)

         {'Rob': 85, 'Kevin': 70, 'Tony': 56}

In [35]: check = 'kory' in score
         print("If kory is there in dictionary -",check)

         If kory is there in dictionary - False
```

- check2 = 'Tony' in score
  print("If Tony is there in dictionary -",check2)



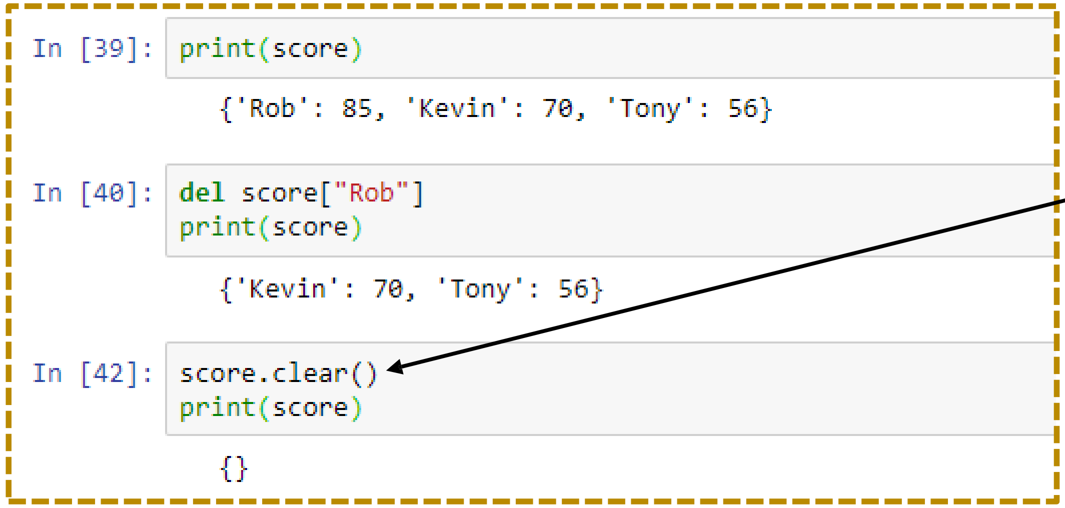
```
In [38]: check2 = 'Tony' in score
         print("If Tony is there in dictionary -",check2)

         If Tony is there in dictionary - True
```

## Dictionary

You can **clear** a dictionary (i.e., remove all its items) using the clear() method:

-> score.clear()

The clear() method **deletes** all items whereas **del**() deletes just one:



```
In [39]: print(score)

         {'Rob': 85, 'Kevin': 70, 'Tony': 56}

In [40]: del score["Rob"]
         print(score)

         {'Kevin': 70, 'Tony': 56}

In [42]: score.clear()
         print(score)

         {}
```

All the values of “score” is cleared by the function “clear”

## Conditional Statements

- We use conditional statements for checking the condition of the statement and to change its behavior.
- Some of the conditional statements are as follows…

1. if
2.if else
3.Nested if else

Now lets discuss about If statement.

## 1. If statement

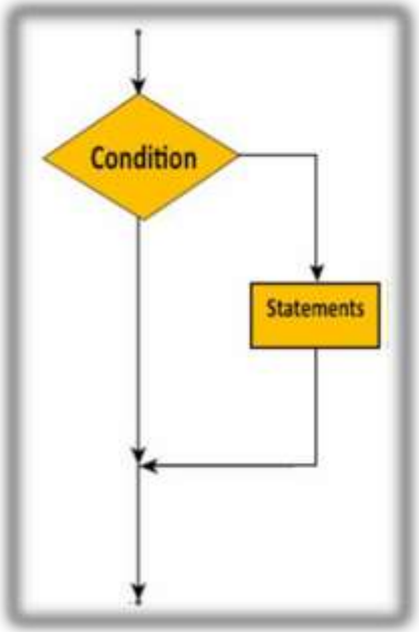
Syntax for if statement is as follows

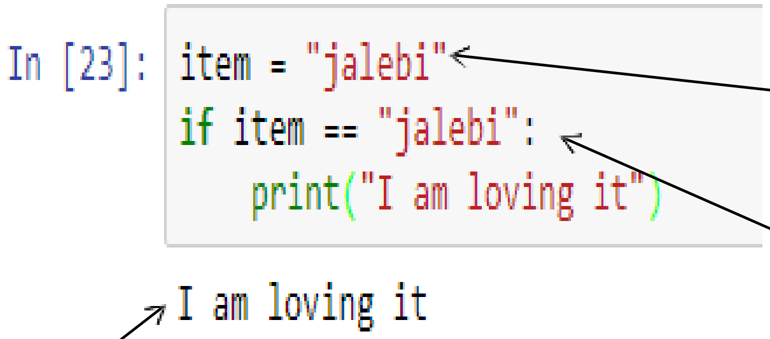
```
if expression:
        statement(s)
```

## If statement

- A flowchart is given for better understanding ..

- The boolean expression after the if statement is called the **condition**. If it is true, then all the given statements are executed . Otherwise no
- Example as shown below.





```
In [23]: item = "jalebi"
         if item == "jalebi":
             print("I am loving it")

I am loving it
```

Here there is an expression "=" which tells that variable is assigned some value.

Here there is an expression "==" which tells that there is a condition

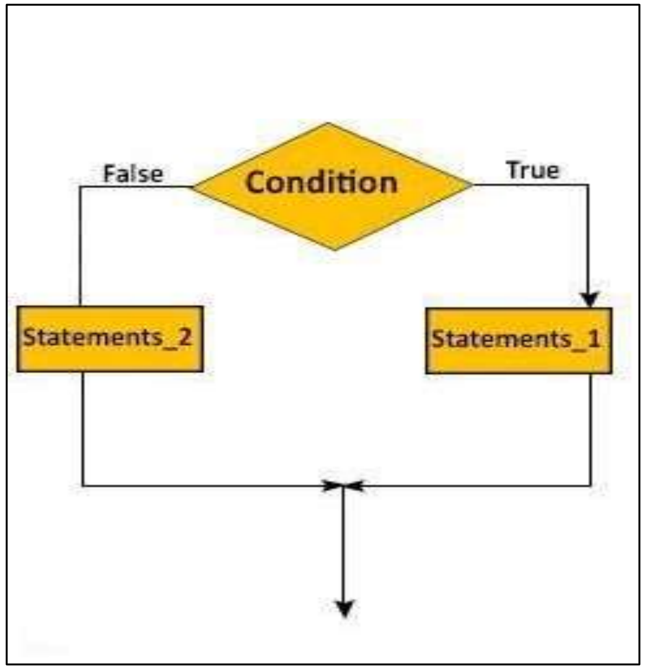
The statement is executed as the condition was true.

## 2.If Else

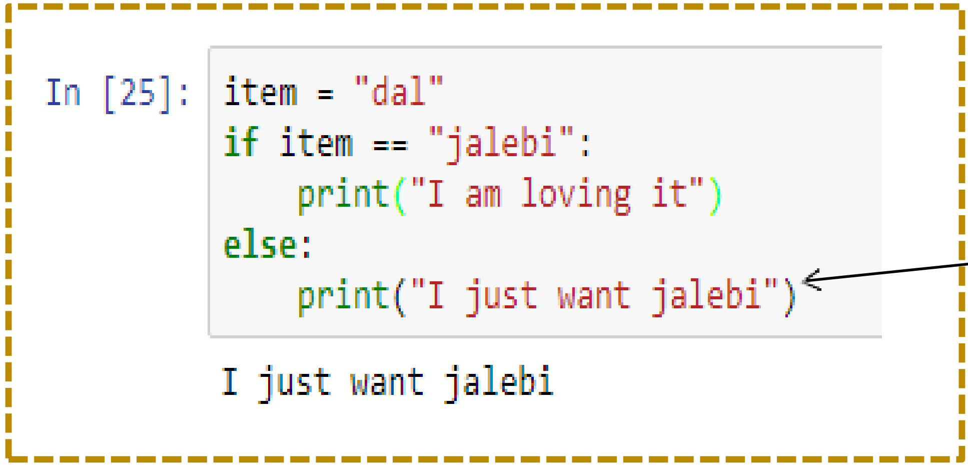
- One thing to happen when a condition it true, otherwise **else** to happen when it is false.
- Syntax:

```
if expression:
        statement(s)
else:
        statement(s)
```

- If the given condition is true then statement one will display other wise second statement

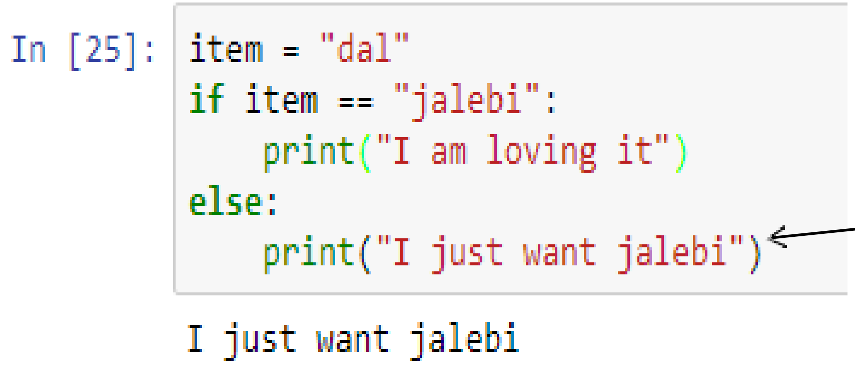


Example

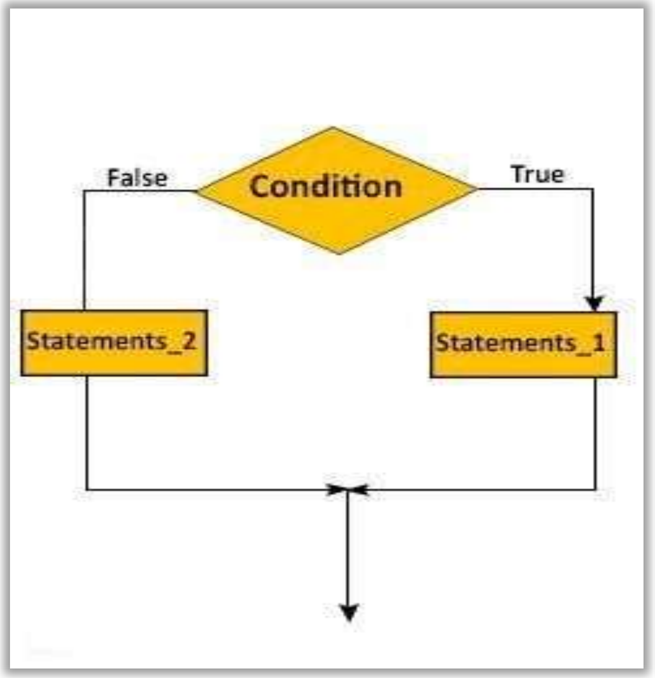


```
In [25]: item = "dal"
         if item == "jalebi":
             print("I am loving it")
         else:
             print("I just want jalebi")

         I just want jalebi
```

Here the “if” condition hasn’t met, so “else” statement has been executed



```
In [25]: item = "dal"
         if item == "jalebi":
             print("I am loving it")
         else:
             print("I just want jalebi")

         I just want jalebi
```

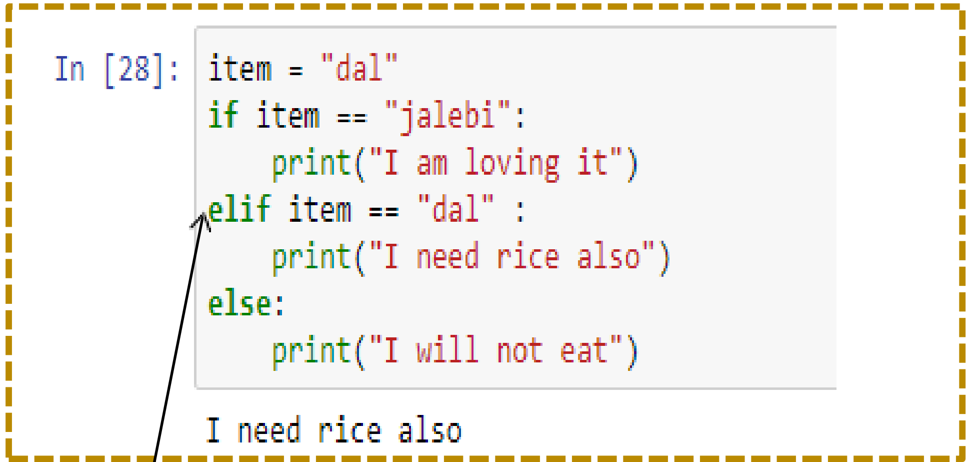


Here the “if” condition hasn’t met, so “else” statement has been executed

## 3. Nested If Else

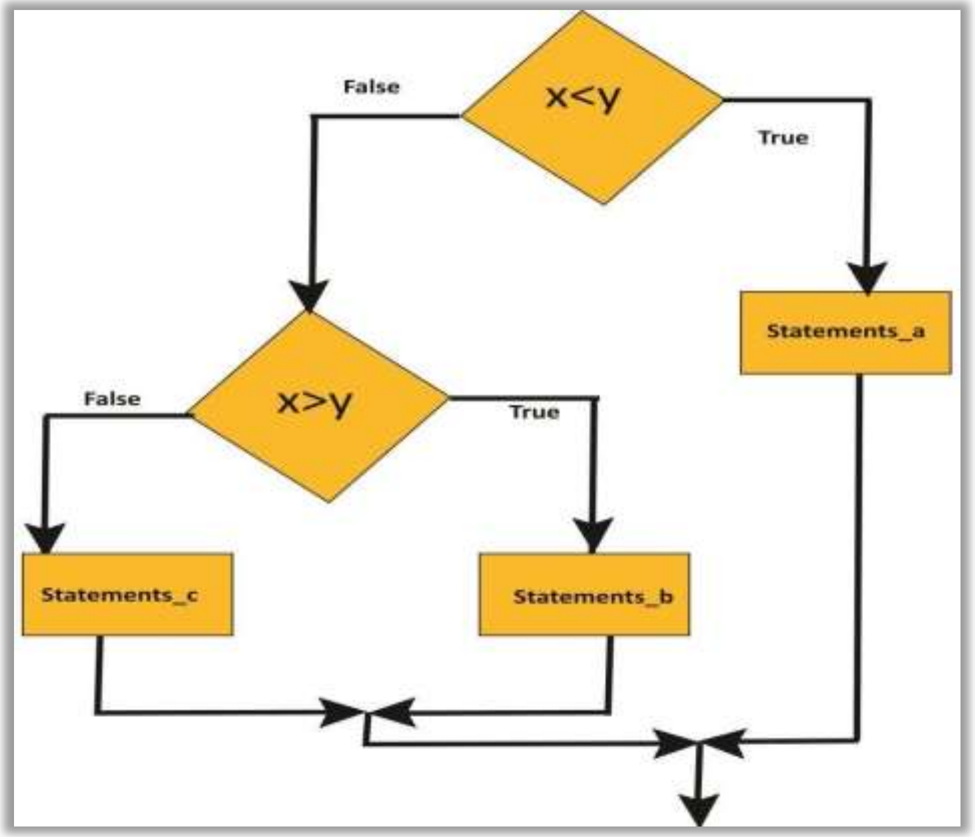
- If the first condition is true( x<y), it displays the first one. If it is false, then checks second condition (x>y) if it is true then displays second statement . If the second condition is also not satisfied, then it displays the third statement.

- Example :



```
In [28]: item = "dal"
         if item == "jalebi":
             print("I am loving it")
         elif item == "dal" :
             print("I need rice also")
         else:
             print("I will not eat")

         I need rice also
```

To use nested else if in python, we have a key word ”elif”. It will be executed after “if”, and before “else”



## Loops

- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. It is a continuous process.
- Syntax:

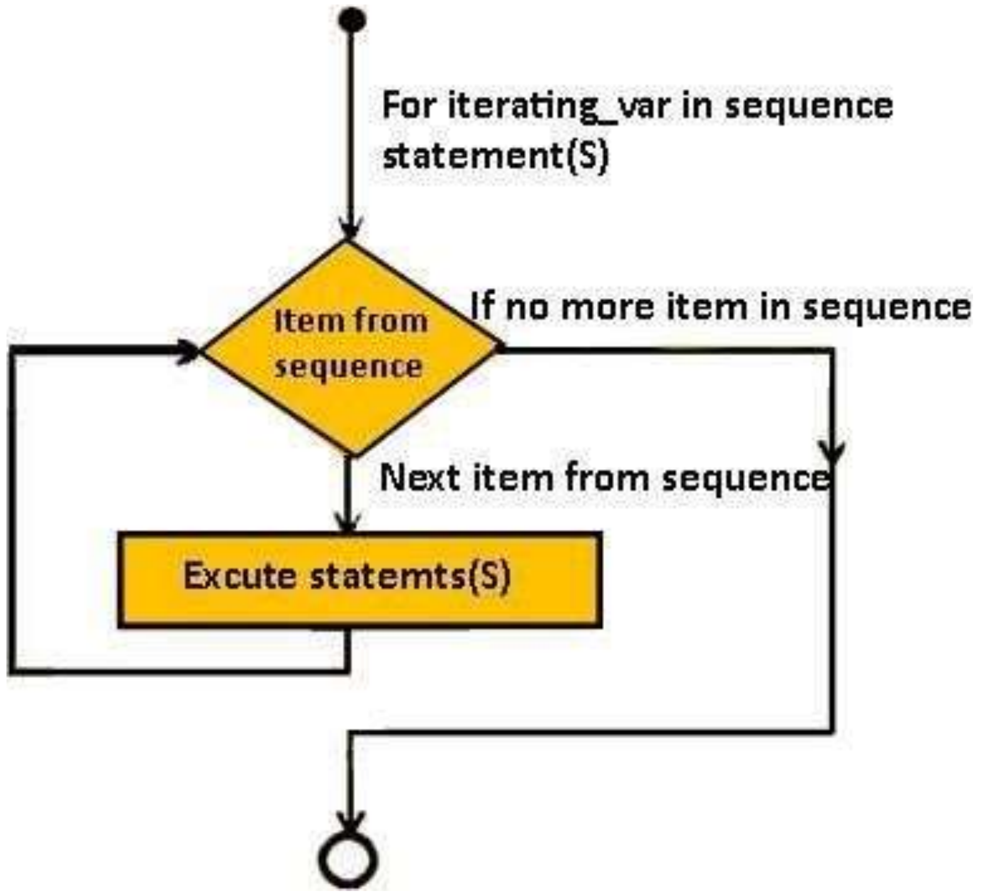
  –for iterating_var in sequence:

  ```
  for iterating_var in sequence:
          statements(s)
  ```

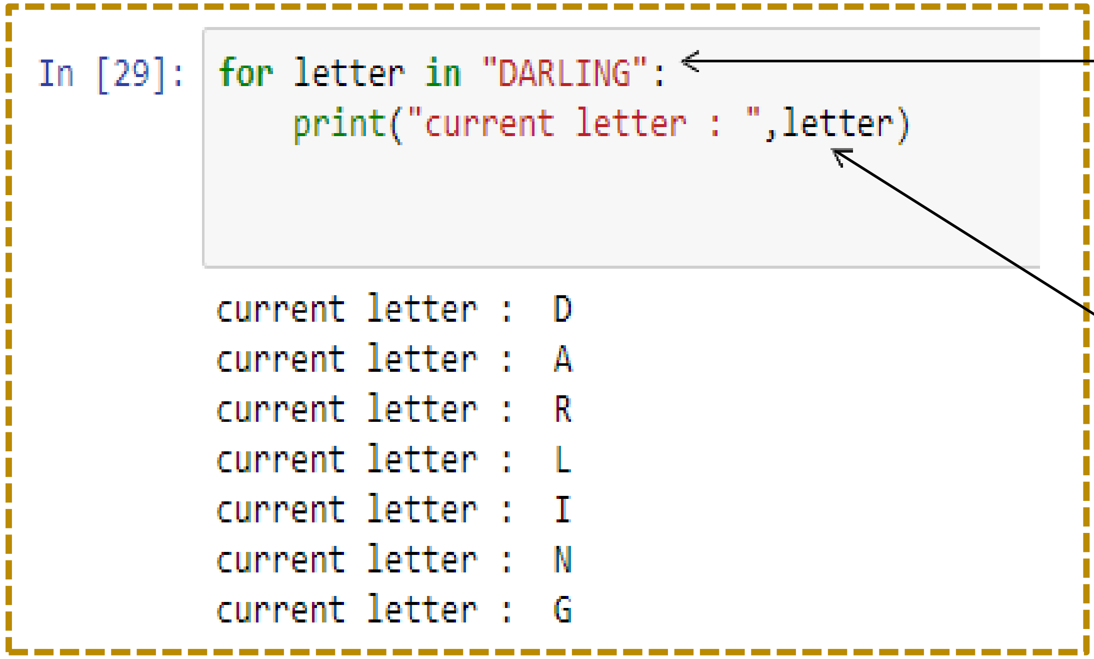
- The below flow chart gives how for loop works

## For Loop

- Each iteration assigns the loop variable to the next element in the sequence, and then executes the statements in the body. Statement will be completed if it reaches the last element in the given sequence.



- Example :



```
In [29]: for letter in "DARLING":
             print("current letter : ",letter)

         current letter :  D
         current letter :  A
         current letter :  R
         current letter :  L
         current letter :  I
         current letter :  N
         current letter :  G
```

To use “for “ loop, we have to use keyword “in”

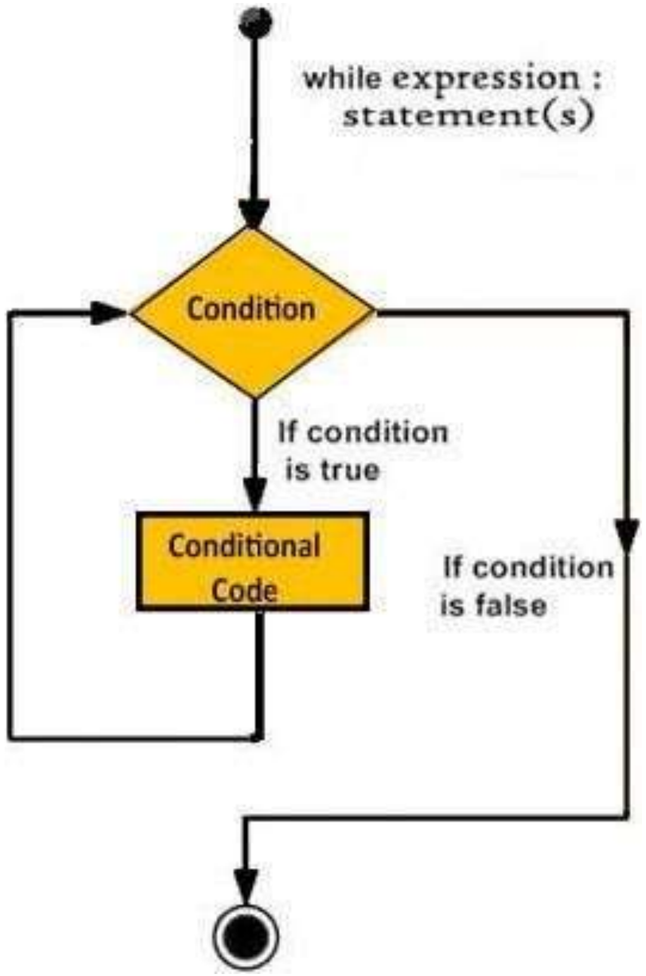
Under variable “letter”, every letter will pick one by one in each loop.

## While Loop

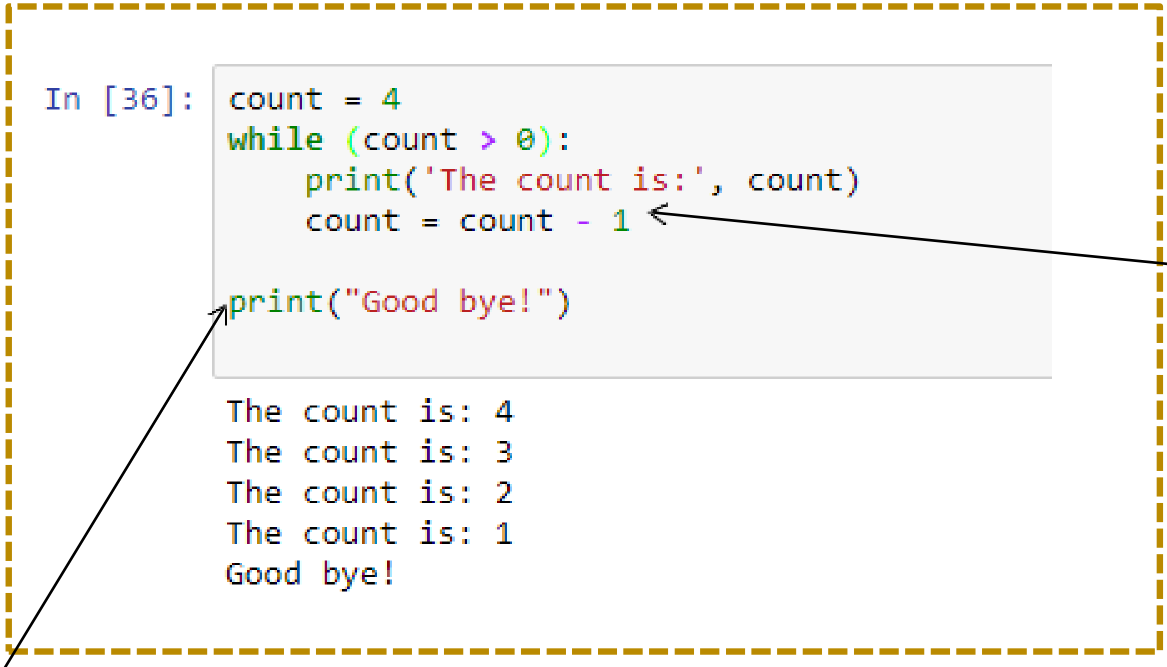
- while loop executes repeatedly the target statement as long as a the given condition becomes true.
- Syntax
  - –while expression:

            statement(s)

•Flow chart gives how does the while loop works.



- Example



```
In [36]: count = 4
         while (count > 0):
             print('The count is:', count)
             count = count - 1

         print("Good bye!")
```

```
The count is: 4
The count is: 3
The count is: 2
The count is: 1
Good bye!
```

"while" loop continues or repeats its conditions until its last statement become true. Here loop will run, until "count" will become equal to or less than zero.

Last "print" statement will be executed out of "while" loop because it is not have four space indentation

## 3.7 Suite

3.7. Multiple statement groups as suites

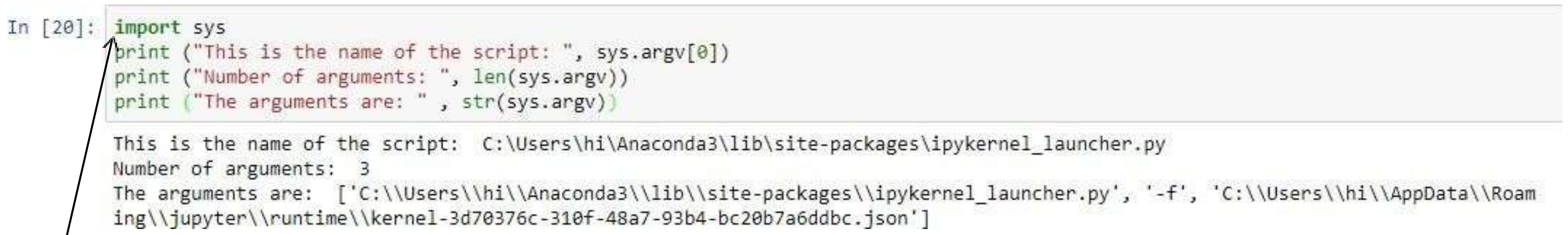
A compound statement consists of one or more 'clauses.' A clause consists of a header and a 'suite.' The clause headers of a particular compound statement are all at the same indentation level. Each clause header begins with a uniquely identifying keyword and ends with a colon. **A suite is a group of statements controlled by a clause.** A suite can be one or more semicolon-separated simple statements on the same line as the header, following the header's colon, or it can be one or more indented statements on subsequent lines.

```
if_stmt ::= "if" expression ":" suite
                 ( "elif" expression ":" suite )*
                 ["else" ":" suite]
```

If condition is met, then clause is executed Each such clause is suite.

## 3.8. Command Line Arguments

An argument sent to a program being called. More than one command lines are accepted by the program.



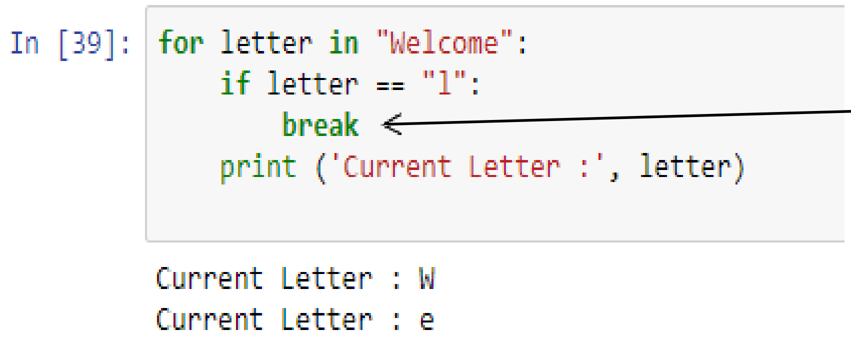
```
In [20]: import sys
         print ("This is the name of the script: ", sys.argv[0])
         print ("Number of arguments: ", len(sys.argv))
         print ("The arguments are: " , str(sys.argv))

         This is the name of the script:  C:\Users\hi\Anaconda3\lib\site-packages\ipykernel_launcher.py
         Number of arguments:  3
         The arguments are:  ['C:\\Users\\hi\\Anaconda3\\lib\\site-packages\\ipykernel_launcher.py', '-f', 'C:\\Users\\hi\\AppData\\Roam
         ing\\jupyter\\runtime\\kernel-3d70376c-310f-48a7-93b4-bc20b7a6ddbc.json']
```

sys.argv is a list in Python, which contains the command-line arguments passed to the script.
With the len(sys.argv) function you can count the number of arguments.
If you are going to work with command line arguments, you probably want to use sys.argv.
To use sys.argv, you will first have to import the **sys module**.

## break Statement

- It terminates the current **loop** and resumes execution at the next **statement**.

•Example:

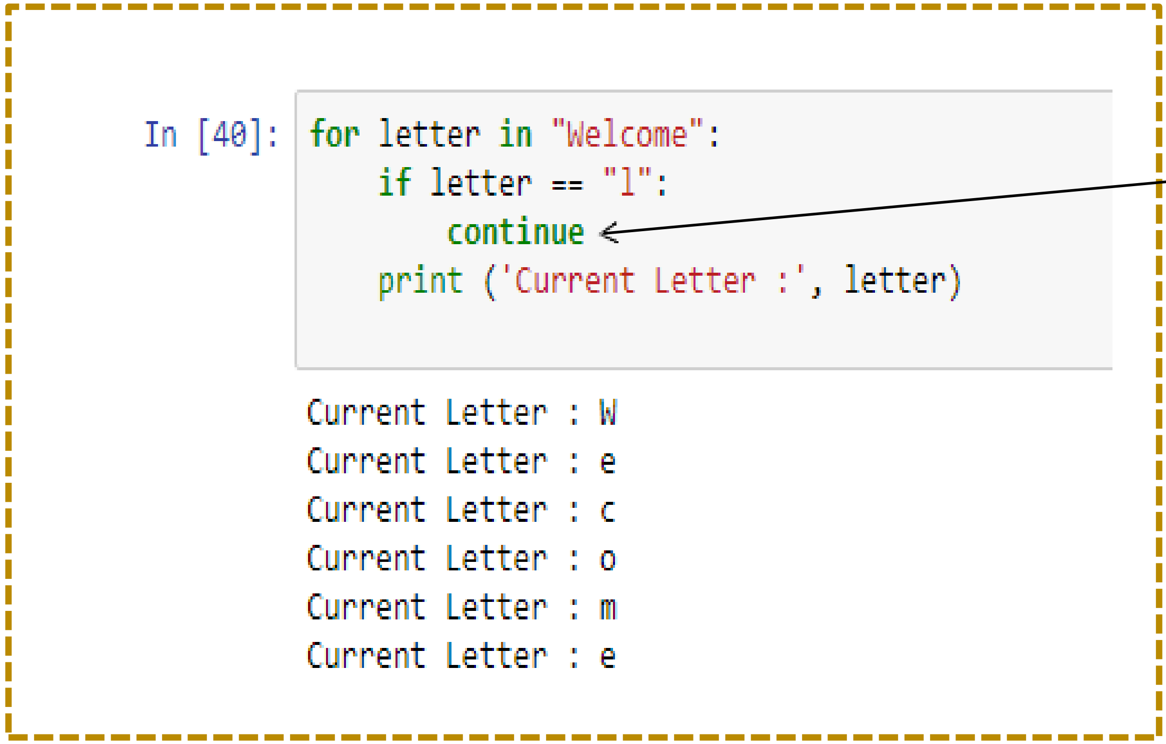


```
In [39]: for letter in "Welcome":
             if letter == "l":
                 break
             print ('Current Letter :', letter)

         Current Letter : W
         Current Letter : e
```

"break" statement breaks the loop if "if" condition comes true.

## continue Statement

- It returns the control to the beginning of the while loop.. The **continue statement** rejects all the remaining **statements** in the current iteration of the loop and moves the control back to the first of the loop.



```
In [40]: for letter in "Welcome":
             if letter == "l":
                 continue
             print ('Current Letter :', letter)
```

```
Current Letter : W
Current Letter : e
Current Letter : c
Current Letter : o
Current Letter : m
Current Letter : e
```

"continue" statement allows the loop to skip the step, if "if" condition comes true.

## Functions and Methods

- Function is a collection of statements which is used to perform a specific task.
- A method is a function that takes a class instance as its first parameter
- *Syntax*

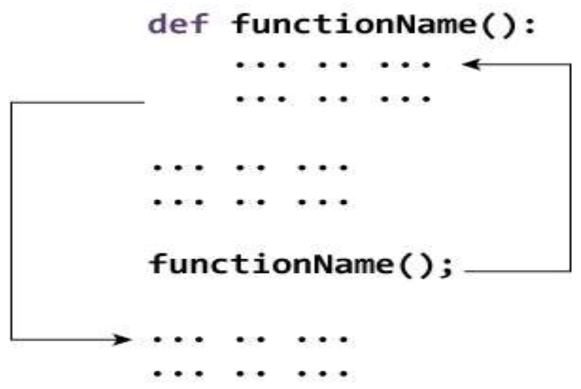
```
def function _name(parameters):
"""doc string"""
        statement(s)
```

- Function begins with *def* Keyword followed by function name and parentheses
- Function naming follows the same rules of writing Python identifiers.

## Functions and Methods

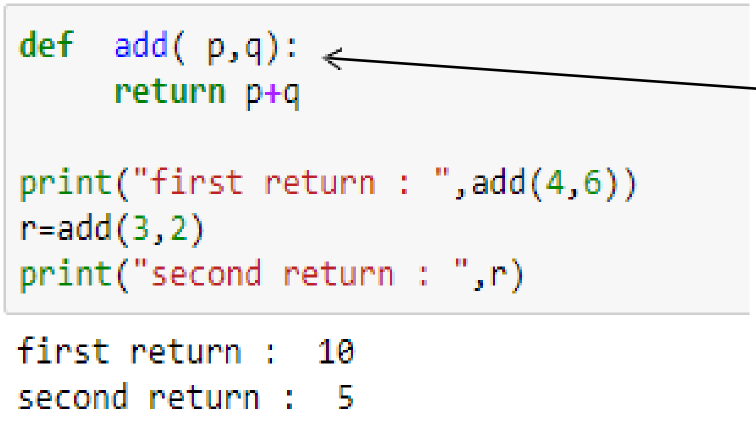
- We pass values through parameters or arguments and these are placed within the parenthesis.
- Colon is used to end the function header.
- Documentation string( *doc string* ) is used to explain what the function does.
- And optional return statement to return value from function.

*How Function works in Python*.



```
def functionName():
    ... .. ...
    ... .. ...

... .. ...
... .. ...

functionName();

... .. ...
... .. ...
```

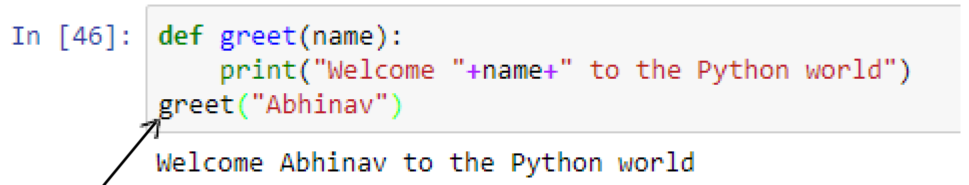
- Example 1:



```
In [43]: def  add( p,q):
              return p+q

         print("first return : ",add(4,6))
         r=add(3,2)
         print("second return : ",r)

         first return :  10
         second return :  5
```

"add" function is created, sum of two parameters is returned in this function

- Example 2:

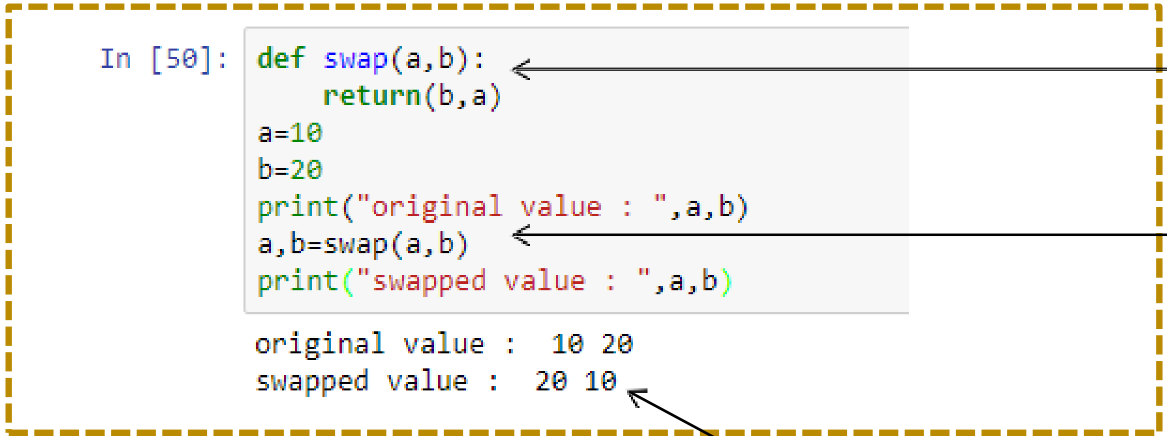


```
In [46]: def greet(name):
             print("Welcome "+name+" to the Python world")
         greet("Abhinav")

         Welcome Abhinav to the Python world
```

To call a function, simply use the function name with apt parameters or arguments. Here the apt parameter is name(string). If string variable would not be provided, error will come

## Function With parameters

- Parameters or arguments are used for passing values. Immutable objects cannot be changed inside the functions or outside. But mutable objects or opposite to Immutable objects, which can be changed outside the function.
- Example :



```
In [50]: def swap(a,b):
             return(b,a)
         a=10
         b=20
         print("original value : ",a,b)
         a,b=swap(a,b)
         print("swapped value : ",a,b)

         original value :  10 20
         swapped value :  20 10
```

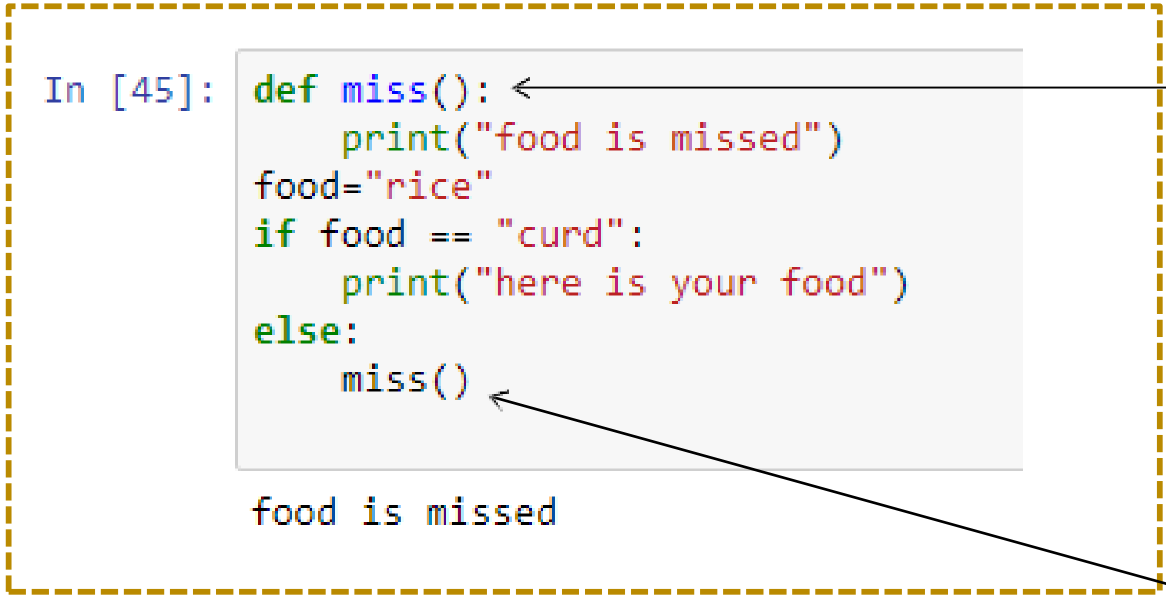
"swap" function is created, to swap the values of parameter.

Value is assigned to variables from "swap" function, which is returned by it

First print statement returned "original value" i.e. the value which is assigned initially.
After that swap statement swaps the value, so then b value is changed to a and vice versa.

## Function without parameter

There is no compulsion for a function to be associated with parameters or arguments.



```
In [45]: def miss():
             print("food is missed")
         food="rice"
         if food == "curd":
             print("here is your food")
         else:
             miss()

         food is missed
```

"miss" function is without parameter.
It will return a constant value which is kept under the function.

"miss" function is called here. It would be more practical to use, when we have to print similar statements multiple times. In such case, each time, we have to just call the function, in place of writing complete print statement.

- In functions there are four parameters they are as follows

1.Required Arguments or positional

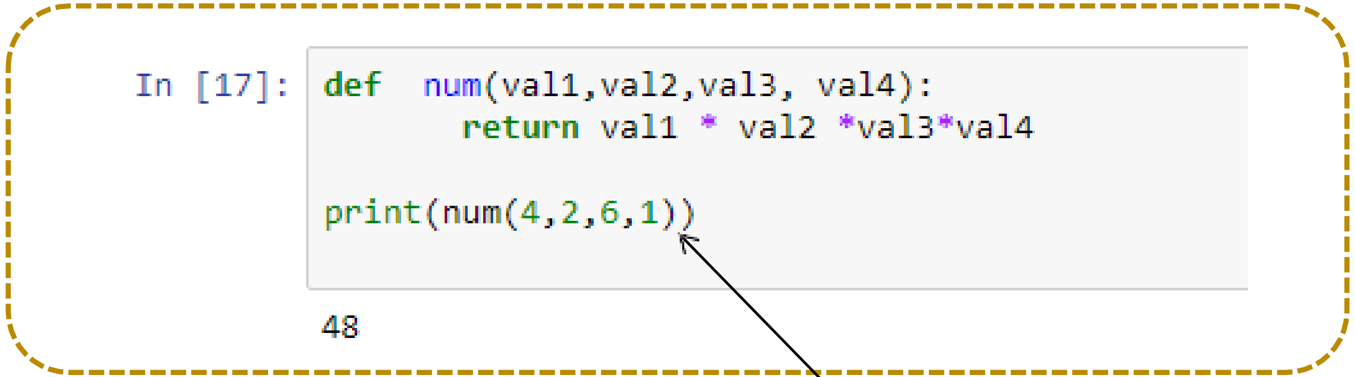
2 .Keyword Parameters

3. Default parameters

4. Collecting parameters

## 1.REQUIRED ARGUMENTS OR POSITIONAL  PARAMETERS

- The arguments or parameters which are in a positional format in a( parameters must be in a order) function is nothing but Positional Arguments or parameters. Parameters in function call will perfectly match with the function definition.
- Example



```
In [17]: def  num(val1,val2,val3, val4):
                return val1 * val2 *val3*val4

         print(num(4,2,6,1))

         48
```

Here we had given 4 positions so when we give values like 4,2,6,1 then they will take those values in a order.

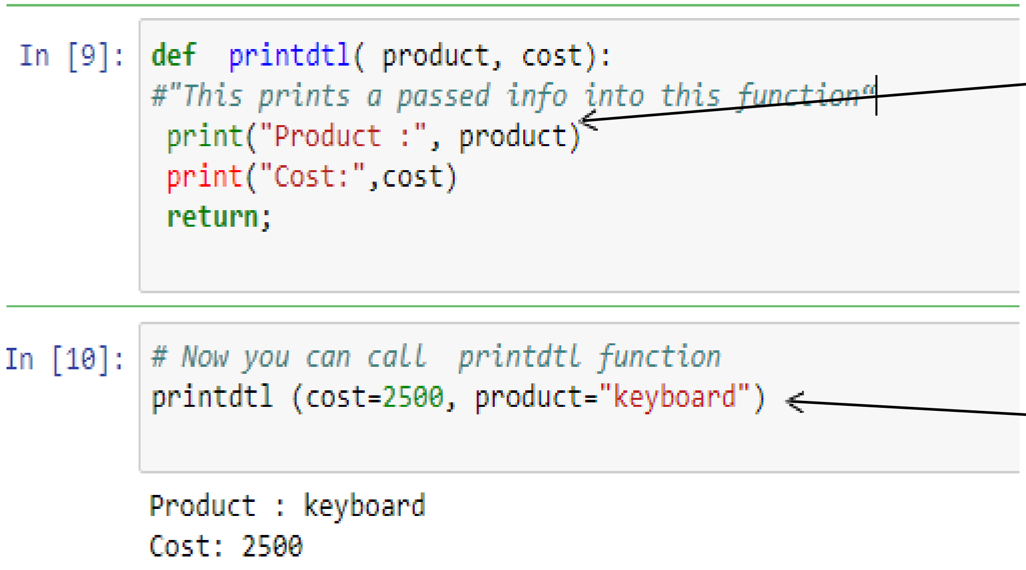
- Given values are 4,2,6,1 those can be placed in order by these positional argument.

| Val 1 | val2 | val3 | val4 |
|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ |
| **4** | | | |
| 4 | 2 | | |
| 4 | 2 | 6 | |
| 4 | 2 | 6 | 1 |

- First value 4 will enter into val1 and then second value will be to val2 so on …

## 2 .KEYWORD PARAMETERS

- We can identify the function call as Keyword argument by its argument name.
- In this argument no need of maintaining the parameters in an order.
- Example :



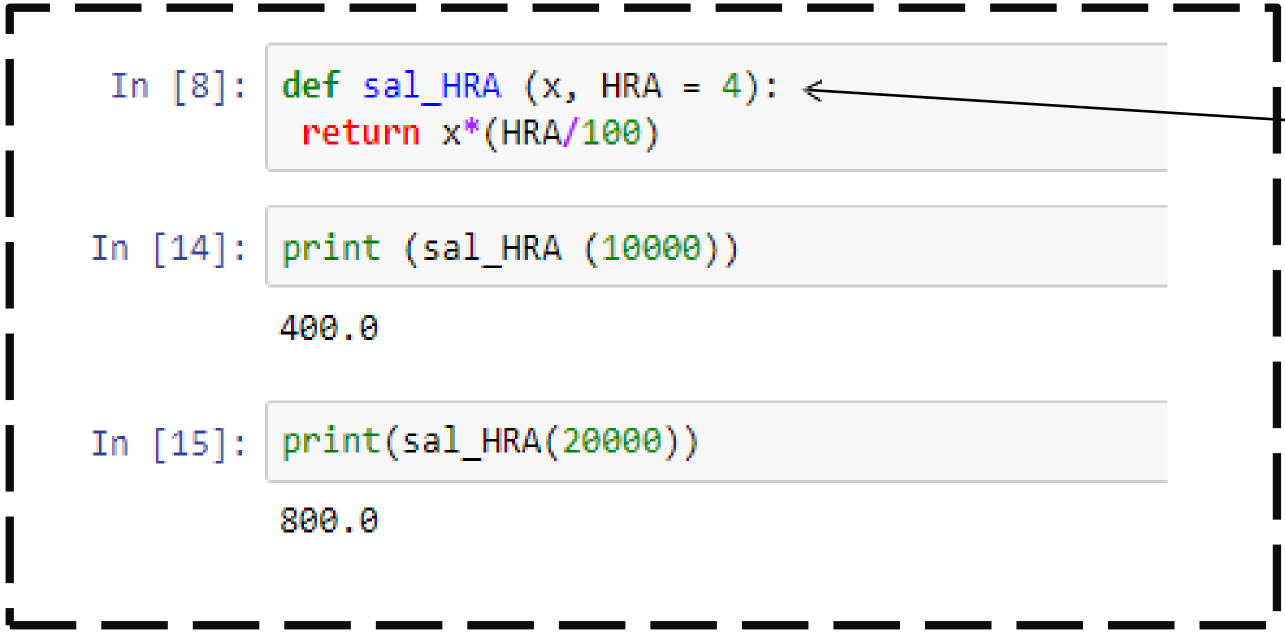
```
In [9]:  def  printdtl( product, cost):
         #"This prints a passed info into this function"
          print("Product :", product)
          print("Cost:",cost)
          return;

In [10]: # Now you can call  printdtl function
         printdtl (cost=2500, product="keyboard")

         Product : keyboard
         Cost: 2500
```

The "product" and "cost" are keyword parameters.

The value of the parameter is taken from function in different order as declared in the function.

## 3. DEFAULT PARAMETERS

- Default parameter is nothing but parameter which is assigned default when value is not assigned in a argument when function is called.
- These values will evaluate once the function is called or defined.
- Example :

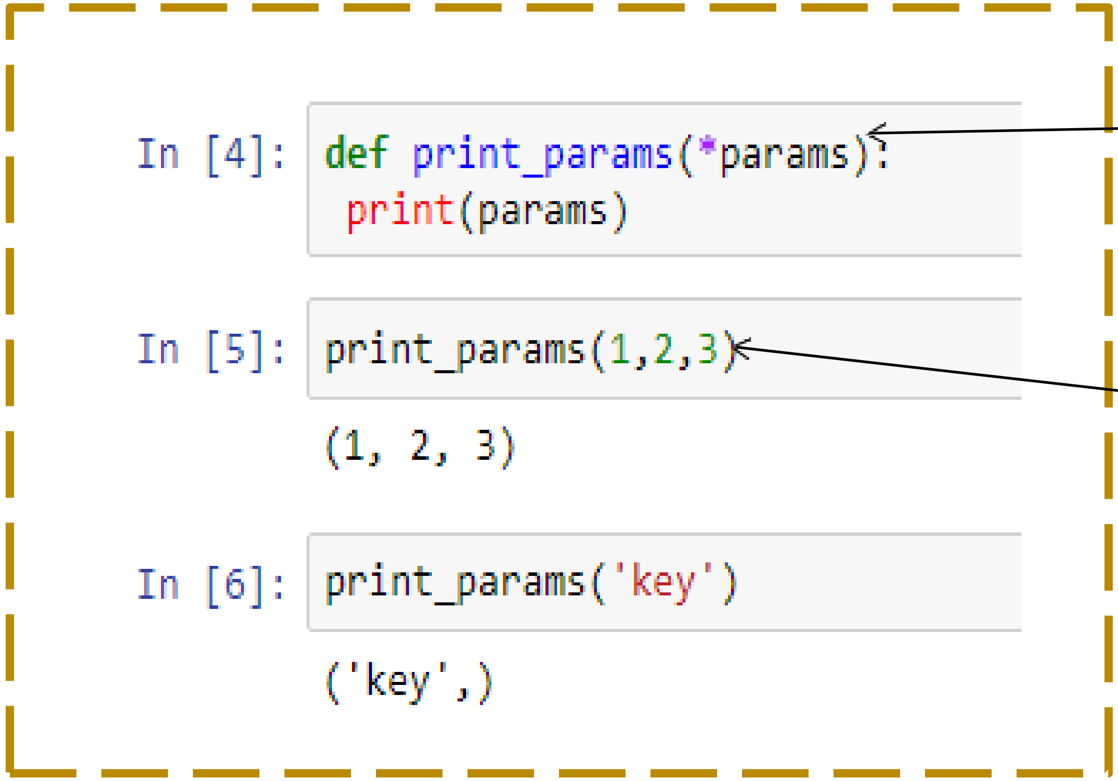


```
In [8]:  def sal_HRA (x, HRA = 4):
          return x*(HRA/100)

In [14]: print (sal_HRA (10000))

         400.0

In [15]: print(sal_HRA(20000))

         800.0
```

Here HRA is default parameter. HRA(given as percent) to value x is returned

# 4.COLLECTING PARAMETERS

- Sometimes it can be useful to allow the user to supply any number of parameters.
- Actually that's quite possible with the help of collecting parameters



```
In [4]: def print_params(*params):
         print(params)

In [5]: print_params(1,2,3)

        (1, 2, 3)

In [6]: print_params('key')

        ('key',)
```

Here parameter is passed through (*) which allows user to input any number of parameters

Here 3 values to the parameter is passed which is accepted by function

## SCOPE OF VARIABLES

- All variables cannot be accessed at a same place or at all locations. They are accessed at different places .

- Usually they are some Scopes for allocating variables at a particular location.
- They are :

LOCAL VARIABLES

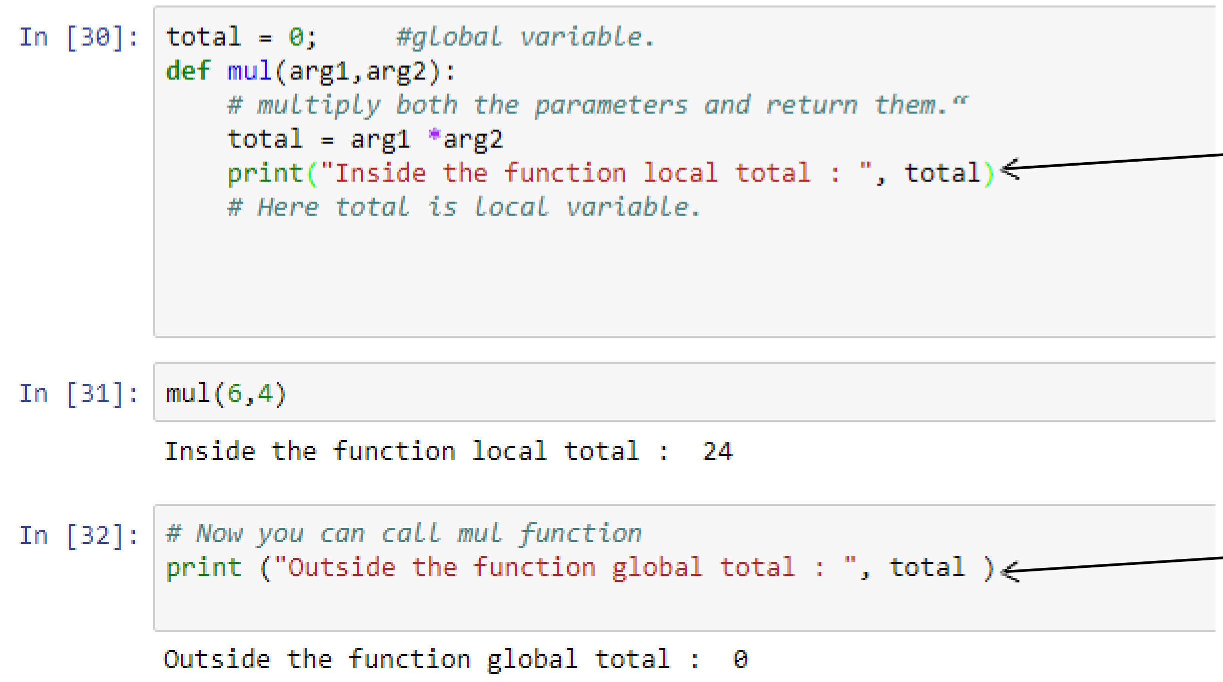
- Variables that are accessed inside the function is called Local Variables.

GLOBAL VARIABLES

- Variables that are accessed throughout the function or outside the program is called Global Variables.

# Local & Global Variable

- If we want to search where the variable location is allocated, first search will be in Local variable if it is not found in local variable then the search will be in Global variable . If the variable still cannot be found in Global variable then search will be *in Built in predifined Name space .*
- If still variables are not found there then an exception will be raised i.e. *name is not defined*



```
In [30]: total = 0;     #global variable.
         def mul(arg1,arg2):
             # multiply both the parameters and return them."
             total = arg1 *arg2
             print("Inside the function local total : ", total)
             # Here total is local variable.

In [31]: mul(6,4)

         Inside the function local total :  24

In [32]: # Now you can call mul function
         print ("Outside the function global total : ", total )

         Outside the function global total :  0
```

To check the output of local variable : its value is printed under function

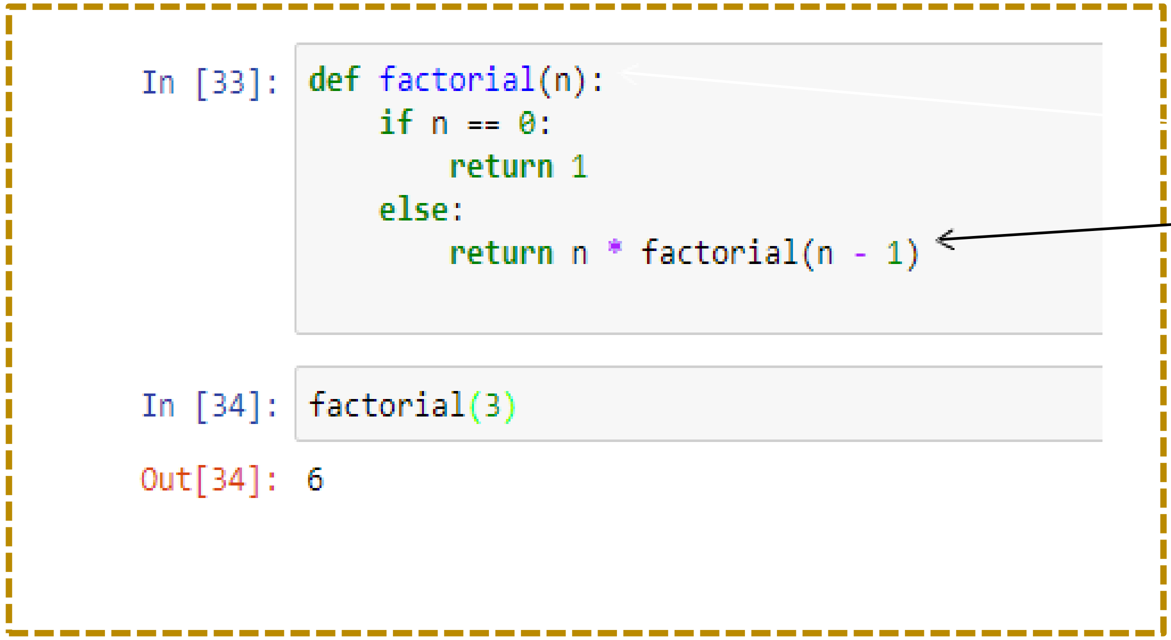
To check the output of global variable : its value is printed outside the function

## RETURN

- Return statement can return any type of value which python identifies.
- In Python all functions return at least one value. If they are not defined then there is no return value.

## RECURSION

- We can call a function in recursive, i.e. the function will call itself.
- Example :



```
In [33]: def factorial(n):
             if n == 0:
                 return 1
             else:
                 return n * factorial(n - 1)

In [34]: factorial(3)

Out[34]: 6
```

The function which is created, itself is returned -> Recursive function

- Function defining by using recursion is too costly for executing the program or due its memory location.

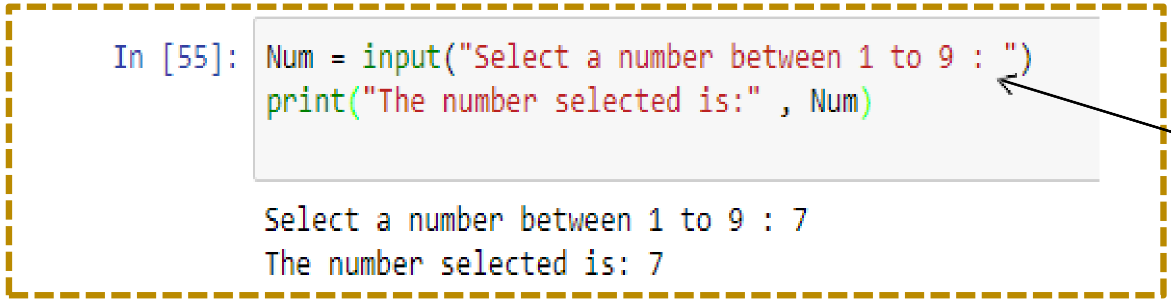
## FILES

- In python there is a possibility to manipulate files .Files are used to store objects permanently. The following are the steps to use Files

1. Printing on screen
2. Reading data from keyboard
3. Opening and closing file
4. Reading and writing files
5. Reading CSV, txt and excel files

## 2.READING DATA FROM KEYBOARD

- For reading data from keyboard they are 2 built in functions in python.
- They are:
  - Raw input Function.
  - Input function.
- #For version greater than python-3.0



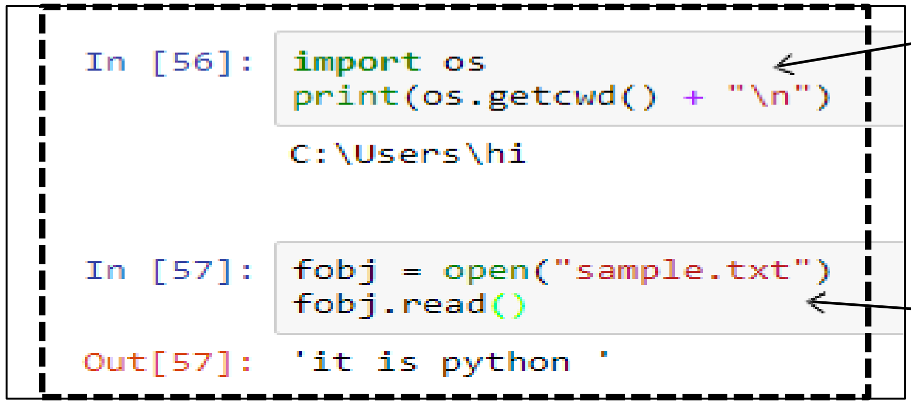
```
In [55]: Num = input("Select a number between 1 to 9 : ")
         print("The number selected is:" , Num)

         Select a number between 1 to 9 : 7
         The number selected is: 7
```

The "input" function is used to get input from the keyboard.

- #For version less than python-3.0
  - Num = raw_input("Select a number between 1 to 9")
  - print("The number selected is: ", Num)

# READING AND WRITING FILES

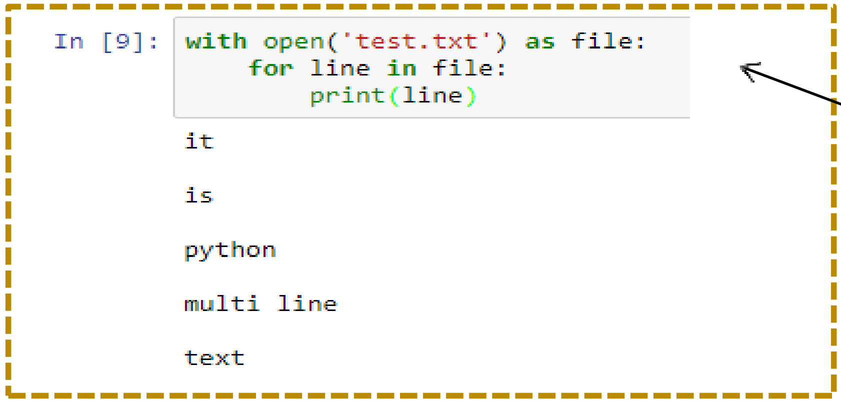
| Name | open("filename") opens the given file for reading,and returns a file object |
|---|---|
| name.read() | file's entire contents as a string |



```
In [56]: import os
         print(os.getcwd() + "\n")

         C:\Users\hi


In [57]: fobj = open("sample.txt")
         fobj.read()

Out[57]: 'it is python '
```

To know where is working directory

Open and read function is used here to read the file



```
In [9]: with open('test.txt') as file:
            for line in file:
                print(line)

        it

        is

        python

        multi line

        text
```

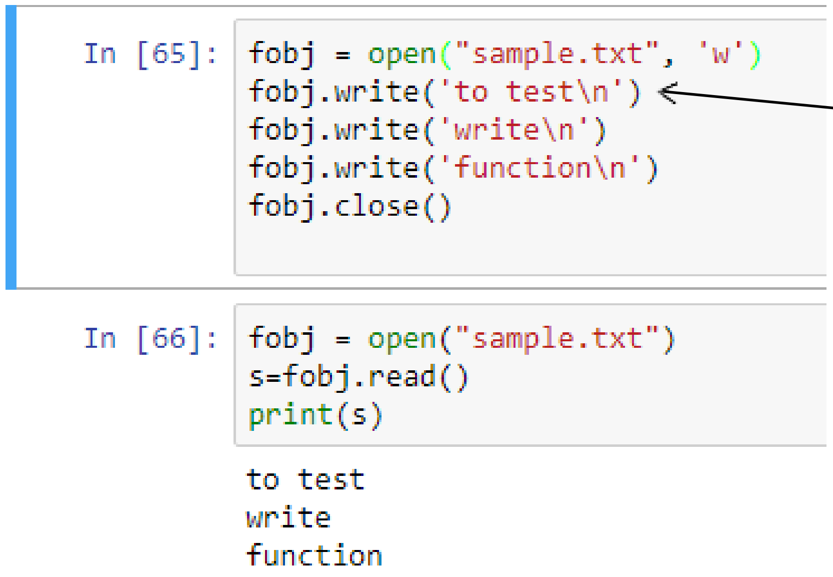
To read multiple lines, we have to open it with “with open “ keyword and print it in “for” loop

## WRITING FILES IN PYTHON

- The *write()* method writes any string to an open file.

- Syntax

  fileObject.write(string);



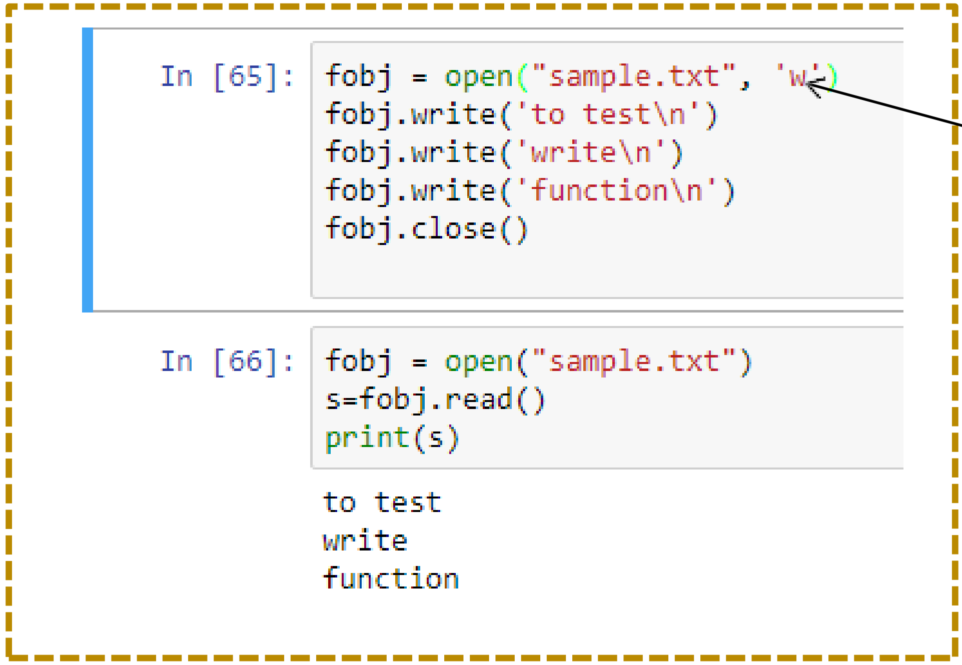
```
In [65]: fobj = open("sample.txt", 'w')
         fobj.write('to test\n')
         fobj.write('write\n')
         fobj.write('function\n')
         fobj.close()

In [66]: fobj = open("sample.txt")
         s=fobj.read()
         print(s)

         to test
         write
         function
```

Used the “write” function to write multiple strings in the file. “\n” changes line

# OPENING AND CLOSING FILES

- **The open Function**:
- By using Python's built-in open() function we can open a file.
- **Syntax**
- file object = open (file_name)**:** ([, access_mode][, buffering])

1. **file_name:**Name of file.
2. **access_mode:** The access_mode determines the mode in which the file has to be opened i.e. read, write

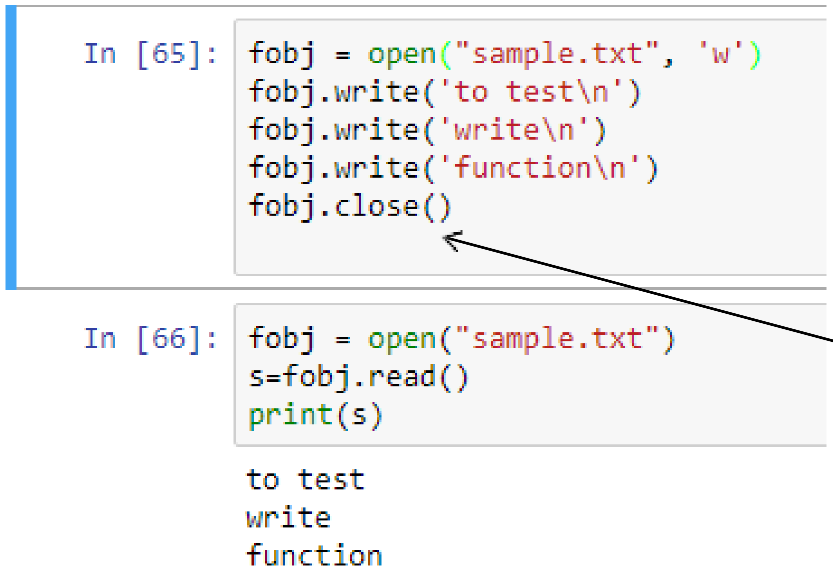


```
In [65]: fobj = open("sample.txt", 'w')
         fobj.write('to test\n')
         fobj.write('write\n')
         fobj.write('function\n')
         fobj.close()

In [66]: fobj = open("sample.txt")
         s=fobj.read()
         print(s)

         to test
         write
         function
```

Used the access mode ‘w’ in “open” function in the file

# OPENING AND CLOSING FILES

- **The close Function:**
- when the reference object of a file is reassigned to another file. Python automatically closes that file.
- **SYNTAX:**
  - fileObject.close();



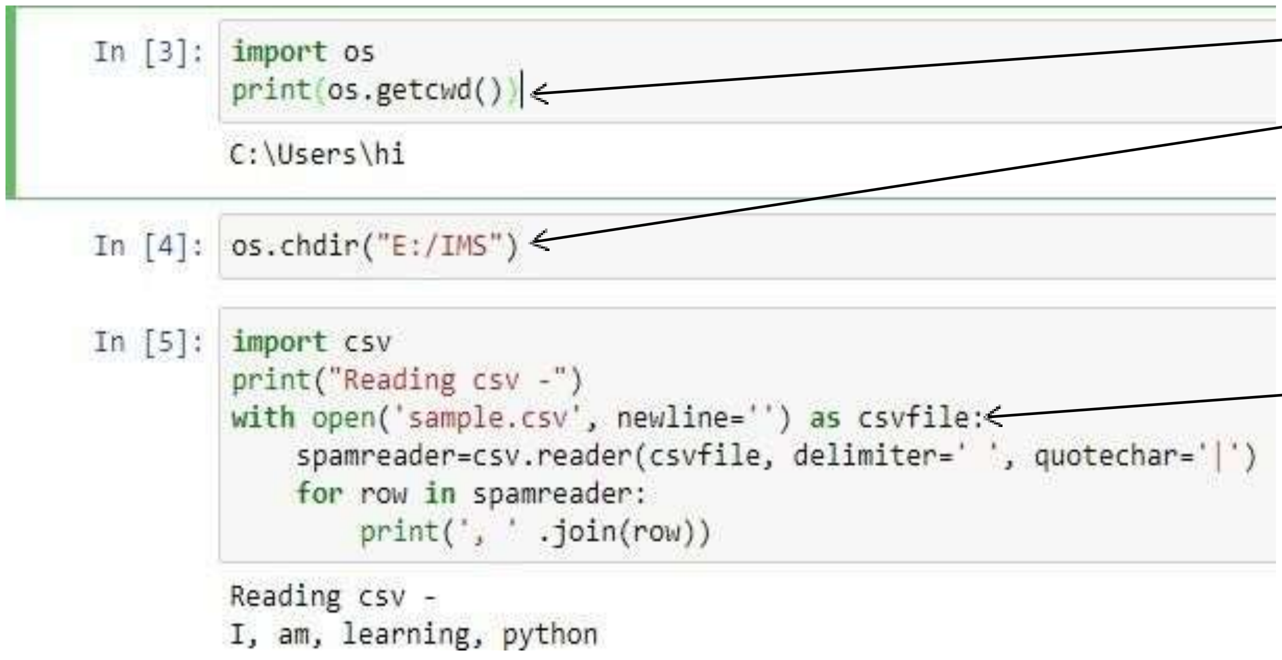
```
In [65]: fobj = open("sample.txt", 'w')
         fobj.write('to test\n')
         fobj.write('write\n')
         fobj.write('function\n')
         fobj.close()

In [66]: fobj = open("sample.txt")
         s=fobj.read()
         print(s)

         to test
         write
         function
```

Used the function “close” to close the file

## READING CSV, TXT AND EXCEL FILES

- A CSV file contains a number of rows and columns separated by commas.

- **Reading CSV Files**
- To read data from a CSV file, use the reader function to create a reader Object. Use writer() to create an object for writing, then iterate over the rows, using writerow() to print them.



```
In [3]: import os
        print(os.getcwd())

        C:\Users\hi

In [4]: os.chdir("E:/IMS")

In [5]: import csv
        print("Reading csv -")
        with open('sample.csv', newline='') as csvfile:
            spamreader=csv.reader(csvfile, delimiter=' ', quotechar='|')
            for row in spamreader:
                print(', '.join(row))

        Reading csv -
        I, am, learning, python
```

os.getcwd() helps in getting working directory, while os.chdir is used to change working directory

"with open" keyword and csv.reader() function can be used to read file

## Important packages in Python

- Numpy

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed

Using NumPy, a developer can perform the following operations :−

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

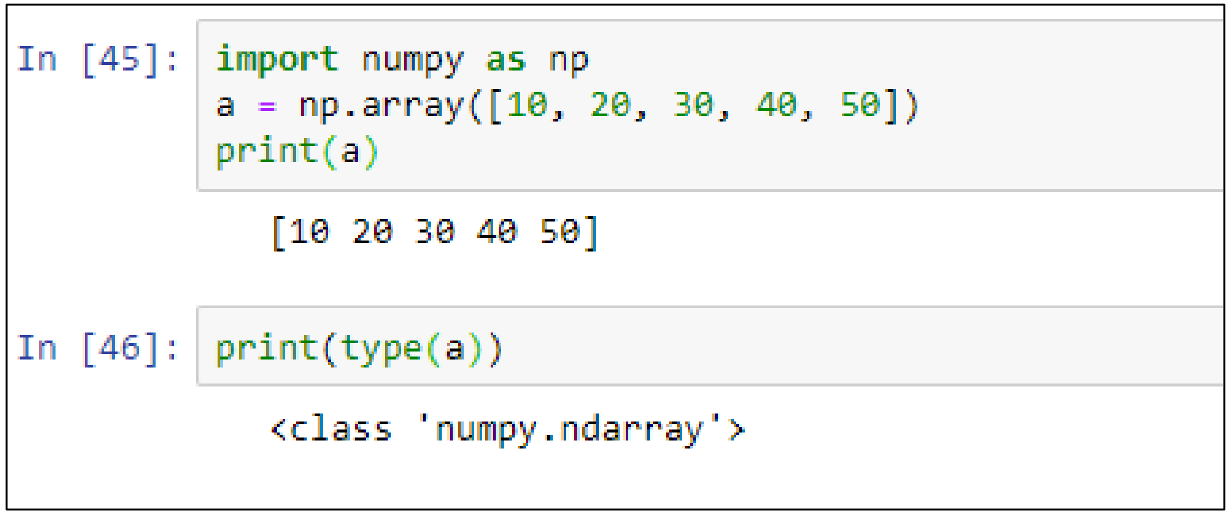
Standard Python distribution doesn't come bundled with NumPy module. An alternative is to import NumPy using popular Python package installer, **pip**.

## Numpy

To import NumPy package we write :

import numpy as np

Here, we are telling python to import NumPy package and store it as “np”. This is mostly useful when you work on data manipulation using NumPy arrays.



```
In [45]: import numpy as np
         a = np.array([10, 20, 30, 40, 50])
         print(a)

            [10 20 30 40 50]

In [46]: print(type(a))

            <class 'numpy.ndarray'>
```

## Pandas

Pandas is an open-source, Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

In 2008, developer Wes McKinney started developing pandas when in need of high performance, flexible tool for analysis of data.

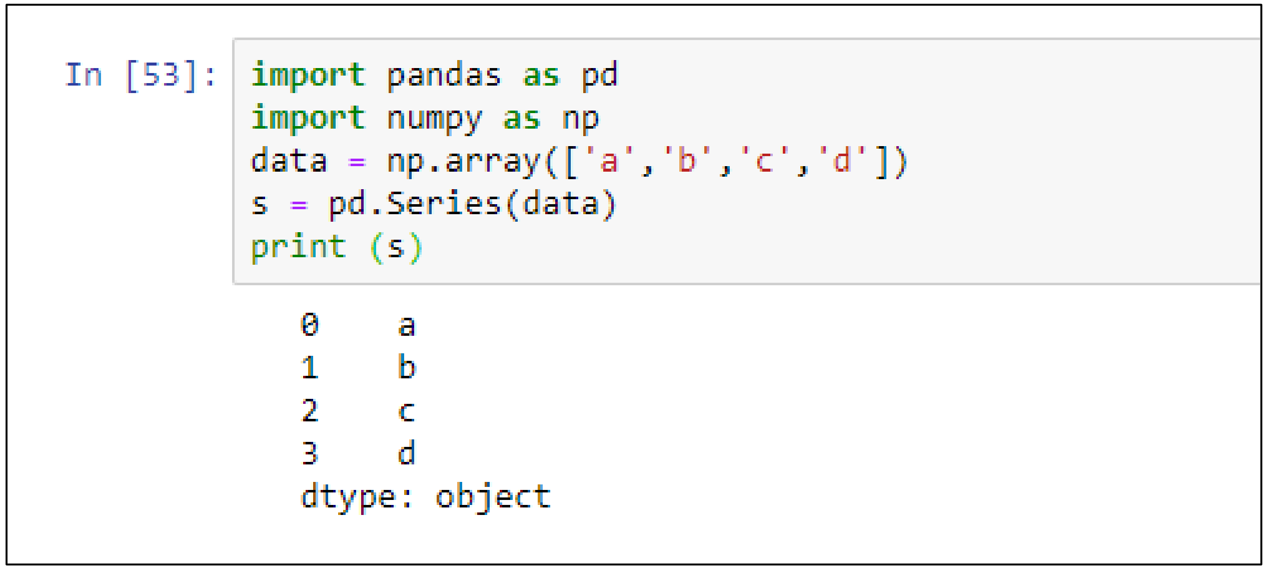
Prior to Pandas, Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.

## Pandas

To import Pandas package we write :

import pandas as pd

Here, we are telling python to import Pandas package and store it as “pd”. This is mostly used for load, prepare, manipulate, model, and analyse data.

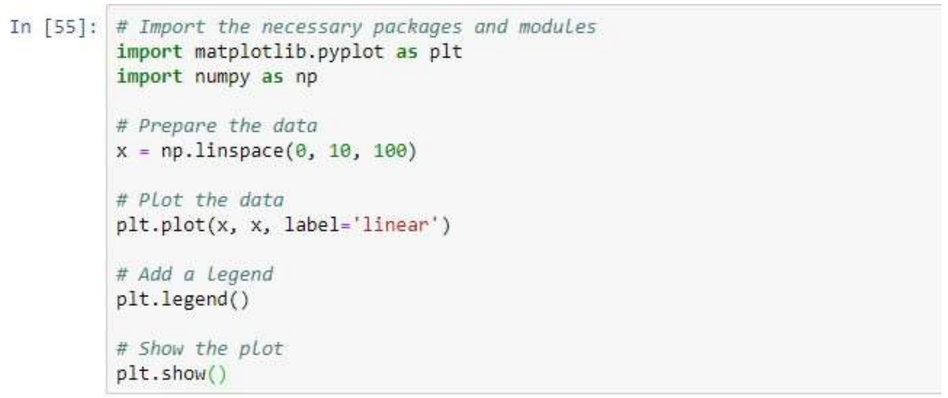


```
In [53]: import pandas as pd
         import numpy as np
         data = np.array(['a','b','c','d'])
         s = pd.Series(data)
         print (s)

           0    a
           1    b
           2    c
           3    d
           dtype: object
```

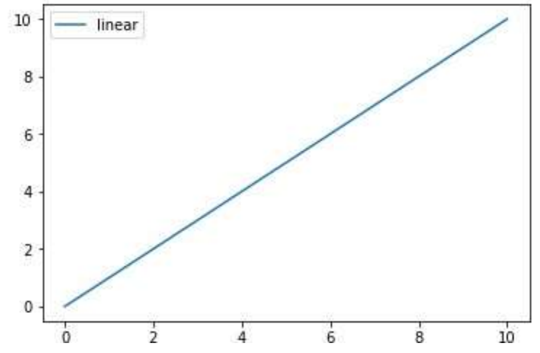
# Matplotlib

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc.

It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.



```
In [55]: # Import the necessary packages and modules
         import matplotlib.pyplot as plt
         import numpy as np

         # Prepare the data
         x = np.linspace(0, 10, 100)

         # Plot the data
         plt.plot(x, x, label='linear')

         # Add a legend
         plt.legend()

         # Show the plot
         plt.show()
```



The package is imported into the Python script by adding the following statement –

import matplotlib.pyplot as plt

# Thank You.