

Python for Data Science

Ashwini Kumar Jha



Python IDE(Integrated Development Environment)

- Python download and install Python 3.9version
- IDLE editor .py
- Pycharm
- Atom
- PyWin
- Anaconda
- Browser based programming
- Jupyter Notebook
- Extension .ipynb
- Interactive python notebook



Google Colaborator

- there are 2 version of python
- Python 2
- Python 3



What is Python

Python is a simple, general purpose, high level, and object-oriented programming language.

Python is an interpreted scripting language also. *Guido Van Rossum* is known as the founder of Python programming.

Definition:

Python is a general purpose, dynamic, high-level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Pythons Attraction

Python is *easy to learn* yet powerful and versatile scripting language, which makes it attractive for Application Development.

Python's syntax and *dynamic typing* with its interpreted nature make it an ideal language for scripting and rapid application development.

Python supports *multiple programming pattern*, including object-oriented, imperative, and functional or procedural programming styles.

Python is not intended to work in a particular area, such as web programming. That is why it is known as *multipurpose* programming language because it can be used with web, enterprise, 3D CAD, etc.



Pythons Attraction:

We don't need to use data types to declare variable because it is *dynamically typed* so we can write `a=10` to assign an integer value in an integer variable.

Python makes the development and debugging *fast* because there is no compilation step included in Python development, and edit-test-debug cycle is very fast.

Python History

Python was invented by **Guido van Rossum** in 1991 at CWI in Netherland. The idea of Python programming language has taken from the ABC programming language or we can say that ABC is a predecessor of Python language.

There is also a fact behind the choosing name Python. Guido van Rossum was a fan of the popular BBC comedy show of that time, "**Monty Python's Flying Circus**". So he decided to pick the name **Python** for his newly created programming language.

Python has the vast community across the world and releases its version within the short period.

Why learn Python?

Python provides many useful features to the programmer. These features make it most popular and widely used language. We have listed below few-essential feature of Python.

- Easy to use and Learn
- Expressive Language
- Interpreted Language
- Object-Oriented Language
- Open Source Language
- Extensible



Learn Standard Library



GUI Programming
Support



Integrated



Embeddable



Dynamic Memory
Allocation



Wide Range of
Libraries and
Frameworks

Where is Python used?

Python is a general-purpose, popular programming language and it is used in almost every technical field. The various areas of Python use are given below.

- Data Science
- Data Mining
- Desktop Applications
- Console-based Applications
- Mobile Applications
- Software Development
- Artificial Intelligence
- Web Applications
- Enterprise Applications
- 3D CAD Applications
- Machine Learning
- Computer Vision or Image Processing Applications.
- Speech Recognitions

Python Basic Syntax

There is no use of curly braces or semicolon in Python programming language. It is English-like language.

But Python uses the indentation to define a block of code. Indentation is nothing but adding whitespace before the statement when it is needed. **For example -**

```
def func():  
    statement 1  
    statement 2  
    .....  
    .....  
    statement N
```

the statements that are same level to right belong to the function. Generally, we can use four whitespaces to define indentation.

Python First
Program
Python provides the
facility to execute
the code using few
lines. **For example** -
Suppose we want to
print the "**Hello
World**" program in
Java and Python

```
public class HelloWorld {  
    public static void main(String[]  
        args){  
        // Prints "Hello, World" to the t  
        erminal window.  
        System.out.println("Hello Wor  
        ld");  
    }  
}
```

```
print("Hello World")
```

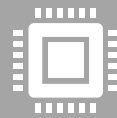
Note*

Both programs will print the
same result, but it takes only
one statement without using a
semicolon or curly braces in
Python.

Python Popular Frameworks and Libraries
Python has wide range of libraries and frameworks widely used in various fields such as machine learning, artificial intelligence, web applications, etc. We define some popular frameworks and libraries of Python as follows.



Web development (Server-side) - Django
Flask, Pyramid, CherryPy



GUIs based applications - Tk, PyGTK, PyQt, PyJs, etc.



Machine Learning - TensorFlow, PyTorch, **Scikit-learn**, Matplotlib, Scipy, etc.



Mathematics - Numpy, Pandas, etc.

Python print() Function

- The **print()** function displays the given object to the standard output device (screen) or to the text stream file.
- Example - 1: Return a value

```
print("Welcome to python.")
```

```
a = 10
```

```
# Two objects are passed in print() function
```

```
print("a =", a)
```

```
b = a
```

```
# Three objects are passed in print function
```

```
print('a =', a, '= b')
```

Output:

```
Welcome to python.
```

```
a = 10
```

```
a = 10 =b
```

Taking Input to the User
Python provides the **input()** function which is used to take input from the user.

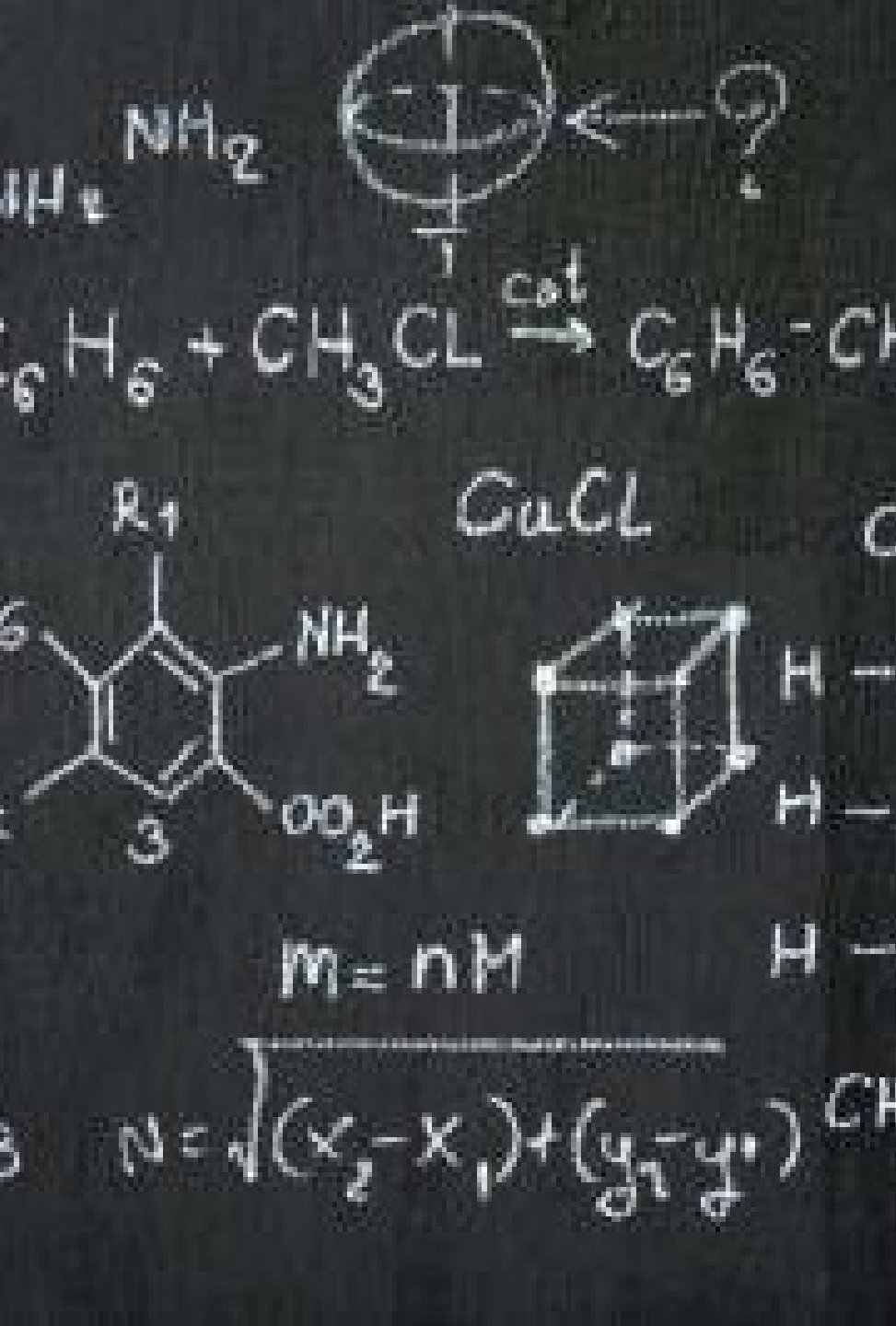
```
name = input("Enter a name of student:")  
print("The student name is: ", name)
```

Output:

```
Enter a name of student:  
Devansh
```

```
The student name is: Devansh
```

- By default, the **input()** function takes the string input but what if we want to take other data types as an input.
- If we want to take input as an integer number, we need to typecast the **input()** function into an integer



Example:

```
a = int(input("Enter first number: "))  
b = int(input("Enter second number: "))
```

```
print(a+b)
```

Output:

Enter first number: 50

Enter second number: 100

150

We can take any type of values using **input()** function.

Python Data Structures

Data structures are referred which can hold some data together or they are used to store the data in organized way. Python provides built-in data structures such as **list, tuple, dictionary, and set.**

Python List

Python list holds the ordered collection of items. We can store a sequence of items in a list. Python list is mutable which means it can be modified after its creation. The items of lists are enclosed within the square bracket [] and separated by the comma.

```
L1 = ["John", 102, "USA"]
```

```
L2 = [1, 2, 3, 4, 5, 6]
```

If we try to print the type of L1, L2, using type() function then it will come out to be a list.

```
print(type(L1))
```

```
print(type(L2))
```

Output:

```
<class 'list'>
```

```
<class 'list'>
```


Python Tuple

Python Tuple is used to store the sequence of immutable Python objects. The tuple is similar to lists since the value of the items stored in the list can be changed, whereas the tuple is immutable, and the value of the items stored in the tuple cannot be changed.

```
tup = ("Apple", "Mango" , "Orange" , "Banana")  
print(type(tup))  
print(tup)  
o/p:  
<class 'tuple'>  
('Apple', 'Mango', 'Orange', 'Banana')
```

If we try to add new to the tuple, it will throw an error.

```
tup = ("Apple", "Mango" , "Orange" ,  
"Banana")
```

```
tup[2] = "Papaya"
```

```
print(tup)
```

Traceback (most recent call last):

```
File "C:/Users/DEVANSH  
SHARMA/PycharmProjects/Hello/gamewi  
thturtle.py", line 3, in
```

```
tup[2] = "Papaya"
```

TypeError: 'tuple' object does not support item assignment

Python String

Python string is a sequence of characters. It is a collection of the characters surrounded by single quotes, double quotes, or triple quotes. It can also define as collection of the Unicode characters.

```
# Creating string using double quotes
```

```
str1 = "Hi Python"
```

```
print(str1)
```

```
# Creating string using single quotes
```

```
str1 = 'Hi Python'
```

```
print(str1)
```

```
# Creating string using triple quotes
```

```
str1 = """Hi Python"""
```

```
print(str1)
```

Hi Python

Hi Python

Hi Python

Python doesn't support the character data-type. A single character written as 'p' is treated as a string of length 1.

Strings are also immutable. We can't change after it is declared.

Dictionaries

Python Dictionary is a most efficient data structure and used to store the large amount of data. It stores the data in the key-value pair format. Each value is stored corresponding to its key.

- Keys must be a unique and value can be any type such as integer, list, tuple, etc.
- It is a mutable type; we can reassign after its creation. Below is the example of creating dictionary in Python.
- **Example -**
- ```
employee = {"Name": "John", "Age": 29, "salary": 250000, "Company": "GOOGLE"}
```
- ```
print(type(employee))
```
- ```
print("printing Employee data ")
```
- ```
print(employee)
```

- Output:
- `<class 'dict'>`
- Printing Employee data
- `{'Name': 'John', 'Age': 29, 'salary': 250000, 'Company': 'GOOGLE'}`



Python Sets

A Python set is a collection of unordered elements. Each element in set must be unique and immutable. Sets are mutable which means we can modify anytime throughout the program.

Creating Set

```
Month = {"January", "February", "March", "April", "May",  
        "June", "July"}
```

```
print(Month)
```

```
print(type(Month))
```

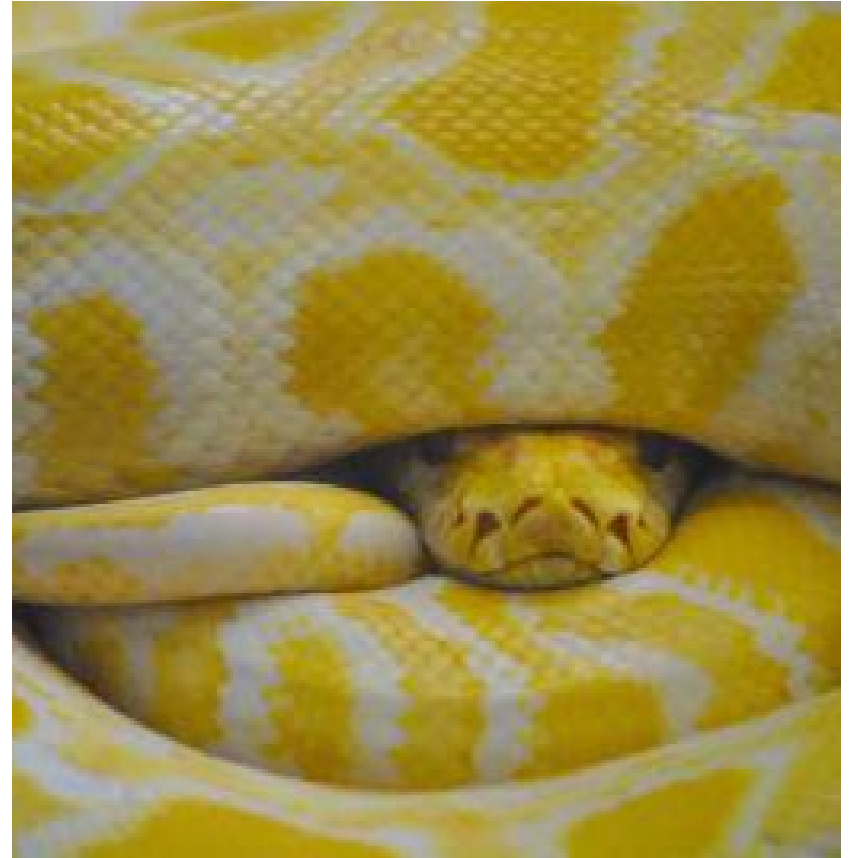
Output:

```
{'March', 'July', 'April', 'May', 'June', 'February', 'January'}
```

```
<class 'set'>
```

Python Oops Concepts

- Everything in Python is treated as an object including integer values, floats, functions, classes, and none. Apart from that, Python supports all oriented concepts.
- Classes and Objects - Python classes are the blueprint of the object. An object is a collection of data and method that act on the data.
- Inheritance - An inheritance is a technique where one class inherits the properties of other classes.
- Constructor - Python provides a special method `__init__()` which is known as a constructor. This method is automatically called when an object is instantiated.
- Data Member - A variable that holds data associated with a class and its objects.



Libraries and Frameworks



Python consists of vast libraries and various frameworks. Libraries are essential to work with the domain specific projects

1. TensorFlow - It is an artificial intelligence library which allows us to create large scale AI based projects.
2. Django - It is an open source framework that allows us to develop web applications. It is easy, flexible, and simple to manage.
3. Flask - It is also an open source web framework. It is used to develop lightweight web applications.
4. Pandas - It is a Python library which is used to perform scientific computations.
5. Keras - It is an open source library, which is used to work around the neural network.
6. There are many libraries in Python. mentioned a few of them.

Usage of Python

Python is a general purpose, open source, high-level programming language and also provides number of libraries and frameworks. Python has gained popularity because of its simplicity, easy syntax and user-friendly environment.

Desktop Applications

Web Applications

Data Science

Artificial Intelligence

Machine Learning

Scientific Computing

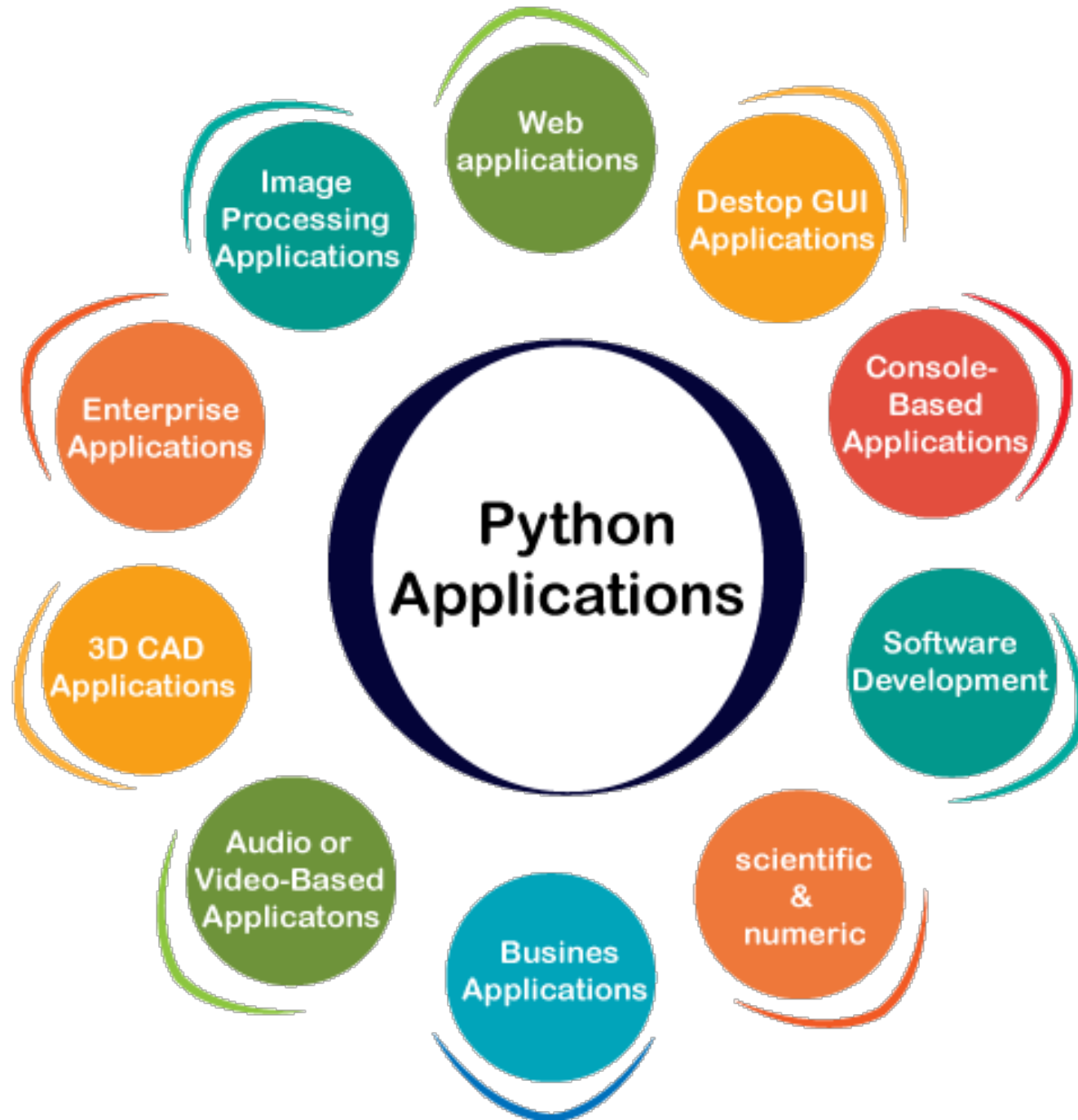
Robotics

Internet of Things (IoT)

Gaming

Mobile Apps

Data Analysis and
Preprocessing



Python Applications

Python is known for its general-purpose nature that makes it applicable in almost every domain of software development.

Python makes its presence in every emerging field. It is the fastest-growing programming language and can develop any application.

Python Applications

Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, BeautifulSoup, Feedparser, etc.

One of Python web-framework named Django is used on **Instagram**. Python provides many useful frameworks, and these are given below:

1. Django and Pyramid framework(Use for heavy applications)
2. Flask and Bottle (Micro-framework)
3. Plone and Django CMS (Advance Content management)

Desktop GUI Applications

The GUI stands for the Graphical User Interface, which provides a smooth interaction to any application. Python provides a Tk GUI library to develop a user interface. Some popular GUI libraries are given below.

1. Tkinter or Tk
2. wxWidgetM
3. Kivy (used for writing multitouch applications)
4. PyQt or Pyside

Scientific and Numeric

Python language is the most suitable language for Artificial intelligence or machine learning.

Implementing machine learning algorithms require complex mathematical calculation.

Python has many libraries for scientific and numeric such as Numpy, Pandas, Scipy, Scikit-learn, etc.

Few popular frameworks of machine libraries are given below.

1. SciPy
2. Scikit-learn
3. NumPy
4. Pandas
5. Matplotlib

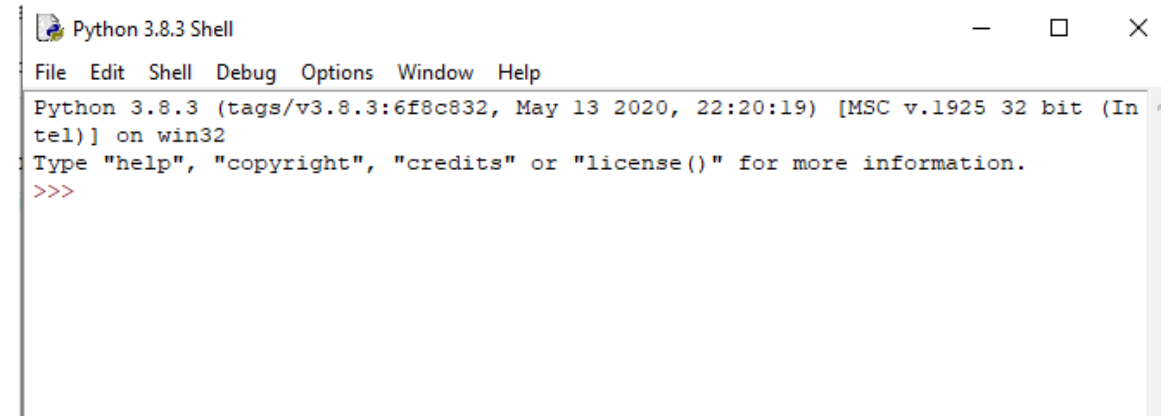
Image Processing Application

Python contains many libraries that are used to work with the image. The image can be manipulated according to our requirements. Some libraries of image processing are given below.


1. OpenCV
2. Pillow
3. SimpleITK

Interactive interpreter prompt

- Python provides us the feature to execute the Python statement one by one at the interactive prompt. It is preferable in the case where we are concerned about the output of each line of our Python program.



```
Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

 Python 3.8.3 Shell



File Edit Shell Debug Options Window Help

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

```
>>> print("Hello World")
```

```
Hello World
```

```
>>> |
```

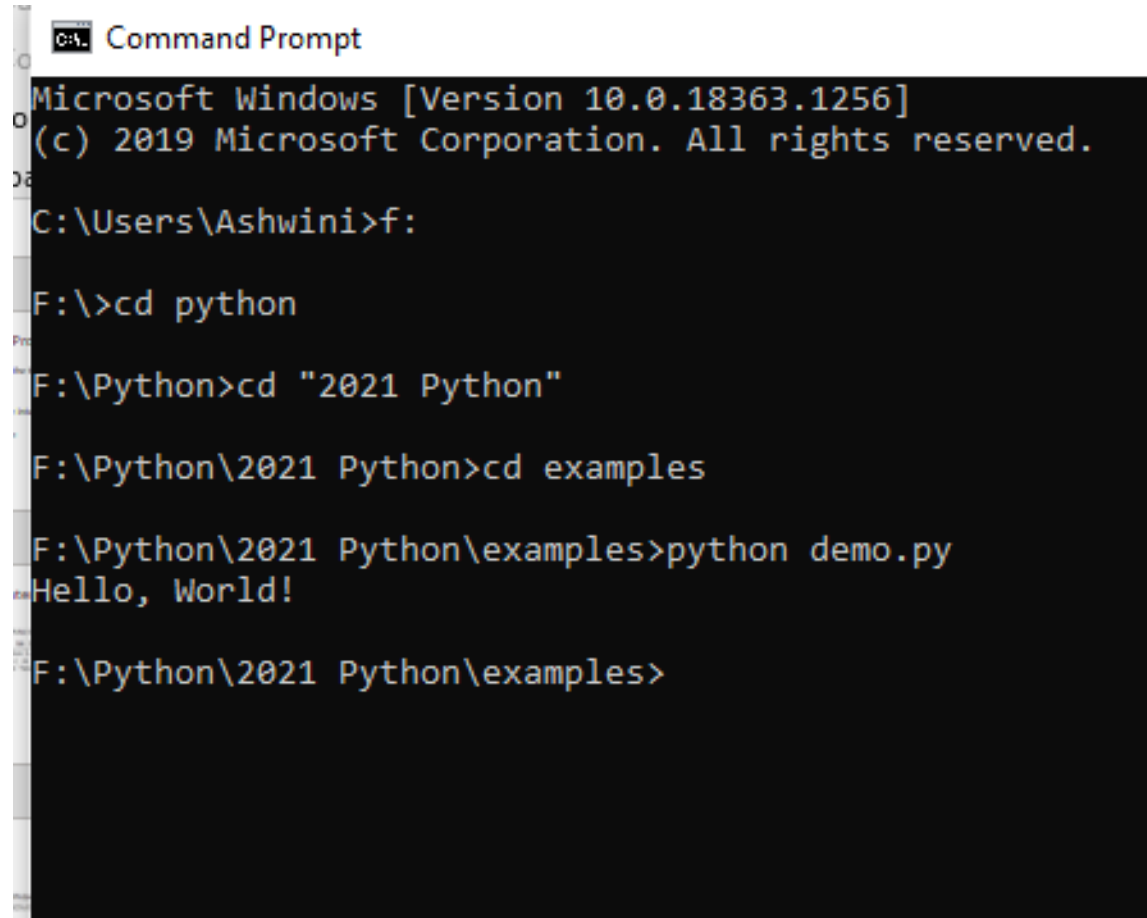
Using a script file (Script Mode Programming)

- The interpreter prompt is best to run the single-line statements of the code. However, we cannot write the code every-time on the terminal. It is not suitable to write multiple lines of code.
- Using the script mode, we can write multiple lines code into a file which can be executed later.

```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: F:/Python/2021 Python/examples/demo.py =====
Hello, World!
>>> |
```

Command Prompt

- we can also run the file using the operating system terminal. But, we should be aware of the path of the directory where we have saved our file.
- Open the command line prompt and navigate to the directory.



```
Command Prompt
Microsoft Windows [Version 10.0.18363.1256]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Ashwini>f:

F:\>cd python

F:\Python>cd "2021 Python"

F:\Python\2021 Python>cd examples

F:\Python\2021 Python\examples>python demo.py
Hello, World!

F:\Python\2021 Python\examples>
```

Multi-line Statements

- name = "ABC"
- branch = "Computer Science"
- age = "25"
- **print**("My name is: ", name,)
- **print**("My age is: ", age)

```
===== RESTART: F:  
Hello, World!  
My name is:  ABC  
My age is:  25  
>>> |
```

Basic Syntax of Python

Indentation and Comment in Python

- Indentation is the most significant concept of the Python programming language. Improper use of indentation will end up "**IndentationError**" in our code.
- statements that are the same level to the right belong to the same block. We can use four whitespaces to define indentation.

```
list1=[1, 2, 3, 4, 5]
for i in list1:
    print(i)
    if i==4:
        break
print("End of for loop")
|
```

```
===== RESTART:
1
2
3
4
End of for loop
>>>
```


Comments in Python:

Comments are essential for defining the code and help us and other to understand the code

- Single-Line Comment - Single-Line comment starts with the hash # character followed by text for further explanation.
- Multi-Line Comments - Python doesn't have explicit support for multi-line comments but we can use hash # character to the multiple lines. Example

we are defining for loop

To iterate the given list.

run this code.

- We can also use another way.

"""

This is an example

Of multi-line comment

Using triple-quotes

"""

'''

'''

Python Identifiers:

Python identifiers refer to a name used to identify a variable, function, module, class, module or other objects. There are few rules to follow while naming the Python Variable.

- A variable name must start with either an English letter or underscore (_).
- A variable name cannot start with the number.
- Special characters are not allowed in the variable name.
- The variable's name is case sensitive.

Example:

```
number = 10
```

```
print(num)
```

```
_a = 100
```

```
print(_a)
```

```
x_y = 1000
```

```
print(x_y)
```

Python Variables
Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.


- we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.
- Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

Identifier Naming:

1. The first character of the variable must be an alphabet or underscore (_).
2. All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
3. Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &,*).
4. Identifier name must not be similar to any keyword defined in the language.
5. Identifier names are case sensitive; for example, my name, and MyName is not the same.
6. Examples of valid identifiers: a123, _n, n_9, etc.
7. Examples of invalid identifiers: 1a, n%4, n 9, etc.



Declaring Variable and Assigning Values

- Python does not bind us to declare a variable before using it in the application. It allows us to create a variable at the required time.
 - We don't need to declare explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.
 - The equal (=) operator is used to assign value to a variable.
 - Dynamically typed language
- 



Object References

Python is the highly object-oriented programming language; that's why every data item belongs to a specific type of class.

```
print("Ashwini")
```

Output:

Ashwini

In the print statement, we have created a string object. Let's check the type of it using the Python built-in type() function.

```
type("John")
```

Output:

```
<class 'str'>
```



`a = 50` the variable `a` refers to an integer object. .

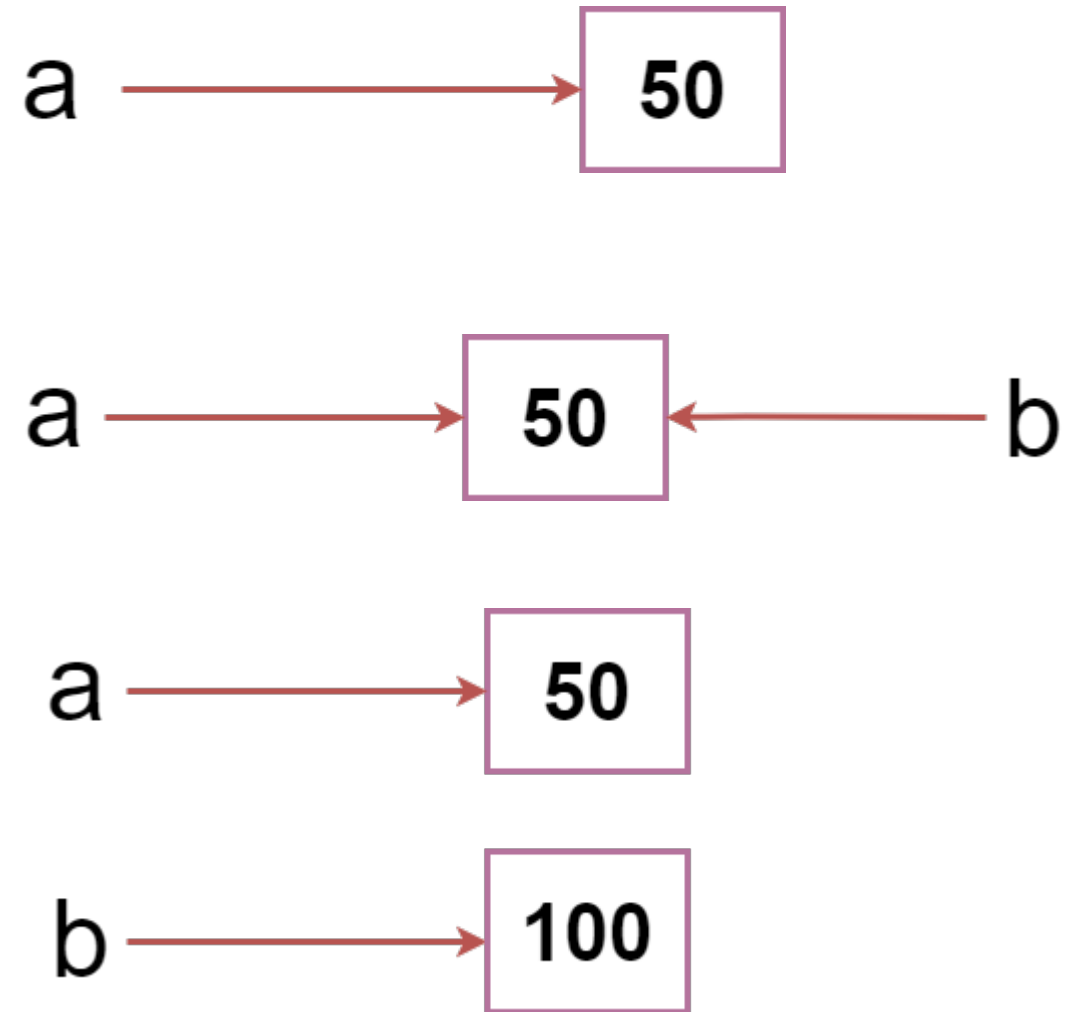
Suppose we assign the integer value 50 to a new variable `b`

`a = 50`

`b = a` The variable `b` refers to the same object that `a` points to because Python does not create another object.

Let's assign the new value to `b`. Now both variables will refer to the different objects.

Python manages memory efficiently if we assign the same variable to two different values.



Object Identity:

Every created object identifies uniquely in Python. Python provides the guarantee that no two objects will have the same identifier. The built-in `id()` function, is used to identify the object identifier.

```
a = 50
```

```
b = a
```

```
print(id(a))
```

```
print(id(b))
```

```
# Reassigned variable a
```

```
a = 500
```

```
print(id(a))
```

Output:

```
140734982691168
```

```
140734982691168
```

```
2822056960944
```

Variable Names:

- Variable names can be any length can have uppercase, lowercase (A to Z, a to z), the digit (0-9), and underscore character(_)

```
name = "Devansh"
```

```
age = 20
```

```
marks = 80.50
```

```
print(name)
```

```
print(age)
```

```
print(marks)
```

Output:

```
Devansh
```

```
20
```

```
80.5
```

```
name = "A"
```

```
Name = "B"
```

```
naMe = "C"
```

```
NAME = "D"
```

```
n_a_m_e = "E"
```

```
_name = "F"
```

```
name_ = "G"
```

```
_name_ = "H"
```

```
na56me = "I"
```

```
print(name,Name,naMe,NAME,n_a_m_e, NAME,  
n_a_m_e, _name, name,_name, na56me)
```

Output:

```
A B C D E D E F G F I
```


Multi-word keywords can be created by the following method.

- **Camel Case** - In the camel case, each word or abbreviation in the middle of begins with a capital letter. There is no intervention of whitespace. For example - `nameOfStudent`, `valueOfVariable`, etc.
- **Pascal Case** - It is the same as the Camel Case, but here the first word is also capital. For example - `NameOfStudent`, etc.
- **Snake Case** - In the snake case, Words are separated by the underscore. For example - `name_of_student`, etc.

Multiple Assignment

Assigning single value to multiple variables

```
x=y=z=50
```

```
print(x)
```

```
print(y)
```

```
print(z)
```

Output:

50

50

50

Assigning multiple values to multiple variables:

```
a,b,c=5,10,15
```

```
print a
```

```
print b
```

```
print c
```

Output:

5

10

15

Python Variable Types

Local Variable

Local variables are the variables that declared inside the function and have scope within the function. Let's understand the following example.

Declaring a function

def add():

 # Defining local variables. They has scope only within a function

 a = 20

 b = 30

 c = a + b

print("The sum is:", c)

Calling a function

add()

Output:

The sum is: 50

we declared a function named add() and assigned a few variables within the function. These variables will be referred to as the local variables which have scope only inside the function. If we try to use them outside the function, we get a following error.

add()

Accessing local variable outside the function

print(a)

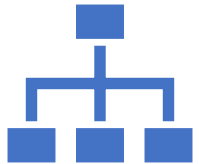
Output:

The sum is: 50

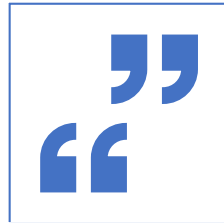
print(a)

NameError: name 'a' is not defined

Global Variables



Global variables can be used throughout the program, and its scope is in the entire program. We can use global variables inside or outside the function.



A variable declared outside the function is the global variable by default. Python provides the global keyword to use global variable inside the function. If we don't use the global keyword, the function treats it as a local variable

```
# Declare a variable and initialize it
```

```
x = 101
```

```
# Global variable in function
```

```
def mainFunction():
```

```
    # printing a global variable
```

```
    global x
```

```
    print(x)
```

```
    # modifying a global variable
```

```
    x = 'Welcome To MBIT'
```

```
    print(x)
```

```
mainFunction()
```

```
print(x)
```

Output: 101

Welcome To MBIT

Welcome To MBIT

Python Data Types

- Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.
- `a = 5`
- The variable `a` holds integer value five and we did not define its type.

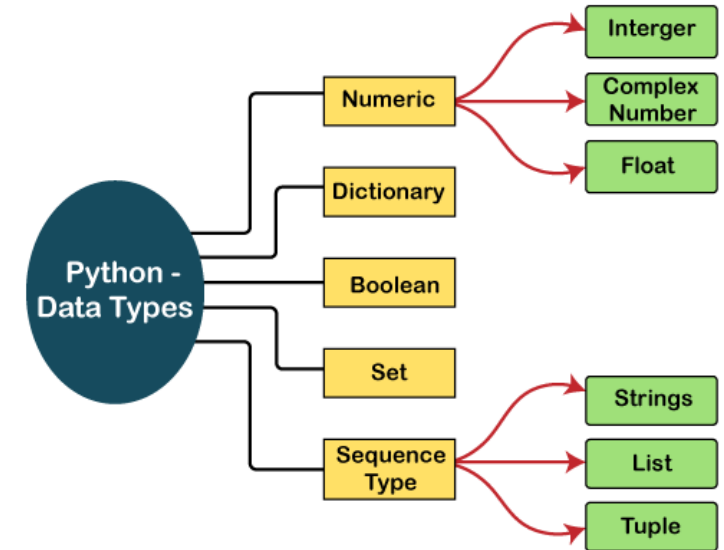
```
a=10
b="Hi Python"
c = 10.5
print(type(a))
print(type(b))
print(type(c))
```

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

- A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.
- The data types defined in Python are given below.
- [Numbers](#)
- [Sequence Type](#)
- [Boolean](#)
- [Set](#)
- [Dictionary](#)



Numbers: Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type. Python provides the **type()** function to know the data-type of the variable. Similarly, the **isinstance()** function is used to check an object belongs to a particular class.

```
a = 5
print("The type of a", type(a))

b = 40.5
print("The type of b", type(b))

c = 1+3j
print("The type of c", type(c))
print(" c is a complex number",
      isinstance(1+3j,complex))
```

Output:

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class
'complex'>
c is complex number: True
```

Python supports three types of numeric data

- 1. int** - Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to **int**
- 2. Float** - Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.
- 3. complex** - A complex number contains an ordered pair, i.e., $x + iy$ where x and y denote the real and imaginary parts, respectively. The complex numbers like $2.14j$, $2.0 + 2.3j$, etc.

Sequence Type

String

The string can be defined as the sequence of characters represented in the quotation marks. In Python, we can use single, double, or triple quotes to define a string.

String handling in Python is a straightforward task since Python provides built-in functions and operators to perform operations in the string.

In the case of string handling, the operator + is used to concatenate two strings as the operation "hello"+"python" returns "hello python".

The operator * is known as a repetition operator as the operation "Python" *2 returns 'Python Python'.

```
str = "string using double quotes"
```

```
print(str)
```

```
s = """A multiline
```

```
string"""
```

```
print(s)
```

Output:

string using double quotes

A multiline

string

```
str1 = 'hello python' #string str
1
str2 = ' how are you' #string str
2
print (str1[0:2]) #printing first t
wo character using slice operat
or
print (str1[4]) #printing 4th cha
racter of the string
print (str1*2) #printing the stri
ng twice
print (str1 + str2) #printing the
concatenation of str1 and str2
```

```
he
o
hello pythonhello python
hello python how are you
```

- Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].
- We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

```
list1 = [1, "hi", "Python", 2]  
#Checking type of given list  
print(type(list1))
```

```
#Printing the list1  
print (list1)
```

```
# List slicing  
print (list1[3:])
```

```
# List slicing  
print (list1[0:2])
```

```
# List Concatenation using + operator  
print (list1 + list1)
```

```
# List repetition using * operator  
print (list1 * 3)
```

Output:

```
<class 'list'>
```

```
[1, 'hi', 'Python', 2]
```

```
[2]
```

```
[1, 'hi']
```

```
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]
```

```
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2, 1, 'hi',  
'Python', 2]
```

- A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().
- A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

```
tup = ("hi", "Python", 2)
# Checking type of tup
print (type(tup))
#Printing the tuple
print (tup)
# Tuple slicing
print (tup[1:])
print (tup[0:1])

# Tuple concatenation using + operator
print (tup + tup)

# Tuple repetition using * operator
print (tup * 3)

# Adding value to tup. It will throw an error.
t[2] = "hi"
```

A large orange circle is positioned on the left side of the slide, partially cut off by the edge.

Output:

```
<class 'tuple'>
('hi', 'Python', 2)
('Python', 2)
('hi',)
('hi', 'Python', 2, 'hi', 'Python', 2)
('hi', 'Python', 2, 'hi', 'Python', 2, 'hi', 'Python', 2)
```

Traceback (most recent call last):

File "main.py", line 14, in <module>

t[2] = "hi";

TypeError: 'tuple' object does not support item assignment

A yellow dashed line is located in the bottom right corner of the slide.

Dictionary

- Dictionary is an unordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type, whereas value is an arbitrary Python object.
- The items in the dictionary are separated with the comma (,) and enclosed in the curly braces {}.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}
```

```
# Printing dictionary
```

```
print (d)
```

```
# Accesing value using keys
```

```
print("1st name is "+d[1])
```

```
print("2nd name is "+ d[4])
```

```
print (d.keys())
```

```
print (d.values())
```

```
1st name is Jimmy
```

```
2nd name is mike
```

```
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
```

```
dict_keys([1, 2, 3, 4])
```

```
dict_values(['Jimmy', 'Alex', 'john', 'mike'])
```

Creating Empty set

```
set1 = set()
```

```
set2 = {'James', 2, 3, 'Python'}
```

Python Set is the unordered collection of the data type. It is iterable, mutable(can modify after creation), and has unique elements. In set, the order of the elements is undefined; it may return the changed sequence of the element.

The set is created by using a built-in function **set()**, or a sequence of elements is passed in the curly braces and separated by the comma. It can contain various types of values.

#Printing Set value

```
print(set2)
```

Adding element to the set

```
set2.add(10)
```

```
print(set2)
```

#Removing element from the set

```
set2.remove(2)
```

```
print(set2)
```

Output:

```
{3, 'Python', 'James', 2}
```

```
{'Python', 'James', 3, 2, 10}
```

```
{'Python', 'James', 3, 10}
```


Python Keywords

Python Keywords are special reserved words that convey a special meaning to the compiler/ interpreter. Each keyword has a special meaning and a specific operation. These keywords can't be used as a variable. Following is the List of Python Keywords.

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

assert - This keyword is used as the debugging tool in Python. It checks the correctness of the code. It raises an **AssertionError** if found any error in the code and also prints the message with an error.

```
a = 10
b = 0
print('a is dividing b
y Zero')
assert b != 0
print(a / b)
```

a is dividing by Zero

Runtime Exception:

Traceback (most recent call last):

File
"/home/40545678b342ce
3b70beb1224bed345f.py",
line 4, in

assert b != 0, "Divide by
0 error"

AssertionError: Divide by
0 error

def - This keyword is used to declare the function in Python.

```
def my_func(a,b):  
    c = a+b  
    print(c)  
my_func(10,20)
```

Output:

30

- **class** - It is used to represents the class in Python. The class is the blueprint of the objects. It is the collection of the variable and methods.

```
class Myclass:  
    #Variables.....  
    def function_name(self):  
        #statements.....
```

- **continue** - It is used to stop the execution of the current iteration.

```
a = 0  
while a < 4:  
    a += 1  
    if a == 2:  
        continue  
    print(a)
```

Output:

1
3
4

- **break** - It is used to terminate the loop execution and control transfer to the end of the loop.

```
for i in range(5):  
    if(i==3):  
        break  
    print(i)  
print("End of execution")
```

Output:

0

1

2

End of execution

- **If** - It is used to represent the conditional statement. The execution of a particular block is decided by if statement.

```
i = 18
```

```
if (1 < 12):
```

```
    print("I am less than 18")
```

Output:

I am less than 18