Python Programming

Unit-I

❖ Python Basics:

➢ Introduction to Python:

- Python is a general purpose, object-oriented, high level, and interpreted programming language.
- The implementation of Python was started in the December 1989 by **Guido Van Rossum** at CWI in Netherland.

Features of Python:

Python provides lots of features that are listed below.

- Easy to Learn and Use
  Python is easy to learn and use. It is developer-friendly and high level programming language.

- Interpreted Language
  Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

- Cross-platform Language
  Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

- Free and Open Source
  Python language is freely available at address. The source-code is also available. Therefore it is open source.

- Object-Oriented Language
  Python supports object oriented language and concepts of classes and objects come into existence.

- GUI Programming Support
  Graphical user interfaces can be developed using Python.

- Integrated
  It can be easily integrated with languages like C, C++, and JAVA etc.

Python Applications

Python is known for its general purpose nature that makes it applicable in almost each domain of software development. Python as a whole can be used in any sphere of development.

Here, we are specifying applications areas where python can be applied.

- *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, Feed parser etc. It also provides Frameworks such as Django, Pyramid, Flask etc to design and develop web based applications.

- *Desktop GUI Applications*

Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.

- *Software Development*

Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.

- *Scientific and Numeric*

Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

- *Business Applications*

Python is used to build Business applications like ERP and e-commerce systems. Tryton is a high level application platform.

- *Console Based Application*

We can use Python to develop console based applications. For example: IPython.

- Audio or Video based Applications

Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

➢ First Python Program:

Python provides us the two ways to run a program:

- Using Interactive interpreter prompt
- Using a script file

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.

To open the interactive mode, open the terminal (or command prompt) and type python (python3 in case if you have python2 and python3 both installed on your system).

Example:

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32

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

>>> print ("Hello World")

Hello World

>>> print ("Welcome to Python")

Welcome to Python
```

Using a script file:

Interpreter prompt is good to run the individual statements of the code. However if we want to execute multiple python statements at a time instead of executing one by one, then we can use script file.

We need to write our code into a file which can be executed later. For this purpose, open an editor like notepad, create a file named filename.py (python used .py extension) and write the code in it.

Example:

first.py

```
Print ("hello world"); #here, we have used print () function to print the message on the console.
```

To run this file named as first.py, we need to run the following command on the terminal.

$ python3 first.py

```
hello world
```

- **Variables:**

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a loosely typed language.

Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use lowercase letters for variable name. 'SUM' and 'sum' both are two different variables.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

Example: **Vardemo.py**

```
a=10
b="Madhu"
c=12.5
print(a)
print(b)
print(c)
```

output:

$python3 Vardemo.py

```
10
Madhu
12.5
```

Python allows us to assign a value to multiple variables and multiple values to multiple variables in a single statement which is also known as multiple assignment.

Example: **Vardemo1.py**

```
x=y=z=50
print x
print y
print z
a,b,c=5,10,15
print a
print b
print c
```

output:

$python3 Vardemo1.py

```
50
50
50
5
10
15
```

- **Comments:**

Comments in Python can be used to explain any program code. It can also be used to hide the code as well.

Comment is not a part of the program, but it enhances the interactivity of the program and makes the program readable.

Python supports two types of comments:

Single Line Comment:

In case user wants to specify a single line comment, then comment must start with '#'

Eg:

```
# This is single line comment.
print "Hello Python"
```

Output:

Hello Python

Multi Line Comment:

Multi lined comment can be given inside triple quotes.

eg:

```
''' This
    Is
    Multiline comment'''
```

eg:

```
#single line comment
print "Hello Python"
'''This is
Multiline comment'''
```

Output:

Hello Python

➢ Operators:

The operator can be defined as a symbol which is responsible for a particular operation between two operands.

Python provides a variety of operators described as follows.

- Arithmetic operators
- Comparison operators
- Identity Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators

❖ Arithmetic operators

Arithmetic operators are used to perform arithmetic operations between two operands.

It includes

+ (addition)	eg: a=20; b=10 then a+b=30
- (subtraction)	eg: a=20; b=10 then a-b=10
***** (multiplication)	eg: a=20; b=10 then a*b=200
/ (divide)	eg: a=20; b=10 then a/b=2
%(reminder)	eg: a=20; b=10 then a%b=0
// (floor division)	eg: a=24; b=7 then a//b=3
** (exponent)	eg: a=2; b=3 then a**b=8

❖ Comparison operators

Comparison operators are used to comparing the value of the two operands and returns boolean true or false accordingly.

It includes

== (Equal to)

!= (Not equal to)

<= (Less than or equal)

>= (Greater than or equal)

< (Less than)

> (Greater than)

❖ Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location

It includes

is (Returns true if both variables are the same object)

is not (Returns true if both variables are not the same object)

❖ **Assignment operators**

The assignment operators are used to assign the value of the right expression to the left operand.

It includes

= (Assigns to)

+= (Assignment after Addition)

-= (Assignment after Subtraction)

***=** (Assignment after Multiplication)

/= (Assignment after Division)

%= (Assignment after Modulus)

****=** (Assignment after Exponent)

//= (Assignment after floor division)

❖ **Bitwise operators**

The bitwise operators perform bit by bit operation on the values of the two operands.

It includes

& (binary and)

| (binary or)

^ (binary xor)

~ (negation)

<< (left shift)

>> (right shift)

❖ **Logical Operators**

The logical operators are used primarily in the expression evaluation to make a decision.

It includes

and (logical and)

or (logical or)

not (logical not)

❖ **Membership Operators**

Membership operators are used to check the membership of value inside a data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

It includes

in (it returns true if the first operand is found in the second operand (list, tuple, or dictionary))

not in (it returns true if the first operand is not found in the second operand (list, tuple, or dictionary))

- **Data Types:**

Variables can hold values of different data types. Python is a dynamically typed or loosely typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python provides us the **type ()** function which enables us to check the type of the variable.

Python provides following standard data types, those are

- Numbers
- String

- **Numbers:**

Number stores numeric values. Python creates Number type variable when a number is assigned to a variable.

There are three numeric types in Python:

1. int
2. float
3. complex

1. int:

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example:

```
a=10
b=-12
c=123456789
```

2. float:

Float or "floating point number" is a number, positive or negative, containing one or more decimals.

Example:

```
X=1.0
Y=12.3
Z=-13.4
```

3. complex:

Complex numbers are written with a "j" as the imaginary part.

Example:

```
A=2+5j
B=-3+4j
C=-6j
```

❖ **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.

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

Example:
S1='Welcome'
S2="NNRG"
S3="'College"'

Example: datatypesdemo.py

```
a=10
b="NNRG"
c = 10.5
d=2.14j
print("Data type of Variable a :",type(a))
print("Data type of Variable b :",type(b))
print("Data type of Variable c :",type(c))
print("Data type of Variable d :",type(d))
```

Output:

python datatypesdemo.py

```
Datatype of Variable a : <class 'int'>
Datatype of Variable b : <class 'str'>
Datatype of Variable c : <class 'float'>
Datatype of Variable d : <class 'complex'>
```

➢ **Type Conversion:**

Python provides type conversion functions to directly convert one data type to another. Python supports following functions

- **int ()** : This function converts any data type to integer**.**
- **float()** : This function is used to convert any data type to a floating point number.
- **str()** : This function is used to convert any data type to a string.

Example:

```
x = int(2.8)
y = int("3")
z = float(2)
s = str(10)
print(x)
print(y)
print(z)
```

output:

```
2
3
2.0
10
```

- **Conditional Statements:**

Conditional Statements in Python performs different computations or actions depending on conditions.

In python, the following are conditional statements

- if
- if –else
- if – elif –else

Indentation:

For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code.

In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.

Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation.

If statement:

The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.

Syntax:

```
if expression:
    statement
```

Example: **ifdemo.py**

```
a = 33
b = 200
if b > a:
   print ("b is greater than a")
```

Output:

python ifdemo.py

b is greater than a

Note:

input () function is used to get input from user.

Example: a=input ("Enter a value")

If – else statement:

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

Syntax:

```
if condition:
    #block of statements
else:
    #another block of statements (else-block)
```

Example:

ifelsedemo.py

```
age = int (input("Enter your age : "))
if age>=18:
   print("You are eligible to vote !!")
else:
   print("Sorry! you have to wait !!")
```

output:

python ifelsedemo.py

Enter your age: 19

You are eligible to vote!!

If –elif -else statement:

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.

We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

Syntax:

```
if expression 1:
    # block of statements

elif expression 2:
    # block of statements

elif expression 3:
    # block of statements

else:
    # block of statements
```

Example:

grade.py

```
marks = int(input("Enter the marks :"))
if marks > 85 and marks <= 100:
   print("Congrats ! you scored grade A ...")
elif marks > 60 and marks <= 85:
   print("You scored grade B + ...")
elif marks > 40 and marks <= 60:
   print("You scored grade B ...")
elif (marks > 30 and marks <= 40):
   print("You scored grade C ...")
else:
   print("Sorry you are fail ?")
```

output:

python grade.py

Enter the marks: 70

You scored grade B +...

Example:

maxnum.py

```
a=int(input("Enter a value : "))
b=int(input("Enter b value : "))
c=int(input("Enter c value : "))
if (a>b) and (a>c):
        print("Maximum value is :",a)
elif (b>a) and (b>c):
        print("Maximum value is :",b)
else:
        print("Maximum value is :",c)
```

output:

python maxnum.py

Enter a value: 10

Enter b value: 14

Enter c value: 9

Maximum value is: 14

- **Loop Statements:**

Sometimes we may need to alter the flow of the program. If the execution of a specific code may need to be repeated several numbers of times then we can go for loop statements.

For this purpose, the python provide various types of loops which are capable of repeating some specific code several numbers of times. Those are,

- while loop
- for loop

while loop:

With the while loop we can execute a set of statements as long as a condition is true. The while loop is mostly used in the case where the number of iterations is not known in advance.

Syntax:

```
while expression:
    Statement(s)
```

Example: **whiledemo.py**

```
i=1;
while i<=3:
   print(i);
   i=i+1;
```

Output: **python** whiledemo.py

1

2

3

- **Using else with while loop**

Python enables us to use the while loop with the else block also. The else block is executed when the condition given in the while statement becomes false.

Example: **wedemo.py**

```
i=1;
while i<=3:
   print (i)
   i=i+1;
else: print("The while loop terminated");
```

Output: **python** wedemo.py

1

2

3

The while loop terminated

for loop:

The for loop in Python is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

Syntax:

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

Example: **fordemo.py**

```
i=1
n=int(input("Enter n value : "))
for i in range(i,n+1):
    print(i,end = ' ')
```

Output: **python** fordemo.py

```
Enter n value: 5
1 2 3 4 5
```

- **Using else with for loop**

Python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

Example: **fedemo.py**

```
for i in range(0,5):
    print(i)
else:print("for loop completely exhausted, since there is no break.");
```

Output: **python** fedemo.py

```
0
1
2
3
4
```

Note:

The **range ()** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example:

range (6) means the values from 0 to 5.

range (2,6) means the values from 2 to 5.

Examples:

table.py

```
# Program to printing the table of the given number
i=1;
num = int(input("Enter a number:"));
for i in range(1,11):
    print("%d X %d = %d"%(num,i,num*i))
```

Output: **python** table.py

```
Enter a number: 10
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

- **Nested for loop in python**

Python allows us to nest any number of for loops inside a for loop. The inner loop is executed n number of times for every iteration of the outer loop.

Syntax:

```
for iterating_var1 in sequence:
    for iterating_var2 in sequence:
        #block of statements
#other statements
```

Example: **nsfordemo.py**

```
# Program to printing the rows of stars of the given number
n = int(input("Enter the number of rows you want to print : "))
for i in range(0,n):
        for j in range(0,i+1):
                print("*",end="")
        print()
```

Output: **python** nsfordemo.py

```
Enter the number of rows you want to print: 4
*
**
***
****
```

- **break statement:**

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops.

In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition.

Syntax:

```
break
```

Example:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Output:

```
1
2
3
```

- **continue statement:**

The continue statement in python is used to bring the program control to the beginning of the loop.

The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

Syntax:

```
continue
```

Example: **continuedemo.py**

```
str =input("Enter any String : ")
for i in str:
    if i == 'a':
        continue;
    print(i,end=" ");
```

Output: **python** continuedemo.py

```
Enter any String: madhu
m d h u
```

- **Strings:**

In python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

Example:

str=”Hello Python”

In python, strings are treated as the sequence of characters which means that python doesn't support the character data type instead a single character written as 'p' is treated as the string of length 1.

Strings indexing

Like other languages, the indexing of the python strings starts from 0. For example, the string "HELLO" is indexed as given in the below figure.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0]=H

str[1]=E

str[4]=O

A string can only be replaced with a new string since its content cannot be partially replaced. Strings are immutable in python.

str = "HELLO"
str[0] = "B" → this is not possible

str = "HELLO"
str = "BYE" → this is possible

String Operators:

+	It is known as concatenation operator used to join the strings.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. To define any string as a raw string, the character r or R is followed by the string. Such as "hello \n python".
%	It is used to perform string formatting. It makes use of the format specifies used in C programming like %d or %f to map their values in python.

Example: **stringopdemo.py**

```
str1 = "Hello"
str2 = " World"
print(str1*3) # prints HelloHelloHello
print(str1+str2)# prints Hello world
print(str1[4]) # prints o
print(str1[2:4]) # prints ll
print('w' in str1) # prints false as w is not present in str1
print('Wo' not in str2) # prints false as Wo is present in str2.
print(r'Hello\n world') # prints Hello\n world as it is written
print("The string str1 : %s"%(str1)) # prints The string str : Hello
```

Output:

```
python  stringopdemo.py
HelloHelloHello
Hello World
o
ll
False
False
Hello\n world
The string str1 : Hello
```

String functions:

Python provides various in-built functions that are used for string handling. Those are

- lower()
- upper()
- replace()
- split()
- find()
- index()
- isalnum()
- isdigit()
- isnumeric()
- islower()
- isupper()
- join()
- len()

- **lower ():**

In python, **lower()** method returns all characters of given string in lowercase.

Syntax:

lower()

Example:

str=”PyTHOn”

print(str.lower())

output:

python

- **upper ():**

In python **upper()** method converts all the character to uppercase and returns a uppercase string.

Syntax:

upper()

Example:

str=”PyTHOn”

print(str.upper())

output:

PYTHON

- **replace():**

In python **replace()** method replaces the old sequence of characters with the new sequence. If the optional argument *count* is given, only the first *count* occurrences are replaced.

Syntax:

replace(old, new[, count])

old : An old string which will be replaced.

new : New string which will replace the old string.

count : The number of times to process the replace.

Example:

```
str = "Java is Object-Oriented and Java is Portable "
# Calling function
str2 = str.replace("Java","Python")  # replaces all the occurences
# Displaying result
print("Old String: \n",str)
print("New String: \n",str2)
str3 = str.replace("Java","Python",1) # replaces first occurance only
# Displaying result
print("\n Old String: \n",str)
print("New String: \n",str3)
str4 = str1.replace(str1,"Python is Object-Oriented and Portable")
print("New String: \n",str4)
```

output:

Old String:

Java is Object-Oriented and Java is Portable

New String:

Python is Object-Oriented and Python is Portable

Old String:

Java is Object-Oriented and Java is Portable

New String:

Python is Object-Oriented and Java is Portable

New String:

Python is Object-Oriented and Portable

- **split():**

In python **split()** method splits the string into a comma separated list. It separates string based on the separator delimiter. The string splits according to the space if the delimiter is not provided.

Syntax:

split([sep="delimiter"])

sep: It specifies the delimiter as separator.

Example:

```
str1 = "Java is a programming language"
str2 = str1.split()
# Displaying result
print(str1)
print(str2)
str1 = "Java,is,a,programming,language"
str2 = str1.split(sep=',')
# Displaying result
print(str1)
print(str2)
```

output:

Java is a programming language

['Java', 'is', 'a', 'programming', 'language']

Java, is, a, programming, language

['Java', 'is', 'a', 'programming', 'language']

- **find()**

In python **find()** method finds substring in the whole string and returns index of the first match. It returns -1 if substring does not match.

Syntax:

find(sub[, start[,end]])

sub :it specifies sub string.

start :It specifies start index of range.

end : It specifies end index of range.

Example:

```
str1 = "python is a programming language"
str2 = str1.find("is")
str3 = str1.find("java")
str4 = str1.find("p",5)
str5 = str1.find("i",5,25)
print(str2,str3,str4,str5)
```

output:

```
7   -1   12   7
```

- **index()**

In python **index()** method is same as the **find()** method except it returns error on failure. This method returns index of first occurred substring and an error if there is no match found.

Syntax:

index(sub[, start[,end]])

sub :it specifies sub string.

start :It specifies start index of range.

end : It specifies end index of range.

Example:

```
str1 = "python is a programming language"
str2 = str1.index("is")
print(str2)
str3 = str1.index("p",5)
print(str3)
str4 = str1.index("i",5,25)
print(str4)
str5 = str1.index("java")
print(str5)
```

output:

```
7
12
7
Substring not found
```

- **isalnum():**

In python **isalnum()** method checks whether the all characters of the string is alphanumeric or not. A character which is either a letter or a number is known as alphanumeric. It does not allow special chars even spaces.

Syntax:

isalnum()

Example:

```
str1 = "python"
str2 = "python123"
str3 = "12345"
str4 = "python@123"
str5 = "python 123"
print(str1. isalnum())
print(str2. isalnum())
print(str3. isalnum())
print(str4. isalnum())
print(str5. isalnum())
```

Output:

```
True
True
True
False
False
```

- **isdigit():**

In python **isdigit()** method returns True if all the characters in the string are digits. It returns False if no character is digit in the string.

Syntax:

isdigit()

Example:

```
str1 = "12345"
str2 = "python123"
str3 = "123-45-78"
str4 = "IIIV"
str5 = "/u00B23" # 2₃
str6 = "/u00BD"  # 1/2
print(str1.isdigit())
print(str2.isdigit())
print(str3.isdigit())
print(str4.isdigit())
print(str5.isdigit())
print(str6.isdigit())
```

Output:

```
True
False
False
False
True
False
```

- **isnumeric():**

In python **isnumeric()** method checks whether all the characters of the string are numeric characters or not. It returns True if all the characters are numeric, otherwise returns False.

Syntax:

isnumeric()

Example:

```
str1 = "12345"
str2 = "python123"
str3 = "123-45-78"
str4 = "IIIV"
str5 = "/u00B23" # 2₃
str6 = "/u00BD"  # 1/2
print(str1.isnumeric())
print(str2.isnumeric())
print(str3.isnumeric())
print(str4.isnumeric())
print(str5.isnumeric())
print(str6.isnumeric())
```

Output:

```
True
False
False
False
True
True
```

- **islower():**

In python string **islower()** method returns True if all characters in the string are in lowercase. It returns False if not in lowercase.

Syntax:

islower()

Example:

```
str1 = "python"
str2="PytHOn"
str3="python3.7.3"
print(str1.islower())
print(str2.islower())
print(str3.islower())
```

Output:

```
True
False
True
```

- **isupper():**

In python string **isupper()** method returns True if all characters in the string are in uppercase. It returns False if not in uppercase.

Syntax:

isupper()

Example:

```
str1 = "PYTHON"
str2="PytHOn"
str3="PYTHON 3.7.3"
print(str1.isupper())
print(str2.isupper())
print(str3.isupper())
```

Output:

```
True
False
True
```

- **join():**

Python **join()** method is used to concat a string with iterable object. It returns a new string which is the concatenation of the strings in iterable.

It allows various iterables like: List, Tuple, String etc.

Syntax:

join(iterable object)

Example:

```
str1 = ":"   # string
str2 = "NNRG"    # iterable object
str3= str1.join(str2)
print(str3)
```

Output:

```
N:N:R:G
```

- **len():**

In python string **len()** method returns length of the given string.

Syntax:

len(string)

Example:

```
str1="Python Language"
print(len(str1))
```

Output:

15

- **List:**

In python, a list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets [].

Syntax:

List=[value1, value2, value3,….]

Example:

```
L0 = []            #creates empty list
L1 = [123,"python", 3.7]
L2 = [1, 2, 3, 4, 5, 6]
L3 = [ "Madhu","Naveen","Python"]
Print(L0)
print(L1)
print(L2)
print(L3)
```

output:

```
[]
[123, 'python', 3.7]
[1, 2, 3, 4, 5, 6]
['Madhu', 'Naveen', 'Python']
```

List indexing:

The indexing are processed in the same way as it happens with the strings. The elements of the list can be accessed by using the slice operator [].

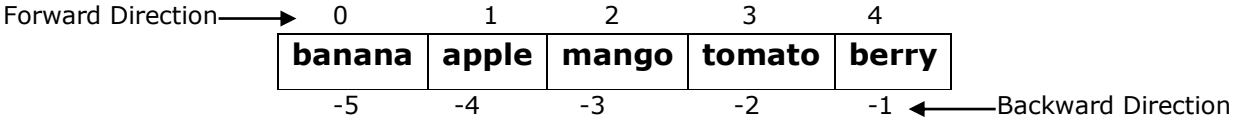
The index starts from 0, the first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

mylist=[‘banana’,’apple’,’mango’,’tomato’,’berry’]

banana	apple	mango	tomato	berry
0	1	2	3	4

mylist[0]="banana" mylist[1:3]=["apple","mango"]
mylist[2]="mango"

Unlike other languages, python provides us the flexibility to use the negative indexing also. The negative indices are counted from the right. The last element (right most) of the list has the index -1, its adjacent left element is present at the index -2 and so on until the left most element is encountered.



mylist[-1]="berry" mylist[-4:-1]=["apple",mango","tomoto"]
mylist[-3]="mango"

Updating List values:

Python allows us to modify the list items by using the slice and assignment operator. Python also provide us the append() method which can be used to add values to the list.

Example:

```
num=[1,2,3,4,5]
print(num)
num[2]=30
print(num)
num[1:3]=[25,36]
print(num)
num[4]="NNRG"
print(num)
```

Output:

```
[1, 2, 3, 4, 5]
[1, 2, 30, 4, 5]
[1, 25, 36, 4, 5]
[1, 25, 36, 4, 'NNRG']
```

Python allows us to delete the list items by using the **del** keyword. Python also provides us the remove() method if we do not know which element is to be deleted from the list.

Example:

```
num=[1,2,3,4,5]
print(num)
del num[1]
print(num)
del num[1:3]
print(num)
```

Output:

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

List Operators:

Python provides various list operations to perform different operations on list. Those are,

+	It is known as concatenation operator used to concatenate two lists
*	It is known as repetition operator. It concatenates the multiple copies of the same list.
[]	It is known as slice operator. It is used to access the list item from list.
[:]	It is known as range slice operator. It is used to access the range of list items from list.
In	It is known as membership operator. It returns if a particular item is present in the specified list.
not in	It is also a membership operator and It returns true if a particular list item is not present in the list.

Example:

```
num=[1,2,3,4,5]
lang=['python','c','java','php']
print(num+lang)
print(num*2)
print(lang[2])
print(lang[1:4])
print('cpp' in lang)
print(6 not in num)
```

Output:

```
[1, 2, 3, 4, 5, 'python', 'c', 'java', 'php']
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
java
['c', 'java', 'php']
False
True
```

Iterating a List

A list can be iterated by using a for - in loop. A simple list containing four strings can be iterated as follows.

Example:

```
lang=['python','c','java','php']
print("The list items are \n")
for i in lang:
        print(i)
```

Output:

```
The list items are
python
c
java
php
```

List Functions:

Python provides the following built-in functions which can be used with the lists.

len(list)	sort()/sorted()
max(list)	reverse()
min(list)	count()
list(sequence)	index()
sum(list)	insert()
append()	pop()
remove()	clear()

- **len(list):**

In Python **len()** is used to find the length of list,i.e it returns the number of items in the list.

Syntax:

len(list)

Example:

```
num=[1,2,3,4,5,6]
print("length of list :",len(num))
```

Output:

```
length of list : 6
```

- **max(list):**

In Python **max()** is used to find maximum value in the list

Syntax:

max(list)

Example:

```
num=[1,2,3,4,5,6]
lang=['java','c','python','cpp']
print("Max of list :",max(num))
print("Max of list :",max(lang))
```

Output:

```
Max of list : 6
Max of list : python
```

- **min(list):**

In Python **min()** is used to find minimum value in the list

Syntax:

min(list)

Example:

```
num=[1,2,3,4,5,6]
lang=['java','c','python','cpp']
print("Min of list :",min(num))
print("Min of list :",min(lang))
```

Output:

```
Min of list : 1
Min of list : c
```

- **sum(list):**

In python, **sum(list)** function returns sum of all values in the list. List values must in number type.

Syntax:

sum(list)

Example:

```
num=[1,2,3,4,5,6]
print("sum of list items :",sum(num))
```

Output:

```
sum of list items: 21
```

- **sorted(list):**

In python, **sorted(list)** function is used to sort all items of list in an ascending order.

Syntax:

sorted(list)

Example:

```
num=[1,3,2,4,6,5]
lang=['java','c','python','cpp']
print(sorted(num))
print(sorted(lang))
```

Output:

```
[1, 2, 3, 4, 5, 6]
['c', 'cpp', 'java', 'python']
```

- **list (sequence):**

The **list()** method takes sequence types and converts them to lists. This is used to convert a given string or tuple into list.

Syntax:

list(sequence)

Example:

```
str="python"
list1=list(str)
print(list1)
```

Output:

```
['p', 'y', 't', 'h', 'o', 'n']
```

- **append ():**

In python **append()** method adds an item to the end of the list.

Syntax:

list.append(item)

-item may be number,string,list and etc

Example:

```
num=[1,2,3,4,5]
lang=['python','c','java','php']
num.append(6)
print(num)
lang.append("cpp")
print(lang)
```

Output:

```
[1, 2, 3, 4, 5, 6]
['python', 'c', 'java', 'php', 'cpp']
```

- **remove():**

In python **remove()** method removes the first item from the list which is equal to the passed value. It throws an error if the item is not present in the list.

Syntax:

list.remove(item)

-item may be number,string,list and etc

Example:

```
num=[1,2,3,4,5]
lang=['python','c','java','php','c']
num.remove(2)
print(num)
lang.remove("c") # first occurence will remove
print(lang)
lang.remove("cpp")
print(lang)
```

Output:

```
[1, 3, 4, 5]
['python', 'java', 'php', 'c']
ValueError: list.remove(x): x not in list
```

- **sort():**

In python **sort()** method sorts the list elements. It also sorts the items into descending and ascending order. It takes an optional parameter 'reverse' which sorts the list into descending order. By default, list sorts the elements into ascending order.

Syntax:

list.sort ([reverse=true])

-reverse, which displays items in descending

Example:

```
lang = ['p', 'y', 't', 'h', 'o','n'] # Char list
even = [6,8,2,4,10] # int list
print(lang)
print(even)
lang.sort()
even.sort()
print("\nAfter Sorting:\n")
print(lang)
print(even)
print("In Descending Order :\n")
even.sort(reverse=True)
print(even)
```

Output:

```
['p', 'y', 't', 'h', 'o', 'n']
[6, 8, 2, 4, 10]
After Sorting:
['h', 'n', 'o', 'p', 't', 'y']
[2, 4, 6, 8, 10]
In Descending Order :
[10, 8, 6, 4, 2]
```

- **reverse():**

In python **reverse()** method reverses elements of the list. If the list is empty, it simply returns an empty list. After reversing the last index value of the list will be present at 0 index.

Syntax:

list. reverse ()

Example:

```
lang = ['p', 'y', 't', 'h', 'o','n']
lang = ['p', 'y', 't', 'h', 'o','n']
print("After reverse")
lang.reverse()
print(lang)
```

Output:

After reverse

['n', 'o', 'h', 't', 'y', 'p']

- **count():**

In python **count()** method returns the number of times element appears in the list. If the element is not present in the list, it returns 0.

Syntax:

list. count (item)

Example:

```
num=[1,2,3,4,3,2,2,1,3,4,5,7,8]
cnt=num.count(2)
print("Count of 2 is:",cnt)
cnt=num.count(10)
print("Count of 10 is:",cnt)
```

Output:

Count of 2 is: 3

Count of 10 is : 0

- **index():**

In python **index ()** method returns index of the passed element. This method takes an argument and returns index of it. If the element is not present, it raises a ValueError.

If list contains duplicate elements, it returns index of first occurred element.

This method takes two more optional parameters start and end which are used to search index within a limit.

Syntax:

list.index(x[, start[, end]])

Example:

```
lang = ['p', 'y', 't', 'h', 'o','n','p','r','o','g','r','a','m']
print("index of t is:",lang.index('t'))
print("index of p is:",lang.index('p'))
print("index of p is:",lang.index('p',3,10))
print("index of p is:",lang.index('z'))
```

Output:

index of t is: 2

index of p is: 0

index of p is: 6

ValueError: 'z' is not in list

- **insert():**

In python **insert()** method inserts the element at the specified index in the list. The first argument is the index of the element before which to insert the element.

Syntax:

list.insert(i,x)

i : index at which element would be inserted.
x : element to be inserted.

Example:

```
num=[10,20,30,40,50]
num.insert(4,60)
print("updated list is :",num)
num.insert(7,70)
print("updated list is :",num)
```

Output:

```
updated list is : [10, 20, 30, 40, 60, 50]
updated list is : [10, 20, 30, 40, 60, 50, 70]
```

- **pop():**

In python **pop()** element removes an element present at specified index from the list. It returns the popped element.

Syntax:

list.pop([i])

Example:

```
num=[10,20,30,40,50]
num.pop()
print("updated list is :",num)
num.pop(2)
print("updated list is :",num)
num.pop(7)
print("updated list is :",num)
```

Output:

```
updated list is : [10, 20, 30, 40]
updated list is : [10, 20, 40]
IndexError: pop index out of range
```

- **clear():**

In python **clear()** method removes all the elements from the list. It clears the list completely and returns nothing.

Syntax:

list.clear()

Example:

```
num=[10,20,30,40,50]
num.clear()
print("After clearing ",num)
```

Output:

After clearing [ ]

- **Tuple:**

In Python **Tuple** is used to store the sequence of immutable python objects. 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.

A tuple can be written as the collection of comma-separated values enclosed with the small brackets.

Syntax:

Tuple= (value1, value2…)

Example:

```
T1 = ()
T2 = (10, 30, 20, 40, 60)
T3 = ("Apple", "Banana", "Orange")
T4 = (501,"abc", 19.5)
T5 = (90,)
print(T1)
print(T2)
print(T3)
print(T4)
print(T5)
```

Output:

```
()
(10, 30, 20, 40, 60)
('Apple', 'Banana', 'Orange')
(501, 'abc', 19.5)
(90,)
```

Tuple indexing:

The indexing and slicing in tuple are similar to lists. The indexing in the tuple starts from 0 and goes to length (tuple) - 1.

The items in the tuple can be accessed by using the slice operator. Python also allows us to use the colon operator to access multiple items in the tuple.

mytuple=('banana','apple','mango','tomato','berry')

banana	apple	mango	tomato	berry
0	1	2	3	4

mytuple[0]="banana" mytuple[1:4]=("apple","mango","tomato")
mytuple[2]="mango"

Unlike other languages, python provides us the flexibility to use the negative indexing also. The negative indices are counted from the right. The last element (right most) of the tuple has the index -1, its adjacent left element is present at the index -2 and so on until the left most elements is encountered.

Forward Direction →

0	1	2	3	4
banana	apple	mango	tomato	berry
-5	-4	-3	-2	-1

← Backward Direction

mytuple [-1]="berry" mytuple[-4:-1]=("apple",mango","tomoto")
mytuple[-3]="mango"

Unlike lists, the tuple items cannot be updated or deleted as tuples are immutable. To delete an entire tuple, we can use the **del** keyword with the tuple name.

Example:

```
mytuple=('banana','apple','mango','tomato','berry')
mytuple[3]="Orange"          # 'tuple' object does not support item assignment
print(mytuple)
del mytuple[3]               # 'tuple' object doesn't support item deletion
print(mytuple)
del mytuple                  #deletes entire tuple
```

Output:

'tuple' object does not support item assignment

'tuple' object doesn't support item deletion

Tuple Operators:

Python provides various tuple operations to perform different operations on tuple. Those are,

+	It is known as concatenation operator used to concatenate two tuples
*	It is known as repetition operator. It concatenates the multiple copies of the same tuple.
[]	It is known as slice operator. It is used to access the item from tuple.
[:]	It is known as range slice operator. It is used to access the range of items from tuple.
In	It is known as membership operator. It returns if a particular item is present in the specified tuple.
not in	It is also a membership operator and It returns true if a particular item is not present in the tuple.

Example:

```
num=(1,2,3,4,5)
lang=('python','c','java','php')
print(num+lang)
print(num*2)
print(lang[2])
print(lang[1:4])
print('cpp' in lang)
print(6 not in num)
```

Output:

```
(1, 2, 3, 4, 5, 'python', 'c', 'java', 'php')
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
java
('c', 'java', 'php')
True
True
```

Iterating a Tuple

A tuple can be iterated by using a for - in loop. A simple tuple containing four strings can be iterated as follows.

Example:

```
lang=('python','c','java','php')
print("The tuple items are \n")
for i in lang:
        print(i)
```

Output:

```
The tuple items are
python
c
java
php
```

List Vs Tuple

List	Tuple
The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
The List is mutable.	The tuple is immutable.
The List has the variable length.	The tuple has the fixed length.
The list provides more functionality than tuple.	The tuple provides less functionality than the list.

Tuple Functions:

Python provides the following built-in functions which can be used with the tuples.

- len(tuple)
- max(tuple)
- min(tuple)
- tuple(sequence)
- sum(tuple)
- sorted(tuple)
- index()
- count()

- **len(tuple):**

In Python **len()** is used to find the length of tuple,i.e it returns the number of items in the tuple.

Syntax:

len(tuple)

Example:

```
num=(1,2,3,4,5,6)
print("length of tuple :",len(num))
```

Output:

```
length of tuple : 6
```

- **max(tuple):**

In Python **max()** is used to find maximum value in the tuple.

Syntax:

max(tuple)

Example:

```
num=(1,2,3,4,5,6)
lang=('java','c','python','cpp')
print("Max of tuple :",max(num))
print("Max of tuple :",max(lang))
```

Output:

```
Max of tuple : 6
Max of tuple : python
```

- **min(tuple):**

In Python **min()** is used to find minimum value in the tuple.

Syntax:

min(tuple)

Example:

```
num=(1,2,3,4,5,6)
lang=('java','c','python','cpp')
print("Min of tuple :",min(num))
print("Min of tuple :",min(lang))
```

Output:

```
Min of tuple: 1
Min of tuple : c
```

- **sum(tuple):**

In python, **sum(tuple)** function returns sum of all values in the tuple. Tuple values must in number type.

Syntax:

sum(tuple)

Example:

```
num=(1,2,3,4,5,6)
print("sum of tuple items :",sum(num))
```

Output:

```
sum of tuple items: 21
```

- **sorted(tuple):**

In python, **sorted (tuple)** function is used to sort all items of tuple in an ascending order. It also sorts the items into descending and ascending order. It takes an optional parameter 'reverse' which sorts the tuple into descending order.

Syntax:

sorted (tuple[,reverse=True])

Example:

```
num=(1,3,2,4,6,5)
lang=('java','c','python','cpp')
print(sorted(num))
print(sorted(lang))
print(sorted(num,reverse=True))
```

Output:

```
(1, 2, 3, 4, 5, 6)
('c', 'cpp', 'java', 'python')
(6, 5, 4, 3, 2, 1)
```

- **tuple (sequence):**

The **tuple()** method takes sequence types and converts them to tuples. This is used to convert a given string or list into tuple.

Syntax:

tuple(sequence)

Example:

```
str="python"
tuple1=tuple(str)
print(tuple1)
num=[1,2,3,4,5,6]
tuple2=tuple(num)
print(tuple2)
```

Output:

```
('p', 'y', 't', 'h', 'o', 'n')
(1, 2, 3, 4, 5, 6)
```

- **count():**

In python **count()** method returns the number of times element appears in the tuple. If the element is not present in the tuple, it returns 0.

Syntax:

tuple.count (item)

Example:

```
num=(1,2,3,4,3,2,2,1,3,4,5,7,8)
cnt=num.count(2)
print("Count of 2 is:",cnt)
cnt=num.count(10)
print("Count of 10 is:",cnt)
```

Output:

```
Count of 2 is: 3
Count of 10 is : 0
```

- **index():**

In python **index ()** method returns index of the passed element. This method takes an argument and returns index of it. If the element is not present, it raises a ValueError.

If tuple contains duplicate elements, it returns index of first occurred element.

This method takes two more optional parameters start and end which are used to search index within a limit.

Syntax:

tuple.index(x[, start[, end]])

Example:

```
lang = ('p', 'y', 't', 'h', 'o','n','p','r','o','g','r','a','m')
print("index of t is:",lang.index('t'))
print("index of p is:",lang.index('p'))
print("index of p is:",lang.index('p',3,10))
print("index of p is:",lang.index('z'))
```

Output:

```
index of t is: 2
index of p is: 0
index of p is: 6
ValueError: 'z' is not in tuple.
```

- **Set:**

In python, the set can be defined as the unordered collection of various items enclosed within the curly braces. The elements of the set cannot be duplicate. The elements of the python set must be immutable.

Unlike other collections in python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together or we can get the list of elements by looping through the set.

Creating a set

The set can be created by enclosing the comma separated items with the curly braces.

Syntax:

```
Set={value1, value2….}
```

Example:

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}
print(Days)
print(type(Days))
print("Looping through the set elements ... ")
for i in Days:
   print(i)
```

Output:

```
{'Wednesday', 'Tuesday', 'Sunday', 'Friday', 'Thursday', 'Saturday', 'Monday'}
<class 'set'>
Looping through the set elements...
Wednesday
Tuesday
Sunday
Friday
Thursday
Saturday
Monday
```

Set Operators:
In Python, we can perform various mathematical operations on python sets like union, intersection, difference, etc

- **Union (|) Operator:**

The union of two sets are calculated by using the or (|) operator. The union of the two sets contains the all the items that are present in both the sets.

Example:

```
Days1={"Mon","Tue","Wed","Sat"}
Days2={"Thr","Fri","Sat","Sun","Mon"}
print(Days1 | Days2)
```

Output:

```
{'Thr', 'Fri', 'Sun', 'Tue', 'Wed', 'Mon', 'Sat'}
```

- **Intersection (&) Operator:**

The & (intersection) operator is used to calculate the intersection of the two sets in python. The intersection of the two sets are given as the set of the elements that common in both sets.

Example:

```
Days1={"Mon","Tue","Wed","Sat"}
Days2={"Thr","Fri","Sat","Sun","Mon"}
print(Days1 & Days2)
```

Output:

```
{'Mon', 'Sat'}
```

- **Difference (-) Operator:**

The difference of two sets can be calculated by using the subtraction (-) operator. The resulting set will be obtained by removing all the elements from set 1 that are present in set 2.

Example:

```
Days1={"Mon","Tue","Wed","Sat"}
Days2={"Thr","Fri","Sat","Sun","Mon"}
print(Days1 & Days2)
```

Output:

```
{'Tue', 'Wed'}
```

Set Functions:
Python contains the following methods to be used with the sets. Those are

- len(set)
- max(set)
- min(set)
- sum(set)
- sorted(set)
- set()
- add()
- update()
- discard()
- remove()
- pop()
- clear()
- union()
- intersection()
- difference()
- issubset()
- issuperset()

- **len(set):**

In Python **len()** is used to find the length of set,i.e it returns the number of items in the set.

Syntax:

len(set)

Example:

```
num={1,2,3,4,5,6}
print("length of set :",len(num))
```

Output:

```
length of set : 6
```

- **max(set):**

In Python **max()** is used to find maximum value in the set.

Syntax:

max(set)

Example:

```
num={1,2,3,4,5,6}
lang={'java','c','python','cpp'}
print("Max of set :",max(num))
print("Max of set :",max(lang))
```

Output:

```
Max of set : 6
Max of set : python
```

- **min(set):**

In Python **min()** is used to find minimum value in the set.

Syntax:

min(set)

Example:

```
num={1,2,3,4,5,6}
lang={'java','c','python','cpp'}
print("Min of set :",min(num))
print("Min of set :",min(lang))
```

Output:

```
Min of set: 1
Min of set : c
```

- **sum(set):**

In python, **sum(set)** function returns sum of all values in the set. Set values must in number type.

Syntax:

sum(set)

Example:

```
num={1,2,3,4,5,6}
print("sum of set items :",sum(num))
```

Output:

```
sum of set items: 21
```

- **sorted(set):**

In python, **sorted (set)** function is used to sort all items of set in an ascending order. It also sorts the items into descending and ascending order. It takes an optional parameter 'reverse' which sorts the set into descending order.

Syntax:

sorted (set[,reverse=True])

Example:

```
num={1,3,2,4,6,5}
lang={'java','c','python','cpp'}
print(sorted(num))
print(sorted(lang))
print(sorted(num,reverse=True))
```

Output:

```
{1, 2, 3, 4, 5, 6}
{'c', 'cpp', 'java', 'python'}
{6, 5, 4, 3, 2, 1}
```

- **set():**

The **set()** method takes sequence types and converts them to sets. This is used to convert a given string or list or tuple into set.

Syntax:

set(sequence)

Example:

```
set1=set("PYTHON")
print(set1)
days=["Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun"]
set2 = set(days)
print(set2)
days=("Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun")
set3 = set(days)
print(set3)
```

Output:

```
{'N', 'O', 'T', 'H', 'P', 'Y'}
{'Fri', 'Thur', 'Tue', 'Sun', 'Mon', 'Sat', 'Wed'}
{'Fri', 'Thur', 'Tue', 'Sun', 'Mon', 'Sat', 'Wed'}
```

- **add():**

In python, the **add()** method used to add some particular item to the set.

Syntax:

set.add (item)

Example:

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"}
print("\n printing the original set ... ")
print(Days)
Days.add("Saturday");
Days.add("Sunday");
print("\n Printing the modified set...");
print(Days)
```

Output:

```
printing the original set ...
{'Wednesday', 'Friday', 'Thursday', 'Tuesday', 'Monday'}

 Printing the modified set...
{'Wednesday', 'Sunday', 'Friday', 'Thursday', 'Tuesday', 'Saturday', 'Monday'}
```

- **update():**

Python provides the **update ()** method to add more than one item in the set.

Syntax:

set.update ([item1, item2…])

Example:

```
Months={"Jan","Feb","Mar","Apr"}
print("\n Printing the original set ... ")
print(Months)
Months.update (["May","Jun","Jul"])
print("\n Printing the modified set...");
print(Months)
```

Output:

```
Printing the original set ...
{'Mar', 'Apr', 'Jan', 'Feb'}
 Printing the modified set...
{'Mar', 'Apr', 'Jan', 'Jun', 'May', 'Jul', 'Feb'}
```

- **discard():**

Python provides **discard ()** method which can be used to remove the items from the set. If item doesn't exist in the set, the python will not give the error. The program maintains its control flow.

Syntax:

set.discard (item)

Example:

```
Months={"Jan","Feb","Mar","Apr"}
print("\n printing the original set ... ")
print(Months)
Months.discard("Apr")
print("\n Printing the modified set...");
print(Months)
Months.discard("May")            #doesn't give error
print("\n Printing the modified set...");
print(Months)
```

Output:

```
printing the original set ...
{'Jan', 'Apr', 'Mar', 'Feb'}
 Printing the modified set...
{'Jan', 'Mar', 'Feb'}
 Printing the modified set...
{'Jan', 'Mar', 'Feb'}
```

- **remove():**

Python provides **remove ()** method which can be used to remove the items from the set. If item doesn't exist in the set, the python will give the error.

Syntax:

set.remove (item)

Example:

```
Months={"Jan","Feb","Mar","Apr"}
print("\n printing the original set ... ")
print(Months)
Months.remove("Apr")
print("\n Printing the modified set...");
print(Months)
Months.remove("May")          #it give error
print("\n Printing the modified set...");
print(Months)
```

Output:

```
printing the original set ...
{'Feb', 'Jan', 'Apr', 'Mar'}
Printing the modified set...
{'Feb', 'Jan', 'Mar'}
KeyError: 'May' doesn't exist.
```

- **pop():**

In Python, **pop ()** method is used to remove the item. However, this method will always remove the last item.

Syntax:

set.pop ()

Example:

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"}
print("\n printing the original set ... ")
print(Days)
Days.pop()
print("\n Printing the modified set...");
print(Days)
```

Output:

```
Printing the original set...
{'Monday', 'Wednesday', 'Friday', 'Tuesday', 'Thursday'}
Printing the modified set...
{'Wednesday', 'Friday', 'Tuesday', 'Thursday'}
```

- **clear():**

In Python, **clear ()** method is used to remove the all items in set.

Syntax:

set.clear ()

Example:

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"}
print("\n printing the original set ... ")
print(Days)
Days.clear()
print("\n Printing the modified set...");
print(Days)
```

Output:

```
printing the original set ...
{'Monday', 'Wednesday', 'Friday', 'Tuesday', 'Thursday'}
Printing the modified set...
set()
```

- **union ():**

In Python, the union () method is used to perform union of two sets. The union of the two sets contains the all the items that are present in both the sets.

Syntax:

set1.union (set2)

Example:

```
Days1={"Mon","Tue","Wed","Sat"}
Days2={"Thr","Fri","Sat","Sun","Mon"}
print(Days1.union(Days2))
```

Output:

```
{'Thr', 'Fri', 'Sun', 'Tue', 'Wed', 'Mon', 'Sat'}
```

- **intersection ():**

In Python, the intersection () is used to calculate the intersection of the two sets in python. The intersection of the two sets is given as the set of the elements that common in both sets.

Syntax:

set1.intersection (set2)

Example:

```
Days1={"Mon","Tue","Wed","Sat"}
Days2={"Thr","Fri","Sat","Sun","Mon"}
print(Days1.intersection(Days2))
```

Output:

```
{'Mon', 'Sat'}
```

- **difference ():**

The difference of two sets can be calculated by using the difference () method. The resulting set will be obtained by removing all the elements from set 1 that are present in set 2.

Syntax:

set1.difference (set2)

Example:

```
Days1={"Mon","Tue","Wed","Sat"}
Days2={"Thr","Fri","Sat","Sun","Mon"}
print(Days1.difference(Days2))
```

Output:

```
{'Tue', 'Wed'}
```

- **issubset ():**

The issubset() method returns True if all elements of a set are present in another set (passed as an argument). If not, it returns False.

Syntax:

set1.issubset (set2)

Example:

```
set1={1,2,3,4}
set2={1,2,3,4,5,6,7,8,9}
print(set1.issubset(set2))
print(set2.issubset(set1))
```

Output:

```
True
False
```

- **issuperset ():**

The issuperset () method returns True if a set has every elements of another set (passed as an argument). If not, it returns False.

Syntax:

set1.issuperset (set2)

Example:

```
set1={1,2,3,4}
set2={1,2,3,4,5,6,7,8,9}
print(set1.issuperset(set2))
print(set2.issuperset(set1))
```

Output:

```
False
True
```

- **Dictionary:**

In python a dictionary is the collection of key-value pairs where the value can be any python object whereas the keys are the immutable python object, i.e., Numbers, string or tuple.

The dictionary can be created by using multiple key-value pairs which are separated by comma(,) and enclosed within the curly braces {}.

Synatx:

Dict={key1:value1,key2:value2,…….}

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
print(student)
```

Output:

```
{'Name': 'Kiran', 'Age': 22, 'Regno': 562, 'Branch': 'CSE'}
```

Accessing the dictionary values:

The data can be accessed in the list and tuple by using the indexing.

However, the values can be accessed in the dictionary by using the keys as keys are unique in the dictionary.

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
print("Name : ",student["Name"])
print("Age : ",student["Age"])
print("RegNo : ",student["Regno"])
print("Branch : ",student["Branch"])
```

Output:

```
Name :  Kiran
Age :  22
RegNo :  562
Branch :  CSE
```

Updating dictionary values:

The dictionary is a mutable data type, and its values can be updated by using the specific keys.

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
print("printing student data .... ")
print(student)
student["Name"]="Kishore"
print("printing updated data .... ")
print(student)
```

Output:

```
printing student data ....
{'Name': 'Kiran', 'Age': 22, 'Regno': 562, 'Branch': 'CSE'}
printing updated data ....
{'Name': 'Kishore', 'Age': 22, 'Regno': 562, 'Branch': 'CSE'}
```

Deleting dictionary values:

The items of the dictionary can be deleted by using the del keyword.

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
print("printing student data .... ")
print(student)
del student["Branch"]
print("printing the modified information ")
print(student)
```

Output:

```
printing student data ....
{'Name': 'Kiran', 'Age': 22, 'Regno': 562, 'Branch': 'CSE'}
printing the modified information
{'Name': 'Kiran', 'Age': 22, 'Regno': 562}
```

Iterating Dictionary:

A dictionary can be iterated using the **for** loop. We can able to access only keys, only values and both keys & values.

print all the keys of a dictionary

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
print("Keys are :")
for x in student:
        print(x)
```

Output:

```
Keys are :
Name
Age
Regno
Branch
```

print all the values of a dictionary

Example:

```
print("values are :")
for x in student:
        print(student[x])
```

Output:

```
values are :
Kiran
22
562
CSE
```

print all the Keys & values of a dictionary

Example:

```
print("Key and values are :")
for x,y in student.items():
        print(x,y)
```

Output:
Key and values are :
Name Kiran
Age 22
Regno 562
Branch CSE

Dictionary Functions:
Python supports following in-bulit functions, Those are

- len()
- copy()
- get()
- keys()
- items()
- values()
- update()
- pop()
- clear()

- **len():**

In python, len() function is used to find length of given dictionary.

Syntax:

len(dictionary)

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
print("Length of Dictionary is:",len(student))
```

Output:

Length of Dictionary is: 4

- **copy():**

It returns another copy of given dictionary.

Syntax:

dictionary.copy()

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
student2=student.copy()
print(student2)
```

Output:

{'Name': 'Kiran', 'Age': 22, 'Regno': 562, 'Branch': 'CSE'}

- **get():**

In python, get() is used to get the value of specified key form dictionary.

Syntax:

dictionary.get()

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
print("Name is :",student.get("Name"))
print("RegNo is :",student.get("Regno"))
```

Output:

Name is : Kiran
RegNo is : 562

- **keys():**

In python keys() method is used to fetch all the keys from the dictionary

Syntax:

dictionary.keys()

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
for x in student.keys():
    print(x)
```

Output:

```
Name
Age
Regno
Branch
```

- **items():**

In python items() method returns a new view of the dictionary. This view is collection of key value tuples.

Syntax:

dictionary.items()

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
for x in student.items():
    print(x)
```

Output:

```
('Name', 'Kiran')
('Age', 22)
('Regno', 562)
('Branch', 'CSE')
```

- **values():**

In python values() method is used to collect all the values from a dictionary.

Syntax:

Dictionary.values()

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
for x in student.values():
    print(x)
```

Output:

```
Kiran
22
562
CSE
```

- **update():**

In python update() method updates the dictionary with the key and value pairs. It inserts key/value if it is not present. It updates key/value if it is already present in the dictionary.

Syntax:

Dictionary.update({key:value,….})

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
student.update({"Regno":590})
student.update({"phno":56895})
print(student)
```

Output:

```
{'Name': 'Kiran', 'Age': 22, 'Regno': 590, 'Branch': 'CSE', 'phno': 56895}
```

- **pop():**

In python pop() method removes an element from the dictionary. It removes the element which is associated to the specified key.

If specified key is present in the dictionary, it remove and return its value.

If the specified key is not present, it throws an error KeyError.

Syntax:

Dictionary.remove(key)

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
student.pop('Age')
print(student)
student.pop('hallno')
print(student)
```

Output:

```
{'Name': 'Kiran', 'Regno': 562, 'Branch': 'CSE'}
KeyError: 'hallno'
```

- **clear():**

In python, clear() is used to delete all the items of the dictionary.

Syntax:

Dictionary.clear()

Example:

```
student = {"Name": "Kiran", "Age": 22, "Regno":562,"Branch":"CSE"}
print(student)
student.clear()
print(student)
```

Output:

```
{'Name': 'Kiran', 'Age': 22, 'Regno': 562, 'Branch': 'CSE'}
{}
```