Table of contents

Created by **Vijay Kumar** Data Engineer & Python Programmer

1.Introductions to Python

What is Python?

- It is powerful, general-purpose, high level,object oriented programming language.
- It is developed by Guido van Rossum
- It is released in 1991

Features of Python?

- Platform Independent: Python is called platform independent because a Python program can be run on differnt kinds of platform for example Wimdow os,Linux os etc.
- Object Oriented: Python supports object oriented programming structure so it is called object oriented.
- Flexible: An application developed in Python can be modified as per user requirement so it is called flexible programmimg language.
- Stucture Oriented: To write program in Python there is a fixed structure so ti is called structure oriented.
- Portable: A PYthon program written in one system can be run in any other system . In simple a Python program can be transferred from one system to another.
- Simple: Python is very simple and easy to learn.

Application of Python?

- Web Application
- Software developement
- Database GUI Application
- Scientific and Numeric Computing

Why Use Python?

- It is very simple and easy to learn
- It powerful, fast and secure.
- It has very powerful svripting language.
- It can be run on different kind of platform for example Window, Mac, Linux etc.

First Program in Python

- It is very simple to write and execute any Python Program .
- Python is case sensitive language

```
# First python output with 'print' fuctions
print('Hello World!')
print('Hi, Python!')
# for Integer
print(7)
#for float
print(7.7)
#for boolean
print(True, False)
#mixed
print('Hello',7,7.7,True,False)
print('hello',7,7.7,True,False, sep='/')
print('hello',7, 7.7, True, False,sep="-")
print('hello\n',7,'\n',7.7,'\n',True)
print('hello',end="-")
print('keep Practice')
print("double quotation string print")
print("'triple quatation string print'")
print('"see the difference"')
print('123,234.4343,...........,ax+by+c=0,you are geneous')
print(">>>>>>>>>>,<<<<<<<<<<<<<,??????????,//////////,||||||||||,@@@@,####,$$$$,^^^,&&&&&,*****,((__---))")
print(bool(1)) #output =True
print(bool(0)) # output =False
```

```
Hello World!
Hi, Python!
7
7.7
True False
Hello 7 7.7 True False
hello/7/7.7/True/False
hello-7-7.7-True-False
hello
 7
 7.7
 True
hello-keep Practice
double quotation string print
'triple quatation string print'
"see the difference"
123,234.4343,...........,ax+by+c=0,you are geneous
>>>>>>>>>>,<<<<<<<<<<<<<,??????????,//////////,||||||||||,@@@@,####,$$$$,^^^,&&&&&,*****,((__---))
True
False
```

importand lessons

.when you write code and try to run ,some error accure

.read those error carefully try to uderstand types of error get accure and why

below we are going to see many common errors

Indentation in Python

- C,C++ , Java etc languages use braces to indicate blocks of code for class amd function definitions or flow control.
- But Python uses indentation to indicate a block of code
- We can use at least one space for indentation

```
#print string as error message
frint('hello world')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-71-eab541ccd755> in <cell line: 2>()
      1 #print string as error message
----> 2 frint('hello world')

NameError: name 'frint' is not defined
```

SEARCH STACK OVERFLOW

```
# Build-in error message
print('Hello world)
```

```
  File "<ipython-input-1-a09e8f5f17e7>", line 2
    print('Hello world)
          ^
SyntaxError: unterminated string literal (detected at line 2)
```

SEARCH STACK OVERFLOW

```
a = 5
b =10
if  a>b:
  print('a is greater than b') # there is proper indentation
else:                       # there is not error because space is same
  print('b is greater than a')
```

```
b is greater than a
```

```
a = 5
b =10
if  a>b:
print('a is greater than b') 3# this line will raise an error  because no indentation
else:                       # there is error because space is not same
  print('b is greater than a')
```

```
  File "<ipython-input-14-6c5365e7036f>", line 4
    print('a is greater than b') 3# this line will raise an error  because no indentation
    ^
IndentationError: expected an indented block after 'if' statement on line 3
```

SEARCH STACK OVERFLOW

```
a = 5
b =10
if  a>b:
                                        print('a is greater than b')
else:                       # You can use number of spaces for indentation but spaces must be same in the same block of  code
                                        print('b is greater than a')
```

```
b is greater than a
```

Variables in Python

- It is a name of storage space which is used to store data.
- It's value may be changed.
- It always contains last value stored to it.
- There is no need to declare a bariable in Python
- Variable is created by assgning a value to it

Rules to define a variable

- The first letter of a variable should be alphabet or underscore(_). `var, NUM, _var,var1 etc.`
- The first letter of variable should not be digit. `8var,1_var,3433 etc.`
- After first character it may be combination of alphabets and digits.
- Blanks spaces are not be a keyword. `v ar=, var 1=`
- Variable name should not be a keyword `print, dtype, appened etc.`
- Variable names are case sensitive for example marks and MARKS are different variables

  $not: *in Python every keyword written in small case accept of True, False*

Local Variable

- *A local Varable declared inside the body of the function is called local variable.*
- *Local variable can be used only inside that function in which it is defined.*

Global variable

- *A variable which is created outside a function is called global variable.*
- *It can be used anywhere in the program.*

```
# Variable Initialization
movie='Bahubali'
buget_in_caror= '200,00000'
kamai= 650,09009.0009
#printing value of variables
print(movie)
print(buget_in_caror)
print(kamai)

```

```
Bahubali
200,00000
(650, 9009.0009)
```

```
# another exaple
scoty_name='Sakharam'
scoty_number=420
milege_in_one_liter=32.4,'km'
# printing the initialized variable
print(scoty_name)
print(scoty_number)
print(milege_in_one_liter)
```

```
Sakharam
420
(32.4, 'km')
```

```
# Assigning single value  to multiple nariable
# we can assign a single value to multiple variables using single statement in python
a=b=c=90
print('a',+a)
print('b',b)
print('c',-c)
```

```
a 90
b 90
c -90
```

```
# Assigning  multiple values to multiple variables
#we can also assign multiples values to multiple variables using single statement in python
a,b,c=70,80,90
print('a',a)
print('b=',+b)
print('c-',+c)
```

```
a 70
b= 80
c- 90
```

```
#Local varialbe example

def add(): #function definition

  x=50 # creating variable
  y=30
  z=x+y
  print("add=",z)
add()
```

```
add= 80
```

```
def add(): #function definition

  x=50 # creating variable
  y=30
  z=x+y
  print("add=",z)
```

```
add()
print(x) # we can not call a local variable from outside
```

```
add= 80
50
```

```
#creating a global variable
x=50
y=30
def add():

  print("inside function Sum=",x+y)

  #calling function
add()
# accessing global variable
print('outside function Sum=',x+y)
```

```
inside function Sum= 80
outside function Sum= 80
```

```
# local and global vaiable with same name
var=50
def add():
  var=20
  # this line will print 20
  print('Inside function var=',var)

  #calling function
add()
 # this line will print 50
print('Outside function var',var)
```

```
Inside function var= 20
Outside function var 50
```

```
# global keyword
var=50
def add():
  global var
  #updating
  var=20
  #this line will print 20
  print('inside function var=',var)

add()
print('outside function x=',var)

```

```
inside function var= 20
outside function x= 20
```

Literals in Python

In Python, a literal is a notation used to represent a fixed value in source code. It's a direct representation of a value, like a number, string, boolean, or other basic data types.

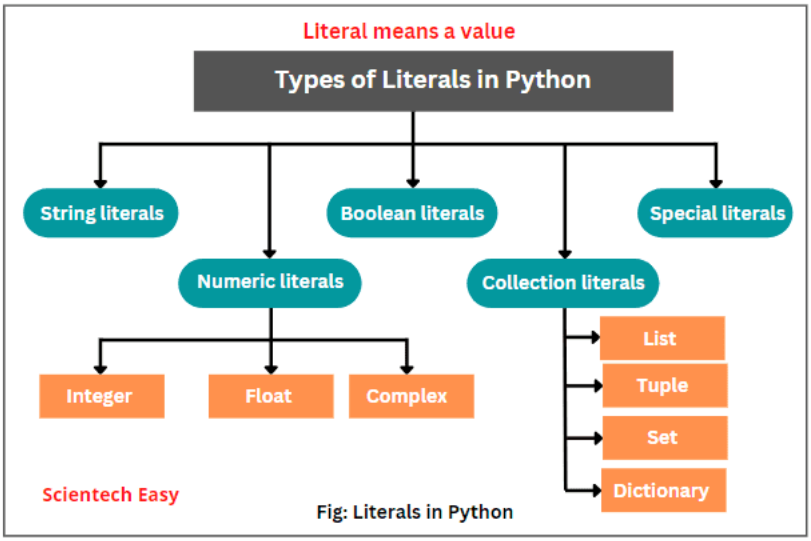


Fig: Literals in Python

```python
# Integer
print(8)
# 1*10^309
print(1e309)
```

```
8
inf
```

```python
# Decimal/Float
print(8.55)
print(1.7e309)
```

```
8.55
inf
```

```python
# Boolean
print(True)
print(False)
```

```
True
False
```

```python
# Text/String
print('Hello World')
```

```
Hello World
```

```python
# complex
print(5+6j)
```

```
(5+6j)
```

```python
# List-> C-> Array
print([1,2,3,4,5])
```

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

```python
# Tuple
print((1,2,3,4,5))
```

```
(1, 2, 3, 4, 5)
```

```python
# Sets
print({1,2,3,4,5})
```

```
{1, 2, 3, 4, 5}
```

```python
# Dictionary
print({'name':'Nitish','gender':'Male','weight':70})
```

```
{'name': 'Nitish', 'gender': 'Male', 'weight': 70}
```

```python
a = 0b1010 #Binary Literals
b = 100 #Decimal Literal
c = 0o310 #Octal Literal
d = 0x12c #Hexadecimal Literal

#Float Literal
float_1 = 10.5
float_2 = 1.5e2 # 1.5 * 10^2
float_3 = 1.5e-3 # 1.5 * 10^-3

#Complex Literal
x = 5+3.14j

print(a, b, c, d)
print(float_1 , float_2 ,float_3)
print(x," ",x.imag, "",x.real)
```

```
10 100 200 300
10.5 150.0 0.0015
(5+3.14j)   3.14  5.0
```

```python
string = 'This is Python'
strings = "This is Python"
char = "C"
multiline_str = """This is a multiline string with more than one line code."""
unicode = u"\U0001f600\U0001F606\U0001F923" # for imojy or image adress
```

```
raw_str = r"raw \n string"

print(string)
print(strings)
print(char)
print(multiline_str)
print(unicode)
print(raw_str)
```

```
This is Python
This is Python
C
This is a multiline string with more than one line code.
😐😆🤣
raw \n string
```

Data Types in Python

- It is a type of data which is used in the program.
- There is no need of data type to declare a variable in Python.
- `type()` is predefined function of Python which is used to get data type of any data.

Types of Data Type

python contains following data types:

- Numbers
- Strings
- List
- Tuple
- Set
- Dictionary

```
name='Rocky'
rollno=205
marks=85.6

print('Name Type:',type(name))
print('roll no. type:',type(rollno))
print('marks type:',type(marks))
```

```
Name Type: <class 'str'>
roll no. type: <class 'int'>
marks type: <class 'float'>
```

```
#Type function
#String
print(type('2333,dlkd,>>>>>>>>>,<<<<,????????,*****&,&&&&&&&,%%%%,$$$$$$<####,'))
print(type('Hello, World!'))
#Integer
print()   # to add a space between two outputs, use 'print()' function
print(type(15))
print(type(-34))
print(type(0))
# Float
print()
print(type(3.14))
print(type(-0.9))
print(type(1.0))
# Boolean
print()
print(type(True))
print(type(False))
```

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

<class 'int'>
<class 'int'>
<class 'int'>

<class 'float'>
<class 'float'>
<class 'float'>
```

```
<class 'bool'>
<class 'bool'>
```

User input in Pyhton

1. `input()` is a predefined function which is used to take user input in Python.

> -User input of String Value `str` Default user input is of type string
>
> -User input of Integer Value `int()` is a predefined function which is used to convert string into integer.
>
> -User input of Float Value `float()` is a predefined function which is used to convert string into float.

```
#from os import name
# User input of string value
name = input("Enter your name:")
print("Your name :",name)
print(type(name))
```

```
Enter your name:hgfhf
Your name : hgfhf
<class 'str'>
```

```
# User input of integer value
#method 1
number = input("Enter any number:")
num =int(number)
print("This number is:",num)
print("Type of numer:",type(num))

```

```
Enter any number:54
This number is: 54
Type of numer: <class 'int'>
```

```
#from pygments.token import Number
#method 2
number1 = int(input('Enter any number:'))
print('Given Number :',number1)
print(type(number))
```

```
Enter any number:58
Given Number : 58
<class 'str'>
```

```
#user input of Float value
marks= float(input("Enter your marks:"))
print("Yourk Marks is:",marks)
print("Type of number:",type(marks))
```

```
Enter your marks:56.325
Yourk Marks is: 56.325
Type of number: <class 'float'>
```

Converting an Object (datatype var) to another object type

```
#let's convert the integer number 6 to a string and a float
number = 6
print(str(number))
print(float(number))
print()
print(type(number))
print(type(str(number)))
print(type(float(number)))
str(number)
```

```
6
6.0

<class 'int'>
<class 'str'>
<class 'float'>
'6'
```

```
# lets convert the float number 3.14 to a string and an integer
number= 3.14
```

```python

print(str(number))
print(int(number))
print(type(number))
print(type(str(number)))
print(type(int(number)))
str(number)
```

```
3.14
3
<class 'float'>
<class 'str'>
<class 'int'>
'3.14'
```

```python
# let's convert the booleans to an integer, a float, and a string
bool_1= True
bool_2= False

print(int(bool_1))
print(int(bool_2))
print(float(bool_1))
print(float(bool_2))
print(str(bool_1))
print(str(bool_2))
print(bool(1))
print(bool(0))
```

```
1
0
1.0
0.0
True
False
True
False
```

Comments in Python

- It is used to explain the code to make it more readable.
- It is not condidered as a part of program , in other word we can say that compiler ignores the comment.
- Python comments are statements that are not executed by the compiler.

Types of Comments in Python

- Signle line

```
-it is used to comment only one line
-Hash symbol(#) is used for
```

- Mulitiline comments

```
-Multiline comment is used to comment a block of code.
-triple single/double quotes(''' text''') or (""" text""").
```

```python
# this is a single line comment
# this is the standard way used by industry
print('you are excellent')
```

```
you are excellent
```

```python
'''
*   It is used to explain the code to make it more readable.

*   It is not condidered as a part of program , in other word we can say that compiler ignores the comment.
*   Python comments are statements that are not executed by the compiler.
'''
print('multiline comment text')
```

```
multiline comment text
```

```python
"""
*   It is used to explain the code to make it more readable.

```

```
*   It is not condidered as a part of program , in other word we can say that compiler ignores the comment.
*   Python comments are statements that are not executed by the compiler.
"""
print( 'both are appliacable')
```

```
both are appliacable
```

```
# Let's find the data types of 9/3 and 9//4

print(9/3)
print(9//4)
print(type(9/3))
print(type(9//4)) # floor // scapped reminder value
```

```
3.0
2
<class 'float'>
<class 'int'>
```

Operator

```
Operand-> X + Y <-Operand
            ^
            |
            Operator
```

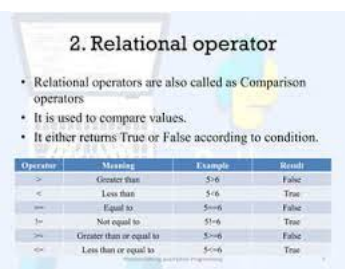
Operator : It is a special symbol which is used to perform logical or mathematical operation on data variable.

Operand: It is data or variable on which the operation is to be performed.

Type of Operator

- Arithmetic Operators

- Relational Operators



2. Relational operator

- Relational operators are also called as Comparison operators
- It is used to compare values.
- It either returns True or False according to condition.

Operator	Meaning	Example	Result
>	Greater than	5>6	False
<	Less than	5<6	True
==	Equal to	5==6	False
!=	Not equal to	5!=6	True
>=	Greater than or equal to	5>=6	False
<=	Less than or equal to	5<=6	True

- Logical Operator

Operator	Meaning	Example	Result
and	Logical and	(5<2) and (5>3)	False
or	Logical or	(5<2) or (5>3)	True
not	Logical not	not (5<2)	True

- Assignment Operators

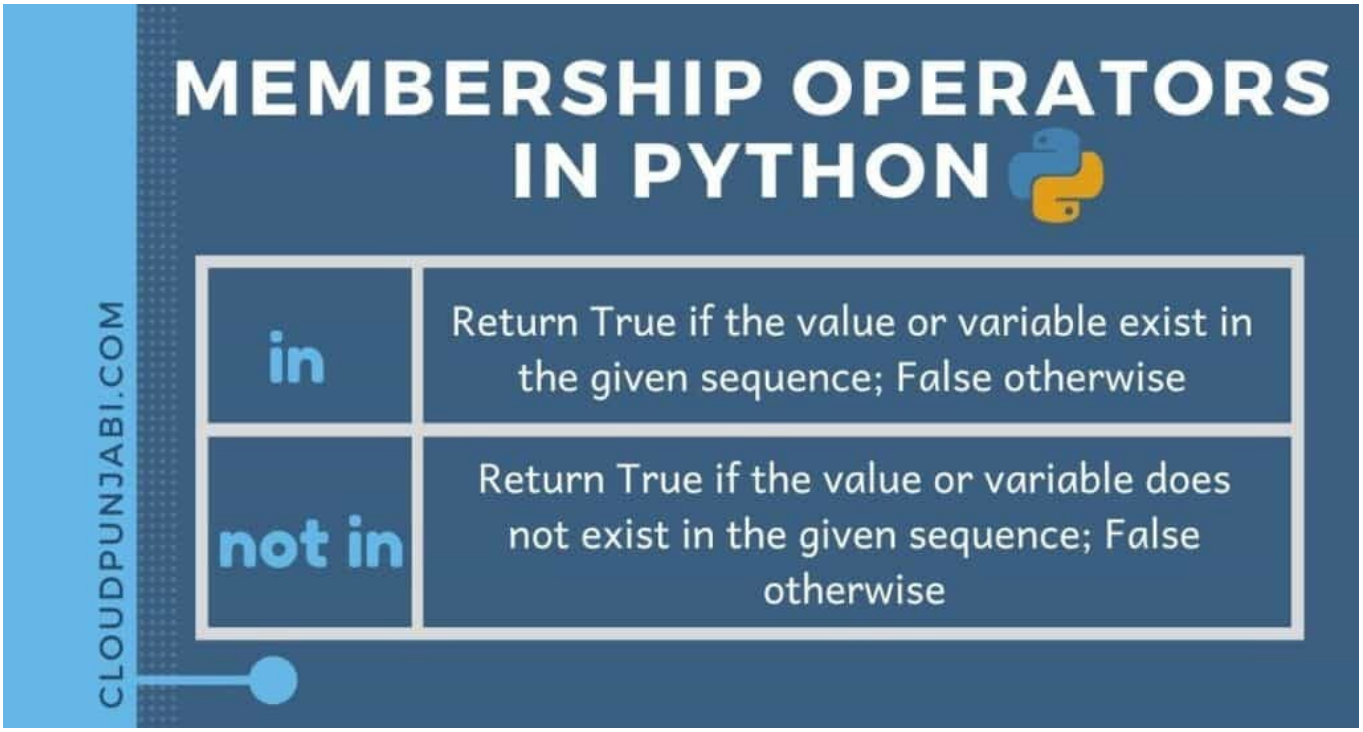
Operator	Example	Equivalent Expression (m=15)	Result
=	y = a+b	y = 10 + 20	30
+=	m +=10	m = m+10	25
-=	m -=10	m = m-10	5
*=	m *=10	m = m*10	150
/=	m /=10	m = m/10	1.5
%=	m %=10	m = m%10	5
**=	m**=2	m = m**2 or $m = m^2$	225
//=	m//=10	m = m//10	1

- Bitwise Operators

Types of Bitwise Operators

Operator	Name	Example	Result
&	Bitwise AND	6 & 3	2
|	Bitwise OR	10 | 10	10
^	Bitwise XOR	2^2	0
~	Bitwise 1's complement	~9	-10
<<	Left-Shift	10<<2	40
>>	Right-Shift	10>>2	2

- Membership Operators



- Identity Operators

Operator	Description
is	It returns true if two variables point the same object and false otherwise
is not	It returns false if two variables point the same object and true otherwise

Expression and variables

```
# Addition
x = 56 + 65+ 89+ 45+ 78.5+ 98.2
print(x)
print(type(x))
```

```
431.7
<class 'float'>
```

```
#substraction
x= 85-52-21-8
print(x)
print(type(x))
```

```
4
<class 'int'>
```

```
# Multiplication

x= 8*74
```

```
print(x)
print(type(x))
```

```
592
<class 'int'>
```

```
# Division
x= 125/24
print(x)
print(type(x))
```

```
5.208333333333333
<class 'float'>
```

```
# Floor division
x=125// 24
print(x)
print(type(x))
```

```
5
<class 'int'>
```

```
# Modulus
x= 125 % 24
print(x)
print(type(x))
```

```
5
<class 'int'>
```

```
# Exponentiation
x=2**3
print(x)
print(type(x))

```

```
8
<class 'int'>
```

```
# Mathematic expression
x= 45+3*89
y=(45+3)*89

print(x)
print(y)
print(x+y)
print(x-y)
print(x*y)
print(x/y)
#print(x**y)
print(x//y)
print(x%y)
```

```
312
4272
4584
-3960
1332864
0.07303370786516854
0
312
```

```
# Arithmetic operator
x= 5
y=2
print('x + y=', x+y)
print('x - y=',x-y)
print("x*y=",x*y)
print('x/y=',x/y)
print('x%y',x%y)
print('x**y',x**y)
```

```
x + y= 7
x - y= 3
x*y= 10
x/y= 2.5
x%y 1
x**y 25
```

```
# Logical Operator
x = int(input('Enter first number:'))
```

```
y = int(input('Enter second number:'))
z = int(input("Enter third number:"))

if x>y and y>z:
  print(x,"is greatest")
if y>x and y>z:
  print(y,"is greatest")
if z>x and z>y:
  print(z,'is greatest')
```

```
Enter first number:52
Enter second number:65
Enter third number:42
65 is greatest
```

```
# Assignment operator
x1 =9
y1 = 4
x1 +=y1 # x1 = x1 + y1
print(x1)

print() # for space

x2 =10
y2 =11
x2 -=y2 #x2 =x2-y2
print(x2)

print()
x3 =15
y3 =7
x3 *=y3 #x3 =x3*y3
print(x3)

print()
x4 =125
y4 =21
x4 /=y4 # x4= x4/y4
print(x4)

print()

x5 = 20
y5 =3
x5 **=y5 # x5 =x5**y5
print(x5)

print()

x6 =9
y6 =4
x6 //=y6  #x6=x6//y6
print(x6)
```

```
13

-1

105

5.9523809523809526

8000

2
```

```
# Bitwise Operators
# To learn this operator  you have knoledge about Decimal to Binary conversion
x=6
y=3
print('x&y=',x&y) # & = Bitwise AND
print()
print('x|y=',x|y) # | = Bitwise OR
print()
print('x<<y=',x<<y) # << = Shift Left
print()
print('x>>y=',x>>y) # >> = Shift Right
print()
print('x^y=',x^y) # ^ = X-OR
```

```
x&y= 2

x|y= 7

x<<y= 48

x>>y= 0

x^y= 5
```

```python
# Membership Operator
list=['Amir','Salman','Saharukh']
print('Amir' in list) # True
print('Virat' not in list) #True
print()
print('Salman' not in list) # False
print('Virat'in list) # False
```

```
True
True

False
False
```

```python
# Identity Operator
x = 10
y = 20
z = 20
print(x is y) # True because of same identity
print(x is not y) # False because of not same identity
print(x is z) # False
print( x is not z) #True
```

```
False
True
False
True
```

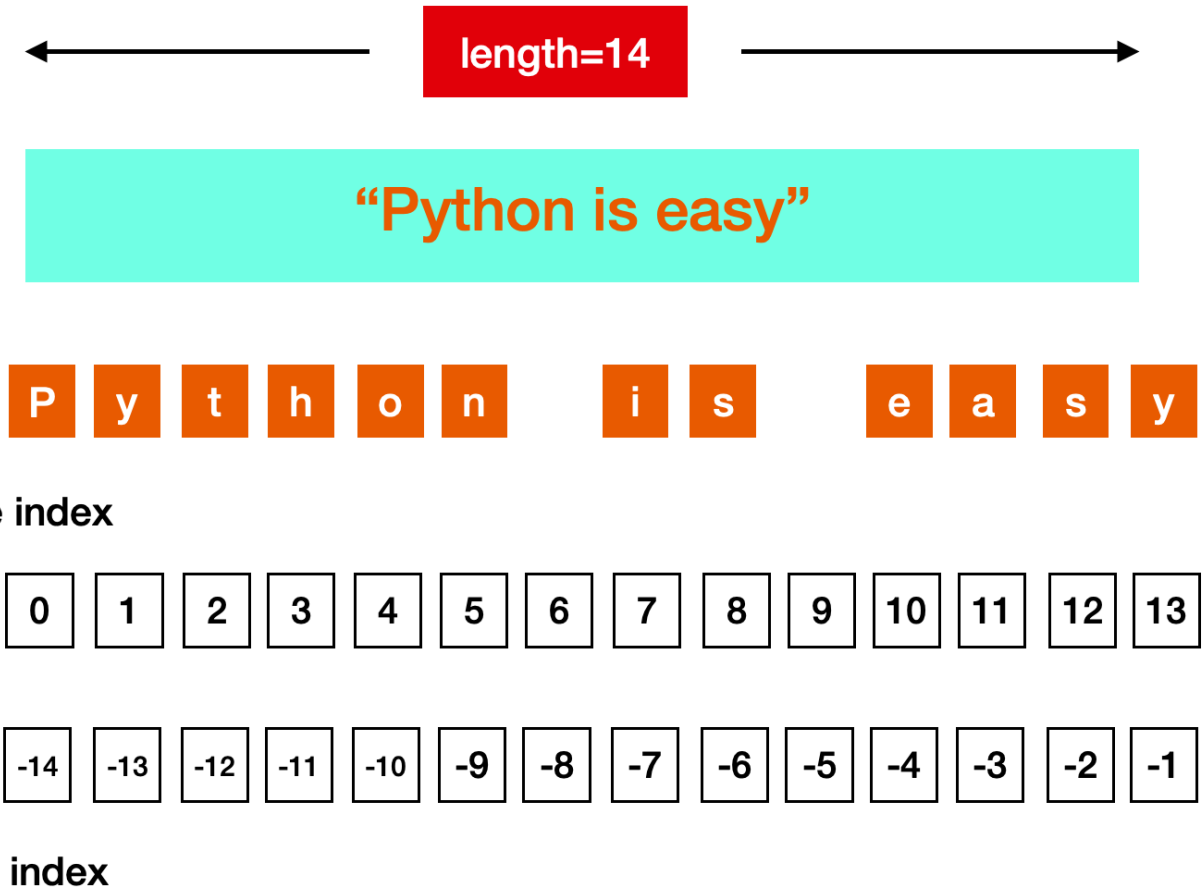
```
1
```

---

2. String in Python

- String is a collection of characters.
- It is created using single quotes or double quotes or triple quotes.
- We can print string using `print()` function.
- We can also print particular character of string using index number.
- String are stored in the form of array

Python String



In Python specifically, strings are a sequence of Unicode Characters

- Creating Strings
- Accessing Strings
- Adding Chars to Strings
- Editing Strings
- Deleting Strings
- Operations on Strings
- String Functions

```
# Employ double quotation marks for describing a string
"Hello World!"
```

```
'Hello World!'
```

```
# Employ single quotation marks for describing  a string
'Python is easy'

```

```
'Python is easy'
```

```
# Digital and spaces in a string
'3 6 9 3 5 4 2 3'
```

```
'3 6 9 3 5 4 2 3'
```

```
# Specific characters in a string
'@#$%^&*?'
```

```
'@#$%^&*?'
```

```
# Assigning a string to a varialble 'message'
message= 'Hello World'
print(message)
message
```

```
Hello World
'Hello World'
```

```
# string with single quotes
str1 = 'Python is easy'
#string with double quotes
str2 = "Python is easy"
#string with triple quotes
str3 ='''Python is easy'''

print(str1)
print(str2)
print(str3)
```

```
Python is easy
Python is easy
Python is easy
```

```
# printing the above string(str1)
print(str1)
#printing first character
print("First character:",str1[0])
#printing second character
print('Second character:',str1[1])
#printing last character
print("Last character:",str1[-1])
```

```
Python is easy
First character: P
Second character: y
Last character: y
```

```
# lenght of a string includiing space
len(str1)
```

```
14
```

```
# print "i" from str1
print(str1[7])
```

```
i
```

Accessing particular character or range of charaters from string.

- We can access range of characters from string using slicing operator colon(:)
- [start : befor end, difference]

Operations on Strings

- Indexing , Slicing and striding of a String
- Arithmetic Operations
- Relational Operations
- Logical Operations
- Loops on Strings
- Membership Operations

```
# creating string
str= "I love PyTHON"
# printing the lenght of string including spaces
print(len(str))
# printing the string
print('string',str)
#printing indexing 2 to 6 character
print("str[2:6]:",str[2:6])
#pirnting character between 7th and last character
print("str[7:-1]:",str[7:-1])
```

```
13
string I love PyTHON
str[2:6]: love
str[7:-1]: PyTHO
```

```
# to select every second element in the variable 'message'
str[::2]
```

```
'Ilv yHN'
```

```
# corporation of slicing and striding
# get every second element in range from index 0 to 6
str[0:13:2]
```

```
'Ilv yHN'
```

```
# Slicing
s = 'hello world'
print(s[6:0:-2]) # remember when we use negative indexing start should be great value
```

```
wol
```

```
# to reverse any string
print(s[::-1])
```

```
dlrow olleh
```

```
s = 'hello world'
print(s[-1:-6:-1])
```

```
dlrow
```

Update String

- We can Update string value by reassigning new value to the same variable.
- But we can't update the particular character of the string.

```
# creating string
str="Python Programmig"
print("Original string:",str)
#update string str
str="Java Programming"
print("Updated string:",str)
```

```
Original string: Python Programmig
Updated string: Java Programming
```

```
# but we can't ,update the particular character of the string
# Updating character P with M
# this line will raise an error
str[0]="M"
print("Updated String:",str)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-62-014df6a68db6> in <cell line: 4>()
      2 # Updating character P with M
      3 # this line will raise an error
----> 4 str[0]="M"
      5 print("Updated String:",str)

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

SEARCH STACK OVERFLOW

```
1
```

String Concatenation

- We can combine two or more than two string using plus (+) operator.
- List item

```
# creating string
str1 ="I love"
str2=" python programming"
str3= str1+str2
print("String1:",str1)
print("String2:",str2)
print("String3:",str3)
```

```
String1: I love
String2:  python programming
String3: I love python programming
```

```python
from prompt_toolkit.shortcuts.utils import print_container
# we can't combine string with numeric value
book ="xyz"
price=550
# this line will raise an error
str="The price of book"+book+"is Rs."+price
print("Combined String:",str)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-66-7317f5eeaf8a> in <cell line: 6>()
      4 price=550
      5 # this line will raise an error
----> 6 str="The price of book"+book+"is Rs."+price
      7 print("Combined String:",str)

TypeError: can only concatenate str (not "int") to str
```

SEARCH STACK OVERFLOW

```python
# To Combine string with nummeric value formate() function is used
# Combining string with numeric value
str="The price of book {} is Rs. {}".format(book,price)
print("Combined String:",str)
```

```
Combined String: The price of book xyz is Rs. 550
```

`eval()`: functions

This function serves the aim of converting a string to an integer or a float

```python
expression = '8+7'
total = eval(expression)
print('Sum of the expression is', total)
print(type(expression))
print(type(total))

```

```
Sum of the expression is 15
<class 'str'>
<class 'int'>
```

String repetition

- `asterisk(*) symbol` is used to concate multiple copies of the same string.

```python
str="Python "
#print Python 5 times
print(str*5)
```

```
Python Python Python Python Python
```

```python
# printing a string for  4 times
4* "Hello World!"
```

```
'Hello World!Hello World!Hello World!Hello World!'
```

```
1
```

Escape sequences

Escape sequences in Python are special combinations of characters that are used to represent certain non-printable or special characters within strings and other text-based data

\: Backslash

': Single Quote

": Double Quote

\n: Newline

\t: Tab

\r: Carriage Return

\b: Backspace

\f: Form Feed

\v: Vertical Tab

\xHH: Hexadecimal Value (e.g., \x41 represents 'A')

\uHHHH: Unicode Character (e.g., \u03A9 represents Ω)

\UHHHHHHHH: Unicode Character (e.g., \U0001F604 represents 😄)

```
# New line escape sequence
print("Hello World! \nHow many people are living on the earth?")
```

```
Hello World!
How many people are living on the earth?
```

```
# Tab escape sequence
print("Hello World! \tHow many people are living on the earth?")
```

```
Hello World!    How many people are living on the earth?
```

```
path = "C:\\myfolder\\myfile.txt"
#\\: Represents a literal backslash character.
# back slash in a string
print('Hello World! \\ How many people are living on the earth?')
```

```
Hello World! \ How many people are living on the earth?
```

```
#\': Represents a single quote character within a single-quoted string.
message = 'He said, \'Hello!\''
message
```

```
'He said, 'Hello!''
```

```
#\": Represents a double quote character within a double-quoted string.
quote = "She exclaimed, \"Wow!\""
quote
```

```
'She exclaimed, "Wow!"'
```

```
#\n: Represents a newline character, creating a new line.
multiline = "Line 1 \nLine 2"
print(multiline)
```

```
Line 1
Line 2
```

```
#\r: Represents a carriage return character, typically used in some text processing
formatted = "Progress:\r100%"
print(formatted)
```

```
100%
```

```
#\b: Represents a backspace character, useful for erasing the preceding character.
user_input = "Hello \bWorld"
print(user_input)
```

```
HelloWorld
```

```
# r will say python that a string will be show as a raw string
print("Hello World! \How many people are living on the earth?")
```

```
Hello World! \How many people are living on the earth?
```

```
1
```

String Methods & Operations

- Python contains the following string methods.

1. `capitalize()` : It converts the first letter of the string into uppercasse.

```
#from IPython.utils.io import CapturedIO
#.capitalize():
str="easy softwares"
print(str.capitalize())
```

```
Easy softwares
```

2. `casefold()` : It converts string into lowercase.

```
#casefold():
str="Easy Softwares"
print(str.casefold())

```

```
easy softwares
```

3. `center()` :

- It is used to align the string to the center.
- It is has two parameters width and fillchar in which fillchar is optional.

```
#center():
str="Easy"
print("Original String:",str)
#without fillchar
print("Centered String:",str.center(20))
print()
 #center() method will center align the string, using a specified character (space is the default) as the fill character.
message = 'Hallo Leute!'
message.center(50, '-')

```

```
Original String: Easy
Centered String:         Easy

'------------------Hallo Leute!------------------'
```

4. `endswith()` :

- It returns boolean value (True/False).
- The given string ends with specified string returns true otherwise false.

```
#`endswith() `:
str="Easy Softwares"
print(str.endswith("softwares"))
print(str.endswith("Softwares"))
```

```
False
True
```

5. `startswith()` :

- It returns boolean value(True/False).
- The given string starts with specified string returns true otherwise false.

```
#`startswith()`:
str="Easy Softwares"
print(str.startswith('easy'))
print(str.startswith('Easy'))
```

```
False
True
```

6. `find()` :

- It is used to search the specified string in a string.
- If the specified string is found then it * returns the position* of where it is found.

```
#find()`:
str="you can learn python easily."
print(len(str)) # this is return leght of string

#this line will search python in string
print(str.find("python")) # starting index of python is 14
print()
```

```
#this line will search python from index 10
print(str.find('python',10))
# this line will search can between index 3 to 10
print(str.find('can',3,10))

# when substring is not in string it will return -1
print(str.find('k'))
```

```
28
14

14
4
-1
```

7. `index()` : This method is same as find but it raises an error when the specified string is not found.

```
# index()
print(str.index("python")) # this line will search starting index number of python in string
print(str.index('k')) # this is showing error
```

```
14
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-133-ca0c39c52a1a> in <cell line: 3>()
      1 # index()
      2 print(str.index("python")) # this line will search starting index number of python in
string
----> 3 print(str.index('k'))

ValueError: substring not found
```

SEARCH STACK OVERFLOW

8. `format()` :

- It is used to format the string.
- we can insert specified value inside the string using format () method.
- The specified value is inside string using placeholder.
- The placeholder is identified using numbered indexes{0} or empty placeholders{}.

```
# Empty placeholder
name="Tom"
pro ="Python"
print("my name is {} and i love {}".format(name,pro))

#Numbered indexes
print("my name is {0} and i love {1}".format(name,pro))
# Reverse index
print("my name is {1} and i love {0}".format(name,pro))
```

```
my name is Tom and i love Python
my name is Tom and i love Python
my name is Python and i love Tom
```

9. `isalnum()` :

- It is used to check specified string is alphanumaric or not.
- String that contains only alphabet and number is called alphanumeric.
- It returns boolean value(True/False).

```
str1="easy123" # this line contain alphabet and number
str2="easy@123" # but this line has special character as well
print(str1.isalnum())
print(str2.isalnum())
```

```
True
False
```

10. `isalpha()` :

- It is used to check specified string is alphabetic or not.
- It returns boolean value(True/False).
- It retrns true if string is alphabetics otherwise returns false.

```
str1="easy"
str2="easy@123"
print(str1.isalpha())
print(str2.isalpha())
```

```
True
False
```

11. `isdecimal()`:

- It is used to check all the characters of string are decimal or not.
- It returns boolean value(True/False).
- It returns true if all charcters are decimal otherwise returns False.

```
str1="easy"
str2="1233443" # it will check only numbers in string
str3="3434.34"
print(str1.isdecimal())
print(str2.isdecimal())
print(str3.isdecimal())
```

```
False
True
False
```

12. `isdigit()`:

- It is used to check all the characters of string are digit or not.
- It returns boolean value(True/False).
- It returns true if all characters are digit otherwise returns false.

```
print(str1.isdigit())
print(str2.isdigit())
print(str3.isdigit())
```

```
False
True
False
```

13. `isidentifier()`:

- It returns true if the specified string is valid identifier otherwise returns false.

```
str1="easy" # valid
str2="12334"
str3="abc123" # valid
str4="*abc"
str5="ab@cd"
print(str1.isidentifier())
print(str2.isidentifier())
print(str3.isidentifier())
print(str4.isidentifier())
print(str5.isidentifier())
```

```
True
False
True
False
False
```

14. `islower()`:

- It returns true if all the characters of the string is in lowercase otherwise returns false.

```
str1="python"
str2="Python"
print(str1.islower())
print(str2.islower())
```

```
True
False
```

15. `isupper()`:

- It retruns true if all the characters of the string are uppercase otherwise returns false.

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

```
False
False
True
```

16. `split()`:

- It is used to break the sentence into words using separator.
- The default separator is white space.
- split() function returns list.

```
str="I love Python"
print("Original String:",str)
mylist=str.split();
print("New String:",mylist)
```

```
Original String: I love Python
New String: ['I', 'love', 'Python']
```

```
# we can use any character as a separator
str="I*love*Python"

print("Original String:",str)
mylist=str.split('*')
print("New String:",mylist)
```

```
Original String: I*love*Python
New String: ['I', 'love', 'Python']
```

17. `replace()`:

- It replaces the old string with new string

```
str="I love Java"
print("Original String:",str)
# replacing java with Python
str=str.replace("Java","Python")
print("New String:",str)
```

```
Original String: I love Java
New String: I love Python
```

18. `strip()`:

- It removes unwanted white-space from string.

`lstrip()` : It removes left side unwanted white-space from string.

`rstrip()` :It removes right side unwanted white-space from string.

```
str1= "    Python   "
print(str1.lstrip())
print(str1.rstrip())
print(str1.strip())

```

```
Python
    Python
Python
```

logical operations

```
'delhi' != 'delhi'
```

```
False
```

```
'mumbai' > 'pune'
# lexiographically
```

```
False
```

```
'Pune' > 'pune'
```

```
False
```

```
'hello' and 'world'
```

```
'world'
```

```
'hello' or 'world'
```

```
'hello'
```

```
len('hello world')
```

```
11
```

```
max('hello world')
```

```
'w'
```

```
min('hello world')
```

```
' '
```

```
sorted('hello world',reverse=True)
```

```
['w', 'r', 'o', 'o', 'l', 'l', 'l', 'h', 'e', 'd', ' ']
```

```
'hi my name is nitish'.split()
```

```
['hi', 'my', 'name', 'is', 'nitish']
```

```
" ".join(['hi', 'my', 'name', 'is', 'nitish'])
```

```
'hi my name is nitish'
```

```
message = 'hello python!'
print('Before uppercase: ', message )
# convert uppercase the elements in a string
message_upper = message.upper()
print('After uppercase: ', message_upper)
# convert lowercase the elements in a string
message_lower = message.lower()
print('Again lowercase: ', message_lower)
# convert first letter of string to uppercase
message_title = message.title()
print('The first element of the string is uppercase: ', message_title)

```

```
Before uppercase:  hello python!
After uppercase:  HELLO PYTHON!
Again lowercase:  hello python!
The first element of the string is uppercase:  Hello Python!
```

```
```

Search String

- `in` : keyword is used to check specified character or gruop of characters is present or not.
- `in` keyword returns true if specified character is present ohterwise returns false.

```
# creating string
str="I love python programming"
search_str=input("Enter any string to search:")
```

```
if search_str in str:
  print(search_str,"is present")
else:
  print(search_str,"is not present")
```

```
Enter any string to search:i
i is present
```

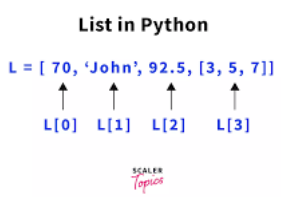
```

```

3.List in Python

- It is a collection of data of different data type.
- It is used to store list of values.
- A list is created by putting list of comma-seperated values between suqare brackets.

Characterstics of a List

- Ordered
- Changeble/Mutable
- Hetrogeneous
- Can have duplicates
- are dynamic
- can be nested
- items can be accessed
- can contain any kind of objects in python
- list elements can be accessed by index.




```
L = [1,2,3,1]
L1 = [3,2,1]

L == L1 # ordered is matter here
```

```
False
```

```
# Create list
#string list
str_list=['Apple','Mango','Orange']
##int list
int_list=[23,51,84,34,21,45,46]
# float list
float_list=[2.4,4.3,5.2,6.23,4.98]
#mixed list
mixed_list=["Apple",True,23,4.35,['a',3,False]]
# we cant print list
print(int_list)
print(str_list)
print(float_list)
print(mixed_list)

```

```
[23, 51, 84, 34, 21, 45, 46]
['Apple', 'Mango', 'Orange']
[2.4, 4.3, 5.2, 6.23, 4.98]
['Apple', True, 23, 4.35, ['a', 3, False]]
```

```
# Empty
print([])
# 1D -> Homo
print([1,2,3,4,5])
# 2D
print([1,2,3,[4,5]])
# 3D
print([[[1,2],[3,4]],[[5,6],[7,8]]])
# Hetrogenous
```

```
print([1,True,5.6,5+6j,'Hello'])
# Using Type conversion
print(list('hello'))
```

```
[]
[1, 2, 3, 4, 5]
[1, 2, 3, [4, 5]]
[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
[1, True, 5.6, (5+6j), 'Hello']
['h', 'e', 'l', 'l', 'o']
```

```
nils=['python',25,2022]
nils
```

```
['python', 25, 2022]
```

```
nils=['python',3.14,2022,[1,1,2,3,5,8],('hello','python',3,14,2022)]
nils

```

```
['python', 3.14, 2022, [1, 1, 2, 3, 5, 8], ('hello', 'python', 3, 14, 2022)]
```

Access Values of list using index

- Value of list can be accessed using index number .
- Index number is always an integer value and starts with 0.

```
fruit_list=['Apple','Orange','Mango']
print('I like',fruit_list[0])
print('I like',fruit_list[1])
print('I like',fruit_list[2])
```

```
I like Apple
I like Orange
I like Mango
```

```
int_list=[5,10,15,30,35,40,50]
# print will start at index 1 to 4
print(int_list[1:4]) # i index value included but, 4 index value not inclueded it befor value get print
```

```
[10, 15, 30]
```

Access value of list using negative index

- Negative indexes start from the end of the list.
- Negative index always starts with -1.

```
fruit_list=['Apple','Orange','Mango']
print('I like',fruit_list[-1])
print('I like',fruit_list[-2])
print('I like',fruit_list[-3])
```

```
I like Mango
I like Orange
I like Apple
```

Access value of list using loop

```
fruit_list=['Apple','Orange','Mango']
for name in fruit_lis:
  print("I like:",name)
```

```
I like: Apple
I like: Orange
I like: Mango
```

Update item of list

```
fruit_list=['Apple','Orange','Mango']
print("Befor Updation",fruit_list)
#this line will replace Orange with banana
fruit_list[1]="Banana"
```

```
print("After Updation",fruit_list)
```

```
Befor Updation ['Apple', 'Orange', 'Mango']
After Updation ['Apple', 'Banana', 'Mango']
```

List Function

- Python contains the following list functions

1. `len()` : It is used to get the numbers of elements in list.

```
fruit_list=['Apple','Orange','Mango']
print("List elements:",fruit_list)
# this line will print length of list
print("Length of list is:",len(fruit_list))

```

```
List elements: ['Apple', 'Orange', 'Mango']
Length of list is: 3
```

max()

- It is used to maximum value from the list.
- In case of string focus on ASCII value of first letter of list items.

`min()` : same think for to get minimum value from the list.

```
fruit_list=['Apple','Orange','Mango']
print("Fruit list:",fruit_list)
print("Max elements:",max(fruit_list))
print()
Animal_list=['zebra',"dog",'elephant']
print('Animal list:',Animal_list)
print("Maximum elements:",max(Animal_list))
print()
int_list=[45,85,36]
print("int list:",int_list)
print("Max element:",max(int_list))
```

```
Fruit list: ['Apple', 'Orange', 'Mango']
Max elements: Orange

Animal list: ['zebra', 'dog', 'elephant']
Maximum elements: zebra

int list: [45, 85, 36]
Max element: 85
```

```
 fruit_list=['Apple','Orange','Mango']
print("Fruit list:",fruit_list)
print("Max elements:",min(fruit_list))
print()
Animal_list=['zebra',"dog",'elephant']
print('Animal list:',Animal_list)
print("Maximum elements:",min(Animal_list))
print()
int_list=[45,85,36]
print("int list:",int_list)
print("Max element:",min(int_list))
```

```
Fruit list: ['Apple', 'Orange', 'Mango']
Max elements: Apple

Animal list: ['zebra', 'dog', 'elephant']
Maximum elements: dog

int list: [45, 85, 36]
Max element: 36
```

`list()`: It is used to convert sequence types (tuple) into list.

```
# tuple is created using parentheses
fruit_tuple=('Apple','Orange','Mango')
print("Tuple items:",fruit_tuple)
# this line convert tuple into list
```

```
fruit_lst=list(fruit_tuple)
```

```
Tuple items: ('Apple', 'Orange', 'Mango')
['Apple', 'Orange', 'Mango']
```

```
# len/min/max/sorted
L = [2,1,5,7,0]

print(len(L))
print(min(L))
print(max(L))
print(sorted(L,reverse=True))
```

```
5
0
7
[7, 5, 2, 1, 0]
```

List Methods

- Python contains the following list methods.

1. `append()` : It is used to add the new element at the end of the list.

```
fruit_list=['Apple','Orange','Mango']
print("Fruits list:",fruit_list)
#this line will add Cherry at the end of list
fruit_list.append("Cherry")
print("New Fruits list:",fruit_list)
```

```
Fruits list: ['Apple', 'Orange', 'Mango']
New Fruits list: ['Apple', 'Orange', 'Mango', 'Cherry']
```

2. `clear()`: This function is used to empty the list.

```
fruit_list=['Apple','Orange','Mango']
print("Fruits list:",fruit_list)
# this line will empty the list
fruit_list.clear()
print("new fruits list:",fruit_list)
```

```
Fruits list: ['Apple', 'Orange', 'Mango']
new fruits list: []
```

3. `count()` : This method counts the number of occurrence of particular item in a list.

```
number_list=[10,16,40,50,16,30,10,16,50]
print("Numbers list:",number_list)
print("Total Count of 16:",number_list.count(16))
```

```
Numbers list: [10, 16, 40, 50, 16, 30, 10, 16, 50]
Total Count of 16: 3
```

4. `copy()` : This function copies the elements of one list into another.

```
fruit_list_1=['Apple','Orange','Mango']
print("List 1",fruit_list_1)
# this line will copy
fruit_list_2=fruit_list_1.copy()
print("List 2",fruit_list_2)
```

```
List 1 ['Apple', 'Orange', 'Mango']
List 2 ['Apple', 'Orange', 'Mango']
```

5. `extend()` : This function is used to join two list.

```
list1=['Apple','Orange','Mango']
list2=['Cherry','Grapes','Melon']
# this line will join list1 and list2
list1.extend(list2)
print(list1)
```

```
['Apple', 'Orange', 'Mango', 'Cherry', 'Grapes', 'Melon']
```

6. `insert()`: this is used to add new items into list at particular index.

```
fruit_list=['Apple','Orange','Mango']
print("Before insertion")
# this line will add Banana at index 1
fruit_list.insert(1,"Banana")
print("After insertion",fruit_list)
```

```
Before insertion
After insertion ['Apple', 'Banana', 'Orange', 'Mango']
```

7. `pop()`:

- This function deletes the element of given index.
- It deletes last item if we do not pass index.

```
fruit_list=['Apple','Orange','Mango','Cherry']
print("Fruits list:",fruit_list)
#this line will delete last element Cherry
fruit_list.pop()
print("Fruits list:",fruit_list)
# this line will delete element at index 1 (Orange)
fruit_list.pop(1)
print('Fruits list:',fruit_list)
```

```
Fruits list: ['Apple', 'Orange', 'Mango', 'Cherry']
Fruits list: ['Apple', 'Orange', 'Mango']
Fruits list: ['Apple', 'Mango']
```

8. `remove()`: remove() function is used to delete or remove item from list.

```
fruit_list=['Apple','Orange','Mango']
print("Before Delete",fruit_list)
# this line will delete Orange from list
fruit_list.remove("Orange")
print("After deletion",fruit_list)
```

```
Before Delete ['Apple', 'Orange', 'Mango']
After deletion ['Apple', 'Mango']
```

10. `reverse()`: This function reverses elements of the list.

```
fruit_list=['Apple','Orange','Mango','Cherry']
print("Fruits list:",fruit_list)
#this will reverse the list items
fruit_list.reverse()
print("Reverse Fruits list:",fruit_list)
```

```
Fruits list: ['Apple', 'Orange', 'Mango', 'Cherry']
Reverse Fruits list: ['Cherry', 'Mango', 'Orange', 'Apple']
```

11. `sort()`:

- This function Sorts the list.
- By using this function we can display the list items in ascending order or descending order.

```
str_list=['Apple','Orange','Mango',"cherry"]
int_list=[58,20,46,36]
char_list=['E','A','S','Y']
print("List item before sorting")
print(str_list)
print(int_list)
print(char_list)
# sort the lists in ascending order
str_list.sort()
int_list.sort()
char_list.sort()
print("List item after sorting")
```

```
print(str_list)
print(int_list)
print(char_list)
```

```
List item before sorting
['Apple', 'Orange', 'Mango', 'cherry']
[58, 20, 46, 36]
['E', 'A', 'S', 'Y']
List item after sorting
['Apple', 'Mango', 'Orange', 'cherry']
[20, 36, 46, 58]
['A', 'E', 'S', 'Y']
```

```
# append
L = [1,2,3,4,5]
L.append(True)
print(L)
L1 = [1,2,3,4,5]
L.append([6,7,8])
print(L1)
```

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

```
# extend
L = [1,2,3,4,5]
L.extend([6,7,8])
print(L)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
L = [1,2,3,4,5]
L.append([6,7,8])
print(L)
```

```
[1, 2, 3, 4, 5, [6, 7, 8]]
```

```
L = [1,2,3,4,5]
L.extend('delhi')
print(L)
```

```
[1, 2, 3, 4, 5, 'd', 'e', 'l', 'h', 'i']
```

```
# insert
L = [1,2,3,4,5]

L.insert(1,100)
print(L)
```

```
[1, 100, 2, 3, 4, 5]
```

```
lis = [1,2,3,4,5,6,7]
print(len(lis))
lis.append(4)
print(lis)
print(lis.count(4)) # How many 4 are on the list 'lis'?
print(lis.index(2)) # What is the index of the number 2 in the list 'lis'?
lis.insert(8, 9) # Add number 9 to the index 8.
print(lis)
print(max(lis)) # What is the maximum number in the list?
print(min(lis)) # What is the minimum number in the list?
print(sum(lis)) # What is the sum of the numbers in the list?
```

```
# conversion of  a string into a list using split() function
message='Python is a programming language.'
message.split() #
```

```
['Python', 'is', 'a', 'programming', 'language.']
```

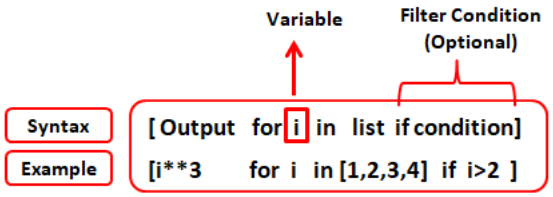
```
text='p,y,t,h,o,n'
text.split(',')
```

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

List comprehension

- List comprehension is a concise way to create lists in many programming languages, including Python.
- It allows you to create a new list by applying an expression to each item in an existing iterable (like a list, tuple, or range).

- and optionally filtering the items based on a condition.



Here's what each part means:

- expression: This is the operation or calculation you want to perform on each item from the iterable to generate the elements of the new list.
- item: This represents each element in the iterable that you're iterating over.
- iterable: This is the collection of elements you want to iterate through, like a list or a range.
- condition (optional): This is an optional part where you can specify a condition that determines whether an item should be included in the new list. If omitted, all items from the iterable will be included.

Advantages of List Comprehension

- More time-efficient and space-efficient than loops.
- Require fewer lines of code.
- Transforms iterative statement into a formula.

```
# Add 1 to 10 numbers to a list
L=[]

for  i in range(1,11):
  L.append(i)
print(L)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
#List comprehension
L=[i for i in range(1,11)]
print(L)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
#Scalar multiplication on a vector
v = [2,3,4]
s=-3
[s*i for i in v]
```

```
[-6, -9, -12]
```

```
# add squares
L=[1,2,3,4,5]
[i**2 for i in L]
```

```
[1, 4, 9, 16, 25]
```

```
# print all numbers divisible by 5 in the range
[i for i in range(1,51) if i%5==0]
```

```
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
```

```
#find the language which starts with letter "p"
lan=['Java','python','php','C','Javascript']
[lan for lan in lan if lan.startswith("p")]
```

```
['python', 'php']
```

```
# Print a (3,3) matrix using list comprehension -> Nested List comprehension
[[i*j for i in range(1,4)] for j in range(1,4)]
```

```
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

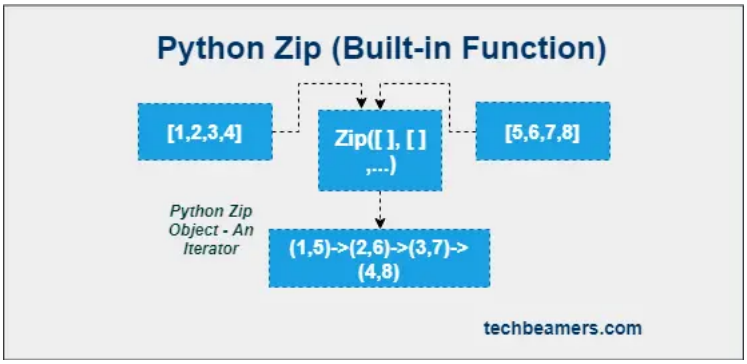
```
# cartesian products -> List comprehension on 2 lists together
L1 = [1,2,3,4]
L2 = [5,6,7,8]

[i*j for i in L1 for j in L2]
```

```
[5, 6, 7, 8, 10, 12, 14, 16, 15, 18, 21, 24, 20, 24, 28, 32]
```

```

```

zip() function :

- The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together.
- If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.




```
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 22]

combined = zip(names, ages)

for name, age in combined:
    print(name, age)

combined_list = list(zip(names, ages))
print(combined_list)


```

```
Alice 25
Bob 30
Charlie 22
[('Alice', 25), ('Bob', 30), ('Charlie', 22)]
```

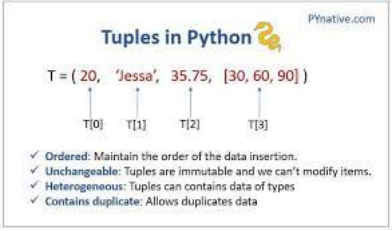
```
# Write a program to add items of 2 lists indexwise

L1 = [1,2,3,4]
L2 = [-1,-2,-3,-4]

list(zip(L1,L2))

[i+j for i,j in zip(L1,L2)]
```

```
[0, 0, 0, 0]
```

```

```

4.Tuple in python

- A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.
- In short, a tuple is an immutable list. A tuple can not be changed in any way once it is created.
- A tuple is created using parentheses.



www.pylenin.com

Indexing Tuples

0	1	2	3	4
(1,	2,	3,	4,	5)
-5	-4	-3	-2	-1

```
1
```

One element tuple

if a tuple includes only one element, you should put a comma after the element.Otherwise ,it is not considered as a tuple.

```
tuple_6 = (0)
print(tuple_6)
print(type(tuple_6))
# Here, you see that the output is an integer

```

```
0
<class 'int'>
```

```
tuple_7 = (0,)
print(tuple_7)
print(type(tuple_7))
# You see that the output is a tuple
```

```
(0,)
<class 'tuple'>
```

```
# creating tuple
str_tuple=("Apple","Orange","Mango")
int_tuple=(15,25,36,84,59)
float_tuple=(2.3,5.6,1.4,9.6)
mixed_tuple= ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
print(str_tuple)
print(int_tuple)
print(float_tuple)
print(mixed_tuple)
```

```
('Apple', 'Orange', 'Mango')
(15, 25, 36, 84, 59)
(2.3, 5.6, 1.4, 9.6)
('Hello', 'Python', 3.14, 1.618, True, False, 32, [1, 2, 3], {1, 2, 3}, {'A': 3, 'B': 8}, (0, 1))
```

Access values of tuple using index

- Value of tuple can be accessed using index number. Index number is always an integer value and starts with 0.
- Negative indexes start from the end of the tuple. Negative index always starts with -1.

For example fruit_tuple=("Apple","Orange","Mango") here index of Mango ,Orange and Apple are -1,-2 and -3.

```
fruit_tuple=("Apple","Orange","Mango")
print("I like ",fruit_tuple[0])
print("I like ",fruit_tuple[2])
```

```
I like  Apple
I like  Mango
```

```
fruit_tuple=("Apple","Orange","Mango")
print(fruit_tuple[-3])#Apple
print(fruit_tuple[-2])#Orange
print(fruit_tuple[-1])#Mango
```

```
Apple
Orange
Mango
```

```
#Access values of tuple using loop

fruit_tuple=("Apple","Orange","Mango")
for name in fruit_tuple:
    print("I like ",name)
```

```
I like  Apple
I like  Orange
I like  Mango
```

Update item of tuple

- We can not change the value of tuple.

```
fruit_tuple=("Apple","Orange","Mango")
#this line will generate error
#because we can't change the value of tuple
fruit_tuple[1]="Banana"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-42-e1f60ce0d290> in <cell line: 4>()
      2 #this line will generate error
      3 #because we can't change the value of tuple
----> 4 fruit_tuple[1]="Banana"

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

SEARCH STACK OVERFLOW

We can update the value of tuple using list.

```
fruit_tuple=("Apple","Orange","Mango")
print("Tuple Before Updation:",fruit_tuple)
#Convert tuple into list
fruit_list=list(fruit_tuple)
#Update Orange with Banana
fruit_list[1]="Banana"
#Convert list into tuple
fruit_tuple=tuple(fruit_list)
print("Tuple after Updation:",fruit_tuple)
```

```
Tuple Before Updation: ('Apple', 'Orange', 'Mango')
Tuple after Updation: ('Apple', 'Banana', 'Mango')
```

Delete item from tuple

- We can't delete item from tuple because it is unchangeable.
- But we can delete a tuple completely using del keyword.

```
fruit_tuple=("Apple","Orange","Mango")
print("Fruit tuple:",fruit_tuple)
del fruit_tuple;
print("Deleted successfully")
#this line will generate error
print(fruit_tuple)
```

```
Fruit tuple: ('Apple', 'Orange', 'Mango')
Deleted successfully
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-46-c9b49cedb3ba> in <cell line: 6>()
      4 print("Deleted successfully")
      5 #this line will generate error
----> 6 print(fruit_tuple)

NameError: name 'fruit_tuple' is not defined
```

SEARCH STACK OVERFLOW

Join two tuples using + symbol

We can join two tuple using plus(+) operator.

```
tuple1=("Apple","Orange","Mango")
tuple2=("Cherry","Grapes","Melon")
#this line will join tuple1 and tuple2
tuple3=tuple1+tuple2
print("tuple3 items")
print(tuple3)
```

```
tuple3 items
('Apple', 'Orange', 'Mango', 'Cherry', 'Grapes', 'Melon')
```

```
#Program to search particular element in tuple

fruit_tuple = ("Apple", "Orange", "Mango")
str=input("Enter any string to search:")
if str in fruit_tuple:
  print(str," is found")
else:
  print("Not found")
```

```
Enter any string to search:Apple
Apple  is found
```

`.max()`

- It is used to get maximum value from the tuple.
- In case of string focus on ASCII value of first letter of tuple items.

```
fruit_tuple=("Apple","Orange","Mango")
print("Fruits tuple :",fruit_tuple)
```

```
print ("Max elements : ", max(fruit_tuple))

animal_tuple=("Zebra","Dog","Elephant")
print("Animal tuple :",animal_tuple)
print ("Max elements : ", max(animal_tuple))

int_tuple=(45,85,36)
print("int tuple :",int_tuple)
print ("Max elements : ", max(int tuple))
```

```
Fruits tuple : ('Apple', 'Orange', 'Mango')
Max elements :  Orange
Animal tuple : ('Zebra', 'Dog', 'Elephant')
Max elements :  Zebra
int tuple : (45, 85, 36)
Max elements :  85
```

`min()`

- It is used to get minimum value from the tuple.
- In case of string focus on ASCII value of first letter of tuple items.

```
fruit_tuple=("Apple","Orange","Mango")
print("Fruits tuple :",fruit_tuple)
print ("Min elements : ", min(fruit_tuple))

animal_tuple=("Zebra","Dog","Elephant")
print("Animal tuple :",animal_tuple)
print ("Min elements : ", min(animal_tuple))

int_tuple=(45,85,36)
print("int tuple :",int_tuple)
print ("Min elements : ", min(int_tuple))
```

```
Fruits tuple : ('Apple', 'Orange', 'Mango')
Min elements :  Apple
Animal tuple : ('Zebra', 'Dog', 'Elephant')
Min elements :  Dog
int tuple : (45, 85, 36)
Min elements :  36
```

`tuple()`

- tuple is used to convert list into tuple.

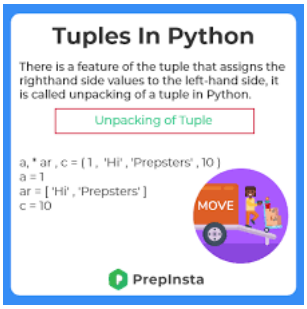
```
#tuple is created using parentheses
fruit_list=["Apple","Orange","Mango"]
print("List Items:",fruit_list)
#this line convert list into tuple
fruit_tuple=tuple(fruit_list)
print("Tuple Items :",fruit_tuple)
```

```
List Items: ['Apple', 'Orange', 'Mango']
Tuple Items : ('Apple', 'Orange', 'Mango')
```

```
#repetition of a tuple
rep_tup = (1,2,3,4)
rep_tup*2

```

```
(1, 2, 3, 4, 1, 2, 3, 4)
```

Tuple Unpacking



```
# tuple unpacking
a,b,c = (1,2,3)
print(a,b,c)
print()
a,b,*others = (1,2,3,4)
print(a,b)
print(others)
```

```
1 2 3

1 2
[3, 4]
```

```
a,b = (1,2,3)
print(a,b)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-22f327f11d4b> in <cell line: 1>()
----> 1 a,b = (1,2,3)
      2 print(a,b)

ValueError: too many values to unpack (expected 2)
```

SEARCH STACK OVERFLOW

```
# using type conversion
t6 = tuple('hello')
print(t6)
```

```
('h', 'e', 'l', 'l', 'o')
```

```

```

tuple function

```
# len/sum/min/max/sorted
t = (1,2,3,4)
print('length',len(t))

print('sum',sum(t))

print('min',min(t))

print('max',max(t))

sorted(t,reverse=True)
```

```
length 4
sum 10
min 1
max 4
[4, 3, 2, 1]
```

```
# count

t = (1,2,3,4,5)

t.count(1) #count frequency of elements
```

```
1
```

Difference between Lists and Tuples

- Syntax
- Mutability
- Speed
- Memory
- Built in functionality
- Error prone
- Usability

```
import time

```

```
L = list(range(100000000))
T = tuple(range(100000000))

start = time.time()
for i in L:
  i*5
print('List time',time.time()-start)

start = time.time()
for i in T:
  i*5
print('Tuple time',time.time()-start)
```

```
List time 9.541415929794312
Tuple time 9.383216142654419
```

```
import sys

L = list(range(1000))
T = tuple(range(1000))

print('List size',sys.getsizeof(L))
print('Tuple size',sys.getsizeof(T))

```

```
List size 8056
Tuple size 8040
```

```
# zipping tuples
a = (1,2,3,4)
b = (5,6,7,8)

tuple(zip(a,b))
```

```
((1, 5), (2, 6), (3, 7), (4, 8))
```

Slicing

- To obtain a new tuple from the current tuple, the slicing method is used.

```
# Obtaining a new tuple from the index 2 to index 6
tuple_1 = ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
tuple_1[2:7]

```

```
(3.14, 1.618, True, False, 32)
```

```
 #Obtaining tuple using negative indexing
tuple_1 = ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
tuple_1[-4:-1]

```

```
([1, 2, 3], {1, 2, 3}, {'A': 3, 'B': 8})
```

```
#reverse
tuple_1[::-1]
```

```
((0, 1),
 {'A': 3, 'B': 8},
 {1, 2, 3},
 [1, 2, 3],
 32,
 False,
 True,
 1.618,
 3.14,
 'Python',
 'Hello')
```

Nested tuple

- In Python , a tuple written inside another tuple is known as a nested tuple.

```
# Take a nested tuple
nested_tuple =('biotechnology', (0, 5), ('fermentation', 'ethanol'), (3.14, 'pi', (1.618, 'golden ratio')) )
nested_tuple

```

```
('biotechnology',
 (0, 5),
 ('fermentation', 'ethanol'),
 (3.14, 'pi', (1.618, 'golden ratio')))
```

```
1
```

5.Sets in Python

- A set is an unordered collection of items.
- Every set element is unique (no duplicates) and must be immutable (cannot be changed).

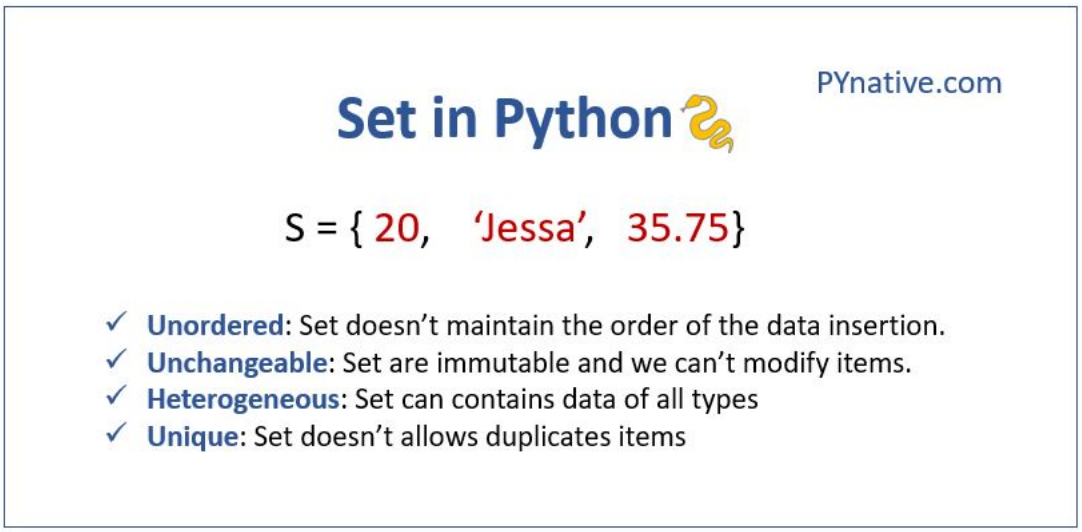
However, a set itself is mutable. We can add or remove items from it.

- A set itself may be modified, but the elements contained in the set must be of an immutable type
- You can denote a set with a pair of curly brackets {}.

Sets can also be used to perform mathematical set operations like

- union,
- intersection,
- symmetric difference, etc.

Characterstics:




```
# The empty set of curly braces denotes the empty dictionary, not empty set
x = {}
print(type(x))

```

```
<class 'dict'>
```

```
# To take a set without elements, use set() function without any items
y = set()
print(type(y))
```

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

```
# create a set
str_set={"Apple","Orange","Mango"}
int_set={15,25,36,84,59}
float_set={2.3,5.6,1.4,9.6}
mixed_set={"Easy",165,25.3}
print(str_set)
print(int_set)
print(float_set)
print(mixed_set)
```

```
{'Apple', 'Orange', 'Mango'}
{36, 84, 25, 59, 15}
{1.4, 2.3, 5.6, 9.6}
{25.3, 165, 'Easy'}
```

Converting list into set

```
# A list can convert to a set
# Take a list
nlis = ['Hello Python!', 3.14, 1.618, 'Hello World!', 3.14, 1.618, True, False, 2022]
# Convert the list to a set
set2 = set(nlis)
set2
```

```
{1.618, 2022, 3.14, False, 'Hello Python!', 'Hello World!', True}
```

Access values of set using loop

- We can't access items of set using index value because it is unordered collection of data.
- We cannot be sure in which order the items will appear because sets are unordered. For better understanding see the example.

```
fruit_set={"Apple","Orange","Mango"}
print("Fruit set :",fruit_set) # run this same code again and again ,see difference in order
```

```
Fruit set : {'Apple', 'Orange', 'Mango'}
```

Update item of set

- We can not change the value of set.

```
ruit_set={"Apple","Orange","Mango"}
#this line will generate error
#because we can't change the value of set
fruit_set[1]="Banana"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-67-c67aaf149714> in <cell line: 4>()
      2 #this line will generate error
      3 #because we can't change the value of set
----> 4 fruit_set[1]="Banana"

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

SEARCH STACK OVERFLOW

`length of set`

- len() function is used to get length of set.

```
fruit_set={"Apple","Orange","Mango"}
print("Length of set is ",len(fruit_set))
```

```
Length of set is  3
```

```
#Add items into set
#add() function is used to add new item in a set.

fruit_set={"Apple","Orange","Mango"}
print("Fruit Set:",fruit_set)
#this line will add
#Cherry at the end of set
fruit_set.add("Cherry")
print("Fruit Set:",fruit_set)
```

```
Fruit Set: {'Apple', 'Orange', 'Mango'}
Fruit Set: {'Apple', 'Cherry', 'Orange', 'Mango'}
```

`updation()`

- Add more than one item to a set
- update() function is used to add more than one item to a set.
- update() and add() function discard the duplicate elements.

```
fruit_set={"Apple","Orange","Mango"}
print("Fruit Set:",fruit_set)
#Add multiple items to a set
print()
fruit_set.update(["Cherry","Banana","Apple"])
print("Fruit Set:",fruit_set)
```

```
Fruit Set: {'Apple', 'Orange', 'Mango'}

Fruit Set: {'Apple', 'Orange', 'Mango', 'Banana', 'Cherry'}
```

Deletion

- Delete item from set using remove() function
- remove() function is used to remove specified item from a set.

```
fruit_set={"Apple","Orange","Mango"}
print("Fruit Set:",fruit_set)
#this line will remove Orange from set
print()
fruit_set.remove("Orange")
print("Fruit Set:",fruit_set)
```

```
Fruit Set: {'Apple', 'Orange', 'Mango'}

Fruit Set: {'Apple', 'Mango'}
```

Discard

- Delete item from set using discard() function
- discard() function is also used to remove specified item from a set.

kye point----The difference between remove() and discard() function is that if the specified element does not exist in the set then remove() function will raise an error but discard() function will not raise an error.

```
fruit_set={"Apple","Orange","Mango"}
print("Fruit Set:",fruit_set)
#this line will remove Orange from set
print()
fruit_set.discard("Orange")
print("Fruit Set:",fruit_set)
```

```
Fruit Set: {'Apple', 'Orange', 'Mango'}

Fruit Set: {'Apple', 'Mango'}
```

Join two sets

We can join two set using union() function

```
set1={"Apple","Orange","Mango"}
set2={"Cherry","Grapes","Melon"}
#this line will join set1 and set2
set3=set1.union(set2)
print("set3 items")
print()
print(set3)
```

```
set3 items

{'Apple', 'Orange', 'Mango', 'Melon', 'Grapes', 'Cherry'}
```

```
#Program to search particular element in set

fruit_set = {"Apple", "Orange", "Mango"}
str=input("Enter any string to search:")
if str in fruit_set:
  print(str," is found")
else:
  print("Not found")
```

```
Enter any string to search:Mango
Mango  is found
```

Clear set

clear() function is used to clear or empty the set.

```
fruit_set={"Apple","Orange","Mango"}
print("Before clear")
print(fruit_set)
#this line will empty the list
fruit_set.clear()
print("After clear")
print(fruit_set)
```

```
Before clear
{'Apple', 'Orange', 'Mango'}
After clear
set()
```

Delete set

del keyword is also used to delete set completely

```
fruit_set={"Apple","Orange","Mango"}
print("Set Items")
print(fruit_set)
#this line will delete  set
del fruit_set
print("Deleted successfully")
```

```
Set Items
{'Apple', 'Orange', 'Mango'}
Deleted successfully
```

set operation

```
# len/sum/min/max/sorted
s = {3,1,4,5,2,7}
len(s)
sum(s)
min(s)
max(s)
sorted(s,reverse=True)
```

```
[7, 5, 4, 3, 2, 1]
```

```
# Take two sets
set4 = set(['Hello Python!', 3.14, 1.618, 'Hello World!'])
set5 = set([3.14, 1.618, True, False, 2022])
# Printing two sets
set4, set5

```

```
({1.618, 3.14, 'Hello Python!', 'Hello World!'},
 {False, True, 1.618, 3.14, 2022})
```

```
#To find the intersect of two sets using &
intersection = set4 & set5
intersection
```

```
{1.618, 3.14}
```

```
#To find the intersect of two sets, use intersection() function
set4.intersection(set5) # The output is the same as that of above

```

```
{1.618, 3.14}
```

difference() function

To find the difference between two sets

```
print(set4.difference(set5))
print(set5.difference(set4))
# The same process can make using subtraction operator as follows:
```

```
print(set4-set5)
print(set5-set4)
```

```
{'Hello Python!', 'Hello World!'}
{False, True, 2022}
{'Hello Python!', 'Hello World!'}
{False, True, 2022}
```

Set comparison

```
print(set4>set5)
print(set5>set4)
print(set4==set5)

```

```
False
False
False
```

`union()` function

it corresponds to all the elements in both sets

```
set4.union(set5)

```

```
{1.618, 2022, 3.14, False, 'Hello Python!', 'Hello World!', True}
```

issuperset() and issubset() functions

To control if a set is a superset or a subset of another set

```
set(set4).issuperset(set5)
```

```
False
```

```
set(set4).issubset(set5)

```

```
False
```

```
print(set([3.14, 1.618]).issubset(set5))
print(set([3.14, 1.618]).issubset(set4))
print(set4.issuperset([3.14, 1.618]))
print(set5.issuperset([3.14, 1.618]))
```

```
True
True
True
True
```

No mutable sequence in a set

A set can not have mutable elements such as list or dictionary in it. If any, it returns error as follows:

```
set6 = {'Python', 1,2,3, [1,2,3]}
set6

```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-91-fa4d977bac6b> in <cell line: 1>()
----> 1 set6 = {'Python', 1,2,3, [1,2,3]}
      2 set6

TypeError: unhashable type: 'list'
```

SEARCH STACK OVERFLOW

Frozenset

Frozen set is just an immutable version of a Python set object

```
# create frozenset
fs1 = frozenset([1,2,3])
fs2 = frozenset([3,4,5])

fs1 | fs2
# what works and what does not
# works -> all read functions
# does't work -> write operations
```

```
frozenset({1, 2, 3, 4, 5})
```

```
#set comprehension
# examples

{i**2 for i in range(1,11) if i>5}
```

```
{36, 49, 64, 81, 100}
```

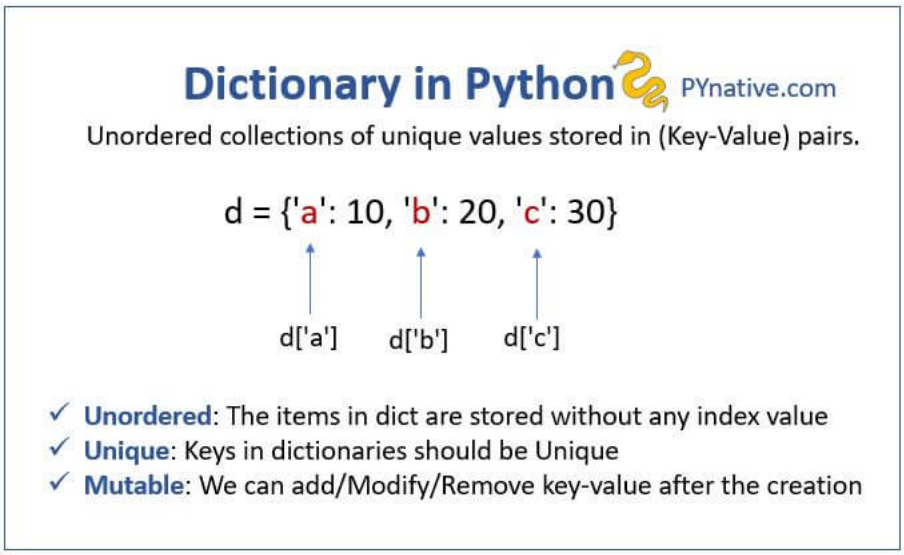
```

```

6.Dictionary in Python

- It is an unordered collection of data of different data types.
- Items of dictionary can be changed.
- A dictionary is created using curly brackets.
- Dictionary items are in the form of key-value pairs.
- Dictionary can nested and can contain another dictionary
- In some languages it is known as map or assosiative arrays.

dict = { 'name' : 'nitish' , 'age' : 33 , 'gender' : 'male' }

Characterstics:




```
#creating dictionay
student={
    'name':'Rocky Singh',
    'rollno':305,
    'percent':85.5
}
student
```

```
{'name': 'Rocky Singh', 'rollno': 305, 'percent': 85.5}
```

```
#empty dictionary
emp_dict={}
print(type(emp_dict))
emp_dict
```

```
<class 'dict'>
{}
```

```
# 1 D dictionary
one_d={
    'name':'nitish','gen':'male'
}
print(one_d)
```

```
{'name': 'nitish', 'gen': 'male'}
```

```
#with mixed keys
d2={
    (1,2,3):1,
    "hello":'world'
}
d2

```

```
{(1, 2, 3): 1, 'hello': 'world'}
```

```
# 2D dictionary -Json
s={
    'name':'nitish',
    'college':'bit',
    'sem':4,
    'subjects':{
        'dsa':50,
        'math':67,
        'english':34

    }
}
s
```

```
{'name': 'nitish',
 'college': 'bit',
 'sem': 4,
 'subjects': {'dsa': 50, 'math': 67, 'english': 34}}
```

```
# sample dictionary
sample_dict={
    'key_1':3.14,
    'key_2':1.618,
    'key_3':True,
    'key_4':[3.14,1.618],
    'key_5':(3.14,1.618),
    'key_6':2022,(23,44,45):'pi and golden ration'
}
sample_dict
```

```
{'key_1': 3.14,
 'key_2': 1.618,
 'key_3': True,
 'key_4': [3.14, 1.618],
 'key_5': (3.14, 1.618),
 'key_6': 2022,
 (23, 44, 45): 'pi and golden ration'}
```

access the items of dictionary

- we can access items of dictionary using key name.

```
student={
    'name':'Rocky Singh',
    'rollno':305,
    'percent':85.5
}
print("student name:",student['name'])
```

```
student name: Rocky Singh
```

- we can also access items of dictionary using get() function.

```
student={
    'name':'Rocky Singh',
    'rollno':305,
    'percent':85.5
}
print("student name:",student.get('name'))
```

```
student name: Rocky Singh
```

```
# Access values of dictionary using loop
# Primary key
student={
    'name':'Rocky Singh',
    'rollno':305,
    'percent':85.5
}
for key in student:
  print(key,student[key])
```

```
name Rocky Singh
rollno 305
percent 85.5
```

Double-click (or enter) to edit

Update items of dictionary

- we can change the value of dictionary

```
student={
    'name':'Rocky Singh',
    'rollno':305,
    'percent':85.5
}
print("dictionary before Update:\n",student)
# this line will replace 85.6 with 92.3
student['percent']=92.3
print('Dictionary after Update:\n',student)
```

```
dictionary before Update:
 {'name': 'Rocky Singh', 'rollno': 305, 'percent': 85.5}
Dictionary after Update:
 {'name': 'Rocky Singh', 'rollno': 305, 'percent': 92.3}
```

Length of dictionary

- `len()` function is used to get length of dictionary

```
student={
    'name':'Rocky Singh',
    'rollno':305,
    'percent':85.5,
    'branch':'Computer Science'
}
print("Length of dictionary :",len(student))
```

```
Length of dictionary : 4
```

Add items into dictionary

- we can also add new item to a dictionary using ney key.

```
student={
    'name':'Rocky Singh',
    'rollno':305,
    'percent':85.5
}
print("Dictinary",student)
# adding new item to a dictionary
student['branch']="Computer Science"
print("Dictionary:",student)
```

```
Dictinary {'name': 'Rocky Singh', 'rollno': 305, 'percent': 85.5}
Dictionary: {'name': 'Rocky Singh', 'rollno': 305, 'percent': 85.5, 'branch': 'Computer Science'}
```

Delete item from dictionary using pop() function

- pop() function is used to remove specified item from a dictionary

```
student={
    'name':'Rocky Singh',
    'rollno':305,
    'percent':85.5
}
print("Dictionary before deletion:",student)
# delete item from dictionary
student.pop('rollno')
print("Dictinary after deletion:",student)
```

```
Dictionary before deletion: {'name': 'Rocky Singh', 'rollno': 305, 'percent': 85.5}
Dictinary after deletion: {'name': 'Rocky Singh', 'percent': 85.5}
```

```
# clear () function is used to clear or empty the dictionary
student={
    'name':'Rocky Singh',
    'rollno':305,
    'percent':85.5
}
print("Dictionary before clear:",student)
# this line will empty the dictionary
student.clear()
print("Dictionary after clear:",student)
```

```
Dictionary before clear: {'name': 'Rocky Singh', 'rollno': 305, 'percent': 85.5}
Dictionary after clear: {}
```

formkeys() function

- It returns a new dictionary with the certain sequence of the items as the keys of the dictionary and the values are assigned with None

```
keys = {'A', 'T', 'C', 'G'}
sequence = dict.fromkeys(keys)
print(sequence)
```

```
{'C': None, 'A': None, 'G': None, 'T': None}
```

```
d = {'name':'nitish','gender':'male','age':33}

for i in d:
  print(i,d[i])
```

```
name nitish
gender male
age 33
```

```
# len/sorted
print(len(d))
print(d)
print(sorted(d,reverse=True))
print(max(d))
```

```
3
{'name': 'nitish', 'gender': 'male', 'age': 33}
['name', 'gender', 'age']
name
```

```
# items/keys/values
print(d)

print(d.items())
print(d.keys())
print(d.values())
```

```
{'name': 'nitish', 'gender': 'male', 'age': 33}
dict_items([('name', 'nitish'), ('gender', 'male'), ('age', 33)])
dict_keys(['name', 'gender', 'age'])
dict_values(['nitish', 'male', 33])
```

Dictionary Comprehension

```
{ key: value for vars in iterable }
```

```
# print 1st to 10 numbers and their squares
{i:i**2 for i in range (1,11)}
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

```
# to take values in the form of tuple
distances = {'delhi':1000,'mumbai':2000,'bangalore':3000}
print(distances.items())
```

```
dict_items([('delhi', 1000), ('mumbai', 2000), ('bangalore', 3000)])
```

```
# using existing dict
distances = {'delhi':1000,'mumbai':2000,'bangalore':3000}
{key:value*0.62 for (key,value) in distances.items()}
```

```
{'delhi': 620.0, 'mumbai': 1240.0, 'bangalore': 1860.0}
```

```
# using zip
days = ["Sunday", "Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]
temp_C = [30.5,32.6,31.8,33.4,29.8,30.2,29.9]

{i:j for (i,j) in zip(days,temp_C)}
```

```
{'Sunday': 30.5,
 'Monday': 32.6,
 'Tuesday': 31.8,
 'Wednesday': 33.4,
 'Thursday': 29.8,
 'Friday': 30.2,
 'Saturday': 29.9}
```

```
# using if condition
products = {'phone':10,'laptop':0,'charger':32,'tablet':0}

{key:value for (key,value) in products.items() if value>0}
```

```
{'phone': 10, 'charger': 32}
```

```
# Nested Comprehension
# print tables of number from 2 to 4
{i:{j:i*j for j in range(1,11)} for i in range(2,5)}
```

```
{2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18, 10: 20},
 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15, 6: 18, 7: 21, 8: 24, 9: 27, 10: 30},
 4: {1: 4, 2: 8, 3: 12, 4: 16, 5: 20, 6: 24, 7: 28, 8: 32, 9: 36, 10: 40}}
```

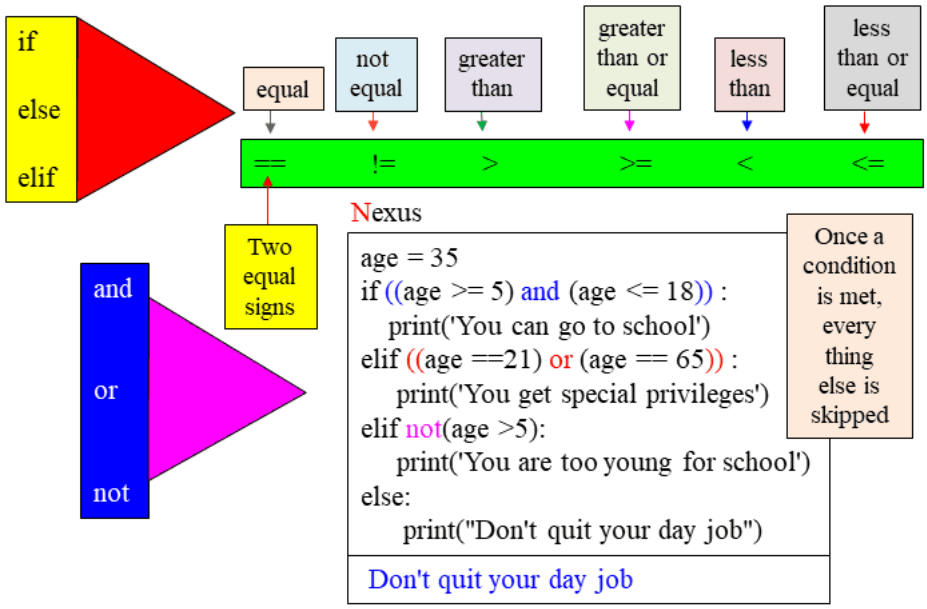
```
```

7. Conditions in Python

Comparison operators

Comparison operations compare some value or operand and based on a condition, produce a Boolean. Python has six comparison operators as below:

- Less than (<)
- Less than or equal to (<=)
- Greater than (>)
- Greater than or equal to (>=)
- Equal to (==)
- Not equal to (!=)



Logical operators

Logical operators are used to combine conditional statements.

- and: Returns True if both statements are true
- or: Returns True if one of the statements is true
- not: Reverse the result, returns False if the result is true

```
# Take a variable
golden_ratio = 1.618
# Condition less than
print(golden_ratio<2) # The golden ratio is lower than 2, thus the output is True
print(golden_ratio<1) # The golden ratio is greater than 1, thus the output is False

```

```
True
False
```

```
# Take a variable
golden_ratio = 1.618
# Condition less than or equal to
print(golden_ratio<=2) # The golden ratio is lower than 2, thus the condition is True.
print(golden_ratio<=1) # The golden ratio is greater than 1, thus the condition is False.
print(golden_ratio<=1.618) # The golden ratio is equal to 1.618, thus the condition is True.

```

```
True
False
True
```

```
# Take a variable
golden_ratio = 1.618
# Condition greater than
print(golden_ratio>2) # The golden ratio is lower than 2, thus the condition is False.
print(golden_ratio>1) # The golden ratio is greater than 1, thus the condition is True
```

```
False
True
```

```
# Take a variable
golden_ratio = 1.618
# Condition greater than or equal to
print(golden_ratio>=2) # The golden ratio is not greater than 2, thus the condition is False.
print(golden_ratio>=1) # The golden ratio is greater than 1, thus the condition is True.
print(golden_ratio>=1.618) # The golden ratio is equal to 1.618, thus the condition is True
```

```
False
True
True
```

```
# Take a variable
golden_ratio = 1.618
```

```
# Condition equal to
print(golden_ratio==2) # The golden ratio is not equal to 1.618, thus the condition is False.
print(golden_ratio==1.618) # The golden ratio is equal to 1.618, thus the condition is True.
```

```
 #Take a variable
golden_ratio = 1.618
# Condition not equal to
print(golden_ratio!=2) # The golden ratio is not equal to 1.618, thus the condition is True.
print(golden_ratio!=1.618) # The golden ratio is equal to 1.618, thus the condition is False.
```

The comparison operators are also employed to compare the letters/words/symbols according to the ASCII

(https://www.asciitable.com/) value of letters.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

```
# Compare strings
print('Hello' == 'Python')
print('Hello' != 'Python')
print('Hello' <= 'Python')
print('Hello' >= 'Python')
print('Hello' < 'Python')
print('Hello' > 'Python')
print('B'>'A') # According to ASCII table, the values of A and B are equal 65 and 66, respectively.
print('a'>'b') # According to ASCII table, the values of a and b are equal 97 and 98, respectively.
print('CD'>'DC') # According to ASCII table, the value of C (67) is lower than that of D (68)
# The values of uppercase and lowercase letters are different since python is case sensitive.
```

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

```

```

if statement

Syntax:

'''

```
if condition :
      body
```

'''

- It is used to test the condition and the result is based on the condition.
- if the condition is true its body will execute otherwise does not execute.

```
# Example 1
no = int(input('Enter any number:'))
if no>5:
  print("Number is greater than 5")
```

```
Enter any number:8
Number is greater than 5
```

```
# Example -2 check given number is positive or  negative or zero
no = int(input('Enter any Number:'))
if no>0:
  print("Number is positive")
if no<0:
  print("Number is negative")
if no==0:
  print("Number is Zero")
```

```
Enter any Number:-8
Number is negative
```

If else statement

syntax:

```
'''
if condition :
  body_of_if
else:
  body_of_else
```

- It is used to test the condition and gives the output both situation either condition true or false.
- If the condition is true body of if will execute otherwise body of else execute.

```
#Example -1
no = int(input("Enter any number:"))
if no> 5:
  print('Number is greater than 5')
else:
  print('Number is less than 5')
```

```
Enter any number:3
Number is less than 5
```

```
# Example -2 check given number is even or odd
no = int(input("Enter any number:"))
if no%2 ==0:
  print('Number is even')
else:
  print("Number is odd")
```

```
Enter any number:11
Number is odd
```

```
# Example-3
number=int(input("Enter a number:"))

print(f'The entered number is:{number}')

if number % 2==0:
  print(f'The entered number {number} is even')
else:
 print(f'The entered number {number} is odd')
```

```
Enter a number:52
The entered number is:52
The entered number 52 is even
```

If else if ladder statement

Syntax:

```
'''
if condition :
    statement 1

elif condition:
    statement 2

elif condition:
    statement 3

else:
    statement 4
```

- It is used to test the condition
- if excutes only one condition at a time
- the condition which is true first from the top will execute

```
# Example 1
no = 7
if no> 10:
  print('Hello1')
elif no >5:
  print('HHello2')
elif no >0:
  print('Hello3')
else:
  print('Hello4')
```

```
HHello2
```

```
# Example-2 show result according to percent
percent = float(input('Enter your percentage.'))
if percent >= 60:
  print('First division')
elif percent >=45:
  print("Second division")
elif percent >= 33:
  print("Third division")
else:
  print("Sorry!!1 You are fail.")

```

```
Enter your percentage.25
Sorry!!1 You are fail.
```

Nested if statement

syntax:

```
'''
if condition:
  #statements
 if condition:
   #statement
   if condition:
    #statement
```

- It is used to test the condition.
- One if inside another if is called nested if.

```
# Example -1 Find greatest value in three number
print('Enter three number:')
a=int(input())
b=int(input())
c=int(input())

if a>b:
  if a>c:
    print(a,"is greatest")
if b>a:
```

```
  if b>c:
    print(b,"is greatest")
if c>a:
  if c>b:
    print(c,"is greatest")
```

```
Enter three number:
2
3
1
3 is greatest
```

```
# min of 3 number

a = int(input('first num'))
b = int(input('second num'))
c = int(input('third num'))

if a<b and a<c:
  print('smallest is',a)
elif b<c:
  print('smallest is',b)
else:
  print('smallest is',c)
```

```
first num4
second num6
third num7
smallest is 4
```

```
# menu driven calculator
menu = input("""
Hi! how can I help you.
1. Enter 1 for pin change
2. Enter 2 for balance check
3. Enter 3 for withdrawl
4. Enter 4 for exit
""")

if menu == '1':
  print('pin change')
elif menu == '2':
  print('balance')
else:
  print('exit')
```

```
Hi! how can I help you.
1. Enter 1 for pin change
2. Enter 2 for balance check
3. Enter 3 for withdrawl
4. Enter 4 for exit
1
pin change
```

```
1
```

8.Introduction to loops in python

- In programming loops are structure that repeats a sequence of instruction until a specific condition is met.
- Loops are most important topic to start any programming language.
- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string)

syntax:

```
'''for (i)_variable in sequence:
      body of for_loop
```

Lets check why loops in python are so important Assume:

you have to print “hello world” for 5 times without using loops and with using loops and see the difference.

```
  #without using loops
  print("hello world ")
  print("hello world ")
  print("hello world ")
  print("hello world ")
```

```
print("hello world")
```

```
hello world
hello world
hello world
hello world
hello world
```

```
#using loops
for i in range(5):
    print("hello world")
```

```
hello world
hello world
hello world
hello world
hello world
```

```
# example for range function
for x in range(2,21,2): # this will print table of 2
 #print(x)

 print(x,end=",")

```

```
2,4,6,8,10,12,14,16,18,20,
```

Advantages of loops

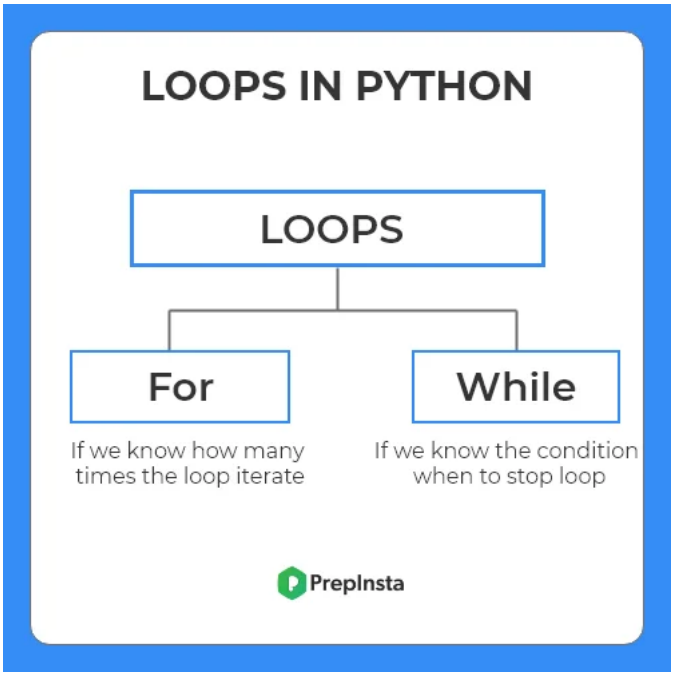
We can reduce the size of the code and code becomes more readable with loops. If we need to do same operation for 100 times then we can't do this by manual typing. So we go for loops.

`range()` function:

- range function is mostly used in for loops to traverse the set of code for specified numbers of times.
- 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.

Syntax:

```
'''range(start,end,incrementer)'''
```



Types of loops in python

- for loop
- while loop

NOTE: There is no do-while loop in python

For loop

Python for loop is different from other programming languages. For loop is used for iterating through string, list, set, tuple, dictionary . For loop is mostly used as iterator in Python.

For loop can be used in two ways , see the below code for understanding.

```
# for loop with string
str ="Python"
#loop iterating to elements of the string
for x in str:
  print(x,end=",")
```

```
P,y,t,h,o,n,
```

```
# for loop with list
years=[2005,2006,2007,2008,2009,2010]

# #loop iterating through the indices of the list
for x in range(6):
  print(years[x], end=",")
print()
#loop iterating to elements of the list
for x in years:
  print(x,end=",")


```

```
2005,2006,2007,2008,2009,2010,
2005,2006,2007,2008,2009,2010,
```

```
# Take an example
years = [2005, 2006, 2007, 2008, 2009, 2010]
for i in range(len(years)):
 print(years[i])

```

```
2005
2006
2007
2008
2009
2010
```

```
# Take a list and print the elements using for loop with indexes
languages = ['Python', 'Java', 'JavaScript', 'C', 'C++', 'PHP']
for i in range(len(languages)):
 print(i, languages[i])
```

```
0 Python
1 Java
2 JavaScript
3 C
4 C++
5 PHP
```

```

```

While loop

- While loop is a entry controlled looping statement.
- While loop works on the condition given in the loop.
- It runs until the given condition is true , once the condition becomes false it exits from the loop.

syntax:

```
'''
while condition:
     body_of_loop
```

- Note: remember to increment i, or else the loop will continue forever.
- The while loop requires relevant (i) variables to be ready or define.

To understand more on while loop see the below code.

```
i=1
while i<= 10:
  print(i)
  i=i+1
```

```
1
2
3
4
5
6
7
8
9
10
```

```
# Example -Find factorial of any number
# factorial of 5!= 1x2x3x4x5

i=1
fact=1
no=int(input("Enter any number:"))
while i<=no:
  fact= fact*i
  i=i+1
print('factorial of',no,"is",fact)
```

```
Enter any number:6
factorial of 6 is 720
```

```
# while loop with string
str="Easy"
i=0
while i < len(str):
  print(str[i])
  i=i+1
```

```
E
a
s
y
```

```
number = int(input('enter the number'))

i = 1

while i<11:
  print(number,'*',i,'=',number * i)
  i += 1
```

```
enter the number2
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
```

```
# while loop with else

x = 1

while x < 3:
  print(x)
  x += 1

else:
  print('limit crossed')
```

```
1
2
limit crossed
```

What is loop control or Jump statements?

loop control statements are the statements which controls the execution of loops.

Types of loop control statements are :

- break

used to terminate the loop

- continue

used to escape from the present iteration

- pass

pass doesn't effect the code but it is used when you need to statement but not to execute that

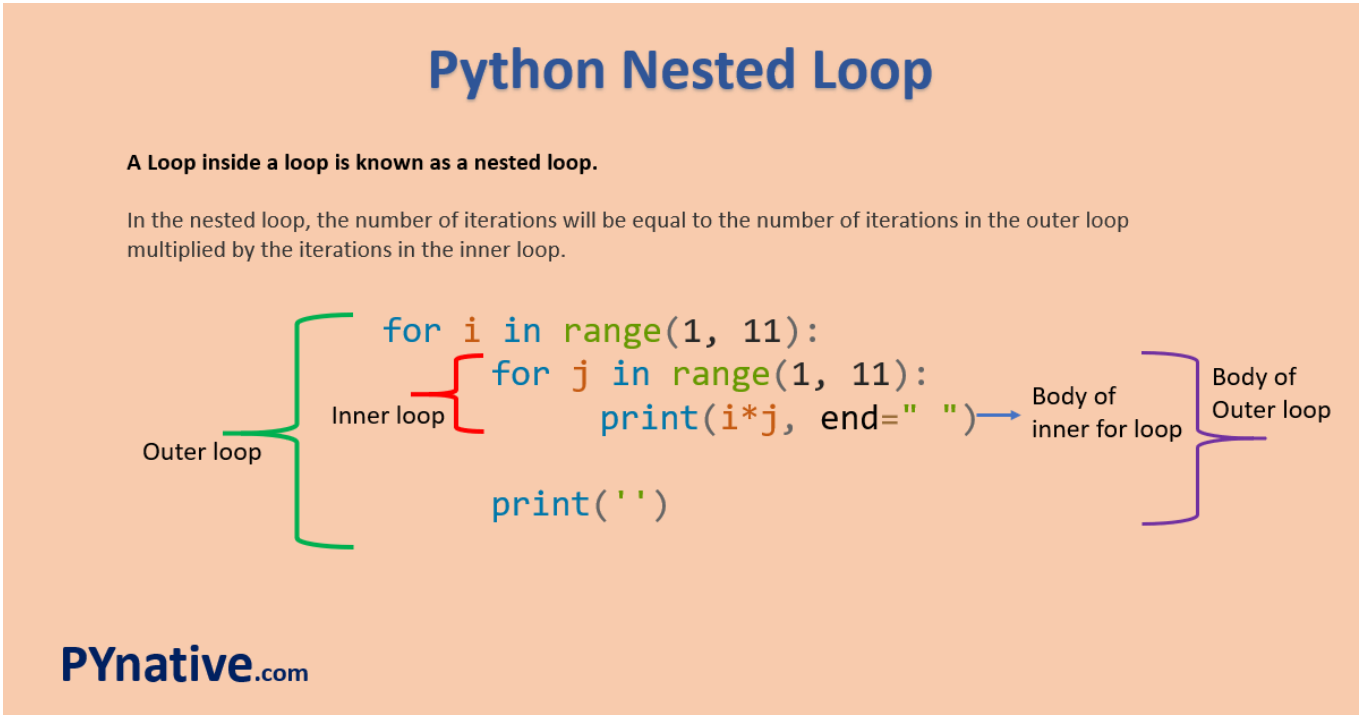
```
#using break statement
n=0
print("Output by using break statement :")
while(n<5):
    if n==3:
        break
    print(n)
    n+=1
```

```
Output by using break statement :
0
1
2
```

```
#using continue and pass statement
n=0
print("Output by using continue statement :")
while(n<5):
    if n==3:
        n=n+1
        continue
    else:
        pass
    print(n)
    n=n+1
```

```
Output by using continue statement :
0
1
2
4
```

```
1
```




```
# Examples -> unique pairs

for i in range(1,5):
```

```
  for j in range(1,5):
```

```
1 1
1 2
1 3
1 4
2 1
2 2
2 3
2 4
3 1
3 2
3 3
3 4
4 1
4 2
4 3
4 4
```

```
# code here

rows = int(input('enter number of rows'))

for i in range(1,rows+1):
  for j in range(1,i+1):
    print('*',end='')
  print()
```

```
enter number of rows3
*
**
***
```

```
# Code here
rows = int(input('enter number of rows'))

for i in range(1,rows+1):
  for j in range(1,i+1):
    print(j,end='')
  for k in range(i-1,0,-1):
    print(k,end='')

  print()
```

```
enter number of rows4
1
121
12321
1234321
```

```
# Write a program to count the number of words in a string without split()

s = input('enter the string')
L = []
temp = ''
for i in s:

  if i != ' ':
    temp = temp + i
  else:
    L.append(temp)
    temp = ''

L.append(temp)
print(L)

```

```
enter the string4
['4']
```

```
# Write a program that can check whether a given string is palindrome or not.
# abba
# malayalam

s = input('enter the string; ')
flag = True
for i in range(0,len(s)//2):
  if s[i] != s[len(s) - i -1]:
    flag = False
    print('Not a Palindrome')
    break

if flag:
```

```
print('Palindrome')

```

```
enter the string; cuttuc
Palindrome
```

```

```

9.Functions in Python