Python – Basic Operators

Python language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Python Arithmetic Operators:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

Python Comparison Operators:

Operato r	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

Python Assignment Operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	c = a + b will assign value of a + b into c
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**=	Exponent AND assignment operator, Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//=	Floor Division and assigns a value, Performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Bitwise Operators:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(a & b) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(a | b) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(a ^ b) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~a) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	a << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 will give 15 which is 0000 1111

Python Logical Operators:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a and b) is false.

Python Membership Operators:

In addition to the operators discussed previously, Python has membership operators, which test for membership in a sequence, such as strings, lists, or tuples.

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here **in** results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here **not in** results in a 1 if x is a member of sequence y.

Python Membership Operators:

Example of membership operator:

```
a=4
list1=[1,2,3,4,5]
    if (a in list1):
                   print("a is available in list")
         else:
                    print("a is not available in list")
Output:
    a is available in list
```

Python Identity Operators:

- Identity operators compare the memory locations of two objects.
- identity operators are used to determine whether a value is of a certain class or type.
- They are usually used to determine the type of data a certain variable contains.

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Python Identity Operators:

Example of Python Identity Operators:

Example:

```
x = 8
if (type(x) is int):
        print("true")
else:
        print("false")
```

Output:

```
true
```

Example:

```
x = 8.5
if (type(x) is not int):
        print("true")
else:
        print("false")
```

Output:

```
true
```

Python Operators Precedence

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Python Output statements:

- We use the print() function to output data to the standard output device (screen).
- syntax of the print() function is:

print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

- Here, objects is the value(s) to be printed.
- The sep separator is used between the values. It defaults into a space character.
- After all values are printed, end is printed. It defaults into a new line.
- The file is the object where the values are printed and its default value is sys.stdout (screen). Here is an example to illustrate this.

Python Output statements:

Example:

```
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep='*')
print(1, 2, 3, 4, sep='#', end='&')
Output:
1 2 3 4
1*2*3*4
1#2#3#4&
```

- we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.
- **Example:**

```
>>> x = 5; y = 10
 >>> print('The value of x is {} and y is {}'.format(x,y))
 The value of x is 5 and y is 10
```

- Here, the curly braces {} are used as placeholders. We can specify the order in which they are printed by using numbers (tuple index).

Python Input statements:

- We want to take the input from the user. In Python, we have the input() function to allow this.
- The syntax for input() is:

  input([prompt])

- Here prompt is the string we wish to display on the screen. It is optional.
- **Example:**

```
>>> num = input('Enter a number: ')
Enter a number: 20
>>> num
‘20‘
>>> int(‘20')
20
>>> float(‘20')
 20.0
```

Python – IF...ELIF...ELSE Statement

- The syntax of the if statement is:

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

```
Example:
   var1 = 100
   if var1:
       print "1 - Got a true expression value"
       print var1
   var2 = 0
   if var2:
       print "2 - Got a true expression value"
       print var2
print "Good bye!"
```

Example:

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even")
```

Output:

```
enter the number? 8
 Number is even
```

Program to print the largest of the three numbers.

Example:

```
a = int(input("Enter a? "));
b = int(input("Enter b? "));
c = int(input("Enter c? "));
if a>b and a>c:
    print("a is largest");
if b>a and b>c:
    print("b is largest");
if c>a and c>b:
    print("c is largest");
```

Output:

```
Enter a? 10
Enter b? 5
Enter c? 15
c is largest
```

Python – IF...ELIF...ELSE Statement

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

Example: Program to check whether a number is even or not.

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even.")
else:
    print("Number is odd.")
```

OutPut:

```
enter the number?8
 Number is even
```

```
var1 = 100
if var1:
   print "1 - Got a true expression value"
   print var1
else:
   print "1 - Got a false expression value"
   print var1

var2 = 0
if var2:
   print "2 - Got a true expression value"
   print var2
else:
   print "2 - Got a false expression value"
   print var2
print "Good bye!"
```

The Nested *if...elif...else* Construct

```
if expression 1:
    # block of statements

elif expression 2:
    # block of statements

elif expression 3:
    # block of statements

else:
    # block of statements
```

Example:

```
var = 100
if var < 200:
   print "Expression value is less than 200"
   if var == 150:
      print "Which is 150"
   elif var == 100:
      print "Which is 100"
   elif var == 50:
      print "Which is 50"
elif var < 50:
   print "Expression value is less than 50"
else:
   print "Could not find true expression"

print "Good bye!"
```

Example:

```
number = int(input("Enter the number?"))
if number==15:
    print("number is equals to 15")
elif number==40:
    print("number is equal to 40");
elif number==100:
    print("number is equal to 100");
else:
    print("number is not equal to 15, 40 or 
100");
```

OutPut:

```
Enter the number?25
number is not equal to 10, 40 or 100
```

Example:

```
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:

```
Enter the marks? 70
You scored grade B + ...
```

Python – while Loop Statements

- The **while** loop is one of the looping constructs available in Python. The **while** loop continues until the expression becomes false. The expression has to be a logical expression and must return either a *true* or a *false* value

  The syntax of the while loop is:

```
while expression:
    statement(s)
```

Example 1:

```
count = 0
while (count < 3):
print 'The count is:', count
count = count + 1
print "Good bye!"
```

Output:

```
The count is:0
The count is:1
The count is:2
Good bye!
```

Example 2:

```
a = [1,3,5,7]
 while a:
    print(a.pop())
```

Output:

```
7
5
3
1
```

The Infinite Loops:

- You must use caution when using while loops because of the possibility that this condition never resolves to a false value. This results in a loop that never ends. Such a loop is called an infinite loop.
- An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

Following loop will continue till you enter CTRL+C :

Example:

```
while var == 1 :  # This constructs an infinite loop
   num = raw_input("Enter a number  :")
   print "You entered: ", num
print "Good bye!“
```

Output:

```
Enter a number :10
You entered: 10
Enter a number :14
You entered: 14
Enter a number :5
You entered: 5........
```

Single Statement Suites:

- Similar to the **if** statement syntax, if your **while** clause consists only of a single statement, it may be placed on the same line as the while header.
- If there are multiple statements in the block that makes up the loop body, they can be separated by semicolons (;).

- Here is the syntax of a one-line while clause:

```
while expression : statement
```

Example:

```
count = 0
while (count < 3): count += 1; print("Hello Python")
```

Output:

```
Hello Python
Hello Python
Hello Python
```

Else Statement with While Loop

- If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

Example:

```
i = 0
while i < 3:
      print i, " is  less than 3"
      i = i + 1
else:
      print i, " is not less than 3"
```

Output:

```
    0 is less than 3
    1 is less than 3
    2 is less than 3
    3 is not less than 3
```

Example: sum = 1+2+3+...+n

```
n = 10
sum = 0
i = 1
while i <= n:
    sum = sum + i
    i = i+1
print("The sum is", sum)
```

Output:

```
The sum is 55
```

Python – for Loop Statements

- The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list or a string.
- The syntax of the loop look is:

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

Example:

```
for letter in 'Python':      # First Example
   print 'Current Letter :', letter

fruits = ['banana', 'apple',  'mango']
for x in fruits:            # Second Example
   print 'Current fruit :', x
print "Good bye!"
```

for Loop with break Statement

- With the break statement we can stop the loop before it has looped through all the items:

Example:

```
fruits =["apple","banana“,”cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

Output:

```
apple
banana
```

Example:

```
fruits = ["apple","banana","cherry"]
for x in fruits:
  if x == "banana":
    break
 print(x)
```

Output:

```
apple
```

For Loop with continue Statement

- With the continue statement we can stop the current iteration of the loop, and continue with the next:

Example:

```
fruits=["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

Output:

```
apple
cherry
```

The range() Function

- To loop through a set of code a specified number of times, we can use the range() function,
- 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:

```
for x in range(3):
    print(x)
```

Output:

```
0
1
2
```

Example:

```
 for x in range(2, 5):
  print(x)
```

Output:

```
2
3
4
```

Example:

```
for x in range(2, 10, 3):
  print(x)
```

Output:

```
2
5
8
```

Else in For Loop

- The else keyword in a for loop specifies a block of code to be executed when the loop is finished:
- The else block will NOT be executed if the loop is stopped by a break statement.

Example:

```
for x in range(4):
  print(x)
else:
  print("Finally finished!")
```

OutPut:

```
0
1
2
3
Finally finished!
```

Example:

```
for x in range(4):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

OutPut:

```
0
1
2
```

Nested Loops

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

Example:

```
adj = ["red", “yellow"]
fruits = ["apple", "banana"]

for x in adj:
  for y in fruits:
    print(x, y)
```

Output:

```
red apple
red banana
yellow apple
yellow banana
```

The pass Statement

- for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.
- The **pass** statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

Example:

```
for x in 'python':
    pass
print('Last Letter :', x)
```

Output:

```
Last Letter :n
```

The pass Statement

Example:

```
for letter in 'India':
   if letter == 'd':
      pass
      print ('This is pass block')
   print ('Current Letter :', letter)
print ("Good bye!")
```

OutPut:

```
Current Letter : I
Current Letter : n
This is pass block
Current Letter : d
Current Letter : i
Current Letter : a
Good bye!
```

Python *break, continue and pass* Statements

The *break* Statement:

- The break statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.

Example:

```
for letter in ‘Hello':
if letter == ‘l':
      break
   print 'Current Letter :',
```

OutPut:

```
Current Letter : H
Current Letter : e
```

Example:

```
var = 10
while var > 0:
   print 'Current value :', var
   var = var -1
   if var == 7:
      break
print "Good bye!"
```

OutPut:

```
Current value : 10
Current value : 9
Current value : 8

Good bye!
```

The continue Statement:

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

Example:

```
for letter in 'Python':
   if letter == 'h':
      continue
   print ("Current Letter :", letter)
```

Ouput:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

Example:

```
var = 5
while var > 0:
   var = var -1
   if var == 3:
      continue
   print ("Current value :", var)
print ("Good bye!")
```

Ouput:

```
Current value : 4
Current value : 2
Current value : 1
Current value : 0
Good bye!
```

The *else* Statement Used with Loops

Python supports to have an **else** statement associated with a loop statements.

- If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.
- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

Example:

```
for num in range(10,20):
   for i in range(2,num):
      if num%i == 0:
         j=num/i
         print ("%d equals %d * %d" %(num,i,j))
         break
   else:
      print (num, "is a prime number")
```

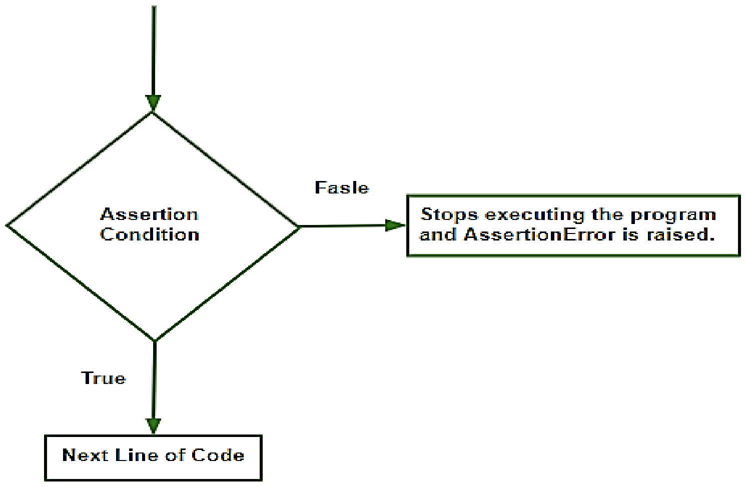
Output:

```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

Python Assert Statement

- Assertions are statements that assert or state a fact confidently in your program.
- **For example,** while writing a division function, you're confident the divisor shouldn't be zero, you assert divisor is not equal to zero.
- Assertions are simply boolean expressions that check if the conditions return true or not.
- If it is true, the program does nothing and moves to the next line of code. However, if it's false, the program stops and throws an error.
- It is also a debugging tool as it halts the program as soon as an error occurs and displays it.
- **Syntax for using Assert in Pyhton**:

```
assert <condition>
assert <condition>,<error message>
```



Assert Statement Example

Example:

```
x = 7
y = 4
print ("x / y value is : ")
assert y != 0, "Divide by 0 error"
print (x / y)
```

Output:

```
x / y value is :
1.75
```

Example:

```
x = 7
y = 0
print ("x / y value is : ")
assert y != 0, "Divide by 0 error"
print (x / y)
```

Output:

```
x / y value is :
Traceback (most recent call last):
  File "C:\Users\admin\anaconda3\a.py", line 4, in <module>
    assert y != 0, "Divide by 0 error"
AssertionError: Divide by 0 error
```

Example:

```
x = "hello"
#if condition returns False, AssertionError is raised:
assert x == "goodbye", "x should be 'hello'"
```

Output:

```
Traceback (most recent call last):
File "demo_ref_keyword_assert2.py", line 4, in <module>
assert x == "goodbye", "x should be 'hello'"
AssertionError: x should be 'hello'
```

Python Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- **User-define functions** : The user-defined functions are those define by the **user** to perform the specific task.
- **Built-in functions** : The built-in functions are those functions that are **pre-defined** in Python.
- **Creating a Function:**

  In Python a function is defined using the def keyword:

  Syntax:

```
def function_name(parameters):
    """docstring"""
    statement(s)
     return expression
```

Example:

```
def my_function():    #create function
  print("Hello ..Function ")
my_function()         #calling function
```

OutPut:

```
Hello ..Function
```

- **Function with Arguments:**
    - Information can be passed into functions as arguments.
    - Arguments are specified after the function name, inside the parentheses.
    - You can add as many arguments as you want, just separate them with a comma.

Example: Python function to find even odd no.

```
def evenOdd(x):                                    #create function with
argument
   if (x % 2 == 0):
            print("even")
   else:
            print("odd")
evenOdd(5)                                         #calling function
evenOdd(8)                                         #calling function
```

OutPut:

```
odd
even
```

Example:

```
def func (name):               #defining the function
   print("Hello ",name)
func("Python")                 #calling the function
```

OutPut:

```
Hello  Python
```

Example:

```
def fibonaci(n):
   a = 0
   b = 1
   for i in range(0, n):
      temp = a
      a = b
      b = temp + b
   return a

n1 = int(input("Enter the number"))

for c in range(0, n1):
   print(fibonaci(c))
```

OutPut:

```
Enter the number10
0
1
1
2
3
5
8
13
21
34
```

- **Number of Arguments**
  - By default, a function must be called with the correct number of arguments.
  - Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Example:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Vaishali", "Patel")
```

OutPut:

```
Vaishali Patel
```

- **Function with return statement:**
  - The return statement is used at the end of the function and returns the result of the function.
  - It terminates the function execution and transfers the result where the function is called.
  - The return statement cannot be used outside of the function.

Example:

```
def sum():
    a = 10
    b = 20
    c = a+b
    return c
print("The sum is:",sum())
```

OutPut:

```
The sum is: 30
```

- **Arbitrary Arguments, *args**
  - If you do not know how many arguments that will be passed into your function, also you don't know the exact number of arguments that you want to pass to a function, you can use the following syntax with *args:
  - This way the function will receive a tuple of arguments, and can access the items accordingly:

Example:

```
def plus(*args):
  total = 0
  for i in args:
    total += i
  return total
print("sum=",plus(1,3,5))
```

Output:

```
sum= 9
```

Example:

```
def plus(*args):
  total = 0
  for i in args:
    total += i
  return total
print("sum=",plus(1,3,5,7,8))
```

Output:

```
sum= 24
```

- **Default Arguments:**
  - Default arguments are those that take a default value if no argument value is passed during the function call.
  - You can assign this default value by with the assignment operator =.
  - **def func(a, b=5) is valid**
  - **def func(a=5, b) is not valid.**
  - just like in the following example:

Example:

```
def plus(a,b = 2):
     return a + b
print("ans=",plus(a=1))
print("ans=",plus(a=1,b=3))
```

OutPut:

```
ans= 3
ans= 4
```

Example:

```
def say(s, times = 1):
          print (s * times)
say('Hello')
say('India', 5)
```

OutPut:

```
Hello
IndiaIndiaIndiaIndiaIndia
```

- **Keyword Arguments:**
  - If we have some functions with many parameters and we want to specify only some parameters, then we can give values for such parameters by naming them. i.e., this is called keyword arguments.
  - We use the name instead of the position which we have been using all along.
  - This has two advantages:
    - Using the function is easier since we do not need to worry about the order of the arguments.
    - We can give values to only those parameters which we want, provided that the other parameters have default argument values.

Example:

```
def func(a, b=5, c=10):
    print("a is", a,"and b is", b,"and c is", c)
func(3,7)
func(25,c=24)
func(c=50,a=100)
```

Output:

```
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
 a is 100 and b is 5 and c is 50
```

- **Passing a List as an Argument:**
  - You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
  - E.g. if you send a List as an argument, it will still be a List when it reaches the function:

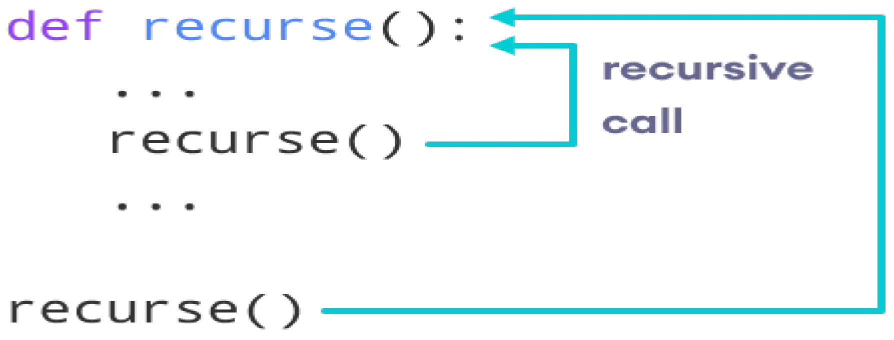
Example:

```
def my_function(colour):
  for x in colour:
    print(x)
mylist = ["Red", "Blue","Green","Pink"]
my_function(mylist)
```

Output:

```
Red
Blue
Green
Pink
```

- **Recursion:**
  - Python also accepts function recursion, which means a defined function can call itself.



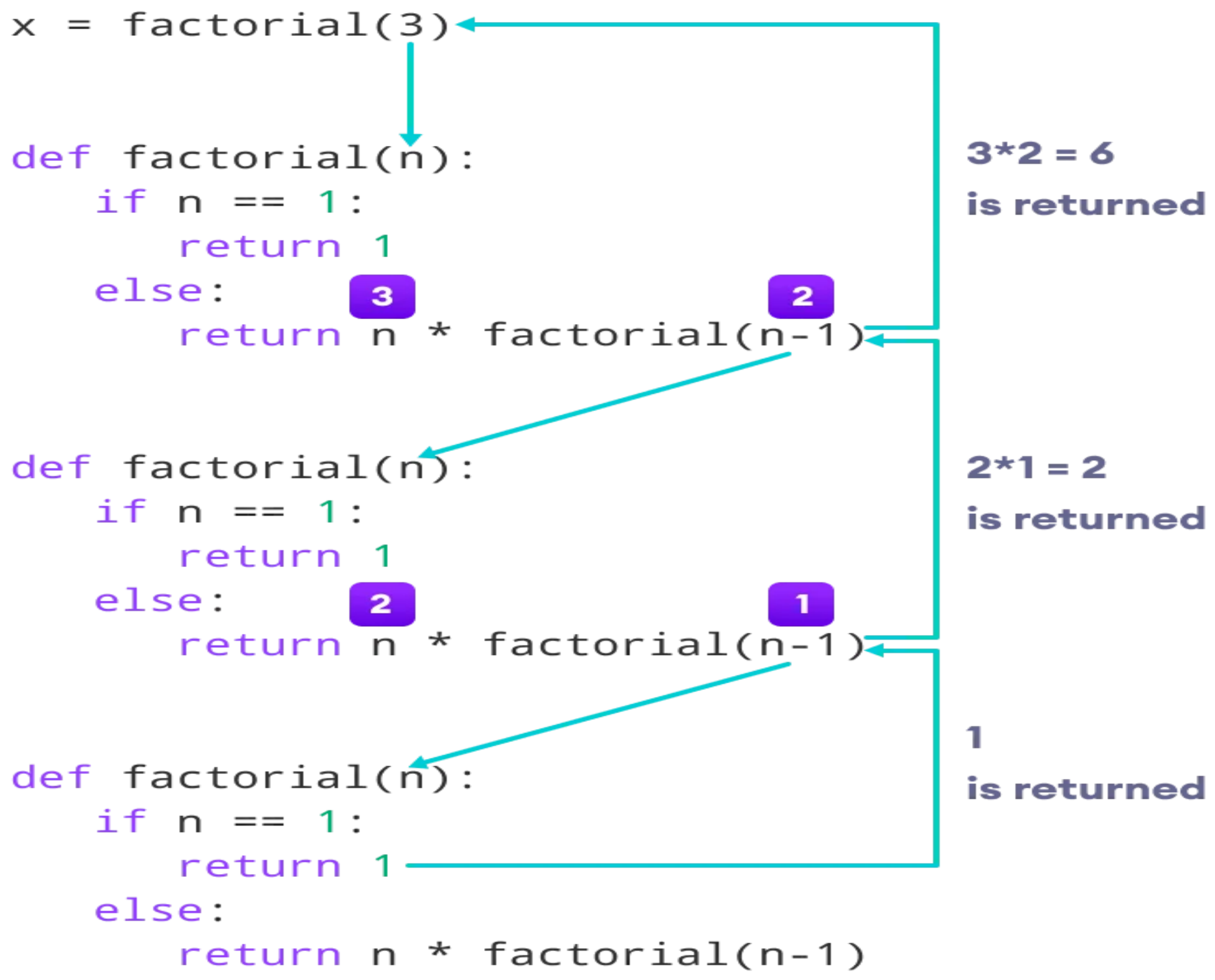
```
def recurse():
    ...
    recurse()
    ...

recurse()
```

Example:

```
def factorial(x):
  if x == 1:
        return 1
  else:
        return (x * factorial(x-1))
num = 5
print("The factorial of", num, "is", factorial(num))
```

Output:

```
The factorial of 5 is 120
```



```
x = factorial(3)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

- **Pass by Object Reference:**
- In Python, call by reference means passing the actual value as an argument in the function.
- All the functions are called by reference, i.e., all the changes made to the reference inside the function revert back to the original value referred by the reference.
- Python utilizes a system, which is known as “Call by Object Reference” or “Call by assignment”.
- In the event that you pass arguments like whole numbers, strings or tuples to a function, the passing is like call-by-value because you can not change the value of the immutable objects being passed to the function.
- Whereas passing mutable objects can be considered as call by reference because when their values are changed inside the function, then it will also be reflected outside the function.

Example: Passing Mutable Object (String)

```
def change_string (str):
    str = str + "Good Morning "
    print("printing the string inside function :",str)

string1 = "Hello India "
change_string(string1)

print("printing the string outside function :",string1)
```

Output:

```
printing the string inside function :
Hello India Good Morning
printing the string outside function :
Hello India
```

Example: Passing Immutable Object (List)

```
def change_list(list1):
    list1.append(5)
    list1.append(6)
    print("list inside function = ",list1)
list1 = [1,2,3,4]
change_list(list1)
print("list outside function = ",list1)
```

Output:

```
list inside function = [1, 2, 3, 4, 5, 6]
list outside function = [1, 2, 3, 4, 5, 6]
```

Python Anonymous/Lambda Function

- In Python, an anonymous function is a function that is defined without a name.
- While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.
- Hence, **anonymous functions** are also called **lambda functions**.

- **Syntax of Lambda Function in python**

  lambda [arg1 [,arg2,.....argn]]:expression

- Lambda functions can have any number of arguments but only one expression.
- The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

- Example:

```
double = lambda x: x * 2
print(double(4))
```

Output:

```
8
```

- Example:

```
sum = lambda arg1, arg2: arg1 + arg2;

print ("Value of sum : ", sum( 3, 4 ))
print ("Value of sum : ", sum( 5, 7 ))
```

Output:

```
Value of sum : 7
Value of sum : 12
```

Using Lambda Function

- Note the abbreviated syntax here:
- There are no parentheses around the argument list
- and the return keyword is missing (it is implied, since the entire function can only be one expression).
- Also, the function has no name .
- But it can be called through the variable it is assigned to.
- We can use a lambda function without even assigning it to a variable.
- It just goes to show that a lambda is just an in-line function.
- To generalize, a lambda function is a function that:
  - Takes any number of arguments and returns the value of a single expression.
  - lambda functions can not contain commands
  - and they can not contain more than one expression.
  - Don't try to squeeze too much into a lambda function; if needed something more complex, define a normal function instead and make it as long as wanted.

- **Lambda functions are used along with built-in functions like filter(), map() etc.**
- The filter() function in Python takes in a function and a list as arguments.
- The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.
- Here is an example use of filter() function to filter out only even numbers from a list.

Example:

```
# Program  to filter out only the even items from a list.
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(filter(lambda x: (x%2 == 0) , my_list))
print(new_list)
```

Output:

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

- The map() function in Python takes in a function and a list.
- The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.
- Here is an example use of map() function to double all the items in a list.

Example:

```
# Program to double each item in a list using map()
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(map(lambda x: x * 2 , my_list))
print(new_list)
```

Output:

```
[2, 10, 8, 12, 16, 22, 6, 24]
```

Decorators in Python

- Python has an interesting feature called decorators to add functionality to an existing code.
- This is also called metaprogramming because a part of the program tries to modify another part of the program at compile time.
- Firstly, every entity in Python is an object, including functions, almost everything, including functions can be assigned to a variable.
- For example, the simple function,
- Example:

```
def hello(name):
    return "Hello " + name
print("function call:",hello("Python"))
```

OutPut:

```
function call: Hello Python
```

- can be assigned to a new variable,

```
greet = hello
```

- which is also a function,

```
greet
```

- and more importantly, it is not attached to the original function hello(),

```
del hello
hello
```

<u>Example:</u>

```
def hello(name):
    return "Hello " + name
print("function call:",hello("Python"))
greet=hello
print("function call:",greet("Java"))
```

<u>OutPut:</u>

```
function call: Hello Python
function call: Hello Java
```

Example:

```
def hello(name):
    return "Hello " + name
print("function call:",hello("Python"))
greet=hello
print("function call:",greet("Java"))
del hello
print("function call:",hello("india"))
print("function call:",greet("Java"))
```

OutPut:

```
Traceback (most recent call last):
File "./prog.py", line 8, in <module>
NameError: name 'hello' is not defined
```

Example:

```
def hello(name):
    return "Hello " + name

print("function call:",hello("Python"))
greet=hello
print("function call:",greet("Java"))
del hello
#print("function call:",hello("india"))
print("function call:",greet("Java"))
```

OutPut:

```
function call: Hello Python
function call: Hello Java
function call: Hello Java
```

Python List

- A list in Python is used to store the sequence of various types of data. Python lists are mutable type its mean we can modify its element after it created.
- 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 [].

  Example:

  A list can be define as below

  ```
  L1 = [“Python", 111, “INDIA"]
  L2 = [1, 2, 3, 4, 5, 6]
  ```

- Characteristics of Lists
  - The lists are ordered.
  - The element of the list can access by index.
  - The lists are mutable types.
  - A list can store the number of various elements.

- Python provides the flexibility to use the negative indexing also. The negative indices are counted from the right.
- The last element (rightmost) of the list has the index -1; its adjacent left element is present at the index -2 and so on until the left-most elements are encountered.

Forward Indexing ⇨	0	1	2	3	4	
	'I'	'N'	'D'	'I'	'A'	
	-5	-4	-3	-2	-1	⇦ Backward Indexing

- Python also provides append() and insert() methods, which can be used to add values to the list.
- The list elements can also be deleted 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.

- Python List Operations:
  - The concatenation (+) and repetition (*) operators work in the same way as they were working with the strings.

- Consider a Lists
  l1 = [1, 2, 3, 4],
  l2 = [5, 6, 7, 8] to perform operation.

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	print ("l1*2=",l1*2) L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]
Concatenation	It concatenates the list mentioned on either side of the operator.	print ("l1+l2=",l1+l2) l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]
Membership	It returns true if a particular item exists in a particular list otherwise false.	print(2 in l1) prints True.
Iteration	The for loop is used to iterate over the list elements.	for i in l1: print(i) Output 1 2 3 4
Length	It is used to get the length of the list	print ("len(l1)=",len(l1)) len(l1) = 4

Python List Built-in functions

Function	Description	Example
cmp(list1, list2)	It compares the elements of both the lists.	This method is not used in the Python 3 and the above versions.

len(list)	It is used to calculate the length of the list.	L1=[10,22,45,89,76,63]
		print(len(L1))
		
		Output: 6

max(list)	It returns the maximum element of the list.	L1=[10,22,45,89,76,63]
		print(max(L1))
		
		Output:89

min(list)	It returns the minimum element of the list.	L1=[10,22,45,89,76,63]
		print(min(L1))
		
		Output:10

list(seq)	It converts any sequence to the list.	str="Python"
		s=list(str)
		print(type(s))
		
		Output:<class 'list'>

Methods to process List

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any inerrable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

- Adding elements to the list
- Removing elements from the list

Example: Add elements to the List

```
l =[]
n = int(input("Enter the number of elements in the list:"))
for i in range(0,n):
    l.append(input("Enter the item:"))
print("printing the list items..")
for i in l:
    print(i, end = "  ")
```

- OutPut:

```
Enter the number of elements in the list:5
Enter the item:3
Enter the item:4
Enter the item:5
Enter the item:6
Enter the item:7
printing the list items..
3  4  5  6  7
```

Example: Remove elements from the List

```
list = [1,2,3,4,5]
print("printing original list: ");
for i in list:
    print(i,end=" ")
list.remove(2)
print("\n printing the list after the removal of first element...")
for i in list:
    print(i,end=" ")
```

- OutPut:

```
printing original list:
1 2 3 4 5
printing the list after the removal of first element...
1 3 4 5
```

Example: Write the program to remove the duplicate element of the list.

```
list1 = [1,2,2,3,4,5,6,6,7,7,8,5]
list2 = []
for i in list1:
    if i not in list2:
        list2.append(i)
print(list2)
```

Output:

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

- Aliasing and Cloning lists:
  - Copy one list to another list with the use of "=" Operators.

Example:

```
list1=[1,2,3,4,5]
list2=list1
print(list2)
```

Output: [1, 2, 3, 4, 5]

- Another way to cloning the list

Example:

```
def Cloning(li1):
    li_copy = list(li1)
    return li_copy
li1 = [1, 2, 3, 4, 5]
li2 = Cloning(li1)
print("Original List:", li1)
print("After Cloning:", li2)
```

Output:

```
Original List: [1, 2, 3, 4, 5]
After Cloning: [1, 2, 3, 4, 5]
```

Nested Lists

- Python list can contain sub-list within itself too. That’s called a nested list.
- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.
- Syntax:

newlist = [*expression* for *item* in *iterable* if *condition* == True]

- The return value is a new list, leaving the old list unchanged.

Example:
sub = ["java", "python", "advancejava", "php", "dbms"]
newlist = [x for x in sub if "p" in x]
print(newlist)
	Example:
	sub = ["java", "python", "advancejava", "php", "dbms"]
	newlist = [x if x != "advancejava" else "c++" for x in sub]
	print(newlist)

OutPut: ['python', 'php']	OutPut: ['java', 'python', 'c++', 'php', 'dbms']

Python Tuples

- Tuples are used to store multiple items in a single variable.
- Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.
- A tuple is a collection which is ordered and unchangeable.
- Tuples are written with round brackets.
- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.
- The tuple has the fixed length.

- A tuple can contain different data types.

Example:

```
T1 = (100, "Python", 22.55)
T2 = ("India", "Canada", "USA")
T3 = 10,20,30,40,50
T4= ()
print(T1)
print(T2)
print(T3)
print(T4)
```

OutPut:

```
(100, 'Python', 22.55)
('India', 'Canada', 'USA')
(10, 20, 30, 40, 50)
()
```

Tuple operations

- Consider a Tuples
  T1 = (1, 2, 3, 4),
  T2 = (6, 7, 8, 9) to perform operation.

Operator	Description	Example
Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	T1*2 = (1, 2, 3, 4, 1, 2, 3, 4)
Concatenation	It concatenates the tuple mentioned on either side of the operator.	T1+T2 = (1, 2, 3, 4, 6, 7, 8, 9)
Membership	It returns true if a particular item exists in the tuple otherwise false	print (2 in T1) prints True.
Iteration	The for loop is used to iterate over the tuple elements.	for i in T1: print(i) Output: 1 2 3 4
Length	It is used to get the length of the tuple.	len(T1) = 4

Tuple functions

Function	Description
cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
len(tuple)	It calculates the length of the tuple.
max(tuple)	It returns the maximum element of the tuple
min(tuple)	It returns the minimum element of the tuple.
tuple(seq)	It converts the specified sequence to the tuple.

- count()
  - Returns the number of times a specified value occurs in a tuple

    Example:

    ```
    T1 = (1,2,3,4,5,4,3,2,2,5)
    x = T1.count(2)
    print(x)
    ```

    OutPut:

    ```
    3
    ```

- index()
  - Searches the tuple for a specified value and returns the position of where it was found

    Example:

    ```
    T1 = (1,2,3,4,5,4,3,2,2,5)
    x = T1.index(4)
    print(x)
    ```

    OutPut:

    ```
    3
    ```