Types of Parameters in Python:

1. Positional Parameters:

- In the case of positional arguments, number of arguments must be same.
- In the case of positional arguments, order of the arguments is important.

2. Keyword (i.e., Parameter name) Parameters:

- In the case of keyword arguments, order of the arguments is not important.
- In the case of keyword arguments, number of arguments must be same.

3. Default Parameters:

- You can define default value for the arguments.
- If you are not passing any argument, then default values by default will be considered.
- After default arguments you should not take normal arguments. (i.e., Default arguments you need to take at last)

4. Variable length Parameters:

- Sometimes we can pass variable number of arguments to our function,such type of arguments are called variable length arguments.
- We can declare a variable length argument with '*' symbol as follows

```
def f1(*n):
```

- We can call this function by passing any number of arguments including zero number. Internally all these values represented in the form of tuple.

Example Programs:

Q1. Write a function to take name of the student as input and print wish message by name.

In [1]:

```
def wish(name):
    print("Hello",name," Good Morning")
wish("Karthi")
wish("Sahasra")
```

```
Hello Karthi  Good Morning
Hello Sahasra  Good Morning
```

Q2. Write a function to take number as input and print its square value.

In [6]:

```
def even_odd(num):
    if num%2==0:
        print(num,"is Even Number")
    else:
        print(num,"is Odd Number")
even_odd(10)
even_odd(15)
```

```
10 is Even Number
15 is Odd Number
```

Q6. Write a function to find factorial of given number.

In [7]:

```
def fact(num):
    result=1
    while num>=1:
        result=result*num
        num=num-1
    return result
for i in range(1,5):
    print("The Factorial of",i,"is :",fact(i))
```

```
The Factorial of 1 is : 1
The Factorial of 2 is : 2
The Factorial of 3 is : 6
The Factorial of 4 is : 24
```

Example programs to demonstrate Positional Arguments.

In [13]:

```
def calc(a,b):       # Here, 'a' & 'b' are called positional arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
a,b,c,d = calc(100,50)                # Positional arguments
print(a,b,c,d)
```

```
150 50 5000 2.0
```

In [14]:

```
def calc(a,b): # Positional Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(100,50)
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Example programs to demonstrate Keyword Arguments.

In [15]:

```
def calc(a,b):        # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(a = 100, b = 50) # keyword arguments Arguments
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

In [16]:

```
def calc(a,b):                  # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(b = 50, a = 100)     # keyword arguments Arguments
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

In [17]:

```
def calc(a,b):           # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(100, b = 50)    # It is perfectly valid
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

In [18]:

```
def calc(a,b): # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(b = 50, 100)
# It is invalid, because positional argument should follow keyword arguments.
# first keyword argument then possitional argument is not allowed.
for x in t:
    print(x)
```

```
  File "<ipython-input-18-70a44f0ef84d>", line 7
    t = calc(b = 50, 100)
                    ^
SyntaxError: positional argument follows keyword argument
```

In [19]:

```
def calc(a,b): # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(50, a = 50) # It is also invalid
for x in t:
    print(x)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-19-2669f311ec68> in <module>
      5     div = a / b
      6     return sum,sub,mul,div
----> 7 t = calc(50, a = 50) # It is also invalid
      8 for x in t:
      9     print(x)

TypeError: calc() got multiple values for argument 'a'
```

Another Example:

In [20]:

```
def wish(name,msg):
    print('Hello',name,msg)
wish(name = 'Karthi',msg = 'Good Morning')
#order is not important, but no.of arguments is important.
wish(msg = 'Good Morning',name = 'Karthi')
```

```
Hello Karthi Good Morning
Hello Karthi Good Morning
```

Example programs to demonstrate Default Arguments.

In [22]:

```
def wish(name ='Guest',msg):
    # After default argument, we should not take non-default argument
    print('Hello',name,msg)
```

```
  File "<ipython-input-22-a83b9dddb70c>", line 1
    def wish(name ='Guest',msg):
            ^
SyntaxError: non-default argument follows default argument
```

In [23]:

```
def wish(msg,name ='Guest'):
    print(msg,name)
wish('Hello','Karthi')
```

```
Hello Karthi
```

In [24]:

```
def wish(msg,name ='Guest'):
    print(msg,name)
wish('Hello')
```

```
Hello Guest
```

In [25]:

```
def wish(msg,name ='Guest'):
    print(msg,name)
wish()
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-25-8435118960b8> in <module>
      1 def wish(msg,name ='Guest'):
      2     print(msg,name)
----> 3 wish()

TypeError: wish() missing 1 required positional argument: 'msg'
```

Note:

- You can give any number of default arguments.

In [26]:

```
def wish(name ='Guest',msg='Good Morning'):
    print('Hello',name,msg)
wish()
```

```
Hello Guest Good Morning
```

In [27]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)
wish(99,48,'Karthi')            # Valid
```

```
Student Name: Karthi
Student Age: 48
Student Marks: 99
Message: Good Morning
```

In [28]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)
wish(age=48,marks = 100)
```

```
Student Name: Guest
Student Age: 48
Student Marks: 100
Message: Good Morning
```

In [29]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)
wish(100,age=46,msg='Bad Morning',name='Karthi')    # valid
```

```
Student Name: Karthi
Student Age: 46
Student Marks: 100
Message: Bad Morning
```

In [30]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)
wish(marks=100,46,msg='Bad Morning',name = 'Karthi') # invalid
```

```
  File "<ipython-input-30-694cd2b4bf85>", line 6
    wish(marks=100,46,msg='Bad Morning',name = 'Karthi') # invalid
                  ^
SyntaxError: positional argument follows keyword argument
```

In [31]:

```
def wish(marks,age,name = 'Guest', msg = 'Good Morning'):
    print('Student Name:',name)
    print('Student Age:',age)
    print('Student Marks:',marks)
    print('Message:',msg)
wish(46,marks=100,msg='Bad Morning',name = 'Karthi')   # invalid
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-31-eb2f97227036> in <module>
      4     print('Student Marks:',marks)
      5     print('Message:',msg)
----> 6 wish(46,marks=100,msg='Bad Morning',name = 'Karthi')   # invalid

TypeError: wish() got multiple values for argument 'marks'
```

Example programs to demonstrate Variable length Arguments.

In [32]:

```
def sum(a,b):
    print(a+b)
sum(10,20)
```

```
30
```

Now it is working correctly. After some time my requiremnet is as follows:

sum(10,20,30)

In [33]:

```
def sum(a,b):
    print(a+b) # This sum() function we can't use for the new requirement.
sum(10,20,30)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-33-5c96eb93c4b3> in <module>
      1 def sum(a,b):
      2     print(a+b) # This sum() function we can't use for the new requ
irement.
----> 3 sum(10,20,30)

TypeError: sum() takes 2 positional arguments but 3 were given
```

In [34]:

```
def sum(a,b,c):
    print(a+b+c)    # we have to go for another sum() function
sum(10,20,30)
```

```
60
```

Now it is working correctly. After some time my requiremnet is as follows:

sum(10,20,30,40)

In [35]:

```
def sum(a,b,c):
    print(a+b+c)
sum(10,20,30,40)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-35-b271960d93ea> in <module>
      1 def sum(a,b,c):
      2     print(a+b+c)
----> 3 sum(10,20,30,40)

TypeError: sum() takes 3 positional arguments but 4 were given
```

Once again the same problem. we should go for another **sum()** function.

In [36]:

```
def sum(a,b,c,d):
    print(a+b+c+d)    # we have to go for another sum() function
sum(10,20,30,40)
```

```
100
```

- If you change the number of arguments, then automatically for every change, compusorily we need to go for new function unnecessarily. Because of this length of the code is going to increase.
- To overcome this problem we should go for **variable length arguments.**

In [37]:

```
def sum(*n):
# Here, 'n' is a variable length argument.
    result =0
    for x in n:
        result = result + x
    print(result)
sum(10,20,30,40)
```

```
100
```

In [38]:

```
def sum(*n):
    result =0
    for x in n:
        result = result + x
    print(result)
sum(10,20,30)
```

```
60
```

In [39]:

```python
def sum(*n):
    result =0
    for x in n:
        result = result + x
    print(result)
sum(10,20)
```

```
30
```

In [40]:

```python
def sum(*n):
    result =0
    for x in n:
        result = result + x
    print(result)
sum(10)
```

```
10
```

In [41]:

```python
def sum(*n):
    result =0
    for x in n:
        result = result + x
    print(result)
sum()
```

```
0
```

In [42]:

```python
def sum(*n):
    result =0
    for x in n:
        result = result + x
    print('The Sum is : ', result)
sum(10,20,30,40)
sum(10,20,30)
sum(10,20)
sum(10)
sum()
```

```
The Sum is :  100
The Sum is :  60
The Sum is :  30
The Sum is :  10
The Sum is :  0
```

Note: Same function is used for variable number of arguments.

Key Point 1:

- We can mix variable length arguments with positional arguments.
- You can take positional arguments and variable length arguments simultaneously.

In [43]:

```
def sum(name,*n):
    result =0
    for x in n:
        result = result + x
    print("The Sum by", name, ": ", result)
sum('Robin',10,20,30,40)
sum('Rahul',10,20,30)
sum('Sachin',10,20)
sum('Sourav',10)
sum('Karthi')
```

```
The Sum by Robin :  100
The Sum by Rahul :  60
The Sum by Sachin :  30
The Sum by Sourav :  10
The Sum by Karthi :  0
```

Note:

Rule: After variable length argumenst,if we are taking any other arguments then we should provide values as keyword arguments.

In [44]:

```
def sum(*n,name):
    result =0
    for x in n:
        result = result + x
    print("The Sum by", name, ": ", result)
sum('Robin',10,20,30,40)
sum('Rahul',10,20,30)
sum('Sachin',10,20)
sum('Sourav',10)
sum('Karthi')
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-44-9cbb9821a5b4> in <module>
      4         result = result + x
      5     print("The Sum by", name, ": ", result)
----> 6 sum('Robin',10,20,30,40)
      7 sum('Rahul',10,20,30)
      8 sum('Sachin',10,20)

TypeError: sum() missing 1 required keyword-only argument: 'name'
```

```
In [45]:
```

```python
def sum(*n,name):
    result =0
    for x in n:
        result = result + x
    print("The Sum by", name, ": ", result)
sum(name = 'Robin',10,20,30,40)
sum(name = 'Rahul',10,20,30)
sum(name = 'Sachin',10,20)
sum(name = 'Sourav',10)
sum(name ='Karthi')
```

```
  File "<ipython-input-45-96d7fd117357>", line 6
    sum(name = 'Robin',10,20,30,40)
                      ^
SyntaxError: positional argument follows keyword argument
```

```
In [46]:
```

```python
def sum(*n,name):
    result =0
    for x in n:
        result = result + x
    print("The Sum by", name, ": ", result)
sum(10,20,30,40,name = 'Robin')
sum(10,20,30,name = 'Rahul')
sum(10,20,name = 'Sachin')
sum(10,name = 'Sourav')
sum(name ='Karthi')
```

```
The Sum by Robin :  100
The Sum by Rahul :  60
The Sum by Sachin :  30
The Sum by Sourav :  10
The Sum by Karthi :  0
```

Another Example:

```
In [47]:
```

```python
def f1(n1,*s):
    print(n1)
    for s1 in s:
        print(s1)
f1(10)
f1(10,20,30,40)
f1(10,"A",30,"B")
```

```
10
10
20
30
40
10
A
30
B
```

Conclusions:

- After variable length arguments, if you are taking any other argument, then we have to provide values as key word arguments only.
- If you pass first normal argument and then variable arguments, then there is no rule to follow. It works correctly.

Key Point 2:

Keyword variable length arguments:

- Now, Suppose if we want to pass any number of keyword arguments to a function, compulsorily we have to identify the difference with the above case (i.e., Passing of any number of positional arguments).
- We can declare key word variable length arguments also. For this we have to use **.
- We can call this function by passing any number of keyword arguments. Internally these keyword arguments will be stored inside a dictionary.

In [48]:

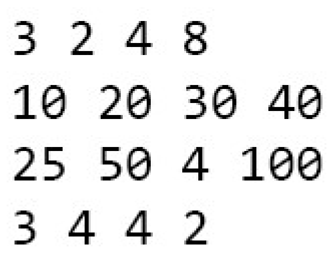
```
def display(**kwargs):
    for k,v in kwargs.items():
        print(k,"=",v)
display(n1=10,n2=20,n3=30)
display(rno=100,name="Karthi",marks=70,subject="Python")
```

```
n1 = 10
n2 = 20
n3 = 30
rno = 100
name = Karthi
marks = 70
subject = Python
```

Case Study:

```
def f(arg1,arg2,arg3=4,arg4=8):
    print(arg1,arg2,arg3,arg4)

f(3,2)       #3 2 4 8

f(10,20,30,40)    # 10,20,30,40

f(25,50,arg4=100)    # 25  50 4 100

f(arg4=2,arg1=3,arg2=4)    # 3 4 4 2

#f()      # TypeError: f() missing 2 required positional arguments: 'arg1' and 'arg2'

#f(arg3=10,arg4=20,30,40)    SyntaxError: positional argument follows keyword argument

#f(4,5,arg2=6)  #TypeError: f() got multiple values for argument 'arg2'

#f(4,5,arg3=5,arg5=6)  #TypeError: f() got an unexpected keyword argument 'arg5'
```

Output:



```
3 2 4 8
10 20 30 40
25 50 4 100
3 4 4 2
```

b) Write a Python program to return multiple values at a time using a return statement.

- In other languages like C,C++ and Java, function can return atmost one value. But in Python, a function can return any number of values.

Eg: Python program to return multiple values at a time using a return statement.

In [8]:

```
def calc(a,b):       # Here, 'a' & 'b' are called positional arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
a,b,c,d = calc(100,50)      # Positional arguments
print(a,b,c,d)
```

```
150 50 5000 2.0
```

Alternate Way:

In [10]:

```
def calc(a,b): # Positional Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(100,50)
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

In [11]:

```
def calc(a,b):            # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(a = 100, b = 50) # keyword arguments Arguments
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

In [12]:

```
def calc(a,b): # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(b = 50, a = 100) # keyword arguments Arguments
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

c) Write a Python program to demonstrate Local and Global variables.

Python supports 2 types of variables.

1. Global Variables
2. Local Variables

1. Global Variables:

- The variables which are declared outside of function are called global variables.
- These variables can be accessed in all functions of that module.

Consider the following example,

In [50]:

```
a = 10          # Global Variables
def f1():
    a = 20      # Local variable to the function 'f1'
    print(a)    # 20
def f2():
    print(a)    # 10
f1()
f2()
```

```
20
10
```

Suppose our requirement is, we don't want local variable. Can you please refer the local variable as the global variable only. How you can do that?

- For that, one special keyword is used, called as **global.**

global keyword:

We can use global keyword for the following 2 purposes:

1. To declare global variables explicitly inside function.
2. To make global variable available to the function so that we can perform required modifications.

In [51]:

```
a=10
def f1():
    a=777
    print(a)
def f2():
    print(a)
f1()
f2()
```

```
777
10
```

In [52]:

```
a=10
def f1():
    global a
# To bring global variable to the function for required modification
    a=777
# we are changing the value of the local variable
    print(a)
def f2():
    print(a)
f1()
f2()
```

```
777
777
```

In [55]:

```
def f1():
    xy = 10         # local variable of 'f1()'
    print(xy)
def f3():
    print(xy) # local variable of 'f1()' can not accessed by function 'f2()'
f1()
f3()
```

```
10

---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-55-b39491c6c549> in <module>
      5     print(xy) # local variable of 'f1()' can not accessed by funct
ion 'f2()'
      6 f1()
----> 7 f3()

<ipython-input-55-b39491c6c549> in f3()
      3     print(xy)
      4 def f3():
----> 5     print(xy) # local variable of 'f1()' can not accessed by funct
ion 'f2()'
      6 f1()
      7 f3()

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

Here, if you make 'xy' of f1() as a global variable, problem will be solved. How can you make 'xy' as global variable?

In [56]:

```
def f1():
    global xy
    xy = 10         # local variable of 'f1()'
    print(xy)
def f3():
    print(xy) # local variable of 'f1()' can not accessed by function 'f2()'
f1()
f3()
```

```
10
10
```

In [57]:

```
def f1():
    global xy = 10          # This syntax is invalid in Python
    print(xy)
def f3():
    print(xy)
f1()
f3()
```

```
  File "<ipython-input-57-8cc65cca44b9>", line 2
    global xy = 10          # This syntax is invalid in Python
              ^
SyntaxError: invalid syntax
```

Another Example:

In [58]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999             # global variable 'a' is overrides the old value.
    print('f2 :',a)
f1()
f2()
```

```
f1 : 888
f2 : 999
```

In [59]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
def f3():
    print('f3 :',a)
f1()
f2()
f3()
```

```
f1 : 888
f2 : 999
f3 : 999
```

In [60]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
def f3():
    print('f3 :',a)
f3()
f1()
f2()
```

```
f3 : 999
f1 : 888
f2 : 999
```

In [61]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
def f3():
    print('f3 :',a)
f3()
f2()
f1()
```

```
f3 : 999
f2 : 999
f1 : 888
```

In [62]:

```
def f1():
    global a
    a = 888
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
def f3():
    a = 1000
    print('f3 :',a)
f3()
f2()
f1()
```

```
f3 : 1000
f2 : 999
f1 : 888
```

Another Example:

In [63]:

```
def f1():
    global a
    a = 888         # global variable 'a' is overrides the old value.
    print('f1 :',a)
def f2():
    global a
    a=999
    print('f2 :',a)
f2()
f1()
```

```
f2 : 999
f1 : 888
```

Note:

- If global variable and local variable having the same name, then we can access global variable inside a function using **globals()** function.

In [64]:

```
a=10          #global variable
def f1():
    a=777     #local variable
    print(a)

f1()
```

```
777
```

In [65]:

```
a=10        #global variable
def f1():
    a=777   #local variable
    print(a)
    print(globals()['a'])

f1()
```

```
777
10
```

Another Example:

In [66]:

```
def f1():
    a = 10
    # SyntaxError: name 'a' is assigned to before global declaration
    global a
    a = 50
    print(a)
f1()
```

```
  File "<ipython-input-66-0a90027f8ff2>", line 4
    global a
    ^
SyntaxError: name 'a' is assigned to before global declaration
```

In [67]:

```
def f1():
    global a
    a = 10
    a = 50
    print(a)
f1()
```

```
50
```

d) Demonstrate lambda functions in Python with suitable example programs.

- Sometimes we can declare a function without any name,such type of nameless functions are called **anonymous functions** or **lambda functions.**
- The main purpose of anonymous function is just for instant use(i.e., for one time usage).

Normal Function:

- We can define by using **def** keyword.

```
def squareIt(n):
    return n*n
```

lambda Function:

- We can define by using lambda keyword

```
lambda n:n*n
```

Syntax of lambda Function:

```
lambda argument_list : expression
```

Note:

- By using Lambda Functions we can write very concise code so that readability of the program will be improved.

Q1. Write a program to create a lambda function to find square of given number.

In [1]:

```
s=lambda n:n*n
print("The Square of 4 is :",s(4))
print("The Square of 5 is :",s(5))
```

```
The Square of 4 is : 16
The Square of 5 is : 25
```

Q2. Write a program to create a Lambda function to find sum of 2 given numbers.

In [2]:

```
s=lambda a,b:a+b
print("The Sum of 10,20 is:",s(10,20))
print("The Sum of 100,200 is:",s(100,200))
```

```
The Sum of 10,20 is: 30
The Sum of 100,200 is: 300
```

Q3.Write a program to create a Lambda Function to find biggest of given values.

In [3]:

```
s=lambda a,b:a if a>b else b
print("The Biggest of 10,20 is:",s(10,20))
print("The Biggest of 100,200 is:",s(100,200))
```

```
The Biggest of 10,20 is: 20
The Biggest of 100,200 is: 200
```

Note:

- Lambda Function internally returns expression value and we are not required to write return statement explicitly.

- Sometimes we can pass a function as argument to another function. In such cases lambda functions are best choice.
- We can use lambda functions very commonly with **filter(),map() and reduce()** functions,because these functions expect function as argument.

1.filter() function:

- We can use filter() function to filter values from the given sequence based on some condition.
- For example, we have 20 numbers and if we want to retrieve only even numbers from them.

Syntax:

```
filter(function,sequence)
```

Where,

- function argument is responsible to perform conditional check.
- sequence can be list or tuple or string.

Q1. Program to filter only even numbers from the list by using filter() function.

Without lambda Function:

In [4]:

```
def isEven(x):
    if x%2==0:
        return True
    else:
        return False
l=[0,5,10,15,20,25,30]
l1=list(filter(isEven,l))
print(l1)
```

```
[0, 10, 20, 30]
```

With lambda Function:

In [5]:

```
l=[0,5,10,15,20,25,30]
l1=list(filter(lambda x:x%2==0,l))
print(l1)                          #[0,10,20,30]
l2=list(filter(lambda x:x%2!=0,l))
print(l2)                          #[5,15,25]
```

```
[0, 10, 20, 30]
[5, 15, 25]
```

2.map() function:

- For every element present in the given sequence,apply some functionality and generate new element with the required modification. For this requirement we should go for map() function.

Syntax:

```
map(function,sequence)
```

- The function can be applied on each element of sequence and generates new sequence.

Q1: For every element present in the list, perform double and generate new list of doubles.

Without lambda:

```
In [6]:
```

```python
l=[1,2,3,4,5]
def doubleIt(x):
    return 2*x
l1=list(map(doubleIt,l))
print(l1)
```

```
[2, 4, 6, 8, 10]
```

With lambda:

```
In [7]:
```

```python
l=[1,2,3,4,5]
l1=list(map(lambda x:2*x,l))
print(l1)
```

```
[2, 4, 6, 8, 10]
```

Q2: Find square of given numbers using map() function.

```
In [8]:
```

```python
l=[1,2,3,4,5]
l1=list(map(lambda x:x*x,l))
print(l1)
```

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

We can apply map() function on multiple lists also.But make sure all list should have same length.

Syntax:

```
map(lambda x,y:x*y,l1,l2))
```

x is from l1 and y is from l2

```
In [9]:
```

```python
l1=[1,2,3,4]
l2=[2,3,4,5]
l3=list(map(lambda x,y:x*y,l1,l2))
print(l3)
```

```
[2, 6, 12, 20]
```

In [10]:

```
l1=[1,2,3,4,5,6,7] # The extra elements will be ignored
l2=[2,3,4,5]
l3=list(map(lambda x,y:x*y,l1,l2))
print(l3)
```

```
[2, 6, 12, 20]
```

3.reduce() function:

- reduce() function reduces sequence of elements into a single element by applying the specified function.

Syntax:

```
reduce(function,sequence)
```

Note:

- reduce() function present in **functools module** and hence we should write import statement.

Eg 1:

In [11]:

```
from functools import *
l=[10,20,30,40,50]
result=reduce(lambda x,y:x+y,l)
print(result) # 150
```

```
150
```

Eg 2:

In [12]:

```
from functools import *
l=sum([10,20,30,40,50])
# result=reduce(lambda x,y:x*y,l)
print(l) #150
```

```
150
```

Eg 3:

In [13]:

```
from functools import *
l=[10,20,30,40,50]
result=reduce(lambda x,y:x*y,l)
print(result) #12000000
```

```
12000000
```

Eg 4:

```
In [14]:
```

```python
from functools import *
result=reduce(lambda x,y:x+y,range(1,101))
print(result) #5050
```

```
5050
```

Good Luck