Function in Python

In [1]:
```
def greet():
    print("hello")
    print("good morning ")
greet()# when we run the code we havent got any output
```

```
hello
good morning
```

In [2]:
```
def greet():
    print('hello')
    print('good morning')
greet() #if you need call multiple times
```

```
hello
good morning
```

In [3]:
```
def greet():
    print('hello')
    print('good morning')
greet() #if you need call multiple times

def greet():
    print('hello')
    print('good morning')
greet() #if you need call multiple times
```

```
hello
good morning
hello
good morning
```

In [4]:
```
def greet():
    print('hello')
    print('good noon')
greet()

greet()
```

```
hello
good noon
hello
good noon
```

In [5]:
```
def add(x,y):
    c=x+y
    print("add=",c)
add(12,23)
```

```
add= 35
```

In [6]:
```
def greet():
    print('hello')
    print('good noon')
greet()

def add(x,y):
    c=x+y
    print(c)
add(5,4)
# you can create mutliple function and call them as many time as you want
```

```
hello
good noon
9
```

In [7]:
```
def greet():
    print('hello')
    print('good noon')

def add(x,y):
    c=x+y
    print(c)

greet()
add(15,4)
# you can create mutliple function and call them as many time as you want
```

```
hello
good noon
19
```

In [12]:
```
def greet():
    print("hello")
    print("good noon")

def add(x,y):
    c=x+y
    return c

greet()
result=add(4,7)
print(result)
```

```
hello
good noon
11
```

as a function we have 2 choice

1- whenever we call the function . function is do the task for you greet() & add()

2- we have another type of function it will return you the value.

In [13]:
```
def add_sub(x,y): # what if i want to return 2 values add_sub & i want to return 2 values & function can ac
    c= x+y
    d= x-y
    return c, d

result = add_sub(4,5)
print(result)
#print(type(result))
```

```
(9, -1)
```

In [15]:
```
def add_sub(x,y):
    c= x+y
    d= x-y
    return c, d

result = add_sub(4,5)
print(result)
print(type(result))
```

```
(9, -1)
<class 'tuple'>
```

In [16]:
```
def add_sub(x,y):
    c= x+y
    d= x-y
    return c, d

result1,result2= add_sub(5,4)
print(result1,result2)
print(type(result1))
print(type(result2))
```

```
9 1
<class 'int'>
<class 'int'>
```

In [17]:
```
a,b = 6
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[17], line 1
----> 1 a,b = 6

TypeError: cannot unpack non-iterable int object
```

- function are always reuseable
- in one code you can write multiple function as well

Function Arguments

- FUNCTION ARGUMENT.
- How to pass parameter to a function & what happend to the variable when you pass to a function & if you modify the value then what happen.
- every code check with debug.

In [19]:
```
def update():
    x=8
    print(x)
update()
```

```
8
```

In [20]:
```
def update(): #update function take the value from the user
    x = 8
    print(x)
update(8)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 4
      2     x = 8
      3     print(x)
----> 4 update(8)

TypeError: update() takes 0 positional arguments but 1 was given
```

In [21]:
```
def update(x): #update function take the value from the user
    x = 8
    print(x)
update(8)
```

```
8
```

In [22]:
```
def update(x): # user want to update the value from 8 to 10
    x = 8
    print(x)
update(10)
```

```
8
```

In [23]:
```
8
def update(x):
    x = 8
    print(x)

a = 10
update(a)
```

```
8
```

In [24]:
```
def update(x):
    x = 8
    print(x)

a = 5
update(a)
print(a) # this print will update 8 to 10
```

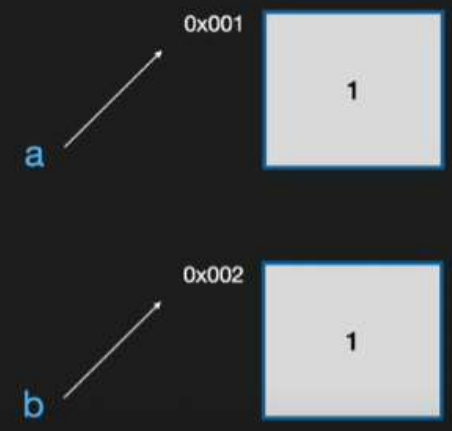
```
8
5
```

function we have 2 thing

- we have concept called as pass by value & pass by reference if you look at below code
- in other programming language

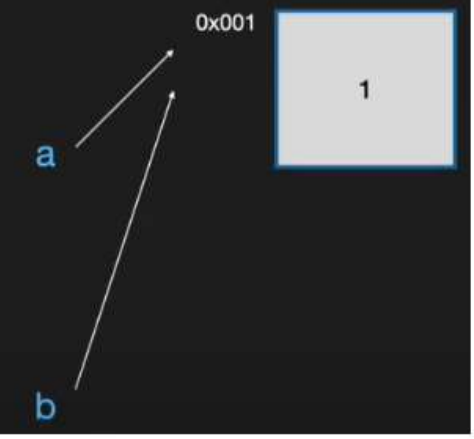
Pass by Value

```
my_function(b):
  #do nothing

a = 1
my_function(a)
```

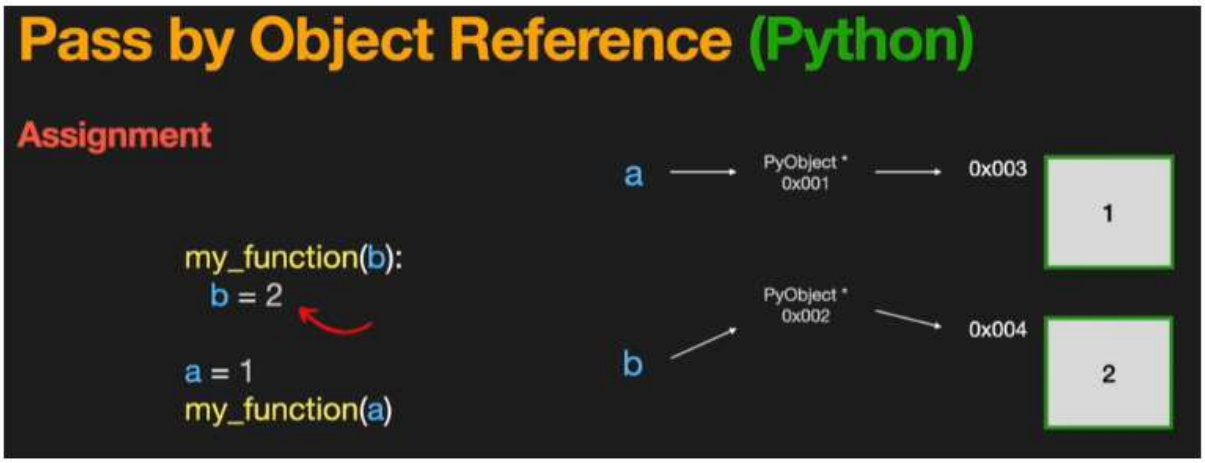


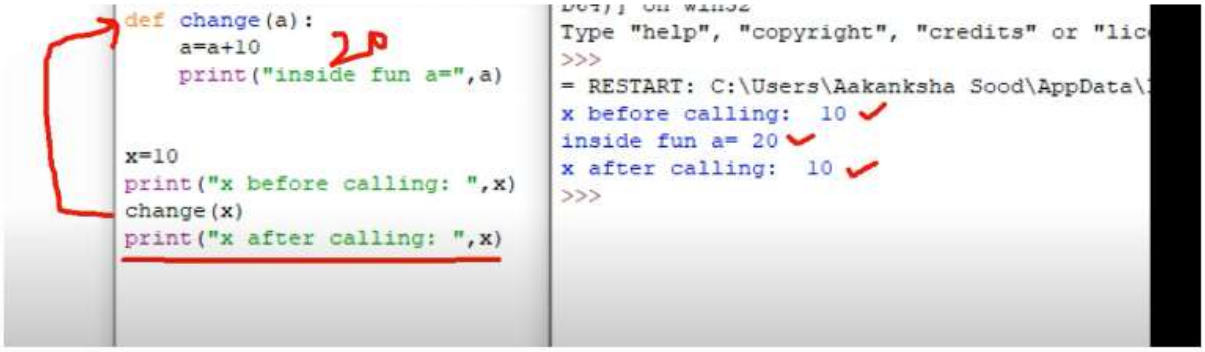
Pass by Reference

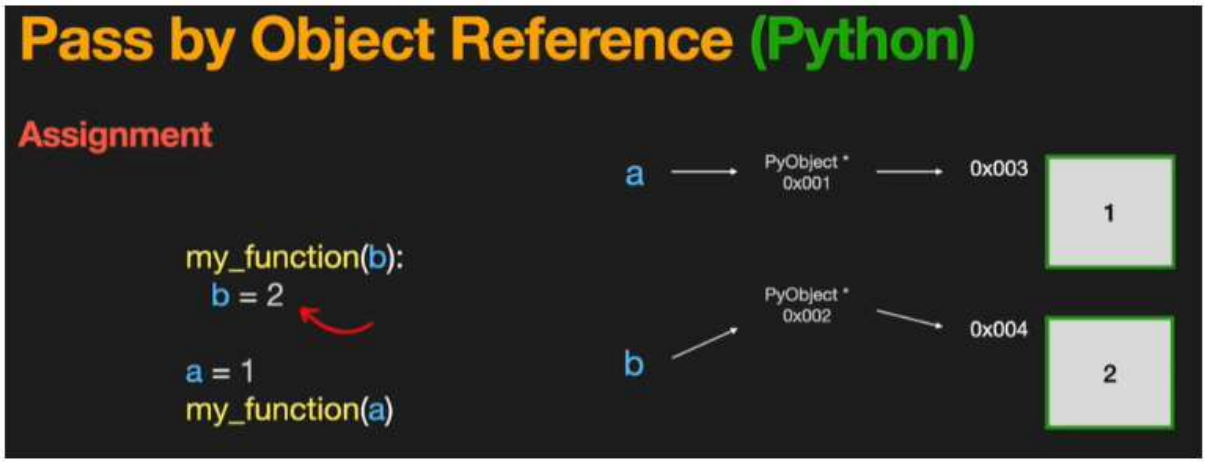
```
my_function(b):
  #do nothing

a = 1
my_function(a)
```

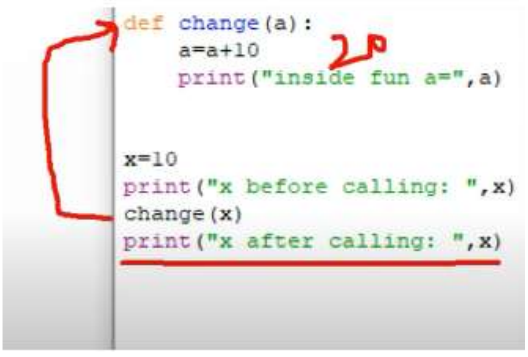


in python no concept called pass by value or pass by function









```
x before calling:  10
inside fun a= 20
x after calling:  10
>>>
```

- if you understand the code well there is no change is x value
- declaration of x is main function
- this is for other programing language & even thoug if i change x value to a still it display same result

In [25]:
```
def change(a):
    a = a+ 10
    print('inside the fun a =',a)

x = 10
print('x before calling:', x)
change(x)
print('x after calling:', x)
```

```
x before calling: 10
inside the fun a = 20
x after calling: 10
```

- even though i changed value x to a stil we got the same result
- THIS CONCEPT CALLED AS (PASS BY VALUE) FOR OTHER PROGRAMING

In [26]:
```
def change(a):
    a = a+ 10
    print('inside the fun a =',a)

a = 10
print('a before calling:', a)
change(a)
print('a after calling:', a)
```

```
a before calling: 10
inside the fun a = 20
a after calling: 10
```

- this is concept of pass by values & in this example both a refered to same address but after change the values only address changed.
- The main reason being in python int & strings are immutable

```
def change(a):
    print("This is original a ",id(a))
    a=a+10
    print("This is new a ",id(a))
    print("inside fun a=",a)


a=10
print("a before calling: ",a)
print("This is main a ",id(a))
change(a)
print("a after calling: ",a)
```

Python

```
a before calling:  10
This is main a  2155972815440
This is original a  2155972815440
This is new a  2155972815760
inside fun a= 20
a after calling:  10
>>>
```

In [30]:
```
def change(a):
    print('This is original a',id(a))
    a = a+ 10
    print('This is the new a =',id(a))
    print('inside the fun a =',a)

a = 10
print('a before calling:', a)
print('This is main a:',id(a))
change(a)
print('a after calling:', a)
```

```
a before calling: 10
This is main a: 140711819842632
This is original a 140711819842632
This is the new a = 140711819842952
inside the fun a = 20
a after calling: 10
```

In [29]:
```
def change(a):
    print('This is original a',id(a))

    print('This is the new a =',id(a))
    a = a+ 10
    print('inside the fun a =',a)

a = 10
print('a before calling:', a)
print('This is main a:',id(a))
change(a)
print('a after calling:', a)
```

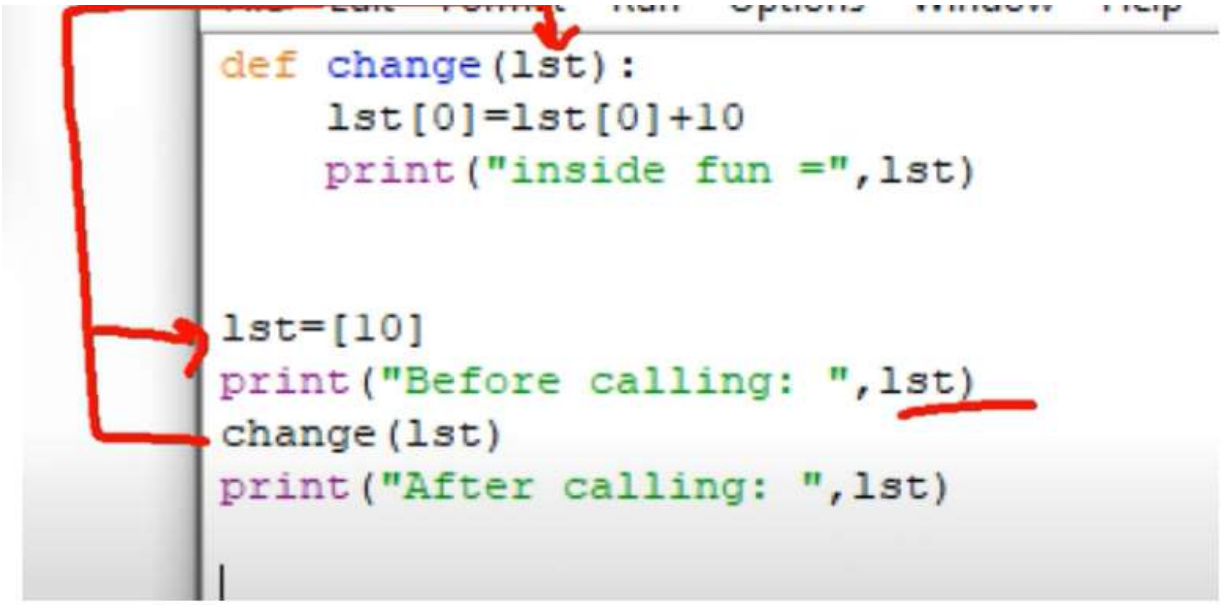
```
a before calling: 10
This is main a: 140711819842632
This is original a 140711819842632
This is the new a = 140711819842632
inside the fun a = 20
a after calling: 10
```

In [31]:
```
def change(a):
    print('This is original a',id(a))
    a = a+ 10
    print('This is the new a =',id(a))
    print('inside the fun a =',a)

a = 10
print('a before calling:', a)
print('This is main a:',id(a))
change(a)
print('a after calling:', a)
```

```
a before calling: 10
This is main a: 140711819842632
This is original a 140711819842632
This is the new a = 140711819842952
inside the fun a = 20
a after calling: 10
```

In [32]:
```
def change(a):
    #print('This is original a',id(a))
    a = a+ 10
    print('This is the new a =',id(a))
    print('inside the fun a =',a)

a = 10
print('a before calling:', a)
print('This is main a:',id(a))
change(a)
print('a after calling:', a)
print('This is original a',id(a))
```

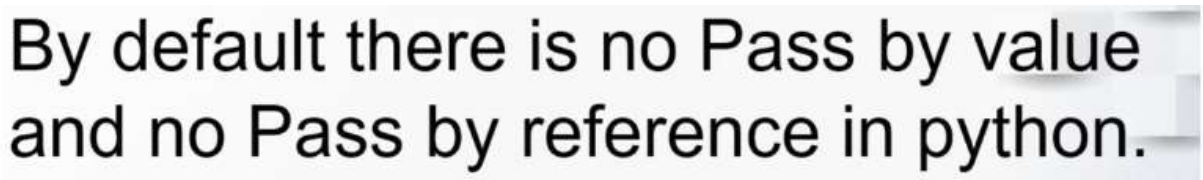
```
a before calling: 10
This is main a: 140711819842632
This is the new a = 140711819842952
inside the fun a = 20
a after calling: 10
This is original a 140711819842632
```



- in pass by reference lets pass the mutable e.g list & lets understand the concept

In [33]:
```
def change(lst):
    lst[0] = lst[0]+10
    print('inside fun =', lst)

lst = [10]
print('Before calling:', lst)
change(lst)
print('After calling:',lst)
```

```
Before calling: [10]
inside fun = [20]
After calling: [20]
```



In [34]:
```
def update(x):
    x = 8
    print('x : ', x)

a = 10
update(a)
print('a : ',a)
```

```
x :  8
a :  10
```

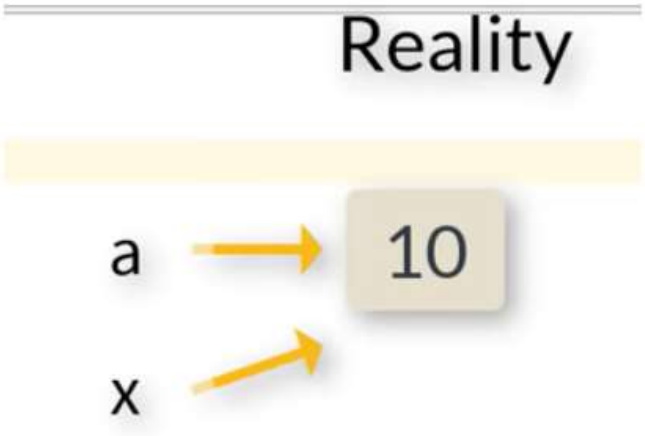
In [35]:
```
def update(x):
    print(id(x))
    x = 8
    #print(id(x))
    print('x', x)

a = 10
print(id(a))
update(a)
print('a',a)
```

```
140711819842632
140711819842632
x 8
a 10
```

In [36]:
```
def update(x):
    #print(id(x))
    x = 8
    print(id(x))
    print('x', x)

a = 10
print(id(a))
update(a)
print('a',a)
```

```
140711819842632
140711819842568
x 8
a 10
```

- if you notice the above result a & x refereing to both belongs to same address



- when you call a function by pass the value they will share the same memory location
- the variabel which you pass & the variable which you accessing heear a & x refer to same obejct
- the above concept is neither pass by value or pass by reference
- Interview(Normally in python we dont use pass by value or pass by reference but other programing language it does & the reason is python is object oriented programing languag

In [37]:
```
def update(x):
    x = 8

    print(id(x))
    print('x', x)

a = 10
print(id(a))

update(a)
print('a',a)

# we will understand more when we learn more
```

```
140711819842632
140711819842568
x 8
a 10
```

In [38]:
```
def update(x):
    print(id(x))
    x = 8
    print(id(x))
    print('x', x)

a = 10
print(id(a))
update(a)
print('a',a)
```

```
140711819842632
140711819842632
140711819842568
x 8
a 10
```

In [39]:
```
def update(lst):
    print(id(lst))

    lst[1] = 25
    print(id(lst))
    print('x', lst)

lst = [10,20,30] #lets pass list hear
print(id(lst))
update(lst)
print('lst',lst)
```

```
2550467809216
2550467809216
2550467809216
x [10, 25, 30]
lst [10, 25, 30]
```

NO concept for pass by value in python (please refer the code below)

In [40]:
```
def modify_integer(x):
    x = 10
    print("Inside function:", x)

my_integer = 5
modify_integer(my_integer)
print("Outside function:", my_integer)
```

```
Inside function: 10
Outside function: 5
```

In [41]:
```
def modify_integer(x):
    x = 10
    print("Inside function:", x)
    print('Inside function:',id(x))

my_integer = 5
modify_integer(my_integer)
print("Outside function:", my_integer)
print('Outside function:',id(x))
```

```
Inside function: 10
Inside function: 140711819842632
Outside function: 5
Outside function: 140711819842632
```

In [43]:
```
def modify_integer(x):
    print('original Inside function:',id(x))
    x = 10
    print("Inside function:", x)
    print('Inside function:',id(x))

my_integer = 5
modify_integer(my_integer)
print("Outside function:", my_integer)
print('Outside function:',id(x))
```

```
original Inside function: 140711819842472
Inside function: 10
Inside function: 140711819842632
Outside function: 5
Outside function: 140711819842632
```

NO concept for pass by reference in python (please refer the code below)

In [44]:
```
def modify_list(my_list):
    my_list.append(4)
    print("Inside function:", my_list)

my_list = [1, 2, 3]
modify_list(my_list)
print("Outside function:", my_list)
```

```
Inside function: [1, 2, 3, 4]
Outside function: [1, 2, 3, 4]
```

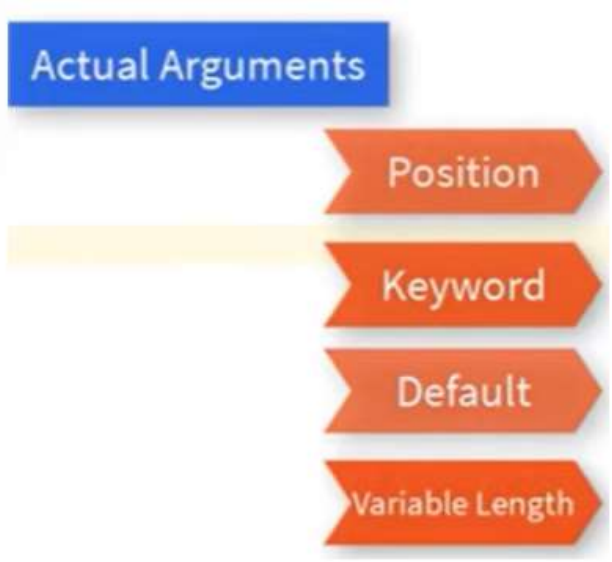
In [45]:
```
def modify_list(my_list):
    print("original Inside function:", id(my_list))
    my_list.append(4)
    print("Inside function:", my_list)
    print("Inside function:", id(my_list))

my_list = [1, 2, 3]
modify_list(my_list)
print("Outside function:", my_list)
print("Outside function:", id(my_list))
```

```
original Inside function: 2550446360128
Inside function: [1, 2, 3, 4]
Inside function: 2550446360128
Outside function: [1, 2, 3, 4]
Outside function: 2550446360128
```

TYPE OF ARGUMENTS --> formal argument & actual argument

In [46]:
```
def add(a,b): # a & b called formal argument
    c = a+b
    print(c)

add(5,6) #5 and 6 we called as actual argument
```

```
11
```



positional argument

In [48]:
```
def person(name,age):
    print(name)
    print(age)

person('nit',20)
# this is called postion argument becuase name we assigned to position name to nit & age to 28
# how it know that name - nit & age - 28 we need to take care of the position, we need to maintain the seque
# lets check we have 10 variable and we need to assign them with position but what if we forget the sequence
```

```
nit
20
```

In [49]:
```
def person(name,age):
    print(name)
    print(age)

person(20,'nit')

# in this case we cant assign name - 20 & age to 'nit' then we can assign them as keyword
# as we keep incorrect position this code trhoughs an error . so how to fix them
# i dont want assign name - 20 & age to nit & in this case we will assing keyword
```

```
20
nit
```

keyword argument

In [50]:
```
def person(name,age):
    print(name)
    print(age)

person(age = 20, name = 'nit')

#this is called keyword arguments
```

```
nit
20
```

default argument

- while you open meta account minumu age criterial is so. by default age is 18

In [52]:
```
def person(name,age): #in this code we expected to print 2 but we got bydefault
    print(name)
    print(age)

person('nit')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[52], line 5
      2     print(name)
      3     print(age)
----> 5 person('nit')

TypeError: person() missing 1 required positional argument: 'age'
```

In [53]:
```
def person(name,age = 18):
    print(name)
    print(age)

person('nit')
```

```
nit
18
```

In [54]:
```
def person(name,age = 18):
    print(name)
    print(age)

person('nit', 38)  #in hear bydefault override the existing default value
```

```
nit
38
```

variable length argument

In [55]:
```
def sum(a, b):
    c = a+b
    print(c)

sum(5,6)
```

```
11
```

In [56]:
```
def sum(a, b):
    c = a+b
    print(c)

sum(5,6)
```

```
11
```

- Everytime we cant add only 2 value we can also pass more the 3 values
- you can define the function when the number of argument are not fixed

In [57]:
```
def sum(a, b):
    c = a+b
    print(c)

sum(5,6,7,8)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[57], line 5
      2     c = a+b
      3     print(c)
----> 5 sum(5,6,7,8)

TypeError: sum() takes 2 positional arguments but 4 were given
```

In [58]:
```
def sum(a, *b): # 1st argument is fixed but for 2nd argument
    c = a+b
    print(c)

sum(5,6,7,8)
# we got error as int & tuple error becuase a is integer & b is tuple
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[58], line 5
      2     c = a+b
      3     print(c)
----> 5 sum(5,6,7,8)

Cell In[58], line 2, in sum(a, *b)
      1 def sum(a, *b): # 1st argument is fixed but for 2nd argument
----> 2     c = a+b
      3     print(c)

TypeError: unsupported operand type(s) for +: 'int' and 'tuple'
```

In [59]:
```
def sum(a, *b): # 1st argument is fixed but for 2nd argument
    #c = a+b
    print(type(a))
    print(type(b))

sum(5,6,7,8)
```

```
<class 'int'>
<class 'tuple'>
```

In [61]:
```
def sum(a, *b): # 1st argument is fixed but for 2nd argument
    #c = a+b
    print(a)
    print(b)

sum(3,5,7,8)
```

```
3
(5, 7, 8)
```

In [62]:
```
def sum(a, *b): # 1st argument is fixed & we fetch each value from the tuple & we can add them.
    c = a
    for i in b:
        c = c + i
        print(c)

sum(5,6,7,8)
```

```
11
18
26
```

In [63]:
```
def sum(a, *b):

    c = a
    for i in b:
        c = c + i
    print(c)

sum(5,6,7,8)
```

```
26
```

In [64]:
```
def sum(a, *b):

    c = a

    for i in b:
        c = c + i
    print(c)

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

```
45
```

In [65]:
```
def sum(a, *b):

    c = 0
    for i in b:
        c = c + i
    print(c)

sum(5,6,7,8)
```

```
21
```

- KWARGs (key worded variable length arguments)

In [66]:
```
def person():
    person('ALEX', 36, 'JOHN', 987767)
```

In [67]:
```
def person(name,*data):
    print('name')
    print(data)

person('ALEX', 36, 'JOHN', 987767)
#hear what is name - is it JOHN or ALEX thats why we assigned keywords varible arguments
```

```
name
(36, 'JOHN', 987767)
```

In [68]:
```
def person(name,*data):
    print(name)
    print(data)

person('ALEX', 36, 'JOHN', 987767)
#hear what is name - is it southcit or AAA thats why we assigned keywords varible arguments
```

```
ALEX
(36, 'JOHN', 987767)
```

In [69]:
```
def person(name,*data):
    print('name')
    print(data)

person('ALEX', age = 36, home_place ='southcity', mob =987767)
# we got error as keyword argument thats why we add another *
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[69], line 5
      2     print('name')
      3     print(data)
----> 5 person('ALEX', age = 36, home_place ='southcity', mob =987767)

TypeError: person() got an unexpected keyword argument 'age'
```

In [70]:
```
def person(name,**data):
    print(name)
    print(data)

person('mark', age = 36, home_place ='southcity', mob =987767)
```

```
mark
{'age': 36, 'home_place': 'southcity', 'mob': 987767}
```

In [71]:
```
def person(name,**data):
    print(name)

    for i, j in data.items():
        print(i, j)

person('john', age = 36, home_place ='southcity', mob =987767, place = 'USA')
```

```
john
age 36
home_place southcity
mob 987767
place USA
```

Global variable vs Local Variable

- We discussed about variable & discussed about function
- Sometimes we are creating variable inside the function and sometimes we are creating variable outside of the function
- when we declare variable outside of the function that concept called as scope

In [72]:
```
a = 10

print(a)
```

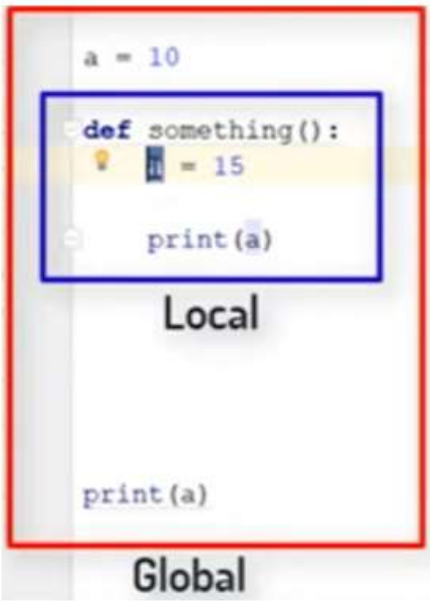
```
10
```

In [73]:
```
a = 10

def something():
    a = 15
    print('in function',a)

print('out function',a)

    # in this code we are declaring 2 variable is this possible
    # first line of a is called outside of the function
    # inside the function is called local variable
```

```
out function 10
```

In [74]:
```
a = 10

def something():
    a = 15

print('in function',a)

print('out function',a)

    # in this code we are declaring 2 variable is this possible
    # first line of a is called outside of the function
    # inside the function is called local variable
```

```
in function 10
out function 10
```



In [75]:
```
a = 10

def something():
    a = 15  #hear a is local variable
    b = 8
    print(a)

#print(b)
print(a)
```

```
10
```

In [76]:
```
a = 10

def something():
    a = 15  #hear a is local variable
    print(a)

print(a)
```

```
10
```

In [77]:
```
a = 10

def something():
    a = 15
    print('in function',a)  # local variable

print('out function',a) #gloabl variable

# In this code we ddint call the function
```

```
out function 10
```

In [78]:
```
a = 10

def something():
    a = 15
    print('in function',a)  # local variable

something()
print('out function',a) #gloabl variable

# 1st preference is always local variable
```

```
in function 15
out function 10
```

In [79]:
```
a = 10

def something():
    #if we remove this variable then can befault it consider as global variable
    print('in function',a)

something()
print('out function',a)
# if we dont assign any variabel inside the functin bydefault both considerd as local variable
```

```
in function 10
out function 10
```

In [80]:
```
a = 10

def something():
    a = 55
    print('in function',a)
something()

print('out function',a)
```

```
in function 55
out function 10
```

In [81]:
```
# if i want to define global variabel inside the function

a = 10

def something():
    global a
    b = 15 # 15 is converted to local when user assigned global a
    print('in function',b)

something()
print('out function',a)

# now in this case we dont have local variable & all variables are global variable only
# so this is how we are assigned to local variabel & global variable
```

```
in function 15
out function 10
```

In [84]:
```
a = 10

def something():
    global a
    a = 15     # we refered local to global
    print('in function',a)

    a = 9 # i want a to be local variable
        #can we assigned loca variabel in the function answer is not cuz bydefault it will treate as global
        # can we declare local & gloabl inside th function
something()

print('out function',a)
```

```
in function 15
out function 9
```

In [85]:
```
import keyword
keyword.kwlist
```

Out[85]:
```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
 'if',
 'import',
 'in',
 'is',
 'lambda',
 'nonlocal',
 'not',
 'or',
 'pass',
 'raise',
 'return',
 'try',
 'while',
 'with',
 'yield']
```

In [86]:
```
# if we used local & global in the same function this is not good idea thats wy introduced to GLOBALS

a = 10
print(id(a))

def something():
    a = 9
    x = globals()['a'] #gloabls can give you all the gloabls

    print(id(x))
    print('in function',a)


something()
print('out function',a)
```

```
140711819842632
140711819842632
in function 9
out function 10
```

In [87]:
```
# now lets introduce special function called globals & using globals we can access global variable address

a = 10
print(id(a))

def something():
    a = 9
    x = globals() # if i dont mention a then it will creat new memory

    print(id(x))
    print('in function',a)

    globals()['a'] = 15


something()
print('out function',a)
```

```
140711819842632
2550446769792
in function 9
out function 15
```

pass list to function

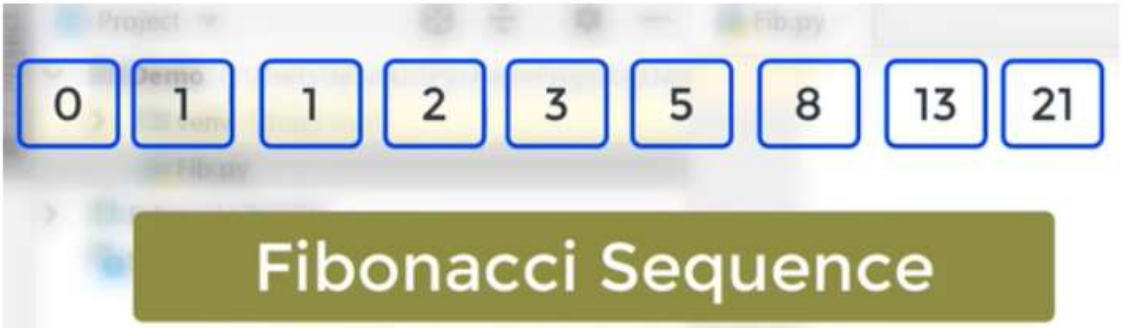
Can we pass list of element in the function that function will return the count of even or odd number from the list

In [88]:
```
def count(lst):

    even = 0
    odd = 0

    for i in lst:
        if i%2 == 0:
            even += 1
        else:
            odd +=1
    return even,odd

lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11]
even,odd = count(lst)

print(even)
print(odd)
```

```
5
6
```

In [89]:
```
def count(lst):

    even = 0
    odd = 0

    for i in lst:
        if i%2 == 0:
            even += 1
        else:
            odd +=1
    return even,odd

lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11]
even,odd = count(lst)

print("Even Number: {} and odd Number : {}".format(even,odd))
#format is function belongs to string& bydefault you need to pass 2 parameter
```

```
Even Number: 5 and odd Number : 6
```



In [90]:
```
def fib(n):
    print(0)
    print(1)

fib(0)

# in the above code we can get the fibonaci series but if the number is large then it takes more time
```

```
0
1
```

In [91]:
```
def fib(n):
    print(0)
    print(1)
    print(1)
    print(2)
    print(3)
    print(5)

fib(0)
```

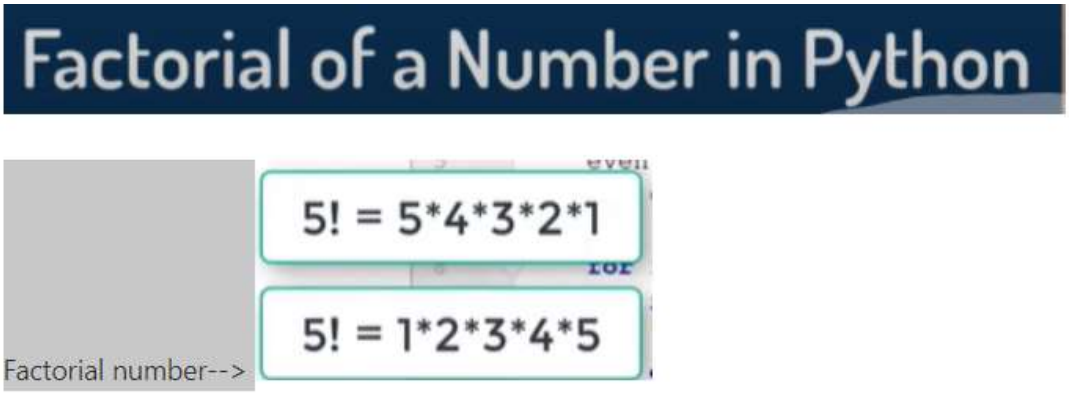
```
0
1
1
2
3
5
```

In [92]:
```
# in progammin we need to continue these process thats why we need to use loop hear

def fib(n):
    a = 0
    b = 1

    print(a)
    print(b)

    for i in range(2, n):
        c = a + b
        a = b
        b = c

        print(c)

fib(5)
```

```
0
1
1
2
3
```

In [93]:
```
# Ignore below code
'''if user wants 5 value then above code is applicable but if user wants only 1 value then if you write
#fib(1) then you will get 2 vales thats why we need to write the condition hear.'''

def fib(n):
    a, b = 0, 1
    if n == 1:
        print(a)
    else:
        print(a)
        print(b)

        for i in range(2, n):
            c = a + b
            a = b
            b = c
            print(c)
fib(2)
```

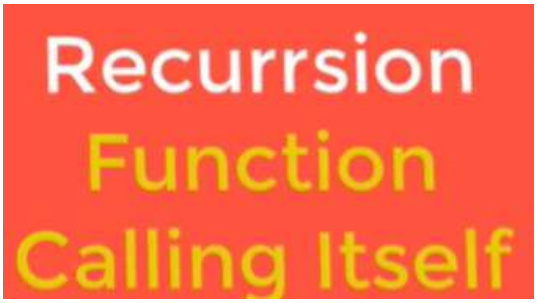
```
0
1
```



In [94]:
```
def fact(n):
    f = 1
    for i in range(1, n+1):
        f = f*i

    return f

x = 5
result = fact(x)
print(result)

# please use debug the code in pycharm for more indetail explanation & breakthrouth point at f = 1
```

```
120
```

RECURSSION - CALLING FUNCTION FROM ITSELF IS CALLED RECURSSION



In [95]:
```
def wish():
    print('hello')
wish()
```

```
hello
```

In [96]:
```
# i want to cal the hello multiple time
# it will execute maximum 1000 time & in below code wish is calling by itself
# bydefault we have 1000 limitation can we extend the recurssion limitation yes we can

def wish(): #---------> 2-greeting function will executed
    print('hello')
wish() # What if i call the function again #3---------> function calls itself is called recurssion

wish() #------------> 1-at this point we are calling wish() function

# it will print infinity time cuz recursion its own function
```

```
hello
hello
```

In [97]:
```
def wish():
    print('hello')
    wish()
wish()
```

```
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

In [98]:
```
import sys
print(sys.getrecursionlimit())
```

```
3000
```

In [99]:
```
sys.setrecursionlimit(2000)
```

In [100]:
```
print(sys.getrecursionlimit())
```

```
2000
```

In [ ]:
```
'''
def wish():
    print('hello')
    wish()
wish()
'''
#kernal will dead
```

In [102]:
```
import sys
sys.setrecursionlimit(150)
print(sys.getrecursionlimit())

i = 0

def wish():
    global  i
    i += 1
    print('hello', i)
    wish()
wish()

# how to know how many wish it printed
# so for best practice i would suggest for
```

```
150
hello 1
hello 2
hello 3
hello 4
hello 5
hello 6
hello 7
hello 8
hello 9
hello 10
hello 11
hello 12
hello 13
hello 14
hello 15
hello 16
hello 17
hello 18
```

FACTORIAL USING RECURSSION

recurssion is funcion calls itself

In [103]:
```
def fact(n):
    if n==0:
        return 1
    return n * fact(n-1)

result = fact(5)

result
```

Out[103]: 120

In [104]:
```
def square(a):
    return a * a
result = square(5)
print(result)

# what if i dont want to call square() multiple times
```

```
25
```

In [105]:
```
#lambda expresion or lambda function
f = lambda a: a * a # hear a is an argument & operation in the argument is a * a
result = f(5)
result
# hear anonymous function is called lambda
# remember lambda alway you need to assgin as function cuz function are object in python
```

Out[105]: 25

In [106]:
```
#lets define function which will add 2 number
# we are defining a function whcih doesnt have name
#lambda expresion or lambda function

f = lambda a, b : a + b

result = f(1,4)
result
```

Out[106]: 5

In [ ]:
```
How can we use lambda in other function like - filter, map & reduce
```

In [107]:
```
#lets take one list & i want to find the list of even numbers
nums = [3,2,6,8,4,6,2,9]

evens = list(filter(is_even, nums)) #is_even is not an inbuild function
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[107], line 4
      1 #lets take one list & i want to find the list of even numbers
      2 nums = [3,2,6,8,4,6,2,9]
----> 4 evens = list(filter(is_even, nums))

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

In [114]:
```
def is_even(n):
    return n % 2 == 0

nums = [3,2,6,8,4,6,2,9]
evens = list(filter(is_even, nums))
print(evens)
# remember filter always takes 2 argument 1- function for the logic 2- sequence or list
```

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

In [115]:
```
def is_odd(n):
    return n % 2 != 0

nums = [3,2,6,8,4,6,2,9]
odd = list(filter(is_odd, nums))
print(odd)
```

```
[3, 9]
```

In [116]:
```
# lets write above function using help of lambda & lambda helps to reduce the line
nums = [3,2,6,8,4,6,2,9]
evens = list(filter(lambda n : n%2 ==0, nums))
print(evens)
```

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

In [117]:
```
nums = [3,2,6,8,4,6,2,9]
odd = list(filter(lambda n : n%2 !=0, nums))
print(odd)
```

```
[3, 9]
```

- What ever even number I have from the assigned list
- I want to double the even number i.e 2 become 4 || 4 become 6 || 6 become 8
- that we will do using map function
- this largly we are using in google map reduce programme
- we can build using user define & lambda

In [118]:
```
def update(n):
    return n*2

nums = [3,2,6,8,4,6,2,9]
evens = list(filter(is_even, nums))
double = list(map(update, evens))

print(double)
```

```
[4, 12, 16, 8, 12, 4]
```

In [119]:
```
nums = [3,2,6,8,4,6,2,9]
evens = list(filter(is_even, nums))
#double = list(map(lambda n : n*2, evens))
double_ = list(map(lambda n : n-2, evens))
#print(double)
print(double_)
```

```
[0, 4, 6, 2, 4, 0]
```

In [120]:
```
nums = [3,2,6,8,4,6,2,9]
evens = list(filter(is_even, nums))
double = list(map(lambda n : n*2, evens))
double_ = list(map(lambda n : n-2, evens))

print(double)
print(double_)
```

```
[4, 12, 16, 8, 12, 4]
[0, 4, 6, 2, 4, 0]
```

In [121]:
```
nums = [3,2,6,8,4,6,2,9]
evens = list(filter(is_even, nums))
double = list(map(lambda n : n*2, evens))
double_ = list(map(lambda n : n-2, evens))
double1 = list(map(lambda n : n+2, evens))

print(double)
print(double_)
print(double1)
```

```
[4, 12, 16, 8, 12, 4]
[0, 4, 6, 2, 4, 0]
[4, 8, 10, 6, 8, 4]
```

- i want to perform reduce now
- i want reduce all the values
- reduce you can add only 2 values
- [4, 12, 16, 8, 12, 4] if you sum everything then you will get 56

In [122]:
```
from functools import reduce

def add_all(a,b):
    return a+b

nums = [3,2,6,8,4,6,2,9]

evens = list(filter(is_even, nums))
double = list(map(lambda n : n*2, evens))
sums = reduce(add_all, double)
sums
#print(sums)
```

Out[122]:
```
56
```

In [123]:
```
from functools import reduce

nums = [3,2,6,8,4,6,2,9]

evens = list(filter(is_even, nums))
double = list(map(lambda n : n*2, evens))
sums = (reduce(lambda a,b : a + b, double))

print(evens)
print(double)
print(sums)
```

```
[2, 6, 8, 4, 6, 2]
[4, 12, 16, 8, 12, 4]
56
```

Special Variable **name**

In [124]:
```
__name__
```

Out[124]:
```
'__main__'
```

In [125]:
```
print(__name__)
```

```
__main__
```

In [ ]:
```

```