In [1]:
```
def Boss():
    print('good morning team')
```

In [2]:
```
def Boss():
    print('good morning team')
Boss()
```

```
good morning team
```

In [3]:
```
def greet():
    print('Hello')
    print('good morning')
```

In [4]:
```
def greet():
    print('Hello')
    print('good morning')
greet()
```

```
Hello
good morning
```

In [5]:
```
def greet():
    print('Hello')
    print('good morning')
greet()

def greet():
    print('Hello')
    print('good morning')
greet()

def greet():
    print('Hello')
    print('good morning')
greet()
```

```
Hello
good morning
Hello
good morning
Hello
good morning
```

In [6]:
```
def greet():

    print('Hello good morning boss')

greet()
```

```
Hello good morning boss
```

In [7]:
```
def greet():

    print('Hello good morning boss')

greet()

greet()
```

```
greet()

greet()
```

```
Hello good morning boss
Hello good morning boss
Hello good morning boss
Hello good morning boss
```

In [8]:
```
def add(*nums):
   print(sum(nums))

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

```
26
```

In [9]:
```
def add(x,y):
    c = x + y
    print(c)

print(5,6)
```

```
5 6
```

In [10]:
```
def add(x,y,z,m):
    c = x + y + z + m
    print(c)
add(1,4,5,2)
```

```
12
```

In [11]:
```
def add(x,y,z,m):
    c=x+y+z+m
    print(c)

add(1,4,5,6)
```

```
16
```

In [12]:
```
def greet():
    print('Hello')
    print('good morning')

greet()
```

```
Hello
good morning
```

In [13]:
```
def add(x,y):
    c = x + y
    print(c)

add(5,6)
```

```
11
```

In [14]:
```
def greet():
    print('Hello')
    print('good morning')
greet()

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

```
    print(c)
add(5,6)
```

```
Hello
good morning
11
```

In [15]:
```
def greet():
    print('Hello')
    print('good mornig')
def add(x,y):
    c = x + y
    print(c)

add(5,6)
greet()
```

```
11
Hello
good mornig
```

In [16]:
```
def greet():
    print('Hello')
    print('good noon')

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

def sub(x,y):
    d = x-y
    print(d)

greet()
add(5,6)
sub(10,2)
```

```
Hello
good noon
11
8
```

In [17]:
```
def add_sub(x,y):
    c = x+y
    d = x-y
    print(c)
    print(d)

add_sub(10,5)
```

```
15
5
```

In [18]:
```
def add_sub(x,y):
    c = x+y
    d = x-y
    return c,d

add_sub(10,5)
```

Out[18]: (15, 5)

In [19]:
```
def add_sub(x,y):
    c = x+y
    d = x-y
    return c,d

result = add_sub(5,4)

print(result)
```

```
(9, 1)
```

In [20]:
```
def add(x,y):
    c = x+y
    print(c)
add(5,6)
```

```
11
```

FORMAL ARGUMENT & ACTUAL ARGUMENT

In [21]:
```
def person(name,age,number):
    print(name)
    print(age)

person('Ruwan',23,34)
```

```
Ruwan
23
```

In [22]:
```
def person(name,age):
    print(name)
    print(age)

person('Ruwan',23)
```

```
Ruwan
23
```

In [23]:
```
def person(name,age):
    print(name)
    print(age)

person(23,'Ruwan')
```

```
23
Ruwan
```

In [24]:
```
def person(name, age):
    age = int(age)
    print(name)
    print(age + 1)

person('Ruwan', 23)
```

```
Ruwan
24
```

In [25]:
```
# Keyword Argument
```

In [26]:
```
def person(name,age):
    print(name)
    print(age+1)

person(age=23,name='Ruwan')
```

```
Ruwan
24
```

In [27]:
```
def person(name,age):
    print(name)
    print(age+1)

person( name='Ruwan',age=23)
```

```
Ruwan
24
```

In [28]:
```
def person(name,age1):
    print(name)
    print(age1+1)

person(age1=23,name='Ruwan')
```

```
Ruwan
24
```

In [29]:
```
def person(name,age,city):
    print(name)
    print(age+1)
    print(city)

person(age=23,name='Ruwan',city='hyd')
```

```
Ruwan
24
hyd
```

In [30]:
```
def person(name,age,city):
    print(name)
    print(age+1)
    print(city)

person(age=23,name='Ruwan',city = 'hyd')
```

```
Ruwan
24
hyd
```

In [31]:
```
def person(name,age=18):
    print(name)
    print(age)

person('Ruwan',24)
```

```
Ruwan
24
```

variable length argument

In [32]:
```
def sum(a,b):
    c = a+b
    return c

sum(5,6)
```

Out[32]: 11

In [33]:
```
def sum(a, *b):
    c = a
    for i in b:
        c += i
    return c

sum(5,6,7,8,9,10)
```

Out[33]: 45

In [34]:
```
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 [35]:
```
def sum(a,*b): # 1st argument is fixed but for 2nd argument
    c = a

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

sum(5,6,7,8,9,10,100,200,300)
```

```
645
```

In [36]:
```
def sum(a,*b):
    c = a
    for i in b:
        c = c+i
    print(c)

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

```
26
```

- Positional Arguments
- Keyword Arguments
- default
- Variable length (* at last arg)|(args)
- keyword + variable length(kwargs)

key word length arguments

Function Arguments we are completed

Global variables vs Local variables

In [37]:
```
a = 10

print(a)
```

```
10
```

In [38]:
```
a = 10
def something():
    b = 15
    return b

b = something()
print('in function',b)
print('out function',a)
```

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

In [39]:
```
a = 10

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

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

```
out function 10
```

In [40]:
```
a = 10
def something():
    a = 15

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

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

In [41]:
```
a = 10

def something():
    b = 15
    print('in Function',b)

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

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

In [42]:
```
# if i want to define global variable inside the function
a = 10

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

```
in function 15
global variable 10
out function 10
```

In [43]:
```
x = 10 # Global varible

def update_x():
    global x # Declare that we are using the global variable x
    x += 5 # Modify the global variable

update_x()
print(x)
```

```
15
```

In [44]:
```
x = 10 # global variable

def updata_x():
    globals()['x'] += 5 # Access and modify the global variable

update_x()
print(x)
```

```
15
```

In [45]:
```
def count():
    lst = [1,2,3,4,8,9,10]
    print(lst)

count()
```

```
[1, 2, 3, 4, 8, 9, 10]
```

Pass By Value

Pass By Reference

Pass by value

In [46]:
```
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 thoughbi changed values x to a still we got the same result.
- THIS CONCEPT CALLED AS (PASS BY VALUE) FOR OTHER PROGRAMING.

In [47]:
```
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
```

In [48]:
```
def change(a):
    print('This is original a',id(a))
    a = a+10
    print('This is the new a =',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: 140726431298760
This is original a 140726431298760
This is the new a = 20
inside the fun a =  20
a after calling: 10
```

In [49]:
```
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: 140726431298760
This is original a 140726431298760
This is the new a =  140726431299080
inside the fun a =  20
a after calling: 10
```

In [50]:
```
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: 140726431298760
This is the new a =  140726431299080
inside the fun a= 20
a after calling: 10
This is original a 140726431298760
```

In [51]:
```
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]
```

By default there is no Pass by value and no Pass by reference in python.

In [52]:
```
def update(x):
    x = 8
    print('x:',x)
a = 10
update(a)
print('a:',a)
```

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

In [53]:
```
def update(x):
    print(id(x))
    x = 8
    # print(id(x))
    print('x : ',x)
```

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

```
140726431298760
140726431298760
x :  8
a 10
```

In [54]:
```
def update(x):
    #print(id(x))
    x = 8
    print(id(x))
    print('x',x)
a = 10
print(id(a))
update(a)
print('a',a)
```

```
140726431298760
140726431298696
x 8
a 10
```

Expectation

a ---> 10

x ---> 10

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

Reality

a ---> 10 x ---> 10

- when you call a function by the value they will share the same memory location.
- The variable which you pass & the variable which you accessing hear a & x refer to same object.

In [55]:
```
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
```

```
140726431298760
140726431298696
x 8
a 10
```

In [56]:
```
def update(x):
    print(id(x))
    x = 8
    print(id(x))
    print('x',x)

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

```
140726431298760
140726431298760
140726431298696
x 8
a 10
```

In [57]:
```
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)
```

```
2465345499328
2465345499328
2465345499328
x [10, 25, 30]
lst [10, 25, 30]
```

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

In [58]:
```
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 [59]:
```
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: 140726431298760
Outside function: 5
Outside function: 140726431298760
```

In [60]:
```
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: 140726431298600
Inside function: 10
Inside function: 140726431298760
Outside function: 5
Outside function: 140726431298760
```

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

In [61]:
```
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 [62]:
```
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: 2465384312064
Inside function: [1, 2, 3, 4]
Inside function: 2465384312064
Outside function: [1, 2, 3, 4]
Outside function: 2465384312064
```

Pass list to function

- can we pass list of element in the function will return the count of even or Odd number from the list.

In [63]:
```
def count (lst):

    even = 0
    odd = 0

    for i in lst:
        if i%2 == 0:
            even+= 1
        else:
            odd += 1
    return even,odd
lst = [10,9,8,2]
even,odd = count(lst)

print(even)
print(odd)
```

```
3
1
```

In [64]:
```
def count(lst):

    even = 0
    odd = 0

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

lst = [1,2,3,4,5,6,7,8,9,10,11,12,13]
even,odd = count(lst)

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

```
Even Number: 6 and odd Number : 7
```

Fibonacci sequence

- 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21
- inbuild function so we need to define the function for it.

In [65]:
```
def fib(n):
    print(0)
    print(1)

fib(0)
```

```
0
1
```

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

In [66]:
```
def fib(n):
    print(0)
    print(1)
    print(1)
    print(2)
    print(3)
    print(5)

fib(0)
```

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

- 0 1 1 2 3 5 8 13 21
- a b
- 0 1 1 2 3 5 8 13 21
- a b c

in program in we need to continue these process thats why we need to use loop hear.

In [67]:
```
def fib(n):
    a = 0
    b = 1

    print(a)
    print(b)

    for i in range(0,n):
        c = a + b
        a = b
        b = c
```

```
        print(c)

fib(10)
```

```
0
1
1
2
3
5
8
13
21
34
55
89
```

Ignore below code

''' if user want 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 values thats why we need to write the condition hear.'''

In [68]:
```
def fib(n):
    a,b = 0,1
    if n==(a):
       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
```

Factorial of a Number in python

5! = 54321

----> Factorial number 5! = 12345

In [69]:
```
def fact (n):
    f = 1
    for i in range(1,n+1):
        f = f*i

    return f

x = 6
result = fact(x)
print(result)
# please use debuge the code in pycharm for more indetial explanation & breakthr
```

```
720
```

Anonymous Function | Lambda

- Function without name is called - Anonymous function or lambda.

In [70]:
```
def square(a):
    return a * a

result = square(5)
print(result)
# what if i dont want to call square() multiple times
```

```
25
```

In [71]:
```
def square(a):
    return a * a

result  = square(5)
print(result)
# what if i want to call square() multiple times
```

```
25
```

In [72]:
```
f = lambda a:a*a
result = f(5)
result
```

Out[72]: 25

In [73]:
```
f = lambda a,b : a + b
f1 = lambda a,b : a - b

result = f(1,4)
```

```
result1 = f1(4,1)

print(result)
print(result1)
```

```
5
3
```

In [74]:
```
f = lambda a,b : a + b
f1 = lambda a,b : a - b
f2 = lambda a,b : a * b

result = f(1,4)
result1 = f1(4,1)
result2 = f2(4,1)

print(result)
print(result1)
print(result2)
```

```
5
3
4
```

In [75]:
```
f = lambda a,b : a + b
f1 = lambda a,b : a - b

result = f(1,4)
result1 = f1(2,3)

print(result)
print(result1)
```

```
5
-1
```

In [76]:
```
import keyword
keyword.kwlist
```

Out[76]:
```
['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']
```

- How can we use lambda in other function like - filter,map & reduce

Filter()

map()

reduce()

In [77]:
```
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)
```

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

In [78]:
```
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 [79]:
```
nums = [3,2,6,8,4,6,2,9]

even = list(filter(lambda n:n%2 == 0,nums))

print(evens)
```

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

In [80]:
```
nums = [3,2,6,8,4,6,2,9]

odd = list(filter(lambda n : n%2 !=0,nums))

print(odd)
```

```
[3, 9]
```

In [81]:
```
nums = [3,2,6,8,4,6,2,9,34,77,120]

evens = list(filter(lambda n : n%2 == 0, nums))
odd = list(filter(lambda n : n%2 !=0,nums))

print(evens)
print(odd)
```

```
[2, 6, 8, 4, 6, 2, 34, 120]
[3, 9, 77]
```

In [82]:
```
nums = [3,2,6,8,4,6,2,9]

evens = list(filter(is_even,nums))
double = list(map(lambda n:n*2,evens))

print(evens)
print(double)
```

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

In [83]:
```
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))
doubble_1 = list(map(lambda n : n-2,evens))

print(evens)
print(double)
print(double_)
print(doubble_1)
```

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

In [84]:
```
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 [85]:
```
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]
```

In [86]:
```
a = [7,8]
print(type(a))
```

```
<class 'list'>
```

In [87]:
```
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
```

PYTHON decorators

- This is one of amazing feacture in python.
- We knew that function are build on certain task.
- lets understand we have some predefined function.

- div() function which takes 2 parameter.

In [88]:
```
def div(a,b):
    print(a/b)
div(4,2)
```

```
2.0
```

In [89]:
```
def div(a,b):
    print(a/b)
div(2,4)
```

```
0.5
```

In [90]:
```
def div(a,b):

    if a<b:
        a,b = b,a
    print(a / b)

div(2,4)
```

```
2.0
```

In [91]:
```
def div(a,b):
    print(a/b)

def div_decorator(func):
    def inner(a,b):
        if a<b:
            a,b = b,a
        return func(a,b)
    return inner
div = div_decorator(div)

div(2,4)
```

```
2.0
```

In [92]:
```
def my_decorator(func):
    def wrapper():
        print("something is happening before the function is called.")
        #func()
        print("something is happening after the function is called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

```
something is happening before the function is called.
something is happening after the function is called.
```

In [93]:
```
def my_decorator(func):
    def wrapper():
        print("something is happening before the function is called.")
        func()
        print("something is happening after the function is called.")
    return wrapper
```

```
@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

```
something is happening before the function is called.
Hello!
something is happening after the function is called.
```

MODULES

- A B C D
- A C B D
- A C B D

Special Variable_name_

- Special Varible name

_ _ name _ _ == " _ _ main _ _"

In [94]:
```
__name__
```

Out[94]:
```
'__main__'
```

In [95]:
```
print(__name__)
```

```
__main__
```

- The moment when we print name then i will get output as main.
- In your project if you have 5 modules or 10 modules there will some modules which will be run first.
- So the first module name is always main. thats why the code start.