# Functions in Python

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

In [2]:
```
def greet():
    print('hello')
    print('good morning')
greet()
```

```
hello
good morning
```

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

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

In [4]:
```
def add(x,y):
    c=x+y
    print(c)
add(15,4)
```

```
19
```

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

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

greet()
result = add(5,4)
print(result)
```

```
hello
good morning
9
```

In [6]:
```
def add_sub(x,y): # what if i want to return 2 values add_sub & i want to return 2 values & function can accept mu
    c= x+y
    d= x-y
    return c, d

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

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

In [10]:
```
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 [11]:
```
def update():
    x = 8
    print(x)
update()
```

```
8
```

In [12]:
```
def update():
    x = 8
    print(x)
update(8)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[12], line 4
      2     x = 8
      3     print(x)
----> 4 update(8)

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

In [13]:
```
def update(x): # user want to update the value from 8 to 10
    x = 8
    print(x)
update(10)
```

```
8
```

In [14]:
```
def update(x):
    x = 8
    print(x)

a = 10
update(a)
```

```
8
```

In [15]:
```
def update(x):
    x = 8
    print(x)

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

```
8
10
```

In [16]:

```
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 [17]:

```
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: 2680456348176
This is original a 2680456348176
This is the new a = 20
inside the fun a = 20
a after calling: 10
```

In [18]:
```
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 [19]:
```
def update(x):
    x = 8
    print('x : ', x)

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

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

In [20]:

```
def update(x):
    print(id(x))
    x = 8
    #print(id(x))
    print('x', x)

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

```
2680456348176
2680456348176
x 8
a 10
```

In [21]:

```
def update(x):
    #print(id(x))
    x = 8
    print(id(x))
    print('x', x)

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

```
2680456348176
2680456348112
x 8
a 10
```

In [22]:
```
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
```

```
2680456348176
2680456348112
x 8
a 10
```

In [23]:
```
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)
```

```
2680564930176
2680564930176
2680564930176
x [10, 25, 30]
lst [10, 25, 30]
```

In [24]:
```python
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 [27]:
```python
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: 2680456348176
Outside function: 5
```

In [28]:
```python
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 [29]:
```
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: 2680565270016
Inside function: [1, 2, 3, 4]
Inside function: 2680565270016
Outside function: [1, 2, 3, 4]
Outside function: 2680565270016
```

## Types of Arguments

### Formal & Actual

In [30]:
```
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
```

## Actual Arguments

```
1.Positional, 2.Keyword, 3.Default, 4.Variable Length
```

## 1.Positional Argument

In [31]:
```
def person(name,age):
    print(name)
    print(age)
person('yogita',20)
```

```
yogita
20
```

In [32]:
```
def person(name,age):
    print(name)
    print(age)
person(20,'yogita')
```

```
20
yogita
```

In [33]:
```
def person(name,age):
    print(name)
    print(age-5)
person('yogita',20)
```

```
yogita
15
```

In [36]:
```
def person(name,age):
    print(name)
    print(age-5)
person(20,'yogita')
```

```
20

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[36], line 4
      2     print(name)
      3     print(age-5)
----> 4 person(20,'yogita')

Cell In[36], line 3, in person(name, age)
      1 def person(name,age):
      2     print(name)
----> 3     print(age-5)

TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

## 2.Keyword Arguments

In [37]:
```
def person(name,age):
    print(name)
    print(age)
person(age = 20, name = 'yogita')
```

```
yogita
20
```

## 3.Default Argument

In [38]:
```
def person(name,age): #in this code we expected to print 2 but we got bydefault
    print(name)
    print(age)
person('yogita')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 4
      2     print(name)
      3     print(age)
----> 4 person('yogita')

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

In [39]:
```
def person(name,age = 18):
    print(name)
    print(age)
person('YOGITA')
```

```
YOGITA
18
```

In [40]:
```
def person(name,age = 18):
    print(name)
    print(age)
person('YOGITA', 38)  #in hear bydefault override the existing default value
```

```
YOGITA
38
```

## 4.Variable Length Argument

In [41]:
```
def sum(a, b):
    c = a+b
    print(c)
sum(5,6)
```
```
11
```

In [42]:
```
def sum(a, b):
    c = a+b
    print(c)
sum(5,6,7,8)
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[42], line 4
      2     c = a+b
      3     print(c)
----> 4 sum(5,6,7,8)

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

In [43]:
```
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[43], line 5
      2     c = a+b
      3     print(c)
----> 5 sum(5,6,7,8)

Cell In[43], 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 [44]:
```
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 [45]:
```
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 [46]:
```
def sum(a, *b):
    c = a
    for i in b:
        c = c + i
    print(c)
sum(5,6,7,8,9)
```

```
35
```

In [47]:
```
def sum(a, *b):
    c = a
    for i in b:
        c = c + i
        print(c)
sum(5,6,7,8,9)
```

```
11
18
26
35
```

## Keyworded Variable Length Argument(KWARGS)

In [48]:
```
def person():
    person('ALEX', 36, 'JOHN', 987767)
```

In [49]:
```
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 [55]:
```
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[55], line 5
      2     print('name')
      3     print(data)
----> 5 person('ALEX',  home_place ='southcity', mob =987767)

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

In [56]:
```
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 [57]:
```
def person(name,**data):
    print('name')
    print(data)

person('mark', age = 36, home_place ='southcity', mob =987767, edu='phd', actor = 'john')
#even though you can keep add the keyword this concept we called as KWARGS
```

```
name
{'age': 36, 'home_place': 'southcity', 'mob': 987767, 'edu': 'phd', 'actor': 'john'}
```

In [58]:
```
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
```

## Local Variable vs Global Variable

In [1]:
```
a = 10
print(a)
```

```
10
```

In [2]:
```
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 [3]:
```
a = 10

def something():
    a = 15

print('in function',a)

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

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

In [4]:
```
a = 10

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

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

```
out function 10
```

In [5]:
```
a = 10

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

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

```
10
```

In [6]:
```
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 [7]:

```python
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 [8]:

```python
a = 10

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

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

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

In [9]:
```
# 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)
```

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

In [10]:
```
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 [11]:

```
# 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)
```

```
1895352173072
1895352173072
in function 9
out function 10
```

In [12]:

```python
# 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)
```

```
1895352173072
1895431192192
in function 9
out function 15
```

## Pass List to Function

In [13]:
```
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 [14]:
```
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
```

## Fibonacci Sequence

In [15]:
```
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 [16]:

```
def fib(n):
    print(0)
    print(1)
    print(1)
    print(2)
    print(3)
    print(5)

fib(0)
```

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

In [17]:
```python
# in progamming 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 [18]:
```
'''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
```

## Factorial of Number

In [19]:
```
def fact(n):
    f = 1
    for i in range(1, n+1):
        f = f*i

    return f

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

```
120
```

## Recursion Function

In [20]:
```
def wish():
    print('hello')
wish()
```

```
hello
```

In [21]:
```
# 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 [ ]:
```
def wish():
    print('hello')
    wish()
wish()
```

In [1]:
```
'''
def wish():
    print('hello')
    wish()
wish()
'''
#kernal will dead
```

Out[1]:
```
"\ndef wish():\n    print('hello')\n    wish()\nwish()\n"
```

In [2]:
```
import sys
print(sys.getrecursionlimit())
```

```
3000
```

In [3]:
```
sys.setrecursionlimit(4000)
```

In [4]:
```
print(sys.getrecursionlimit())
```

```
4000
```

In [5]:
```
import sys
sys.setrecursionlimit(150)
print(sys.getrecursionlimit())

i = 0

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

```
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 Recursion

In [6]:
```
def fact(n):
    if n==0:
        return 1
    return n * fact(n-1)

result = fact(5)

result
```

Out[6]:
```
120
```

## Anonymous Function| Lambda

In [7]:
```
def square(a):
    return a * a
result = square(5)
print(result)

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

```
25
```

In [1]:
```
#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 always you need to assgin as function because function are object in python
```

Out[1]:
```
25
```

## filter(), map(), reduce()

In [2]:
```
#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[2], 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 [3]:
```
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 [4]:
```
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 [5]:
```
# 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 [6]:
```
nums = [3,2,6,8,4,6,2,9]
odd = list(filter(lambda n : n%2 !=0, nums))
print(odd)
```

```
[3, 9]
```

In [7]:
```
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 [8]:
```
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]
```

In [9]:
```
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 [10]:
```
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[10]:
```
56
```

In [11]:
```
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)
```

```
56
```

In [12]:
```
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

In [13]:
```
def div(a,b):
    print(a / b)
div(4,2)
# but what if we pass the value 2, 4
```

```
2.0
```

In [14]:
```
def div(a,b):
    print(a / b)
div(2,4)
# but what if we pass the value 2, 4
```

```
0.5
```

In [15]:
```
def div(a,b):
    if a<b:
        a,b = b,a
    print(a / b)

div(2,4)
```

```
2.0
```

In [16]:
```
# using help of the decorator you can add the extra feature in the exicting function

def div(a,b):
    print(a / b)

def div_decorator(func): # hear div_dectorator will accept the div function
    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 [17]:
```
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

### special variable__name__ ¶

```
__name__="__main__"
```

In [18]:
```
__name__
```

Out[18]:
```
'__main__'
```

In [19]:
```
print(__name__)
```

```
__main__
```

In [ ]:
```
```