Functions in python

- Inbuild functions -- print(), type(), sqrt() etc - User defiend functions - Function is collections of statement - 2 main property of function is -- define the function and calling the function - fuction always define with def & function always declares as () different between variable and function || a - variable || b() - function

In [1]:
```
def greet():
    prinnt('Hello')
    print('Good Morning')
# When we run the code we haven't got any output
```

In [2]:
```
def greet(): # Function defination
    print('Hello World')
    print('Data Science and AI')

greet() # Call the function to get the output
```

```
Hello World
Data Science and AI
```

In [4]:
```
# if we want to print one message multiple times instade of writing multiple tim
def fun():
    print("I'm K Arati Pradhan")
fun()#if you need call multiple times


def fun():
    print("I'm K Arati Pradhan")
fun()#if you need call multiple times
```

```
I'm K Arati Pradhan
I'm K Arati Pradhan
```

In [5]:
```
# here we no need to write codes multiple times we can simply call the function
def fun():
    print("I'm K Arati Pradhan")
fun()
fun()
fun()
```

```
I'm K Arati Pradhan
I'm K Arati Pradhan
I'm K Arati Pradhan
```

FUNCTION WITH ARGUEMENTS

TYPE OF ARGUMENTS --> formal argument & actual argument

In [13]:
```
def add(a,b): # a & b called formal argument
    c = a+b
    print("The sum of {a} and {b} is :",c)
```

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

```
The sum of {a} and {b} is : 11
```

In [12]:
```
def add(a,b,d): # a & b called formal argument
    c = a+b+d
    print(c)

add(5,6,7) #5 and 6 we called as actual argument
```

```
18
```

Actual Arguemens are 4 types

POSITION
KEYWORD
DEFAULT
VARIABLE LENGTH

Positional Arguement

In [19]:
```
def person(name,age): # here we creat two formal arguments name and age
    print("Person's Name is",name)
    print("Person's age is",age)

person('Arati',21)
# this is called postion argument becuase name we assigned to position name to n
```

```
Person's Name is Arati
Person's age is 21
```

In [21]:
```
def f1(name, age): # here we creat two formal arguments name and age
    print("The name :", name)
    print('The age :', age)
f1(21,"Arati") # in this case we cant assign name - 21 & age to 'Arati' then we
# 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 keywor
```

```
The name : 21
The age : Arati
```

In [25]:
```
def f1(name, age): # here we creat two formal arguments name and age
    print("The name :", name)
    print('The age :', age)
f1("Arati") # it need two actual arguements as we assigned 2 formal arguements t
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[25], line 4
      2     print("The name :", name)
      3     print('The age :', age)
----> 4 f1("Arati")

TypeError: f1() missing 1 required positional argument: 'age'
```

In [29]:
```
def f1(name, age=21): # here we give a value 23 to age
    print("The name:", name)
    print('The age :', age)
f1('Arati') # it work properly because age vale a defult value
```

```
The name: Arati
The age : 21
```

In [30]:
```
def person(name,age):
    print(name)
    print(age-5)

person(20,'nit')
# in this case we cant assign name - 20 & age to 'nit' then we can assign them a
# this code not give you error but output is in different format
```

```
20
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[30], line 5
      2     print(name)
      3     print(age-5)
----> 5 person(20,'nit')

Cell In[30], 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'
```

Keyword arguments

In [34]:
```
def f(name, age,state):
    print("Persons's name:", name)
    print("Persons's age :", age)
    print("The state is: ",state)
f(state="Odisha",name="Arati",age=21) # here we pass according to the variable n
```

```
Persons's name: Arati
Persons's age : 21
The state is:  Odisha
```

In [37]:
```
def f1(name, age,state):
    name="Arati" # inside the function we change update the name
    print("The name:", name)
    print('The age :', age)
    print("The state:",state)

f1(state="odisha",name="Aru",age=21) # here we pass a actual arguments as DK
```

```
The name: Arati
The age : 21
The state: odisha
```

Variable Length Arguments

In [42]:
```
def f(a,b):
   c= a+b
   print("the sum is :", c)
f(3,6)
```

```
the sum is : 9
```

In [43]:
```
# we need to pass the more than two value in the function even we use only two f
def f1(a, *b):
    c= a+b
    print("the sum is:", c)
f1(3,66,4,5,6) # here we get error because a=3, and rest value b=(66,4,5,6) beha
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[43], line 5
      3     c= a+b
      4     print("the sum is:", c)
----> 5 f1(3,66,4,5,6)

Cell In[43], line 3, in f1(a, *b)
      2 def f1(a, *b):
----> 3     c= a+b
      4     print("the sum is:", c)

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

In [45]:
```
def f1(a, *b): # Here first arg. is fixed and rest of the value consider as tupl
    print(type(a)) # int
    print(type(b)) # tuple
f1(3,66,4,5,6)
```

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

In [46]:
```
# to acces the value of the tuple we need to apply the for loop
def f(a, *b):
    for i in b:
        a=a+i
        print(a)

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

```
11
18
26
35
45
```

In [47]:
```
def f(a,*b):
    for i in b:
        a=a+i
    print("The Sum is: ",a)

f(4,5,6,7,8,9,0,1,2,3)
```

```
The Sum is:  45
```

Keyword variable length arguments

KWARGs (key worded variable length arguments)

In [48]:
```
def fun(name,*data): # we get error to avoide the eror we need to apply double *
    print("Name:",name)
    print(data) # here first arguments is fixed and rest of all taken as a dicti
fun("Arati", salary=80000,designation=" Data science")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[48], line 4
      2     print("Name:",name)
      3     print(data) # here first arguments is fixed and rest of all taken as
a dictionary
----> 4 fun("Arati", salary=80000,designation=" Data science")

TypeError: fun() got an unexpected keyword argument 'salary'
```

In [49]:
```
def f1(name,**data): # we get error to avoide the eror we need to apply double *
    print("Name:",name)
    print(data)
f1("Arati", salary=80000,designation="Data science")
```

```
Name: Arati
{'salary': 80000, 'designation': 'Data science'}
```

In [50]:
```
def f1(name,**data): # we get error to avoide the eror we need to apply double *
    print(type(name)) # string
    print(type(data)) # dictionary
f1("Arati", salary=80000,designation=" Data science")
```

```
<class 'str'>
<class 'dict'>
```

In [51]:
```
def f1(name,**data): # even we adding more data to it we can get access
    print("Name:",name)
    print(data)
f1("Arati", salary=80000,designation=" Data science", home_town="Odisha",educati
```

```
Name: Arati
{'salary': 80000, 'designation': ' Data science', 'home_town': 'Odisha', 'educati
on': 'B.Tech'}
```

In [53]:
```
# as except the first data all are turned to dictionary we can access by using f
def f1(name,**data):
    print("Name :",name)
    for i,j in data.items():
        print(i,":",j)
f1("Arati", Salary=80000,Designation=" Data science", Home_town="Odisha",Educati
```

```
Name : Arati
Salary : 80000
Designation :  Data science
Home_town : Odisha
Education : B.Tech
```

GLOBAL VARIBALE & LOCAL VARIABLE

Global variable

It's a variable which are declar out side the function is called global variable

The global variable we can use anywhere in the program both inside and out side the function

In [57]:
```
a=1 # global variable
def v1():
    print("The value inside function:",a) # same variable call inside the functi
    print("it works..")
v1() # calling the function
print()
print("The value Outside the function:",a) # same variable call outside the func
```

```
The value inside function: 1
it works..

The value Outside the function: 1
```

Local Variale

The local variable we can declar inside the function The scope the the variable in valid till the scope of the function only we can use the local variable inside the function outside the function we cannot accesss the local vaiable

In [58]:
```
def fun():
    a=4 # local variable
    print("The value inside function:",a) # same variable call inside the functi
    print("it works..")
v1() # calling the function
```

```
The value inside function: 1
it works..
```

In [60]:
```
def v1():
    b=453 # local variable
    print("The value inside function:-",a) # same variable call inside the funct
    print("it works..")
v1() # calling the function
print()
print("The value Outside the function:-",b) # same variable call outside the fun
# in this code we get error because we use the local variable inside the functio
```

```
The value inside function:- 1
it works..
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[60], line 7
      5 v1() # calling the function
      6 print()
----> 7 print("The value Outside the function:-",b)

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

In [61]:
```
a = 10

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

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 [62]:
```
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 [65]:
```
a = 10

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

print(b)
print(a)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[65], line 8
      5     b = 8
      6     print(a)
----> 8 print(b)
      9 print(a)

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

In [66]:
```
a = 10
```

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

print(a)
```

```
10
```

In [67]:
```
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 [68]:
```
a = 10

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

something()

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

# 1st preference is always local variable
```

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

In [69]:
```
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
```

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

In [70]:
```
a = 10

def something():
    b= 25
    #if we remove this variable then can befault it consider as global variable
    print('in function',b)

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

```
in function 25
out function 10
```

In [71]:
```
# 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)
    print('gloabl variable', a)

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

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

```
in function 15
gloabl variable 10
out function 10
```

In [72]:
```
import keyword
keyword.kwlist
```

Out[72]:
```
['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 [73]:
```
# if we used local & global in the same function this is not good idea thats wy

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)
```

```
140731635153624
140731635153624
in function 9
out function 10
```

global variable --

The global keyword is used inside a function to indicate that a variable refers to a global variable, not a local one.

In [74]:
```
x = 10  # Global variable

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

update_x()
print(x)  # Output: 15
```

```
15
```

globals -->

globals() is a built-in function that returns a dictionary representing the current global symbol table. It allows you to access and modify global variables programmatically.

In [75]:
```
x = 10  # Global variable

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

update_x()
print(x)  # Output: 15
```

```
15
```

In [76]:
```
# Global variable
x = 10
def modify_global():
```

```
    # Accessing the global variable
    print("Original x:", globals()['x'])
# Modifying the global variable
    globals()['x'] = 20
    print("Modified x:", globals()['x'])
modify_global()
# Accessing the modified global variable
print("Global x after modification:", x)
```

```
Original x: 10
Modified x: 20
Global x after modification: 20
```

Some program

1.Pass a list to the function count how many even and how many odd number in the list

In [78]:
```
def even_odd(list1):

    even=0
    odd=0
    for i in list:
        if i % 2 ==0:
            even +=1
        else:
            odd +=1

list=[1,2,45,66,54,56,78,99,0,33,43]
even_odd(list)
print("The even number :-",even)
print('The odd number:-',odd)
print("************************************")
print('The even number: {} \nThe odd number: {}'.format(even,odd)) # we can use
print("************************************")
print(f"The even number: {even} \nThe odd number: {odd}") # we can print using f
```

```
The even number :- 6
The odd number:- 5
************************************
The even number: 6
The odd number: 5
************************************
The even number: 6
The odd number: 5
```

FIBONACCI SERIES IN PYTHON

In [79]:
```
# in progammin we need to continue these process thats why we need to use loop h

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
```

FACTORIAL OF A NUMBER

In [80]:
```
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 & breakthro
```

```
120
```