WHAT IS FUNCTION?

- **A FUNCTION IS A REPEATED BLOCK OF CODE WHICH CONSIST OF BUSINESS LOGIC.**
- **THIS BLOCK OF CODE CAN BE CALLED "n" NUMBER OF TIMES BASED ON THE REQUIREMENT.**

> - **A FUNCTION IS A BLOCK OF COIDE THAT ONLY RUNS WHEN IT IS CALLED.**
> - **YOU CAN PASS DATA, KNOWN AS PARAMETERS, INTO A FUCTION.**
> - **IN SIMPLE WORD, FUNCTION IS A PARADIGM.**

WHAT IS PARADIGM?

- **IT IS A STYLE OR WAY TO CLASSIFY PROGRAMMING LANGUAGES. ### TYPES OF PARADIGM IN PYTHON:**

> 1. **PROCEDURAL PARADIGM**
> 2. **FUNCTIONAL PARADIGM**
> 3. **MODULAR PARADIGM**
> 4. **OBJECT ORIENTED PROGRAMMING STRUCTURE**
> 5. **LOGICAL PARADIGM**

WHY USE FUNCTION?

- **NO WASTAGE OF RESOURCES.**
- **ENHANCEMENT IS EASY.**
- **DEBUGGING IS EASY.**
- **TO REUSE THE CODE -- DEFINE THE CODE ONCE AND USE IT ANY TIMES.**

TYPES OF FUNCTION:

1. **BUILT-IN FUNCTIONS:**
   - **THESE FUNCTIONS ARE PRE-DEFINED BY PYTHON.**
   - **EG:-- print(), len(), input() ..**

1. **USER DEFINED FUNCTIONS:**
   - **THESE FUNCTIONS ARE DEFINED AND USED BY THE PROGRMMER AS PER PROJECT NEEDS.**

1. **RECURSIVE FUNCTION**
2. **LAMBDA FUNCTION**

COMPONENTS OF FUNCTION:

a. FUNCTION DEFINATION:

- **FUNCTION DECLARATION**
  - **IT IS FIRST LINE OF THE DEFINITION WITH** `"def"` **KEYWORD.**

- **FUNCTION IMPLEMENTATION**
  - **BLOCK OF CODE UNDER FUNCTION DEFINITION IS CALLED FUNCTION IMPLEMENTATION.**

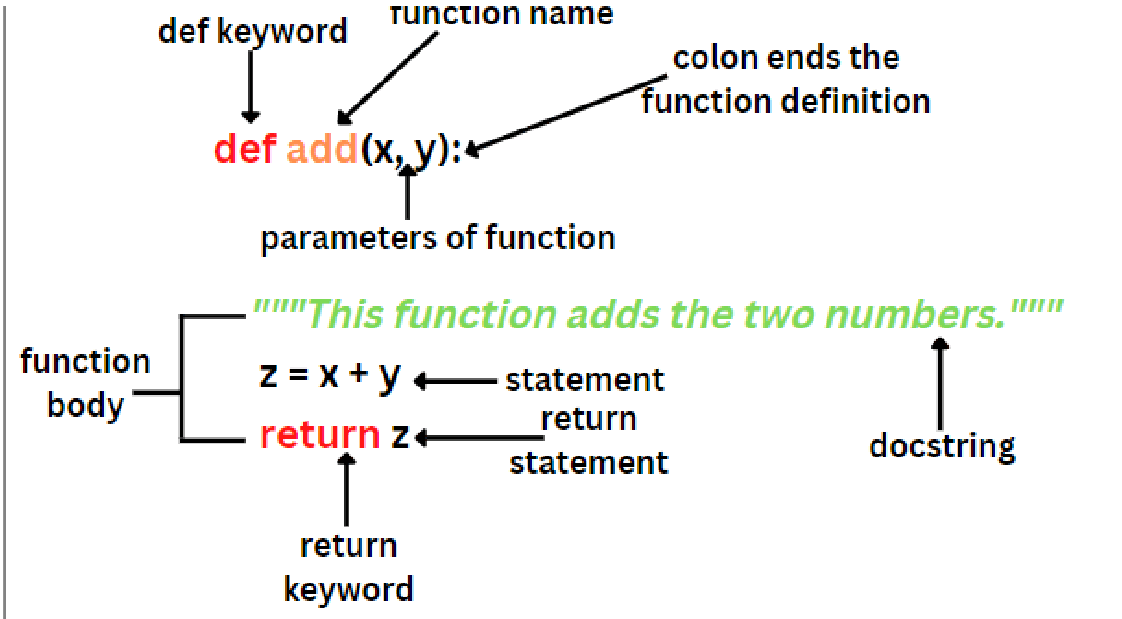


Fig: Function definition

- **FUNCTION DEFINITION SHOULD START WITH `"def"` KEYWORD FOLLOWED BY FUNCTION NAME WHICH IDENTIFIES THE FUNCTIO DEFINITION UNIQUELY.**
- **FUNCTION DEFINITION CAN EXIST WITHOUT CALLING.**
- **FUNCTION DEFINITION SHOULD BE DEFINED BEFORE FUNCTION CALLING.**
- **WE CAN DEFINE MULTIPLE FUNCTION DEFINITION WITHIN THE SAME PROGRAM.**
- **FUNCTION DEFINTION IS EXECUTED WHEN IT IS CALLED.**

b. FUNCTION CALLING:

- **FUNCTION NAME FOLLOWED BY PARANTHESIS IS CONSIDERED AS FUNCTION CALLING.**
- **FUNCTION CALLING SHOULD BE ALWAYS MENTIONED AFTER DEFINITION OF THE FUNCTION.**
- **FUNCTION CALLING CAN BE DONE "n" NUMBER OF TIME.**
- **FUNCTION CALLING CAN NOT EXIST WITHOUT FUNCTION DEFINITION, IT TRIGGERS `ERROR`.**

<function name>()

In [1]:

```
# FUNCTION DEFINATION
def fun_add():
    """
    THIS FUNCTION RETURNS THE ADDITION OF TWO NUMBERS.
    INPUT - ANY VALID INTEGER.
    OUTPUT - ADDITION OF TWO NUMBERS.
    """
    num1=int(input("Enter a Number: "))
    num2=int(input("Enter Second Number: "))
```

```
    a=num1+num2
    print(a)
```

In [2]:

```
# FUNCTION CALL
fun_add()
```

```
Enter a Number: 2
Enter Second Number: 3
5
```

HOW TO SEE THE DOCUMENTATION?

In [3]:

```
fun_add.__doc__
```

Out[3]:

```
'\n    THIS FUNCTION RETURNS THE ADDITION OF TWO NUMBERS.\n    INPUT - ANY VALID INTEGER.
\n    OUTPUT - ADDITION OF TWO NUMBERS.\n    '
```

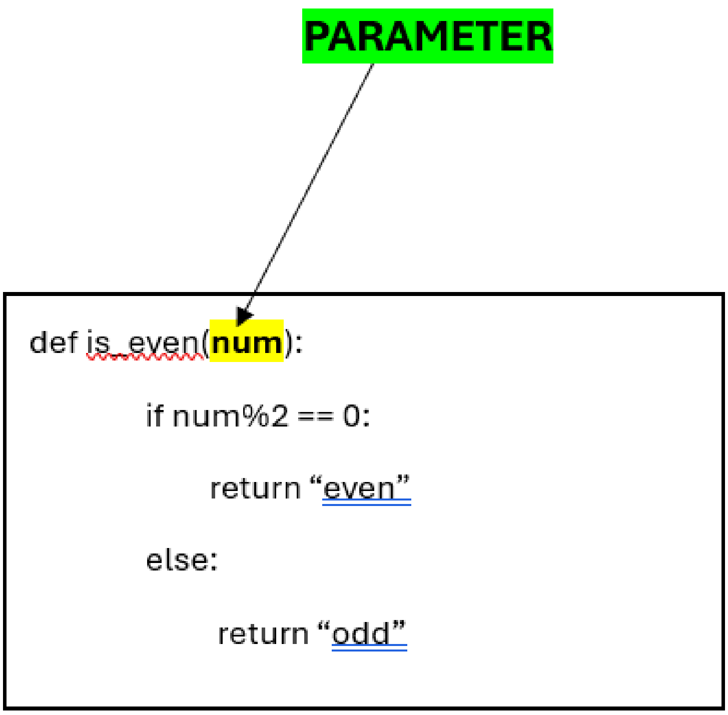
In [4]:

```
print(fun_add.__doc__)
```

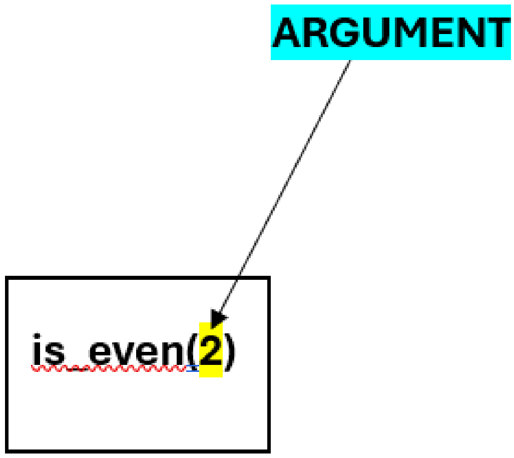
```
    THIS FUNCTION RETURNS THE ADDITION OF TWO NUMBERS.
    INPUT - ANY VALID INTEGER.
    OUTPUT - ADDITION OF TWO NUMBERS.
```

PARAMETER vs ARGUMENT

- **WHEN YOU CREATE A FUNCTION,WHATEVER YOU PASS INSIDE THE PARAENTESIS IS CALLED AS** `PARAMETER`**.**



- **WHEN YOU CALL A FUNCTION,WHATEVER YOU PASS INSIDE THE PARAENTESIS IS CALLED AS** `ARGUMENT`



TYPES OF ARGUMENTS:

a. DEFAULT ARGUMENTS:

- **INITIALIZING ARGUMENTS WITH ALTERNATIVE VALUES AT FUNCTION DEFINITION CALLED AS** `DEFAULT ARGUMENT`.

In [5]:

```
def power(a=1,b=1):
    return a**b
```

In [6]:

```
# eg-1
power()
```

Out[6]:

1

In [7]:

```
# eg-2
power(2,3)
```

Out[7]:

8

In [8]:

```
# eg-3
power(2)
```

Out[8]:

2

In [9]:

```
# eg-4
power(3)
```

Out[9]:

3

b. POSITIONAL ARGUMENTS:

- **VALUES ARE ASSIGNED TO ARGUMENT AT FUNCTION DEFINITION BASED ON POSITION IS CALLED** `POSITIONAL ARGUMENT`**.**

In [10]:

```
# eg:
def fun_add3(a,b,c):
    print(a)
    print(b)
    print(c)
    add=a+b+c
    print(add)

fun_add3(50,60,70)
```

```
50
60
70
180
```

c. KEYWORD ARGUMENTS:

- **INITIALIZING ARGUMENTS WITH RELEVANT VALUES AT FUNCTION CALLIMNG IS CALLED** `KEYWORD ARGUMENT`**.**
- **KEYWORD ARGUMENT SHOULD ALWAYS FOLLOW POSITIONAL ARGUMENT.**

In [11]:

```
def fun_add3(a,b,c):
    print(a)
    print(b)
    print(c)
    add=a+b+c
    print(add)

fun_add3(c=100,b=200,a=300)
```

```
300
200
100
600
```

d. VARIABLE-LENGTH ARGUMENTS (*args):

- **IT IS A SPECIAL TYPE OF VARIABLE WHICH ACCEPT "n" NUMBER OF POSITIONAL ARGUMENT.**
- **THESE VALUES ARE STORED IN THE FORM OF** `TUPLE`**.**

In [12]:

```
# eg-1
def fun_add3(*args):
```

```
        print(args)


fun_add3(50,60,70,80,90,60)
```

```
(50, 60, 70, 80, 90, 60)
```

In [13]:

```
# eg-2
# *args VALUES ARE STORED IN TUPLE.

def fun_add3(*a):
        print(a)


fun_add3(50,60,70,80,90,60,70,80,90,60,70,80,90)
```

```
(50, 60, 70, 80, 90, 60, 70, 80, 90, 60, 70, 80, 90)
```

In [14]:

```
# eg-3
def multiply(*args):
    prdd=1

    for i in args:
        prdd=prdd * i

    return prdd
```

In [15]:

```
multiply(1,2,3,4,5)
```

Out[15]:

```
120
```

e. KEYWORD VARIABLE-LENGTH ARGUMENTS (*kwargs):

- **IT IS SPECIAL TYPE OF VARIABLE WHICH ACCEPT "n" NUMBER OF KEYWORD ARGUMENTS.**
- **THESE KEYWORD ARGUMENTS ARE STORED IN THE FORM OF DICTIONARY.**

In [16]:

```
# eg-1
def fun_add3(**kwargs):

        print(kwargs)


fun_add3(x=10,x2=20,x3=30,x4=40)
```

```
{'x': 10, 'x2': 20, 'x3': 30, 'x4': 40}
```

POINTS TO REMEMBER WHILE USING __`*args & **kwargs`__

- **ORDER OF THE ARGUMENTS MATTER (normal --> `*args` --> `**kwargs`)**
- **THE WORDS "args" & "kwargs" ARE ONLY CONVENTION, YOU CAN USE ANY NAME OF YOUR CHOICE.**

In [17]:

```
# eg
def fun_add3(*args,**kwargs):
```

```
        print(args)
        print(kwargs)


fun_add3(50,60,70,80,90,60,x=10,x2=20,x3=30,x4=40)
```

```
(50, 60, 70, 80, 90, 60)
{'x': 10, 'x2': 20, 'x3': 30, 'x4': 40}
```

__`return`__ KEYWORD

- **A FUNCTION DEFINITION CAN RETURN A VALUE/VALUES TO THE FUNCTION CALLING PART USING `"return"` KEYWORD.**
- **THESE VALUES CAN BE COLLECT AT FUNCTION CALLING PLACE.**
- **RETURN SHOULD BE THE LAST LINE OF THE FUNCTION.**

In [18]:

```
# WITHOUT "return" KEYWORD

# eg-1
def is_even(num):
    if num %2 ==0:
        print("even")
    else:
        print("odd")

print(is_even(4))
```

```
even
None
```

In [19]:

```
# eg-2
l=[1,2,3]
print(l.append(4))
```

```
None
```

In [20]:

```
# eg-3
def fun_add():
    num1=1000
    num2=2000
    a=num1+num2


x=fun_add()
print(x)
```

```
None
```

In [21]:

```
# WITH "return" KEYWORD

def fun_add():
    num1=int(input("Enter a Number: "))
    num2=int(input("Enter Second Number: "))
    a=num1+num2
    return num1,num2,a

x=fun_add()
print(x)
```

```
Enter a Number: 2
```

```
Enter Second Number: 3
(2, 3, 5)
```

TYPES OF VARIABLES:

a. LOCAL VARIABLES:

- **VARIABLES WHICH ARE DECLARED WITHIN A FUNCTION & CAN BE USED WITHIN THE SAME FUNCTION ONLY.**
- **THESE VARIABLES CAN NOT BE USED OUTSIDE OF THE DECLARED FUNCTION.**

In [22]:

```python
# eg
def fun_add():


    num1=int(input("Enter a Number: "))
    num2=int(input("Enter Second Number: "))
    a=num1+num2
    print(a)

def fun_mul():

    b=num1*num2
    print(b)

fun_add()
print("---------------------------------------------------")
fun_mul()
```

```
Enter a Number: 2
Enter Second Number: 3
5
---------------------------------------------------

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14060\1505537858.py in <module>
     15 fun_add()
     16 print("---------------------------------------------------")
---> 17 fun_mul()

~\AppData\Local\Temp\ipykernel_14060\1505537858.py in fun_mul()
     10 def fun_mul():
     11 
---> 12     b=num1*num2
     13     print(b)
     14 

NameError: name 'num1' is not defined
```

> **NOTE: num1 & num2 IS DEFINED IN THE fun_add() FUNCTION, SO IT IS A LOCAL VARIABLE, SO IT IS NOT USED IN fun_mul() FUNCTION.**

a. GLOBAL VARIABLES:

- **GLOBAL VARIABLES WHICH CAN USED ANYWHERE WITHIN THE PROGRAM IS CALLED** `GLOBAL VARIABLE`**.**
- **GLOBAL VARIABLES ARE DECLARED IN TWO WAYS,**

    - **DECLARING A VARIABLE OUTSIDE AL THE FUNCTIONS IS CALLED/CONSIDERED AS**

- DECLARING A VARIABLE OUTSIDE AL THE FUNCTIONS IS CALLED/CONSIDERED AS `GLOBAL VARIABLE`.

- **A VARIABLE WHICH DECLARED WITH `"global"` KEYWORD INSIDE A FUNCTION IS ALOS CONSIDERED AS `GLOBAL VARIABLE`.**

In [23]:

```
# eg
# OUTSIDE OF ALL FUNCTION

num1=int(input("Enter a Number: "))
num2=int(input("Enter Second Number: "))

def fun_add():

    a=num1+num2
    print(a)

def fun_mul():

    b=num1*num2
    print(b)

fun_add()
print("----------------------------------------------------")
fun_mul()
```

```
Enter a Number: 2
Enter Second Number: 3
5
----------------------------------------------------
6
```

In [24]:

```
# eg
# USING "global" KEYWORD

def fun_add():
    global num1,num2

    num1=int(input("Enter a Number: "))
    num2=int(input("Enter Second Number: "))
    a=num1+num2
    print(a)

def fun_mul():

    b=num1*num2
    print(b)

fun_add()
print("----------------------------------------------------")
fun_mul()
```

```
Enter a Number: 2
Enter Second Number: 3
5
----------------------------------------------------
6
```

VARIABLE SCOPE

In [25]:

```
# eg-1
```

```
def g(y):
    print(x)
    print(x+1)
x=5
g(x)
print(5)
```

```
5
6
5
```

In [26]:

```
# eg-2

def f(y):
    x=1
    x +=1
    print(x)
x=5
f(x)
print(x)
```

```
2
5
```

In [27]:

```
# eg-3
# IF THERE IS NO VARIABLE INSIDE A FUNCTION, THEN THE FUNCTION
# USE THE GLOBAL VARIABLE BUT NOT CHANGE IT.

def h(y):
    x +=1
x=5
h(x)
print(x)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14060\4023663422.py in <module>
      6     x +=1
      7 x=5
----> 8 h(x)
      9 print(x)

~\AppData\Local\Temp\ipykernel_14060\4023663422.py in h(y)
      4 
      5 def h(y):
----> 6     x +=1
      7 x=5
      8 h(x)

UnboundLocalError: local variable 'x' referenced before assignment
```

In [28]:

```
# IF YOU WANT TO CHANGE IT, THEN USE "global" KEYWORD.
def h(y):
    global x
    x +=1
x=5
h(x)
print(x)
```

```
6
```

FUNCTION REFERENCE:

- **FUNCTION REFERENCE REFERS TO ADDRESS OF THE FUNCTION.**

- **WHICH EVER VARIABLE HOLDS THE ADDRESS OF THE FUNCTION IS AUTHORIZED TO CALL THE FUNCTION.**
- **THIS ADDRESS CAN BE PASSED FROM ONE VARIABLE TO ANOTHER VARIABLE.**

In [29]:

```
def fun_add():
    num1=1000
    num2=2000
    a=num1+num2
    print(a)


fun_add()
```

```
3000
```

In [30]:

```
fun_add
```

Out[30]:

```
<function __main__.fun_add()>
```

In [31]:

```
s=fun_add
```

In [32]:

```
s
```

Out[32]:

```
<function __main__.fun_add()>
```

In [33]:

```
# THE MEMORY LOCATION OF fun_add & s IS SAME.
```

NESTED FUNCTIONS:

- **NESTED FUNCTION CAN BE EXECUTED WITHIN THE PARENT FUNCTION.**
- **A FUNCTION WITHIN ANOTHER FUNCTION IS CALLED `NESTED FUNCTION`.**
- **THESE NESTED FUNCTION CAN NOT BE EXECUTED.**

In [34]:

```
def f():
    def g():
        print("Inside function g")
    g()
    print("Inside function f")
```

In [35]:

```
f()
```

```
Inside function g
Inside function f
```

In [36]:

```
g()   # INSIDE FUNCTION CAN NOT ACCESSED.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14060\1518445395.py in <module>
----> 1 g()  # INSIDE FUNCTION CAN NOT ACCESSED.

TypeError: g() missing 1 required positional argument: 'y'
```

CLOSURE:

- **PASSING ADDRESS OF NESTED FUNCTION TO MAIN PROGRAM TO MAIN PROGRAM & EXECUTING NESTED FUNCTION TO MAIN PROGRAM IS CALLED AS `CLOSURE PROPERTY`.**
- **FUNCTION REFERENCE, NESTED FUNCTION, CLOSURE PROPERTY --> DECORATOR**

RECURSION FUNCTION:

- **A FUNCTION WHICH CALLS ITSELF UNTIL A PARTICULAR CONDITION IS SATISFIED.**
- **CONDITION WHICH CONTROL THE FLOW OF EXECUTION IS CALLED `BASE CONDITION`.**

In [37]:

```python
# eg-1

def greet():
    print("Good Morning...Everyone")
greet()
```

```
Good Morning...Everyone
```

In [38]:

```python
# eg-2

def num(n):
    print(n)
    if n==10:
        return 0
    else:
        n=n+1
        num(n)
num(1)
```

```
1
2
3
4
5
6
7
8
9
10
```

In [39]:

```python
# eg-3

def fnum(num):
    print(num)
    if num==1:
        return 0
    else:
        num=num-1
        fnum(num)
```

```
fnum(10)
```

```
10
9
8
7
6
5
4
3
2
1
```

LAMBDA FUNCTION(ANONYMOUS FUNCTION):

- **LAMBDA FUNCTION IS CALLED ANONYMOUS FUNCTION.**
- **ANONYMOUS MEANS NOT IDENTIFY FOR THIS FUNCTION.**
- **IT IS AN ANONYMOUS INFINITE FUNCTION WHICH IS DEFINED WITH `"lamda"` KEYWORD & THIS FUNCTION ACCEPTS "n" NUMBER OF ARGUMENTS BUT RETURN ONLY ONE VALUE BASED ON THE EXPRESSION.**

PROPERTIES OF LAMBDA FUNCTION:

- **IT DOESN'T HAVE ANY NAME TO IDENTIFY.**
- **NO `"def"` KEYWORD IS USED TO DEFINE IT.**
- **IT ACCEPTS `"n"` NUMBER OF ARGUMENTS.**
- **IT RETURN ONLY ONE VALUE WITHOUT ANY `"return"` KEYWORD.**
- **IT IS AN `INLINE FUNCTION`.**
- **IT IS CONSIDERED AS LIGHT WEIGHT FUNCTION.**
- **ITS EXECUTION IS VERY FAST COMPARED TO THAT OF TRADITIONAL FUNCTION.**
- **IT IS SINGLE USE FUNCTION (NO REUSABILITY.)**

USE OF LAMBDA FUNCTION:

- **WE CAN INVOKE THE LAMBDA INTO ANOTHER PYTHON OBJECT LIKE (LIST, DICTIONARY ETC..)**
- **IT CAN ACT AS SOURCE OF INPUT TO HIGHER ORDER FUNCTIONS LIKE (MAP, FILTER, REDUCE)**

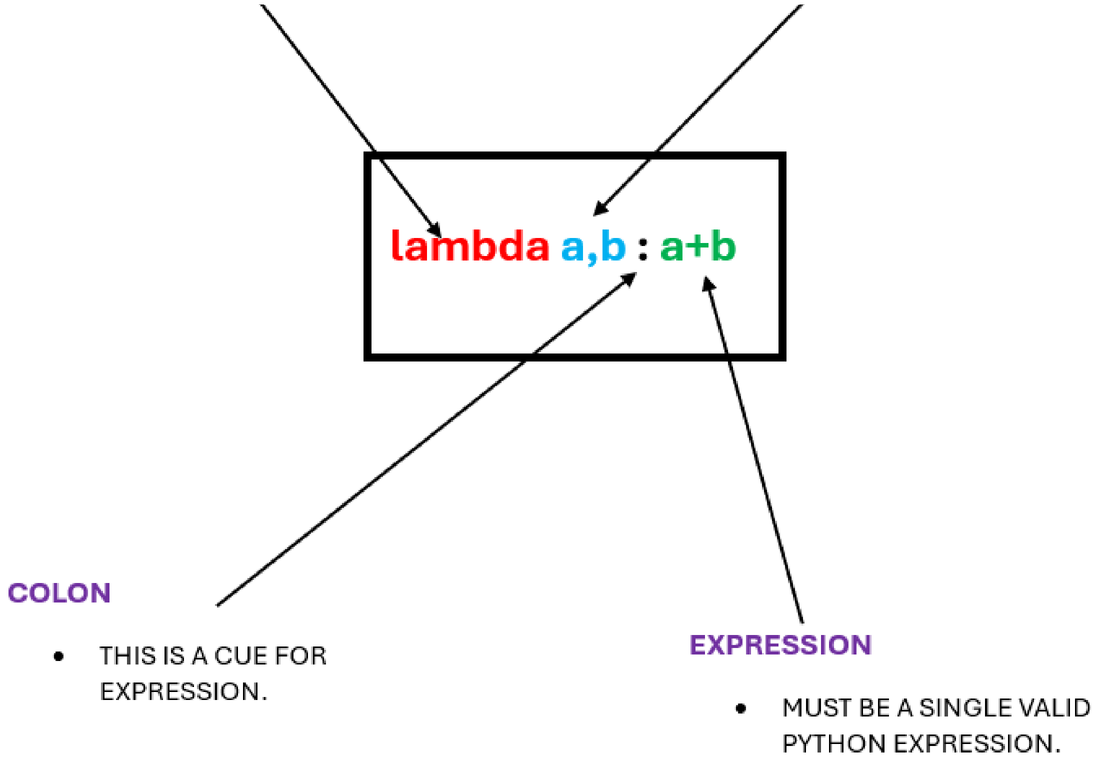
SYNTAX

PARAMETERS

- ONE OR MORE PARAMETERS ARE SUPPOTRTED.
- MUST BE SEPARATED BY A COMMA (,) & NO PARENTHESES

lambda KEYWORD

- CREATES THE LAMBDA FUNCTION



DIFFERENCE BETWEEN LAMBDA vs NORMAL FUNCTION

- **IT DO NOT HAVE ANY NAME.**
- **LAMBDA HAS NO RETURN VALUE.(TECHNICALLY IT RETURNS A FUNCTION)**
- **LMBDA IS WRITTEN IN ONE LINE.**
- **NO REUSABLE.**
- `LAMBDA FUNCTION IS USED WITH HIGHER ORDER FUNCTION.`

In [40]:

```
# eg-1
lambda x,y:x*y   # return the address
```

Out[40]:

```
<function __main__.<lambda>(x, y)>
```

In [41]:

```
# eg-2
# FIRST METHOD TO EXECUTE THE LAMBDA FUNCTION

r=lambda x,y:x*y
r(10,20)
```

Out[41]:

```
200
```

In [42]:

```
# eg-3
# SECOND METHOD TO EXECUTE THE LAMBDA FUNCTION

(lambda a,b:a+b)(20,30)
```

Out[42]:

```
50
```

In [43]:

```
# eg-4
#  LAMBDA WITHOUT ARGUMENTS

e=lambda : "Welcome to Lambda Function"
e()
```

Out[43]:

```
'Welcome to Lambda Function'
```

In [44]:

```
(lambda : "Welcome to Lambda Function")()
```

Out[44]:

```
'Welcome to Lambda Function'
```

In [45]:

```
# eg-5
# LAMBDA WITH ONE ARGUMENT

(lambda a:a**3)(3)
```

Out[45]:

```
27
```

In [46]:

```
# eg-6
# LAMBDA WITH TWO ARGUMENTS

(lambda x,y:x+y)(10,20)
```

Out[46]:

```
30
```

In [47]:

```
# eg-7
# LAMBDA WITH THREE ARGUMENTS

(lambda a,b,c:a+b+c)(20,30,40)
```

Out[47]:

```
90
```

In [48]:

```
c=(lambda a,b,c:a+b+c)
c(59,68,79)
```

Out[48]:

```
206
```

In [49]:

```
# eg-8
# LAMBDA IN LIST

nlist=[100,200,300]
nlist
```

Out[49]:

```
[100, 200, 300]
```

In [50]:

```
lambda_list=[lambda z: z**3, lambda a,b:a*b, lambda x,y,g:x+y+g]

print(lambda_list[2](10,20,30))
print("*********************************")
print(lambda_list[0](4))
print("*********************************")
print(lambda_list[1](40,50))
```

```
60
*********************************
64
*********************************
2000
```

In [51]:

```
# eg-9
# LAMBDA IN DICTIONARY

dlambda={

        "C":lambda z: z**3,
        "M":lambda a,b:a*b,
        "A":lambda x,y,g:x+y+g
}
```

In [52]:

```
dlambda["C"](5)
```

Out[52]:

```
125
```

In [53]:

```
dlambda["M"](20,10)
```

Out[53]:

```
200
```

In [54]:

```
dlambda["A"](200,100,500)
```

Out[54]:

```
800
```

LAMBDA WITH CONDITION

In [55]:

```
# eg-1

r=lambda m,n:m>n
r(66,83)
```

Out[55]:

```
False
```

In [56]:

```
# eg-2

r=lambda m,n:m<n
```

```
r(66,83)
```

Out[56]:

```
True
```

In [57]:

```
# eg-3
# if...else

r1=lambda m1,n1: m1 if m1>n1 else n1
r1(242,573)
```

Out[57]:

```
573
```

In [58]:

```
# eg-4
# odd or even

(lambda a: "even" if a%2==0 else "odd")(4)
```

Out[58]:

```
'even'
```

In [59]:

```
# eg-5

r2=lambda x,y,z: x if (x>=y) and (x>=z) else(y if y>=z else z)
r2(789,864,907)
```

Out[59]:

```
907
```

LAMBDA WITH FOR LOOP

In [60]:

```
# eg-1

r5=lambda n:  [i for i in range(1,n+1)]
r5(10)
```

Out[60]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [61]:

```
# eg-2

r6=lambda n:  [i*2  for i in range(1,n+1)  if i%2==0]
r6(10)
```

Out[61]:

```
[4, 8, 12, 16, 20]
```

NESTED LAMBDA

In [62]:

```
k=lambda x,y:x+y # Normal Lambda Function
k(10,80)
```

Out[62]:

Out[62]:

```
90
```

In [63]:

```
# eg-1

k1=lambda x: lambda y : x+y
k1(80)(40)
```

Out[63]:

```
120
```

In [64]:

```
# eg-2

k2=lambda x: lambda y,z : x+y+z
k2(80)(40,50)
```

Out[64]:

```
170
```

In [65]:

```
# eg-3

k3= lambda a: lambda b,c : lambda x,y,z : a+b+c+x+y+z
k3(34)(45,20)(67,78,89)
```

Out[65]:

```
333
```

HIGHER ORDER FUNCTION:

- **ANY FUNCTION WHICH ACCEPTS OTHER FUNCTION REFERENCE AS ARGUMENT IS CALLED** `HIGHER ORDER FUNCTION`.

a. MAP:

- **IT MAPS ELEMENTS OF ITERABLE WITH FUNCTION TO IMPLEMENT LOGIC.**
- **MAP IMPLEMENTS A BUSSINESS LOGIC TO EACH & EVERY ELEMENT OF THE ITERABLE.**
- **NO. OF INPUT ELEMENTS = NO. OF OUTPUT ELEMENTS.**

SYNTAX

map(<function> , <iterable>)

In [66]:

```
# eg-1

def sqr(n):   # Normal Function
    res=n**2
    return res

sqr(5)
```

Out[66]:

```
25
```

In [67]:

```
def sqr(n):
    res=n**2
    return res

nlist=[9,5,8,7,4,6]

for i in nlist:
    r=sqr(i)
    print(i,end=" :")
    print(r)
```

```
9 :81
5 :25
8 :64
7 :49
4 :16
6 :36
```

In [68]:

```
def sqr(n):
    res=n**2
    return res

nlist=[9,5,8,7,4,6]
l=[]
for i in nlist:
    r=sqr(i)

    l.append(r)
print(l)
```

```
[81, 25, 64, 49, 16, 36]
```

In [69]:

```
def sqr(n):
    res=n**2
    return res

nlist=[9,5,8,7,4,6]

m=map(sqr,nlist)
m
list(m)
```

Out[69]:

```
[81, 25, 64, 49, 16, 36]
```

In [70]:

```
list(map(sqr,[2,3,4,5]))
```

Out[70]:

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

MAP WTH LAMBDA

In [71]:

```
# eg-1

res=list(map(lambda x:x**2 ,   nlist))
res
```

Out[71]:

```
[81, 25, 64, 49, 16, 36]
```

In [72]:

```
res1=list(map(lambda x:x**2 ,   [9,5,8,7,4,6]))
res1
```

Out[72]:

```
[81, 25, 64, 49, 16, 36]
```

In [73]:

```
# eg-2

clist=["INDIA","USA","FRANCE","UK","JAPAN","UAE"]
```

In [74]:

```
list(map(lambda x:len(x)   , clist))
```

Out[74]:

```
[5, 3, 6, 2, 5, 3]
```

In [75]:

```
list(map(lambda x:len(x)>3   , clist))
```

Out[75]:

```
[True, False, True, False, True, False]
```

IMPLEMENTING MAP WITH LAMBDA ON DATA FRAME

In [76]:

```
# Create a Dictionary

std={
    "Name":["A","B","C","D"],
    "English":[60,80,85,65],
    "Science":[78,59,79,90]

}

std
```

Out[76]:

```
{'Name': ['A', 'B', 'C', 'D'],
 'English': [60, 80, 85, 65],
 'Science': [78, 59, 79, 90]}
```

In [77]:

```python
import pandas as pd  # Import pandas Library
```

In [78]:

```python
# Convert the Dictionary to Data Frame

s=pd.DataFrame(std)
s
```

Out[78]:

	Name	English	Science
0	A	60	78
1	B	80	59
2	C	85	79
3	D	65	90

In [79]:

```python
# Add a Column with Values

s["Computer"]=[69,89,76,95]
s
```

Out[79]:

	Name	English	Science	Computer
0	A	60	78	69
1	B	80	59	89
2	C	85	79	76
3	D	65	90	95

In [80]:

```python
# Add another column with values

s["History"]=80
s
```

Out[80]:

	Name	English	Science	Computer	History
0	A	60	78	69	80
1	B	80	59	89	80
2	C	85	79	76	80
3	D	65	90	95	80

In [81]:

```python
# Add total Column

s["Total"]=s["English"]+s["Science"]+s["Computer"]
s
```

Out[81]:

	Name	English	Science	Computer	History	Total
0	A	60	78	69	80	207
1	B	80	59	89	80	228

	Name	English	Science	Computer	History	Total
1	B	80	59	89	80	228
2	C	85	79	76	80	240
3	D	65	90	95	80	250

In [82]:

```
# Add average column

s["Average"]=s["Total"]/3
s
```

Out[82]:

	Name	English	Science	Computer	History	Total	Average
0	A	60	78	69	80	207	69.000000
1	B	80	59	89	80	228	76.000000
2	C	85	79	76	80	240	80.000000
3	D	65	90	95	80	250	83.333333

In [86]:

```
import numpy as np
```

In [87]:

```
# Round the Average column to 2 decimal places

s["Average"]=np.round(s["Average"],2)
s
```

Out[87]:

	Name	English	Science	Computer	History	Total	Average	Grade
0	A	60	78	69	80	207	69.00	Grade C
1	B	80	59	89	80	228	76.00	Grade B
2	C	85	79	76	80	240	80.00	Grade A
3	D	65	90	95	80	250	83.33	Grade A

In [88]:

```
# Final Data Frame

s
```

Out[88]:

	Name	English	Science	Computer	History	Total	Average	Grade
0	A	60	78	69	80	207	69.00	Grade C
1	B	80	59	89	80	228	76.00	Grade B
2	C	85	79	76	80	240	80.00	Grade A
3	D	65	90	95	80	250	83.33	Grade A

In [89]:

```
# eg-1

s["Grade"]=s["Average"].map(lambda x: "Grade A" if x>=80 else( "Grade B"  if x>=70 else
"Grade C"))
s["Grade"]
```

Out[89]:

```
0    Grade C
1    Grade B
2    Grade A
3    Grade A
Name: Grade, dtype: object
```

In [90]:

```
s
```

Out[90]:

	Name	English	Science	Computer	History	Total	Average	Grade
0	A	60	78	69	80	207	69.00	Grade C
1	B	80	59	89	80	228	76.00	Grade B
2	C	85	79	76	80	240	80.00	Grade A
3	D	65	90	95	80	250	83.33	Grade A

In [91]:

```
# eg-2

s["Average"].apply((lambda x: "Grade A" if x>=80 else( "Grade B"  if x>=70 else "Grade C
")))
```

Out[91]:

```
0    Grade C
1    Grade B
2    Grade A
3    Grade A
Name: Average, dtype: object
```

In [92]:

```
s["Grade2"]=s["Average"].apply((lambda x: "Grade A" if x>=80 else( "Grade B"  if x>=70 e
lse "Grade C")))
s["Grade2"]
```

Out[92]:

```
0    Grade C
1    Grade B
2    Grade A
3    Grade A
Name: Grade2, dtype: object
```

In [93]:

```
s
```

Out[93]:

	Name	English	Science	Computer	History	Total	Average	Grade	Grade2
0	A	60	78	69	80	207	69.00	Grade C	Grade C
1	B	80	59	89	80	228	76.00	Grade B	Grade B
2	C	85	79	76	80	240	80.00	Grade A	Grade A
3	D	65	90	95	80	250	83.33	Grade A	Grade A

b. FILTER

- **IT IMPLEMENTS CONDITIONS EACH & EVERY ELEMENTS OF ITERABLE & RETURN ONLY SATISFIED ELEMENTS.**
- **NO. OF OUTPUT ELEMENT EITHER EQUAL OR LESS THAN THE INPUT NUMBER BASED ON THE CONDITION.**

SYNTAX

filter(<function> , <iterable>)

In [94]:

```
# eg-1

clist=["INDIA","USA","FRANCE","UK","JAPAN","UAE"]

list(filter(lambda x:len(x)>3  , clist))
```

Out[94]:

```
['INDIA', 'FRANCE', 'JAPAN']
```

In [95]:

```
import numpy as np
arr=np.random.randint(50,150,(40))
arr
```

Out[95]:

```
array([ 79, 110, 145, 120, 107,  68, 137, 122,  71, 126,  62,  98, 127,
       126,  94,  87,  73, 121,  97, 107,  52, 118,  52,  98, 149,  96,
       118, 118,  95,  59,  87,  78,  89,  93,  99, 135,  98, 104,  89,
        64])
```

In [96]:

```
# eg-2

list(filter(lambda x:x, arr))
```

Out[96]:

```
[79,
 110,
 145,
 120,
 107,
 68,
 137,
 122,
 71,
 126,
 62,
 98,
 127,
 126,
 94,
```

```
 87,
 73,
 121,
 97,
 107,
 52,
 118,
 52,
 98,
 149,
 96,
 118,
 118,
 95,
 59,
 87,
 78,
 89,
 93,
 99,
 135,
 98,
 104,
 89,
 64]
```

In [97]:

```
# eg-3

list(filter(lambda x:x>=100, arr))
```

Out[97]:

```
[110,
 145,
 120,
 107,
 137,
 122,
 126,
 127,
 126,
 121,
 107,
 118,
 149,
 118,
 118,
 135,
 104]
```

In [98]:

```
# eg-4

list(filter(lambda x:x<=100, arr))
```

Out[98]:

```
[79,
 68,
 71,
 62,
 98,
 94,
 87,
 73,
 97,
 52,
 52,
 98,
 96
```

```
 96,
 95,
 59,
 87,
 78,
 89,
 93,
 99,
 98,
 89,
 64]
```

In [99]:

```
# eg-5

list(filter(lambda x:x%3==0, arr))
```

Out[99]:

```
[120, 126, 126, 87, 96, 87, 78, 93, 99, 135]
```

In [100]:

```
sttr="Good Morning Everyone. Welcome to Lambda Class"
```

In [101]:

```
v=["a","e","i","o","u"]
```

In [109]:

```
f=list(sttr)
f
```

Out[109]:

```
['G',
 'o',
 'o',
 'd',
 ' ',
 'M',
 'o',
 'r',
 'n',
 'i',
 'n',
 'g',
 ' ',
 'E',
 'v',
 'e',
 'r',
 'y',
 'o',
 'n',
 'e',
 '.',
 ' ',
 'W',
 'e',
 'l',
 'c',
 'o',
 'm',
 'e',
 ' ',
 't',
 'o',
 ' ',
 'L',
 'a',
```

```
 'm',
 'b',
 'd',
 'a',
 ' ',
 'C',
 'l',
 'a',
 's',
 's']
```

In [110]:

```python
# eg-6

list(filter(lambda s:s in v ,f))
```

Out[110]:

```
['o', 'o', 'o', 'i', 'e', 'o', 'e', 'e', 'o', 'e', 'o', 'a', 'a', 'a']
```

In [112]:

```python
# eg-7

set(list(filter(lambda s:s in v ,f)))
```

Out[112]:

```
{'a', 'e', 'i', 'o'}
```

c. REDUCE

- **IT IS USED TO REDUCE THE FINAL OUTPUT TO A SINGLE OUTPUT.**
- **IT IS A PART OF `functools` LIBRARY.**

SYNTAX

reduce(<function> , <iterable>)

In [113]:

```python
from functools import reduce     # IMPORT THE LIBRARY
```

In [114]:

```python
nlist
```

Out[114]:

```
[9, 5, 8, 7, 4, 6]
```

In [115]:

```
# eg-1

reduce(lambda x,y:x+y ,  nlist)
```

Out[115]:

```
39
```

In [116]:

```
# eg-2

reduce(lambda x,y:x*y ,  nlist)
```

Out[116]:

```
60480
```

In [117]:

```
# eg-3

n1=[9, 5,18, 8, 7, 4, 6]
reduce(lambda x,y: x  if x>y else y ,  n1)
```

Out[117]:

```
18
```

In [118]:

```
# eg-4

n1=[9, 5,18, 8, 7, 4, 6]
reduce(lambda x,y: x  if x<y else y ,  n1)
```

Out[118]:

```
4
```