Function is a block of code that perform specefic task and make our work easier dividng complex problem into small chunks.

Type of function:

- user defined function : this are built-in function, its already in python
- standard library function: we can built function according to our needs.

Function syntax:

```
def function_name(arguments):
    # function body

    return
```

Here,

- `def` - keyword used to declare a function
- `function_name` - any name given to the function
- `arguments` - any value passed to function
- `return` (optional) - returns value from a function

Function without using return

In [1]:
```
#example
# lets create function which will add two values

def addvalues(a,b):    #here we define function name "addvalues" with the help of keyword def and (a,b) is argu
    print(a+b)
    print("You passed two value and result is ", a+b)
```

In [2]:
```
addvalues(5,6)
```

```
11
You passed two value and result is  11
```

In [3]:
```
#what we see above, we created function and after that we did not need to type whole code again, we just pass v
```

In [4]:
```
#lets pass another value in same function
addvalues(8,9)
```

```
17
You passed two value and result is  17
```

In above example, we did not use return, so how we can use return or whats its use?

Function with return:

In [5]:
```
#first lets see similar type of example so that we can be clear about using it
#lets again create function which add 2 values:
def add_values(a,b):
    result=a+b              #here now, we have not decleared about result, so can we directly print result?
    print(result)
```

In [6]:
```
add_values(5,6)
```

```
11
```

In [7]:
```
#yes we are able to print. similarly lets use return, and when we put return in function, python will understan
#function is completed too.
def add_values(a,b):
    result=a+b
    return result
```

In [8]:
```
add_values(1,2)
```

Out[8]:
```
3
```

Standarad Library Function

Some function are already defined in function like square root, powere, to use this type of function we have to import module math. lets see example:

In [9]:
```
import math
#square_root is just variable to store result, we can give any name

square_root=math.sqrt(2)
```

In [10]:
```
print(square_root)
```

```
1.4142135623730951
```

In [11]:
```
#similarly, lets see example of power , power is also standard library function
#suppose lets say we want to do power of 2 by 3
a=pow(2,3)   #will this give result?
print(a)
```

```
8
```

In [12]:
```
#yes we did not keep math. like in sqrt so do we need to keep math.? lets see:
b=sqrt(4)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[12], line 2
      1 #yes we did not keep math. like in sqrt so do we need to keep math.? lets see:
----> 2 b=sqrt(4)

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

In [13]:
```
#Is it error because we did not keep math.?
b=math.sqrt(4)
```

In [14]:
```
print(b)
```

```
2.0
```

Python Function Arguments

In [15]:
```
#We have already see Python function arguments but, lets see in few more details again:
#lets create finction that multiply two numbers and add third one.

def my_function(a=10,b=2,c=3):   #here I defined function my_function and pass 3 arguments, already provided de
    result=(a*b)+c
    return(result)
```

In [16]:
```
#here we created function, now lets say we run it, but someone else want to give other values, can he or she? y
my_function(3,5,6)
```

Out[16]:
```
21
```

In [17]:
```
#if noone gives any value then that time only it return default values
my_function()
```

Out[17]:
```
23
```

In [19]:
```
# Next question? if we keep print also after return, will it run?
#lets create new function to see:
def subtract(a,b):
    if a>b:
        result=a-b
    else:
        result=b-a

    return (result)

    print("Function completed when we put return")
```

In [20]:
```
subtract(5,8)
```

Out[20]:
```
3
```

In [21]:
```
#no print will not work.
```

Python Fuction with arbitrary arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function. To handle this kind of situation, we

can use arbitrary arguments in Python.

In [22]:
```
#example
def addition(*numbers):
    result=0

    for i in numbers:
        result=result+i

    return(result)
```

here we did not know, how many arguments will be there so we cannot have exact body too like a+b or a+b+c+d, thatswhy we keep result=0 and use for loop.

In [23]:
```
addition(1,2,3,5,6)
```

Out[23]:
```
17
```

In [24]:
```
#can we use math function in above code, where we use for loop?
def try_add(*numbers):
    result=sum(numbers)
    return(result)
```

In [25]:
```
try_add(4,5,6,7,8)
```

Out[25]:
```
30
```

In [26]:
```
#yes it works and its easy than using for loop.
```

Python Recursion

When fuction call function. like in start we define function later, it will call inbuilt function too previous function we defined

In [29]:
```
#Example: find the factorial of number

def factorial(a):
    result=math.factorial(a)

    return(result)
```

In [30]:
```
#here we defined function factorail and it used factorial to find factorial of number
factorial(3)
```

Out[30]:
```
6
```

In [37]:
```
#lets practice one more question:

#suppose here is a list, now create function which only give even number as result:

def even_number(number):
    even=[]
    for i in number:
        if i%2==0:
            even.append(i)
    return(even)
```

In [38]:
```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]    #suppose this is question, provide only even from this list
```

In [39]:
```
output=even_number(numbers)
```

In [40]:
```
print(output)
```

```
[2, 4, 6, 8, 10]
```

In [41]:
```
#OR, we can directly provide to our function

even_number([8,16,22,33,11,9,6])
```

Out[41]:
```
[8, 16, 22, 6]
```

In [42]:
```
#Next question (more we practice more we became perfect and clear)
#create a function which separate even number and odd number

def separating_even_odd(number):
    even=[]
    odd=[]

    for i in number:
        if i%2==0:
            even.append(i)
        else:
```

```
            odd.append(i)

    return(even,odd)
```

In [43]:
```
separating_even_odd([7,77,11,2,6,8,88,45])
```

Out[43]:
```
([2, 6, 8, 88], [7, 77, 11, 45])
```

In [46]:
```
#OR
odd_number,even_number=separating_even_odd([7,77,11,2,6,8,88,45])
print(odd_number)
```

```
[2, 6, 8, 88]
```

Python variable Scope: Python has 3 variable according to scope:

1. Local variable
2. Global variable
3. nonlocal variable

In [50]:
```
#1 local variable is that variable which can only access inside function

def greet(message):
    message="Good morning"
    print(message)
```

In [52]:
```
greet('message')
```

```
Good morning
```

In [54]:
```
#but above is not local variable example, just showing simple function
def greet():
    message="hello"
    print(message)
```

In [55]:
```
greet()
```

```
hello
```

In [56]:
```
print(message)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[56], line 1
----> 1 print(message)

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

In [59]:
```
# its show not defined because we create message as local variable inside function.
#Now lets try Global variable:

message="hello"

def chor():
    print(message)
```

In [60]:
```
chor()
```

```
hello
```

In [61]:
```
#now lets just print message as above in local variable
print(message)
```

```
hello
```

In [62]:
```
#see it got printed because we define it above function or outside of function as gloabal variable
```

Now, lets practice some of user defined function using everything upto what we have learn, lets try to learn and Practice.

1. Write a program to create a function that takes two arguments, name and age, and print their value.

In [3]:
```
def aboutyou(name,age):
    print(f"your name is {name}")
    print("your age is", age)
```

In [4]:
```
aboutyou("Ram", 21)
```

```
your name is Ram
your age is 21
```

1. Write a program to create function func1() to accept a variable length of arguments and print their value.

Note: Create a function in such a way that we can pass any number of arguments to this function, and the function should process them and display each argument's value.

In [5]:
```
def addition(*numbers):
    result=sum(numbers)

    return(result)
```

In [6]:
```
addition(5,6,7,8,9)
```

Out[6]:
```
35
```

In [7]:
```
#OR
def values(*numbers):
    for i in numbers:
        print(i)
```

In [8]:
```
values(12,13,14,77)
```

```
12
13
14
77
```

In [9]:
```
#0R, we can do it without for loop too
def printout(*numbers):
    print(numbers)
```

In [10]:
```
printout(1,2,3)
```

```
(1, 2, 3)
```

The function which accepts many arguments or we are unknown how many arguments will be there, in that case we put *, this type of function is called function with arbitrary arguments.

1. Write a program to create function calculation() such that it can accept two variables and calculate addition and subtraction. Also, it must return both addition and subtraction in a single return call.

In [12]:
```
def calculation(a=6,b=7): #lets try default values too
    add=a+b
    sub=b-a

    return(add,sub)
```

In [13]:
```
calculation(10,11)   #we override default value
```

Out[13]:
```
(21, 1)
```

In [14]:
```
#lets not keep any values, it automatically used default value
calculation()
```

Out[14]:
```
(13, 1)
```

In [17]:
```
#OR, let do more in detail and with clear calculation
def calculation(a,b):
    addition=a+b
    if a>b:
        subtraction=a-b
    else:
        subtraction=b-a

    print("addition is: ", addition)
    print("subtraction is: ", subtraction)
```

In [18]:
```
calculation(10,4)
```

```
addition is:  14
subtraction is:  6
```

1. Write a program to create a function show_employee() using the following conditions.

- It should accept the employee's name and salary and display both.
- If the salary is missing in the function call then assign default value 9000 to salary

In [19]:
```
def show_employee(name,salary=9000):
    print("Name of employee is ", name, "and salary is ", salary)
```

In [20]:
```
show_employee("Ram",20000)
```

```
Name of employee is  Ram and salary is  20000
```

In [21]:
```
show_employee("shyam")
```

```
Name of employee is  shyam and salary is  9000
```

1. Create Recursive function: Write a program to create a recursive function to calculate the sum of numbers from 0 to 10.

A recursive function is a function that calls itself again and again.

In [1]:
```
import math

def fsum(*a):
    result=math.fsum(a)
    return(result)
```

In [3]:
```
fsum(0,1,2,3,4,5,6,7,8,9,10)
```

Out[3]:
```
55.0
```

In [1]:
```
#OR
def addition(n):
    if n==0:
        return 0
    else:
        return n + addition(n-1)
```

In [2]:
```
result= addition(10)
```

In [3]:
```
print(result)
```

```
55
```

In [4]:
```
# This above function known as recurssive which run again and again.
```

1. Assign a different name to function and call it through the new name

- Below is the function display_student(name, age). Assign a new name show_student(name, age) to it and call it using the new name.

In [7]:
```
def display_name(name,age):
    return(name,age)

#assigning new name
show_student=display_name
```

In [8]:
```
display_name("roshan",21)
```

Out[8]:
```
('roshan', 21)
```

In [9]:
```
#will show_student also show?
show_student("Ram",54)
```

Out[9]:
```
('Ram', 54)
```

1. Generate a Python list of all the even numbers between 4 to 30

In [13]:
```
 #first of all just create function which will separate even number: create general function
```

In [14]:
```
def even_num(numbers):
    even=[]
    for i in numbers:
        if i%2==0:
            even.append(i)

    print(even)
```

In [15]:
```
a=range(4,31)
```

In [17]:
```
print(a)
```

```
range(4, 31)
```

In [18]:
```
a=list(a)
```

In [19]:
```
even_num(a)
```

```
[4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
```

1. Find the largest from given list: x = [4, 6, 8, 24, 12, 2]

No need to use user defined function, we can use standard library function

In [25]:
```
x = [4, 6, 8, 24, 12, 2]
```

In [27]:
```
print(max(x))
```

```
24
```

In [ ]: