CH - 3: Functions, Scoping And Abstraction

Why functions ?

- Reusability
- Concise code
- Modularity

Defining Function

In [1]:
```
def greet():
    print("Hello python")
```

In [2]:
```
print(greet()) # Here no any value Return so o/p is none
```

```
Hello python
None
```

Function specifications

- It specifies the function name, parameters and the retuen type

In [3]:
```
def add(x,y):
    """
    This function return addition of x and y
    Parameters:
    x(int) : the first Number
    y(int) : the second number

    Returns:
    int : The addition Of x And y
    """
    result = x + y
    return result
```

In [4]:
```
print(add(4,5)) # here retun 9
```

```
9
```

Types of Functions

dir(**builtins**) # a list containing inbuilt functions

User Defined Function

1. No parameters , no returns

In [5]:
```
def printline():
    print("Hello Jp")
printline()
```

```
Hello Jp
```

2. No Parameters, with return

In [6]:
```
def printline():
    return 'Hello Jp'
printline()
```

Out[6]:
```
'Hello Jp'
```

3. With parameters, no return

In [7]:
```
def printline(s):
    print(s)

m = input("Enter Greeting")
printline(m)
```

```
Enter GreetingJp
Jp
```

4. With parameters, With return

In [8]:
```
def printline(s):
    return s
    print("Will this run?") # this won't be executed

m = input("enter greeting - ")
printline(m)
```

```
enter greeting - P@nchal
```

Out[8]:
```
'P@nchal'
```

Returning Multiple values

In [9]:
```
def sumsub(x,y):
    sum = x + y
    sub = x - y
    return sum,sub

m,n =sumsub(7,6)
print(type(m))
print(m)
```

```
<class 'int'>
13
```

In [10]:
```
a = 7,6
type(a)
```

Out[10]:
```
tuple
```

In [11]:
```
a
```

Out[11]:
```
(7, 6)
```

WAP to return Addition, Subtraction, multiplication and division of given two numbers using Function

In [12]:
```
def cal(x,y):
    return (f"Addition={x+y},Subtraction = {x-y},Multipication = {x*y}, Division

cal(1,1)
```

Out[12]:
```
'Addition=2,Subtraction = 0,Multipication = 1, Division = 1.0'
```

PARAMETERS AND ARGUMENTS

- The value in parameters used while defining function are called parameters
- The values passed while calling the function are the Argumennts

Types of Arguments

Default argument

In [13]:
```
def square(x = 20):
    return x*x

print(square())
print(square(10)) # 10 overwrite to 20
```

```
400
100
```

Position arguments

- The number of Arguments and their postions must match
- if we change the order of arguments,the result will way
- if we change the number of arguments, we will get Error

In [14]:
```
def sub(x,y):
    return x - y

print(sub(5,6))
print(sub(6,5))
print(sub(5,6,7)) # Error
```

```
-1
1
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-d0738194e7b2> in <module>
      4 print(sub(5,6))
      5 print(sub(6,5))
----> 6 print(sub(5,6,7)) # Error

TypeError: sub() takes 2 positional arguments but 3 were given
```

Keyword Argument

- In case of all keyword arguments, the order does not matter.
- One can use combination of keyword and positional arguments
- Keyword argument always follows positional argument

In [ ]:
```
def wish(name,msg):
    print("hello",name , msg)
```

In [ ]:
```
wish(name = 'python', msg = 'good Morning')
```

In [ ]:
```
wish( msg = 'good Morning',name = 'python',)
```

In [ ]:
```
wish("c++", msg = "Good Morning")
```

In [ ]:
```
wish("c++", name = "good morning")
```

In [ ]:
```
wish(msg = "Good Mornig", "c++") # (Keyword Argument , positional Argument)
```

Variable length arguments

In [ ]:
```
def sum(*n):
    total = 0
    for i in n:
        total+=i
        print("The sum is ",total)
```

In [ ]:
```
sum(10)
```

In [ ]:
```
sum(10,20)
```

In [ ]:
```
sum(10,-10)
```

In [ ]:
```
sum(10,20,30)
```

FUNCTION SCOPE

Local Variable

- a variable declared inside function has started execution and are lost when the function terminates

In [ ]:
```
x = 5

def fun():
    x = 100 # local variable
    print(x)
```

In [ ]:
```
fun()
```

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

Global keyword

In [16]:
```
x = 5

def fun():
    global x
    x = 100 # local variable
    print(x)
```

In [17]:
```
fun()
```

```
100
```

In [18]:
```
print(x)
```

```
100
```

SCOPING RULE

- LEGb RULE

1. Local
2. Enclosed
3. Global
4. Builtin

Nested Functions

In [29]:
```
def f():
    def g():
        print("inside g function")
    print("inside f function")
```

In [30]:
```
f()
```

```
inside f function
```

In [31]:
```
g()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-31-5fd69ddb5074> in <module>
----> 1 g()

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

In [33]:
```
def f():
    def g():
        print("inside g function")
    g()
    print("inside f function")
```

In [34]:
```
f()
```

```
inside g function
inside f function
```

In [35]:
```
def g(x):
    def h():
        x = 'abc'
        return x
    x = x+1
    print("in g function x is",x)
    print(h())
    return x
```

In [36]:
```
x = 3
z = g(x)
print(z)
```

```
in g function x is 4
abc
4
```

Duplicate Functions

In [40]:
```
def add(x,y):
    return x+y

def add(x,y): # here 1 st method will go in garbage so this same name same paaram
    return x - y
```

In [41]:
```
print(add(8,9))
```

```
-1
```

In [ ]:
```

```