Python Functions

- Function is a group of related statements that perform a specific task.
- A function is a set of statements that take inputs, do some specific computation and produces output.
- It avoids repetition and makes code reusable.
- If we use functions written by others in the form of library, it can be termed as library functions.

Types Of Functions

1. InBuilt Functions
2. User-defined Functions

Built-in Functions

In [1]:
```
#### abs() ---> find the absolute value

num = -100
abs(num)
```

Out[1]:
```
100
```

User-defined Functions

- Functions that we define ourselves to do certain specific task are referred as user-defined functions

Advantages

1. User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
3. Programmars working on large project can divide the workload by making different functions.

Syntax:

```
def function_name(parameters):

    """
    Doc String
    """
    function body (statements)

    return [expression]
```

1. keyword "def" marks the start of function header
2. Parameters (arguments) through which we pass values to a function. These are optional
3. A colon(:) to mark the end of funciton header
4. Doc string describe what the function does. This is optional
5. "return" statement to return a value from the function. This is optional

return

- The return statement is used to exit a function and go back to the place from where it was called.
- return statement can contain an expression which gets evaluated and the value is returned.
- if there is no expression in the statement or the return statement itself is not present inside a function, then the function will return None Object

In [2]:
```
a,b,c,d=3,4,8,9
print(a+b+c+d)
```

```
24
```

In [3]:
```
a,b,c,d=13,24,8,9
print(a+b+c+d)
```

```
54
```

In [4]:
```
# Define the function --> a,bc,d are called as parameters or arguments

def sum_4(a,b,c,d):
    z=a+b+c+d
    return z
```

In [5]:
```
# Function Call --> Once we have defined a function, we can call it from anywh
ere

sum_4(3,4,8,9)
```

Out[5]:
```
24
```

In [6]:
```
sum_4(13,24,8,9)
```

Out[6]:
```
54
```

In [7]:
```
def double(num):
    r=2*num
    return r

double(10)
```

Out[7]:
```
20
```

In [8]:
```
for num in range(1,100):
    if num%5 == 0 and num%7==0:
        print(num)
```

```
35
70
```

In [9]:
```
for num in range(1,200):
    if num%5 == 0 and num%7==0:
        print(num)
```

```
35
70
105
140
175
```

In [10]:
```
for num in range(1,300):
    if num%5 == 0 and num%7==0:
        print(num)
```

```
35
70
105
140
175
210
245
280
```

In [11]:
```
def num_5_7(i):
    for num in range(1,i):
        if num%5 == 0 and num%7==0:
            print(num)
```

In [12]:
```
num_5_7(100)
```

```
35
70
```

In [13]:
```
num_5_7(200)
```

```
35
70
105
140
175
```

In [14]:
```
def pre(x):
    y=x+2
    print(y)
```

In [15]:
```
pre(2)
```

```
4
```

In [16]:
```
pre(125)
```

```
127
```

In [17]:
```
pre("siva")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-426bb8279fa1> in <module>
----> 1 pre("siva")

<ipython-input-14-b193d8f0b882> in pre(x)
      1 def pre(x):
----> 2     y=x+2
      3     print(y)

TypeError: can only concatenate str (not "int") to str
```

In [18]:
```
a=[1,2,3,4]
sum = 0
for num in a:
    sum = sum +num
print(sum)
```

```
10
```

In [19]:
```
# Define the function
def get_sum(a):
    sum = 0
    for num in a:
        sum = sum+num
    return sum

# Function Call
get_sum([1,2,3,4])
```

Out[19]:
```
10
```

In [20]:
```
lst=[8,7,5,4,2,10,50,9]

even_sum=0

odd_sum=0

for i in lst:
    if i%2==0:
        even_sum=even_sum+i
    else:
        odd_sum=odd_sum+i

print(even_sum,odd_sum)
```

```
74 21
```

In [21]:
```
# Define the function
def f_n(l):
    even_sum=0
    odd_sum=0
    for i in l:
        if i%2==0:
            even_sum=even_sum+i
        else:
            odd_sum=odd_sum+i

    print(even_sum,odd_sum)

# Function Call
f_n([8,7,5,4,2,10,50,9])
```

```
74 21
```

Different Forms of Arguments

Function with No Arguments

In [22]:
```
def even_num():
    lb=int(input('enter lower bound values:'))
    ub=int(input('enter upper bound values:'))
    for i in range(lb,ub+1):
        if i%2==0:
            print(i)
```

In [24]:
```
even_num()
```

```
enter lower bound values:10
enter upper bound values:20
10
12
14
16
18
20
```

1. Positional Arguments

In [25]:
```
def greet(name, msg):
    print("Hello", name, msg)
```

In [26]:
```
#call the function with arguments
greet("good morning","satish")
```

```
Hello good morning satish
```

In [27]:
```
#suppose if we pass one argument
greet("satish") #will get an error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-27-8f7ad29300cf> in <module>
      1 #suppose if we pass one argument
----> 2 greet("satish") #will get an error

TypeError: greet() missing 1 required positional argument: 'msg'
```

2. Default Arguments

We can provide a default value to an argument by using the assignment operator (=).

In [28]:
```
def greet(name, msg="Good Morning"):
    print("Hello", name, msg)
```

In [29]:
```
greet("satish", "Good Night")
```

```
Hello satish Good Night
```

In [30]:
```
#with out msg argument
greet("satish")
```

```
Hello satish Good Morning
```

3. Keyword Arguments

In [31]:
```
def greet(name, msg):
    print("Hello", name, msg)
```

In [32]:
```
#call the function with arguments
greet(msg="satish",name="good morning")
```

```
Hello good morning satish
```

In [33]:
```
greet(msg="Good Morning", name="good")
```

```
Hello good Good Morning
```

Example

In [34]:
```
print(1,2,3)  # here, "sep" acts as default argument
```

```
1 2 3
```

In [35]:
```
print(1,2,3,sep=",") # here,"sep"  its keyword argument
```

```
1,2,3
```

4. Arbitary Arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.

In [36]:
```
def greet(*names):
    for name in names:
        print("Hello, {} ".format(name))
```

In [37]:
```
greet("satish", "murali")
```

```
Hello, satish
Hello, murali
```

Recussive Function or Recurison

We know that in Python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.

In [38]:
```
#python program to print factorial of a number using recurion

def fact(num):
    if num<=0:
        return 1
    else:
        return num * fact(num-1)
```

In [39]:
```
fact(5)
```

Out[39]:
```
120
```

In [40]:
```
import math
math.factorial(5)
```

Out[40]:
```
120
```

Python program to make a simple calculator that can add, subtract, multiply and division

In [42]:
```
def add(a, b):
    return a+b

def multiply(a, b):
    return a*b

def subtract(a, b):
    return a-b

def division(a, b):
    return a/b

print("Select Option")
print("1. Addition")
print("2. Subtraction")
print("3. Multiplication")
print("4. Division")

#take input from user
choice = int(input("Enter choice 1/2/3/4:  "))

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number:"))
if choice == 1:
    print("Addition of {} and {} is {}".format(num1, num2, add(num1, num2)))
elif choice == 2:
    print("Subtraction of {} and {} is {}".format(num1, num2, subtract(num1, n
um2)))
elif choice == 3:
    print("Multiplication of {} and {} is {}".format(num1, num2, multiply(num1
, num2)))
elif choice == 4:
    print("Division of {} and {} is {}".format(num1, num2, division(num1, num2
)))
else:
    print("Invalid Choice")
```

```
Select Option
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter choice 1/2/3/4:  1
Enter first number: 10
Enter second number:20
Addition of 10 and 20 is 30
```