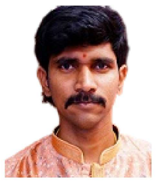


Functions

- 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

Inbuilt Functions

In [1]:

```
print(10)
```

```
10
```

In [2]:

```
abs(-2)         #returns absolute value
```

Out[2]:

```
2
```

User-defined Functions

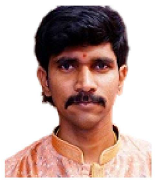
- Functions that we define ourselves to do certain specific task are referred as user-defined functions
- User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
- If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
- 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 [3]:

```
#defining a function with doc string

def square_this(x):
    '''
    this function will return the square of the given number
    '''
    return x**2
```

In [4]:

```
# call the function
square_this(3)
```

Out[4]:

```
9
```

In [5]:

```
# Defining a function with 1 argument
def print_this(x):
    print(x)

#calling a function with 1 argument
print_this("srk")
```

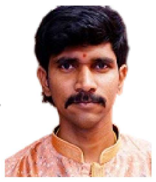
```
srk
```

In [6]:

```
#defining a function with 2 arguments

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

#calling a function with 2 arguments
add(3,9)
```

```
12
```

In [7]:



```
#defining a function with no argument

def print_text():
    print('this is text')


#calling a function with no argument
print_text()
```

```
this is text
```

In [8]:

```
#defining a function which returns return two values

def min_max(num):
    a=min(num)
    b=max(num)
    return a,b


# call the function
a,b = min_max([1, 2, 3])
print(a,b)
```

```
1 3
```

Different types of Arguments

Function with No Arguments

In [9]:

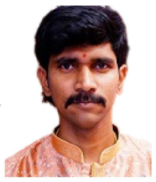
```
#Defining a function
def greet():
    print("Hello","SRK","good Morning")

# Call a function
greet()
```

```
Hello SRK good Morning
```

1. Positional Arguments

- the number of arguments while defining a function & calling a function should be same. Otherwise, it throws error
- order is preserved
- its executes in the same order

In [10]:



```
#Defining a function
def greet(name, msg):
    print("Hello",name,msg)

#call the function
greet("srk","good morning")
greet("morning","srk")
greet(1,2)
```

```
Hello srk good morning
Hello morning srk
Hello 1 2
```

In [11]:

```
#suppose if we pass one argument
greet("morning")                    #will get an error
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[11], line 2
      1 #suppose if we pass one argument
----> 2 greet("morning")

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

2. Default Arguments

While creating a function, we are asssigning a default value to an argument by using the assignment operator (=).

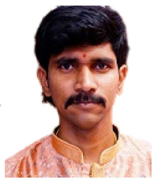
- if you don't assign value for default arguments, it will automatically selects the default value
- if you assign the value for default argument, it will consider that value which you have assigned

In [12]:

```
#Defining a function
def greet_1(name, msg="Good Morning"):
    print("Hello",name, msg)


#call the function
greet_1("srk","good night")
greet_1("srk")
```

```
Hello srk good night
Hello srk Good Morning
```

3.keyword Arguments



In [13]:

```
#Defining a function
def greet(name, msg):
    print("Hello", name, msg)
```

In [14]:

```
greet("Good morning","satish")            #positional arguments
```

```
Hello Good morning satish
```

In [15]:

```
greet(msg="Good Morning", name="satish")   #keyword arguments
```

```
Hello satish Good Morning
```

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.

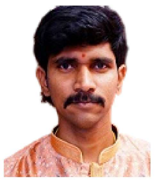
In [16]:

```
#Defining a function
def greet(*n):
    for i in n:
        print("Hello",i)

#call the function
greet(1,2,3,4)
```

```
Hello 1
Hello 2
Hello 3
Hello 4
```

Recussive Function or Recurison

- 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 [17]:

```
def factorial(x):
    if type(x)== int:
        if x==0:
            return 1
        elif x>0:
            return x*factorial(x-1)
    else:
        print("value error")


#call the function
factorial(5)
```

Out[17]:

```
120
```

Anonymous (Lambda) Functions

- Primarily used to temporarily define a function for use by another function

In [18]:

```
# define a function the "usual" way
def squared(x):
    return x**2
```

In [19]:

```
# define an identical function using lambda
squared = lambda x: x**2
```

In [20]:

```
squared(2)
```

Out[20]:

```
4
```