In Python, the function is a block of code defined with a name. We use functions whenever we need to perform the same task multiple times without writing the same code again. It can take arguments and returns the value.

Function improves efficiency and reduces errors because of the reusability of a code. Once we create a function, we can call it anywhere and anytime. The benefit of using a function is reusability and modularity.

Types of Functions Python support two types of functions

1. Built-in function (range(), id(), type(), input(), eval() etc.)
2. User-defined function (Functions which are created by programmer explicitly according to the requirementare called a user-defined function.)

Creating a Function

Use the following steps to to define a function in Python.

1. Use the def keyword with the function name to define a function.
2. Next, pass the number of parameters as per your requirement. (Optional).
3. Next, define the function body with a block of code. This block of code is nothing but the action you wanted to perform.

In Python, no need to specify curly braces for the function body. The only indentation is essential to separate code blocks. Otherwise, you will get an error. While defining a function, we use two keywords, def (mandatory) and return (optional)

In [36]:

```
print("This is python batch")
print('batch consist of 4 students')
print(['Kunal', 'Sharvari','Kanisha','Prajakta'])
```

```
This is python batch
batch consist of 4 students
['Kunal', 'Sharvari', 'Kanisha', 'Prajakta']
```

In [37]:

```
def Python():

    print("This is python batch")
    print('batch consist of 4 students')
    print(['Kunal', 'Sharvari','Kanisha','Prajakta'])
```

In [38]:

```
Python()
```

```
This is python batch
batch consist of 4 students
['Kunal', 'Sharvari', 'Kanisha', 'Prajakta']
```

In [39]:

```
#area of rectangle
length = float(input("Enter the length"))
breadth = float(input("enter the breadth"))
area = length * breadth
print(area)
```

```
Enter the length12
enter the breadth8
96.0
```

In [40]:

```
def area(length=12, breadth=8):#parameter
    return ("Area of rectangle is =", length * breadth)

area()#arguement
```

Out[40]:

```
('Area of rectangle is =', 96)
```

In [41]:

```
area(23)
```

Out[41]:

```
('Area of rectangle is =', 184)
```

Parameter Vs Arguement ?

Generally when people say parameter/argument they mean the same thing, but the main difference between them is that the parameter is what is declared in the function, while an argument is what is passed through when calling the function.

In [42]:

```
#Here, the parameters are a and b, and the arguments being passed through are 5 and 4.
def add(a, b):
     return a+b
add(5, 4)
```

Out[42]:

```
9
```

In [ ]:

In [43]:

```
def course_func(name, course_name):
     print("Hello", name, "Welcome to DeveLearn")
     print("Your course name is", course_name)
# call function
course_func('Students', 'Python')
```

```
Hello Students Welcome to DeveLearn
Your course name is Python
```

In [44]:

```
def my_function(fname, lname):
    print(fname + " " + lname)
my_function("Sanvee")
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp/ipykernel_9404/784823913.py in <module>
      1 def my_function(fname, lname):
      2     print(fname + " " + lname)
----> 3 my_function("Sanvee")

TypeError: my_function() missing 1 required positional argument: 'lname'
```

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

In [45]:

```
def my_function(*kids):
 print("The youngest child is " + kids[2])
my_function("a", "b", "c")
```

```
The youngest child is c
```

In [46]:

```
#You can also send arguments with the key = value syntax.
def my_function(child3, child2, child1):
 print("The youngest child is " + child3)
my_function(child1 = "a", child2 = "b", child3 = "c")
```

```
The youngest child is c
```

Arbitrary Keyword Arguments, **kwargs If you do not know how many keyword arguments that will be passed into your function, add two asterisk:* * before the parameter name in the function definition.

In [47]:

```
def my_function(**girl):
 print("His last name is " + girl["lname"])
my_function(fname = "Sanvee", lname = "Khot")
```

```
His last name is Khot
```

Default Parameter Value

If we call the function without argument, it uses the default value

In [48]:

```
def my_function(country = "Norway"):
 print("I am from " + country)
my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

In [49]:

```
def my_function(food):
 for x in food:
     print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

```
apple
banana
cherry
```

In [50]:

```
my_function(['Kunal','Sharvari','Kanisha','Prajkta','Sanvee'])
```

```
Kunal
Sharvari
Kanisha
Prajkta
Sanvee
```

Return values

To let a function return a value, use the return statement:

In [51]:

```
def my_function(x):
 return 5 * x
print(my_function(3))
```

```
15
```

In [52]:

```
# function
def calculator(a, b):
 add = a + b
 # return the addition
 return add
# call function
# take return value in variable
res = calculator(20, 5)
print("Addition :", res)
```

```
Addition : 25
```

In [53]:

```
def sum(a, b):
 return a + b
total=sum(10, 20)
print(total)
total=sum(5, sum(10, 20))
print(total)
```

```
30
35
```

The return value is nothing but a outcome of function.

The return statement ends the function execution.

For a function, it is not mandatory to return a value.

If a return statement is used without any expression, then the None is returned.

The return statement should be inside of the function block.

In [54]:

```
#Return multiple values
def arithmetic(num1, num2):
 add = num1 + num2
 sub = num1 - num2
 multiply = num1 * num2
 division = num1 / num2
 # return four values
 return add, sub, multiply, division
# read four return values in four variables
a, b, c, d = arithmetic(10, 2)
print("Addition: ", a)
print("Subtraction: ", b)
print("Multiplication: ", c)
print("Division: ", d)
```

```
Addition:  12
Subtraction:  8
Multiplication:  20
Division:  5.0
```

Local and Global variable

In general, a variable that is defined in a block is available in that block only. It is not accessible outside the block. Such a variable is called a local variable. Formal argument identifiers also behave as local variables.

In [55]:

```
#local variable
def greet():
 name1 = 'Sanvee'
 print('Hello ', name1)
greet()
```

```
Hello  Sanvee
```

In [56]:

```
print(name1)
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp/ipykernel_9404/536052042.py in <module>
----> 1 print(name1)

NameError: name 'name1' is not defined
```

In [57]:

```
name = 'Sanvee'#global variable
def greet():
    name = 'Kunal'#local variable
    print("Hello", name)
greet()
```

```
Hello Kunal
```

In [58]:

```
print(name)
```

```
Sanvee
```

In [59]:

```
a = 10
a = 20
print(a)
```

```
20
```

In [60]:

```
name = 'Sanvee'#global variable
def greet():
    global name
    name = 'Kunal'#global variable
    print("Hello", name)
greet()
```

```
Hello Kunal
```

In [61]:

```
print(name)
```

```
Kunal
```

In [62]:

```
def function1():
 # local variable
 loc_var = 888
     print("Value is :", loc_var)
def function2():
     print("Value is :", loc_var)
function1()
function2()
```

```
  File "C:\Users\User\AppData\Local\Temp/ipykernel_9404/3576358708.py", li
ne 4
    print("Value is :", loc_var)
    ^
IndentationError: unexpected indent
```

In [63]:

```
global_var = 999
def function1():
 print("Value in 1nd function :", global_var)
def function2():
 print("Value in 2nd function :", global_var)
function1()
function2()
```

```
Value in 1nd function : 999
Value in 2nd function : 999
```

In [64]:

```
# Global variable
global_var = 5
def function1():
 print("Value in 1st function :", global_var)
def function2():
 # Modify global variable
 # function will treat it as a local variable
 global_var = 555
 print("Value in 2nd function :", global_var)
def function3():
 print("Value in 3rd function :", global_var)
function1()
function2()
function3()
```

```
Value in 1st function : 5
Value in 2nd function : 555
Value in 3rd function : 5
```

In [65]:

```
# Global variable
x = 5
# defining 1st function
def function1():
 print("Value in 1st function :", x)
# defining 2nd function
def function2():
 # Modify global variable using global keyword
 global x
 x = 555
 print("Value in 2nd function :", x)
# defining 3rd function
def function3():
 print("Value in 3rd function :", x)
function1()
function2()
function3()
```

```
Value in 1st function : 5
Value in 2nd function : 555
Value in 3rd function : 555
```

In [ ]:

In [ ]: