**Function**

**Functions** are blocks of reusable code that perform a specific task. They allow us to break down our program into smaller, modular pieces and promote code reuse.

Main advantage of function is code reusability. Function also exists in other programming languages also.

Types of functions:

* **Built in functions:** Those function that are available with the python in built. Eg: id(), print(), type(), input().
* **User defined functions:** Those functions that we user make as per our requirement.

**User defined Functions**

Syntax:

|  |
| --- |
| def function\_name(parameters):  ‘’’ doc string ‘’’  # Function body  return value # Optional |

def keyword mandatory which is used as the definition of a function.

Parameter are the input to the function.

|  |
| --- |
| def hello(name):  return (f"Hello {name}, how are you!")  print(hello("Bhaskar"))  output  Hello Bhaskar, how are you! |

Here Bhaskar is argument and name is parameter.

**Return statement**: A function can return a value using the return statement. If no return statement is provided, the function returns None by default.

|  |
| --- |
| def add(a,b):  sum= a+b  return sum  a= add(10,20)  print(f"sum is:{a}")  output  sum is: 30 |

|  |
| --- |
| **To check odd or even**  def odd\_even(num):  if num%2==0:  print(num," is even")  else:  print(num," is odd")  odd\_even(10)  output  10 is even |

|  |
| --- |
| **To find factorial of a number**  #4!= 4\*3\*2\*1=24  def fact(num):  result=1  while num>=1:  result\*=num  num-=1  return result  print(fact(4))  output  24 |

Python can return multiple values as other programming language cannot. But in reality only one value is return as a tuple.

|  |
| --- |
| def sum\_sub(a,b):  sum=a+b  sub= a-b  return sum,sub  t= sum\_sub(20,10)  print(t) #tuple is printed  a,b=sum\_sub(20,10) #unpacking of tuple  print(f"sum is {a} and sub is {b}")  output:  (30, 10)  sum is 30 and sub is 10 |

**Arguments:** There are 4 type of arguments

* Positional arguments: The order of the function to parameter is positional set and that order is followed. In above example of sum\_sub the position of a=20 and b=10 is defined as positional argument. When the order is changed then the result will also get differed.
* Keyword arguments: We can pass the value in the keywords while calling the function. In the above example we can do as sum\_sub(a=20,b=10) or sum\_sub(b=10,a=20). Here order is not important.
* Default arguments: We can give the default argument if there is no argument passed through the function call. Default argument should be last argument that is after positional argument only default argument is accepeted.

|  |
| --- |
| def sub(a=200,b=20):  print(a-b)  sub(20,10) #argument is passed  sub() #argument is not passed so default value is used  output:  10  180 |

* Variable length arguments: In Python, we can define functions that accept a variable number of arguments. This is useful when we're unsure how many arguments will be passed into the function.

Syntax:

def function\_name(\*args):

# Function body

|  |
| --- |
| def sum(\*n):  total=0  for x in n:  total+=x  print("sum is: ", total)    sum()  sum(10,20)  sum(10,20,30,40)  output  sum is: 0  sum is: 30  sum is: 100 |

Here the variable n become tuple and accept all the values.

|  |
| --- |
| def f(a,\*args):  print(a) #output: 10  print(args) #output:(20,30)  f(10,20,30) |

**Types of variables (functional programming)**

* Global variable: Those type of variable that we declare outside the function. We can access global variable from any part of the program.
* Local variable: Those variables that are declared inside the function. We can access this kind of variables within a functions only. Local variable get more priority than global variable.

|  |
| --- |
| a=10 #global variable  def f1():  b=20 #local variable  print(b)  print(a)  f1()  print(a)  output:  20  10  10 |

We can make a local variable by using global function

Syntax: global b #b is global variable now

b=10 #we cannot do global b=10, at first we must declare then only value is assigned

|  |
| --- |
| a=10 #global variable  def f1():  a=222  b=20 #local variable  print(a)  print(globals().get('a'))  print(globals()['a'])  f1()  output:  222  10  10 |

**Recursive Function**: The function that calls itself then it is said to be recursive function. There are 2 criteria to become a recursive function:

* There must be termination criteria.
* At every call, output should be close.

|  |
| --- |
| **Eg: Factorial number**  def fact(n):  if n==0:  return 1  else:  return n\*fact(n-1)  n=int(input("enter number "))  a=fact(n)  print(f'Factorial of {n} is :',a)  output:  enter number 4  Factorial of 4 is: 24 |

**Anonymous function (lambda function):** Those functions that do not have function name. When we want to use a particular service for a small interval of time probably one time function then we make anonymous function.

|  |
| --- |
| s=lambda n:n\*n # when we assign s then it is not an anonymous function  print("The square is: ",s(4))  s=lambda a,b,c: a if a>b and a>c else b if b>c else c  print("biggest number is: ",s(20,30,40))  output:  The square is: 16  biggest number is 40 |

We have 3 functions that take function as an argument.

* filter()
* map()
* reduce()

**Filter() function**

It filter sequence from a number of sequence. Syntax: fliter(function, sequence)

|  |
| --- |
| #Here We fliter even number from a range  l= range(1,10)  def isEven(n):  if n%2==0:  return True  else: return False  l1=[]  for n in l:  if isEven(n) == True:  l1.append(n)  print(l1)  #now by using fliter function  def iseven(n):  if n%2==0:  return True  else: return False    l= range(1,10)  l1=list(filter(iseven, l))  print(l1)  output:  [2, 4, 6, 8]  [2, 4, 6, 8] |

|  |
| --- |
| #now use of lambda function  l= range(1,10)  l1= list(filter(lambda x:x%2==0, l))  print(l1)  output:  [2, 4, 6, 8] |

**Map() function:**

It gives same number of output as input after doing some operation.

|  |
| --- |
| l = range(1, 6)  def double(x):  return 2 \* x  l1 = list(map(double, l))  print(l1)  l= range(1,6)  l1= list(map(lambda x:x\*x,l))  print(l1)  output:  [2, 4, 6, 8, 10]  [1, 4, 9, 16, 25] |

**Reduce() function:**

It reduces the input by doing some operation into one single output. We cannot directly use reduce function, we import from functools.

|  |
| --- |
| from functools import reduce  l=range(1,6)  sum=reduce(lambda x,y:x+y,l)  print(sum) |

Everything in python is “Object”.

**Function aliasing**: Giving another name to the function.

|  |
| --- |
| def hello(name):  print("Hello",name)  wish= hello #another name as wish to the fucntion.  hello('bhaskar')  wish('isha') |

**Nested function**: The function that contain another function in it.

|  |
| --- |
| def outer():  print("outer function.")  def inner():  print("inner function")  inner()  print('outer function calling inner function')    output:  outer function.  inner function  outer function calling inner function |