Chapter 4

Functions

A group of related statements to perform a specific task is known as a function. Functions help to break the program into smaller units. Functions avoid repetition and enhance code reusability. Functions provide better modularity in programming. Python provides two types of functions.

a) Built-in functions
b) User-defined functions

Functions like input(), print() etc. are examples of built-in functions. The code of these functions will be already defined. We can create our own functions to perform a particular task. These functions are called user-defined functions.

4.1 FUNCTION DEFINITION

The following specifies simple rules for defining a function.

- Function block or Function header begins with the keyword `def` followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. We can also define parameters inside these parentheses.
- The first string after the function header is called the `docstring` and is short for documentation string. It is used to explain in brief, what a function does. Although optional, documentation is a good programming practice.
- The code block within every function starts with a colon (:) and is indented.
- The `return` statement [expression] exits a function, optionally passing back an expression to the caller. A `return` statement with no arguments is the same as `return` None.

Syntax

```
def functionname( parameters ):
    "function_docstring"
    function_suite
    return [expression]
```

Example Program

```
def sum_of_two_numbers(a,b): #Function Header with 2 parameters a and b
    "This Function is to find the sum of two numbers" #Docstring
 sum=a+b
 return sum
```

4.2 FUNCTION CALLING

After defining a function, we can call the function from another function or directly from the Python prompt. The order of the parameters specified in the function definition should be preserved in function call also.

Example Program

```
#Main Program Code
a = int(input("Enter the first number:"))
b = int(input("Enter the second number:"))
s = sum_of_two_numbers(a,b)#Function calling
print("Sum of",a,"and",b,"is", s)
```

Output

```
Enter the first number:4
Enter the second number:3
Sum of 4 and 3 is 7
```

All the parameters in Python are passed by reference. It means if we change a parameter which refers to within a function, the change also reflects back in the calling function. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope. Lifetime of a variable is the period throughout which the variable exists in the memory. The lifetime of variables inside a function is as long as the function executes. All variables in a program may not be accessible at all locations in that program. This depends on where we have declared a variable. The scope of a variable determines the portion of the program where we can access a particular identifier. They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Example Program

```
def value_change(a): #Function Header
    a=10
    print("Value inside function=",a) #prints the value of a inside
    function return
#Main Program Code
a =int(input("Enter a number:"))
```

```
value_change(a)#Function calling
print("Value outside function=",a) #prints value of a outside fun
```

Output

```
Enter a number:3
Value inside function= 10
Value outside function= 3
```

Here in the main program a number is read and stored in variable a. Even thou value is passed to the function value_change, a is assigned another value inside the fu This value is displayed inside the function. After returning from the function, it will d the value read from the main function. The variable a declared inside function value_c is local to that function and hence after returning to the main program, a will displ value stored in the main program.

4.3 FUNCTION ARGUMENTS

We can call the function by any of the following 4 arguments.

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

4.3.1 Required Arguments

Required arguments are the arguments passed to a function in correct positional o Here, the number of arguments in the function call should match exactly with the fun definition. Consider the following example.

Example Program

```
def value_change(a): #Function Header
 a=10
 print("Value inside function=",a) #prints the value of a inside func
 return
#Main Program Code
a =int(input("Enter a number:"))
value_change()#Function calling without passing parameter
print("Value outside function=",a) #prints value of a outside func
```

Output

```
Enter a number:4
Traceback (most recent call last):
  File "main.py", line 7, in <module>
    value_change()#Function calling without passing parameter
TypeError: value_change() takes exactly 1 argument (0 given)
```

The above example contains a function which requires 1 parameter. In the main prog code, the function is called without passing the parameter. Hence it resulted in an error

4.3.2 Keyword Arguments

When we use keyword arguments in a function call, the caller identifies the arguments by the parameter name. This allows us to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

Example Program

```
# Function definition
def studentinfo(rollno,name,course):
    print("Roll Number:",rollno)
    print("Name:",name)
    print("Course:",course)
#Function calling
#order of parameters in function call is shuffled
studentinfo(course="UG",rollno=50,name="Jack")
```

Output

```
Roll Number: 50
Name: Jack
Course: UG
```

4.3.3 Default Arguments

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example shows how default arguments are invoked.

Example Program

```
# Function definition
def studentinfo(rollno,name,course="UG"):
    "This prints a passed info into this function"
    print("Roll Number:",rollno)
    print("Name:",name)
    print("Course:",course)
#Function calling
#Order of parameters is shuffled
studentinfo(course="UG", rollno=50, name="Jack")
#The parameter course is omitted
studentinfo(rollno=51,name="Tom")
```

Output

```
Roll Number: 50
Name: Jack
Course: UG
```

```
Roll Number: 51
Name: Tom
Course: UG
```

In the above example, the first function call to `studentinfo` passes the three parameter In the case of second function call to `studentinfo`, the parameter `course` is omitted. Her it takes the default value "UG" given to course in the function definition.

4.3.4 Variable-Length Arguments

In some cases we may need to process a function for more arguments than specifi while defining the function. These arguments are called *variable-length* arguments and a not named in the function definition, unlike required and default arguments. An asterisk is placed before the variable name that holds the values of all non-keyword variab arguments. This tuple remains empty if no additional arguments are specified during t function call. The following shows the syntax of a function definition with *variable-leng* arguments.

Syntax

```
def functionname([formal_arguments,] *variable_arguments_tuple ):
    "function_docstring"
    function_statements
    return [expression]
```

Example Program

```
# Function definition
def variablelengthfunction(*argument):
    print("Result:",)
    for i in argument: print(i)
#Function call with 1 argument
variablelengthfunction(10)
#Function call with 4 arguments
variablelengthfunction(10,30,50,80)
```

Output

```
Result: 10 #Result of function call with 1 argument
Result: 10 #Result of function call with 4 arguments
30
50
80
```

4.4 ANONYMOUS FUNCTIONS (LAMBDA FUNCTIONS)

In Python, anonymous function is a function that is defined without a name. While norm functions are defined using the def keyword, in Python anonymous functions are define using the lambda keyword. Hence, anonymous functions are also called lambda function The following are the characteristics of lambda functions.

1. Lambda functions can take any number of arguments but return only one value i the form of an expression.
2. It cannot contain multiple expressions.
3. It cannot have comments.
4. A lambda function cannot be a direct call to print because lambda requires ar expression.
5. Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
6. Lambda functions are not equivalent to inline functions in C or C++.

Syntax

```
Lambda [arg1 [,arg2,.....argn]]:expression
```

Example Program

```
# Lambda Function definition
square=lambda x : x*x;
# Usage of lambda function
n=int(input("Enter a number:"))
print("Square of", n,"is", square(n))#Lambda function call
```

Output

```
Enter a number:5
Square of 5 is 25
```

In the above example, lambda is the keyword, x shows the argument passed, and x*x is the expression to be evaluated and stored in the variable square. In the case of calculating the square of a number, only one argument is passed. More than one argument is possible for lambda functions. The following example shows a lambda function which takes two arguments and returns the sum.

Example Program

```
# Lambda Function definition
sum=lambda x,y : x+y;
# Usage of lambda function
m=int(input("Enter first number:"))
n=int(input("Enter second number:"))
print("Sum of", m, "and", n,"is", sum(m,n))#Lambda function call
```

Output

```
Enter first number:10
Enter second number:20
Sum of 10 and 20 is 30
```

Uses of lambda function

We use lambda functions when we require a nameless function for a short period of time. In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like filter(), map() etc.

Example Program with map()

The map() function in Python takes in a function and a list. The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

```
# Lambda Function to increment the items in list by 2
oldlist=[2,31,42,11,6,5,23,44]
print(oldlist)
# Usage of lambda function
newlist=list(map(lambda x: x+2,oldlist))
print("List after incrementation by 2")
print(newlist)
```

Output

```
[2, 31, 42, 11, 6, 5, 23, 44]
List after incrementation by 2
[4, 33, 44, 13, 8, 7, 25, 46]
```

4.4.1 *filter()* Function

We have already discussed about the use of *map()* function along with lambda functions. The function *filter(function, list)* offers an elegant way to filter out all the elements of a list, for which the function returns True. The function *filter(func,lis)* needs a function *func* as its first argument. *func* returns a Boolean value, i.e. either True or False. This function will be applied to every element of the list *lis*. Only if *func* returns True will the element of the list be included in the result list.

Example Program with *filter()*

The filter() function in Python takes in a function and a list as arguments. The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.

```
# Lambda Function to filter out only the odd numbers from a list
oldlist=[2,31,42,11,6,5,23,44]
# Usage of lambda function
newlist=list(filter(lambda x: (x%2!=0),oldlist))
print(oldlist)
print(newlist)
```

Output

```
[2, 31, 42, 11, 6, 5, 23, 44]
[31, 11, 5, 23]
```

4.4.2 *reduce()* Function

The function *reduce(func, seq)* continually applies the function *func()* to the sequence *seq*. It returns a single value.

If *seq* = [s_1, s_2, s_3, ... , s_n], calling *reduce(func, seq)* works like this:

At first the first two elements of *seq* will be applied to *func*, i.e. *func*(s_1,s_2) The list on which *reduce()* works looks now like this: [*func*(s_1, s_2), s_3, ... , s_n]

In the next step *func* will be applied on the previous result and the third element of the list, i.e. *func*(*func*(s_1, s_2),s_3)

The list looks like this now: [*func*(*func*(s_1, s_2),s_3), ... , s_n]

Continue like this until just one element is left and return this element as the result of *reduce()*.

This is a really useful function for performing some computation on a list and returning the result. The following example illustrates the use of *reduce()* function, that computes the product of a list of integers.

Example Program with *reduce()*

```
import functools
list=[1,2,3,4]
product=functools.reduce(lambda x, y: x * y,list)
print(list)
print("Product=",product)
```

Output

```
[1, 2, 3, 4]
Product= 24
```

4.5 RECURSIVE FUNCTIONS

Recursion is the process of defining something in terms of itself. A function can call other functions. It is possible for a function to call itself. This is known as recursion. The following example shows a recursive function to find the factorial of a number.

Example Program

```
def recursion_fact(x):
#This is a recursive function to find the factorial of an integer
  if x == 1: return 1
  else: return (x * recursion_fact(x-1))#Recursive calling
num = int(input("Enter a number: "))
if num >= 1:
  print("The factorial of", num, "is", recursion_fact(num))
```

Output

```
Enter a number: 5
The factorial of 5 is 120
```

Explanation

In the above example, recursion_fact() is a recursive function as it calls itself. When we call this function with a positive integer, it will recursively call itself by decreasing the number. Each function call multiples the number with the factorial of number-1 until the number is

equal to one. This recursive call can be explained in the following steps. Our recursion ends when the number reduces to 1. This is called the base condition. Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely. We must avoid infinite recursion.

```
Recursion_fact(5)                          # 1st call with 5
5 * recursion_fact(4)                      # 2nd call with 4
5 * 4 * recursion_fact(3)                  # 3rd call with 3
5 * 4 * 3 *  recursion_fact(2)             # 4th call with 2
5 * 4 * 3 * 2 * recursion_fact(1)          # 5th call with 1
5 * 4 * 3 * 2 * 1                          # return from 5th call as number=1
5 *4 * 3 * 2                               # return from 4th call
5 * 4 * 6                                  # return from 3rd call
5*24                                       # return from 2nd call
120                                        #return from 1st call
```

4.6 FUNCTION WITH MORE THAN ONE RETURN VALUE

Python has a strong mechanism of returning more than one value at a time. This is very flexible when the function needs to return more than one value. Instead of writing separate functions for returning individual values, we can return all the values within same function. The following shows an example for function returning more than one value.

Example Program

```
#Python Function Returning more than One value
def calc(a,b):
    sum=a+b
    diff=a-b
    prod=a*b
    quotient=a/b
    return sum, diff, prod, quotient
a=int(input("Enter first number:"))
b=int(input("Enter second number:"))
s,d,p,q=calc(a,b)
print("Sum=",s)
print("Difference=",d)
print("Product=",p)
print("Quotient=",q)
```

Output

```
Enter first number:10
Enter second number:5
Sum= 15
Difference= 5
Product= 50
Quotient= 2
```

4.7 SOLVED LAB EXERCISES

1. Write a Python function to check whether a number is even or odd.

Program

```
#Function to Check whether a number is even or odd.
Def oddeven(x):
  if x%2==0: print("The Number", x, "is even")
  else: print("The Number", x, "is odd")
  return;
num = int(input("Enter a number: "))
oddeven(num)#Function calling
```

Output

```
Enter a number: 4
The Number 4 is even
```

2. Write a Python program to calculate the sum of three given numbers, if the values are equal then return thrice of their sum.

Program

```
  def sum_thrice(x, y, z):
     sum = x + y + z
     if x == y == z:
      sum = sum * 3
     return sum
#Main Program
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))
   print("Sum_Thrice:",sum_thrice(a,b,c))
```

Output

```
Enter First Number:3
Enter Second Number:3
Enter Third Number:3
Sum_Thrice: 27
```

3. Write a Python function to get a new string from a given string where "Is" has been added to the front. If the given string already begins with "Is" then return the string unchanged.

Program

```
def new_string(str):
  if str[:2] == "Is":
    return str
  return "Is " + str
#Main Program
str1=input("Enter a String:")
```

```
print("New String:",new_string(str1))
```

Output

```
Enter a String:Good Morning
New String: Is Good Morning
```

4. Write a Python program to get a string which is n (non-negative integer) copies of a given string.

Program

```
def larger_string(str, n):
    result = " "
    for i in range(n):
        result = result + " " + str
    return result
#Main Program
str1=input("Enter a String:")
n=int(input("Number of Copies:"))
print("Final String:",larger_string(str1,n))
```

Output

```
Enter a String:Python Program
Number of Copies:3
Final String:   Python Program Python Program Python Program
```

5. Write a Python function that will return true if the two given integer values are equal o their sum or difference is 5.

Program

```
def test_number5(x, y):
    eq,differ5,sum5,no=False,False,False,False
    if x == 5 or y == 5:
        eq = True
    elif abs(x-y) == 5:
        differ5 = True
    elif (x+y) == 5:
        sum5 = True
    else:
        no = True
    return eq,differ5,sum5,no
a=int(input("Enter first integer:"))
b=int(input("Enter second integer:"))
e,d,s,n = test_number5(a,b)
if e:print(a,"and",b, "are equal:",e)
if d: print("Difference of", a, "and", b,"is",5, "is",d)
if s:print("Sum of ", a, "and", b, "is", 5, "is",s)
if n: print("Sum of numbers is not 5\n Difference of numbers is not 5\r
```

```
Numbers are not equal")
```

Output

```
Enter first integer:6
Enter second integer:8
Sum of numbers is not 5
Difference of numbers is not 5
Numbers are not equal
```

6. Write a Python function that takes two lists and returns True if they have at least one common member.

Program

```
def common_data(list1, list2):
 result = False
 for x in list1:
  for y in list2:
        if x == y:
            result = True
 return result
#main program
list1=input("Enter first list(space separated):")
lis1=list(list1.split())
list2=input("Enter second list(space separated):")
lis2=list(list2.split())
r=common_data(lis1,lis2)
if r:
 print(lis1,"and", lis2,"has at least common member:",r)
else:
 print(lis1,"and", lis2,"has at least common member:",r)
```

Output

```
Enter first list(space separated):1 2 cat rat
Enter second list(space separated):2 3 tiger lion
['1', '2', 'cat', 'rat'] and ['2', '3', 'tiger', 'lion'] has at least
common member: True
```

7. Write a Python function to find the GCD of 2 numbers.

Program

```
#Function to find the GCD of two numbers
def gcd(a,b):
 for i in range (1, min(a,b)+1):
  if (a%i==0 and b%i==0): gcd=i
 print(gcd)
 return;
#Main Function
```

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
gcd(a,b)#Function calling
```

Output

```
Enter First Number:12
Enter Second Number:18
   6
```

Write a Python function to generate all the factors of a number.

Program

#Function to find all the factors of a number

```
def factors(a):
 for i in range (1,(a+1)):
  if (a%i==0) : print(i)
 return;
#Main Function
a=int(input("Enter a Number:"))
factors(a)#Function calling
```

Output

```
Enter a Number:12
1
2
3
4
6
12
```

9. Write a Python function to find the sum of digits of a number.

Program

```
#Function to find sumofdigits of a number
def sumofdigits(n):
 sum=0
 while(n>0):
  digit=n%10
  sum+=digit
  n=n//10
 print("Sum of digits is",sum)
 return;
#Main Function
a=int(input("Enter a Number:"))
sumofdigits(a)#Function calling
```

Output

```
Enter a Number:1234
Sum of digits is 10
```

10. Write a Python function to concatenate two strings.

Program

```
#Function to concatenate two strings
def concatenate(s1,s2):
 print("First String:",s1)
 print("Second String:",s2)
 print("Concatenated String:",(s1+s2))
 return;
#Main Function
s1=input("Enter first string:")
s2=input("Enter second string:")
concatenate(s1,s2)#Function calling
```

Output

```
Enter first string:Hello
Enter second string: Python
First String: Hello
Second String:  Python
Concatenated String: Hello Python
```

11. Write a Python function called compare which takes two strings s1 and s2 and an integer n as arguments. The function should return True if first n characters of both the strings are same else the function should return False.

Program

```
#Function to find similar substrings
def substring(s1,s2,n):
  for i in range(0,n):
   if(s1[i]!= s2[i]):return False
  else : return True
#Main Function
s1=input("Enter first string:")
s2=input("Enter second string:")
n=int(input("Enter a number:"))
#substring(s1,s2,n)#Function calling
if substring(s1,s2,n):
  print("The substrings are equal:")
else:
  print("The substrings are not equal")
```

Output

```
Enter first string: Programming
Enter second string: Program
Enter a number:4
The substrings are equal
```

12. Write a Python function to find whether a number is completely divisible by another number.

Program

```
#Function to find whether a number is completely divisible by another
def divisible(x,y):
    if(x%y==0):
        print(x, "is completely divisible by", y)
        return;
    else :
        print(x, "is not completely divisible by", y)
        return;
#Main Function
x=int(input("First Number:"))
y=int(input("Second Number:"))
divisible(x,y)
```

Output

```
First Number:12
Second Number:5
12 is not completely divisible by 5
```

13. Write a Python program to display Fibonacci series using recursion.

Program

```
"""Recursive function to
   print Fibonacci sequence"""
def recursive_fibo(n):
   if n <= 1:
       return n
   else:
       return(recursive_fibo(n-1) + recursive_fibo(n-2))
# Main function
n = int(input("How many terms? "))
# check if the number of terms is valid
if n <= 0:
   print("Plese enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(n):
       print(recursive_fibo(i))
```

Output

```
How many terms? 5
Fibonacci sequence:
0
1
1
2
3
```

14. Write a Python program to find the sum of n natural numbers using recursion.

Program

```
"""Function to return the sum
   of natural numbers using recursion"""
def recursive_sum(n):
    if n <= 1:
       return n
    else:
       return n + recursive_sum(n-1)
#Main function
num = int(input("Enter a number: "))
if num < 0:
   print("Enter a positive number:")
else:
  print("The sum of first",num, "natural numbers is",recursive_sum(num))
```

Output

```
Enter a number: 5
The sum of first 5 natural numbers is 15
```

15. Write a Python program to convert decimal to binary using recursion.

Program

```
"""Function to print binary number
   for the input decimal using recursion"""
def binary(n):
    if n >= 1:
       binary(n//2)
       print(n%2, end=' ')

#Main function
dec = int(input("Enter an integer: "))
binary(dec)
```

Output

```
Enter an integer: 5
1 0 1
```