Working with Functions Class 12 Notes

Computer Science Class 12 Functions Notes

What is functions in Python

When a single program is divided into small units, it is known as a function. A function is reusable code, meaning the same function can be used multiple times, function helps to organise the code easily. A function is a block of code that only executes when called. Functions can accept inputs (arguments) and produce outputs (return values).

You can define function using “def” keyword in python.

Syntax for defining function –

```
def <function name> (<parameters>) :
    <statement 1>
    <statement 2>
    .
```

.
.

Types of function

In Python, functions are categorised into three types, each having different purposes. The type of function are:

- Built-in functions
- Functions defined in module
- User defined functions

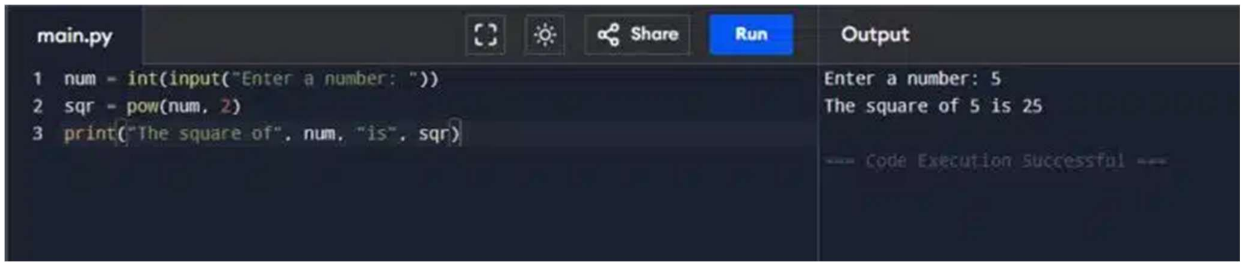
a. Built-in functions

A built-in function is a predefined function like len(), max(), min(), pow() etc. These functions perform common tasks and are always available for use.

Example,

Q. Program to calculate square of a number using built-in function

```
num = int(input("Enter a number: "))
sqr = pow(num, 2)
print("The square of", num, "is", sqr)
```



```
num = int(input("Enter a number: "))
sqr = pow(num, 2)
print("The square of", num, "is", sqr)
```

```
Enter a number: 5
The square of 5 is 25

=== Code Execution Successful ===
```

b. Functions defined in module

A module is like a container which holds pieces of code like functions and variables; this function can be reused in the program. For example, a module is just like a toolbox. Suppose you have different tools, like functions, variables, etc. Whenever you require a specific tool, you import the module and use it in your program. Python offers many built-in modules. Some of the common modules are:

- Math module

- Random module
- Statistics module

c. User defined functions

User-defined functions are the fundamental building block of any programme user-define funcation allow programmers to write their own function with function name.

Creating User Defined Function

A function definition begins with def (short for define). The syntax for creating a user defined function is as follows –

Syntax –

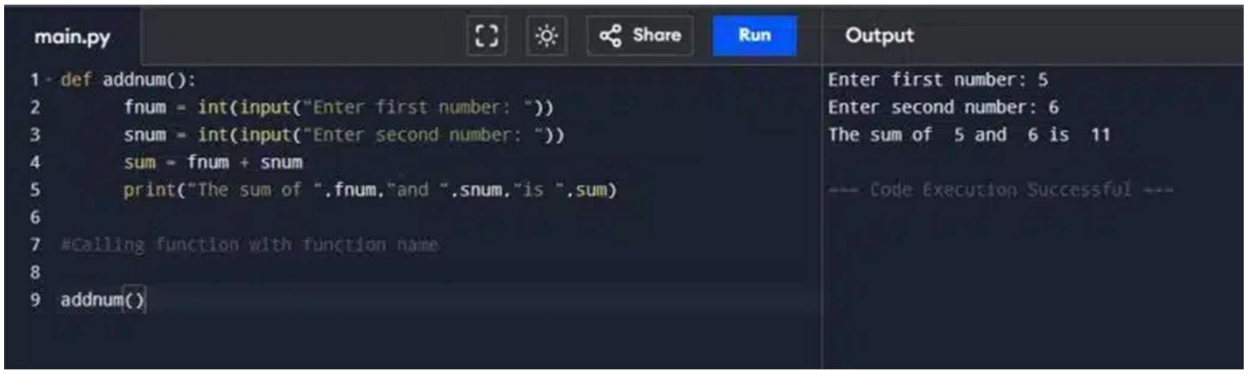
```
def function_name(parameter1, parameter2, ...) :
      statement_1
      statement_2
      statement_3
      ....
```

- The items enclosed in “[ ]” are called parameters and they are optional. Hence, a function may or may not have parameters. Also, a function may or may not return a value.
- Function header always ends with a colon (:).
- Function name should be unique. Rules for naming identifiers also applies for function naming.
- The statements outside the function indentation are not considered as part of the function.

Q. Write a user defined function to add 2 numbers and display their sum.

```
def addnum():
     fnum = int(input("Enter first number: "))
     snum = int(input("Enter second number: "))
     sum = fnum + snum
     print("The sum of ",fnum,"and ",snum,"is ",sum)

#Calling function with function name

addnum()

Output:
```

```
Enter first number: 5
Enter second nu
```



main.py

```python
def addnum():
      fnum = int(input("Enter first number: "))
      snum = int(input("Enter second number: "))
      sum = fnum + snum
      print("The sum of ",fnum,"and ",snum,"is ",sum)

#Calling function with function name

addnum()
```

Output

```
Enter first number: 5
Enter second number: 6
The sum of  5 and  6 is  11

=== Code Execution Successful ===
```

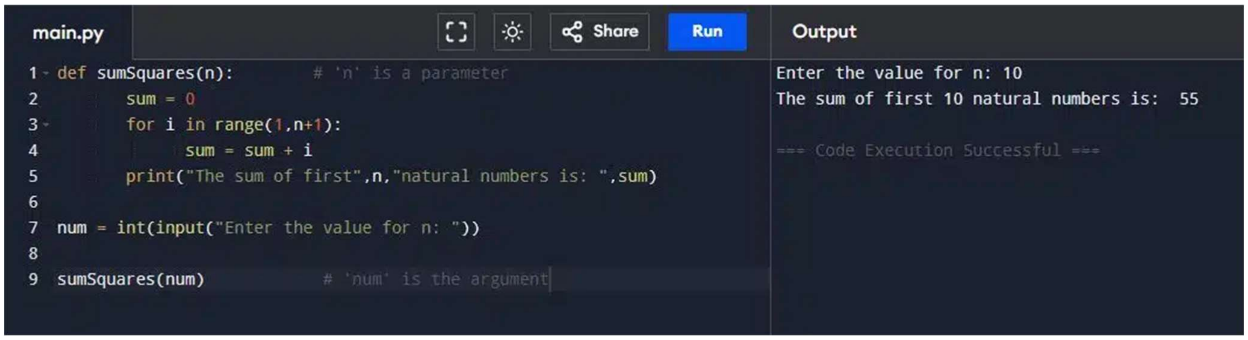
Arguments and Parameters

User-defined function could potentially take values when it is called. A value received in the matching parameter specified in the function header and sent to the function as an argument.

Q. Write a program using a user defined function that displays sum of first n natural numbers, where n is passed as an argument.

```python
def sumSquares(n):      # 'n' is a parameter
    sum = 0
    for i in range(1,n+1):
        sum = sum + i
    print("The sum of first",n,"natural numbers is: ",sum)

num = int(input("Enter the value for n: "))

sumSquares(num)         # 'num' is the argument
```

```
Output
Enter the value for n: 10
The sum of first 10 natural numbers is: 55
```



main.py

```python
def sumSquares(n):         # 'n' is a parameter
    sum = 0
    for i in range(1,n+1):
        sum = sum + i
    print("The sum of first",n,"natural numbers is: ",sum)

num = int(input("Enter the value for n: "))

sumSquares(num)            # 'num' is the argument
```

Output

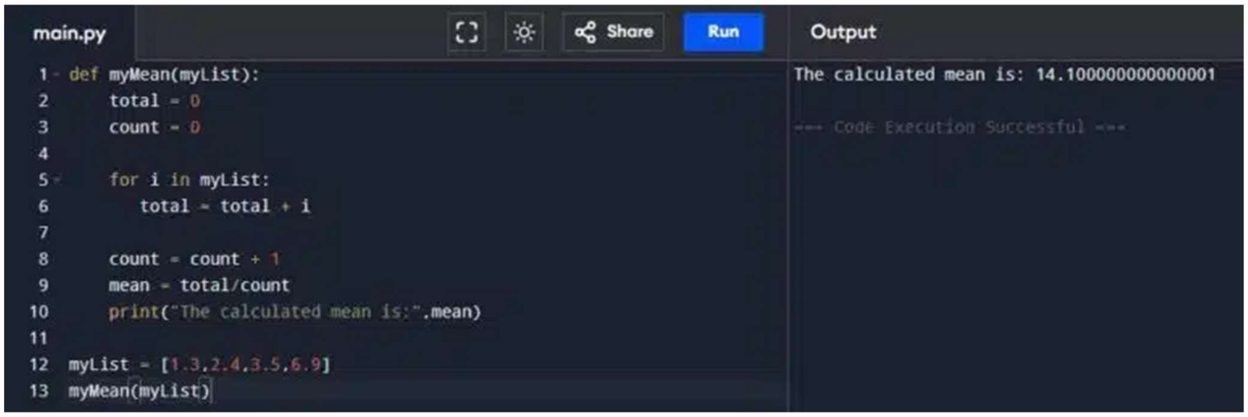
```
Enter the value for n: 10
The sum of first 10 natural numbers is:  55

=== Code Execution Successful ===
```

Q. Write a program using a user defined function myMean() to calculate the mean of floating values stored in a list.

```
def myMean(myList):
    total = 0
    count = 0

    for i in myList:
      total = total + i

    count = count + 1
    mean = total/count
    print("The calculated mean is:",mean)

myList = [1.3,2.4,3.5,6.9]
myMean(myList)
```

Output:

```
The calculated mean is: 14.100000000000001
```



```
def myMean(myList):
    total = 0
    count = 0

    for i in myList:
       total = total + i

    count = count + 1
    mean = total/count
    print("The calculated mean is:",mean)

myList = [1.3,2.4,3.5,6.9]
myMean(myList)
```

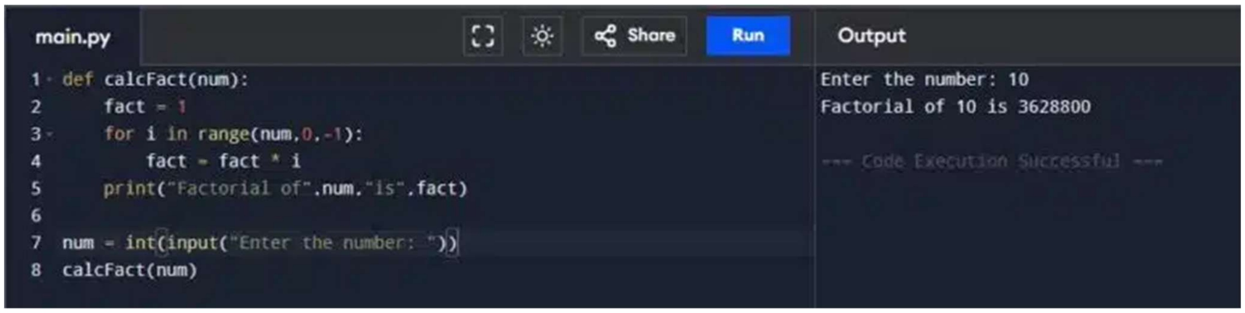
```
The calculated mean is: 14.100000000000001

=== Code Execution Successful ===
```

Q. Write a program using a user defined function calcFact() to calculate and display the factorial of a number num passed as an argument.

```
def calcFact(num):
    fact = 1
    for i in range(num,0,-1):
        fact = fact * i
    print("Factorial of",num,"is",fact)

num = int(input("Enter the number: "))
calcFact(num)
```

Output:

```
Enter the number: 5
Factorial of 5 is 120
```



```python
def calcFact(num):
    fact = 1
    for i in range(num,0,-1):
        fact = fact * i
    print("Factorial of",num,"is",fact)

num = int(input("Enter the number: "))
calcFact(num)
```

```
Enter the number: 10
Factorial of 10 is 3628800

=== Code Execution Successful ===
```

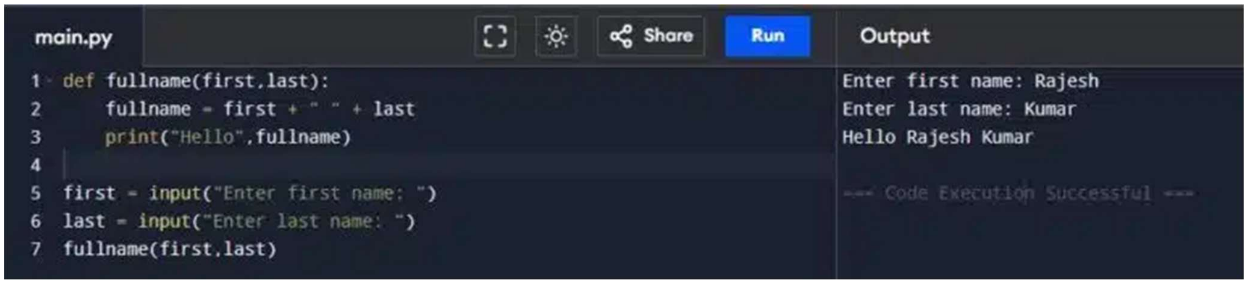
String as Parameters

Some programmes may require the user to supply string values as an argument.

Q. Write a program using a user defined function that accepts the first name and lastname as arguments, concatenate them to get full name and displays the output as:

```python
def fullname(first,last):
    fullname = first + " " + last
    print("Hello",fullname)

first = input("Enter first name: ")
last = input("Enter last name: ")
fullname(first,last)
```

```
Output
Enter first name: Rajesh
Enter last name: Kumar
Hello Rajesh Kumar
```



```python
def fullname(first,last):
    fullname = first + " " + last
    print("Hello",fullname)

first = input("Enter first name: ")
last = input("Enter last name: ")
fullname(first,last)
```

```
Enter first name: Rajesh
Enter last name: Kumar
Hello Rajesh Kumar

=== Code Execution Successful ===
```

Default Parameter

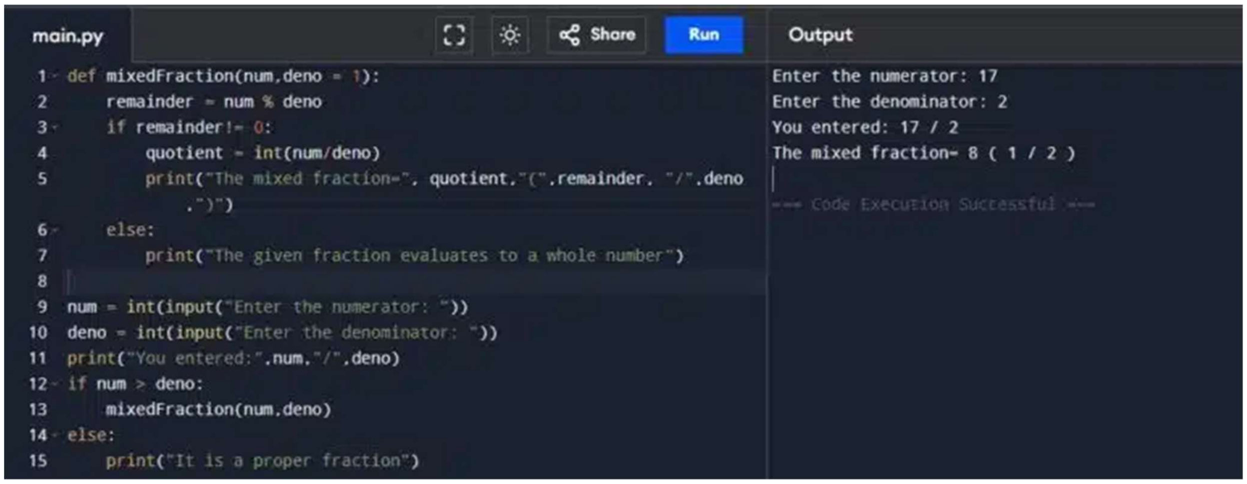
The argument can be given a default value in Python. When a function call doesn’t have its appropriate argument, a default value is chosen in advance and given to the parameter.

Q. Write a program that accepts numerator and denominator of a fractional number and calls a user defined function mixedFraction() when the fraction formed is not a proper fraction. The default value of denominator is 1. The

function displays a mixed fraction only if the fraction formed by the parameters does not evaluate to a whole number.

```
def mixedFraction(num,deno = 1):
    remainder = num % deno
    if remainder!= 0:
        quotient = int(num/deno)
        print("The mixed fraction=", quotient,"(",remainder, "/",deno,")")
    else:
        print("The given fraction evaluates to a whole number")

num = int(input("Enter the numerator: "))
deno = int(input("Enter the denominator: "))
print("You entered:",num,"/",deno)
if num > deno:
    mixedFraction(num,deno)
else:
    print("It is a proper fraction")
```

```
Output
Enter the numerator: 17
Enter the denominator: 2
You entered: 17 / 2
The mixed fraction= 8 ( 1 / 2 )
```



```
def mixedFraction(num,deno = 1):
    remainder = num % deno
    if remainder != 0:
        quotient = int(num/deno)
        print("The mixed fraction=", quotient,"(",remainder, "/",deno
            ,")")
    else:
        print("The given fraction evaluates to a whole number")

num = int(input("Enter the numerator: "))
deno = int(input("Enter the denominator: "))
print("You entered:",num,"/",deno)
if num > deno:
    mixedFraction(num,deno)
else:
    print("It is a proper fraction")
```

```
Enter the numerator: 17
Enter the denominator: 2
You entered: 17 / 2
The mixed fraction= 8 ( 1 / 2 )

=== Code Execution Successful ===
```

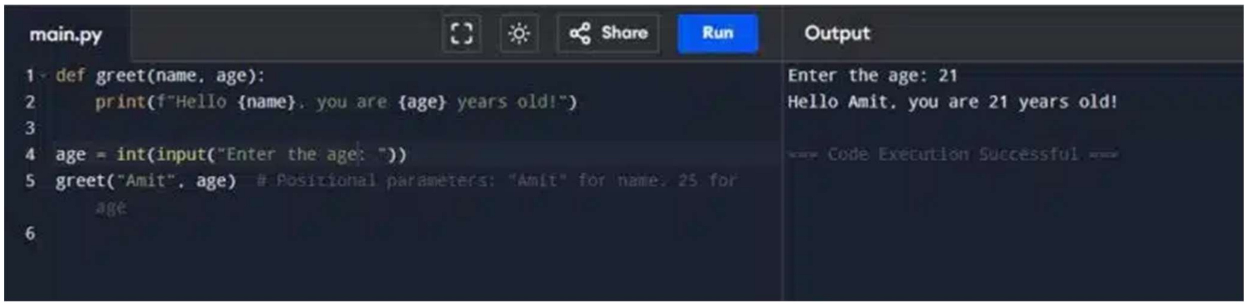
Positional parameters

Positional parameters are the method of passing arguments to functions in a specific order. Arguments that must be presented in the right order are known as positional arguments. When calling a function, the first positional argument must always be listed first, and the second will be second.

Q. Write a program to print the name and age of the user.

```
def greet(name, age):
    print(f"Hello {name}, you are {age} years old!")
```

```
age = int(input("Enter the age: "))
greet("Amit", age) # Positional parameters: "Amit" for name, 25 for age

Output:
Enter the age: 21
Hello Amit, you are 21 years old!
```



```
def greet(name, age):
    print(f"Hello {name}, you are {age} years old!")

age = int(input("Enter the age: "))
greet("Amit", age)  # Positional parameters: "Amit" for name, 25 for age
```

```
Enter the age: 21
Hello Amit, you are 21 years old!

=== Code Execution Successful ===
```

Functions Returning Value

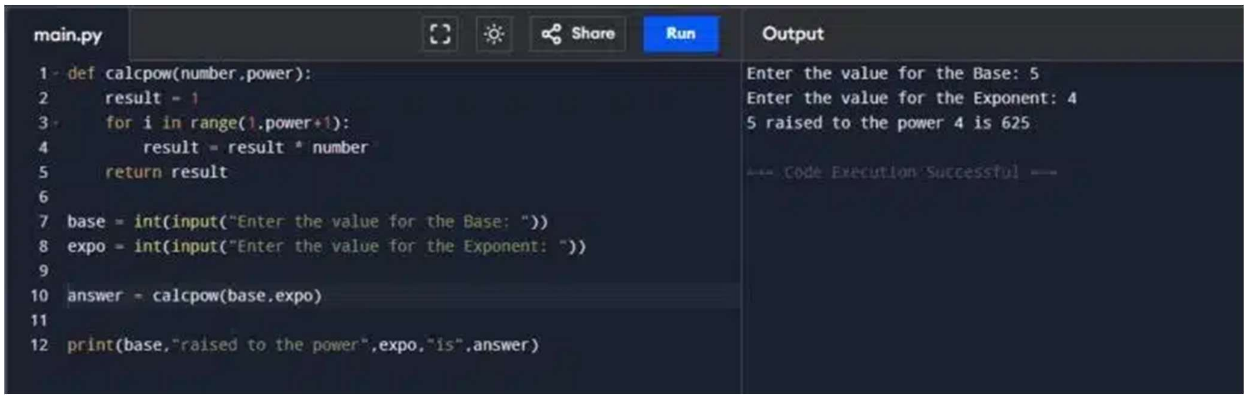
The function’s values are returned using the return statement. A function that has finished its duty will return a value to the script or function that called it.

The return statement does the following –

- returns the control to the calling function.
- return value(s) or None.

Q. Write a program using user defined function calcPow() that accepts base and exponent as arguments and returns the value Baseexponent where Base and exponent are integers.

```
def calcpow(number,power):
    result = 1
    for i in range(1,power+1):
        result = result * number
    return result

base = int(input("Enter the value for the Base: "))
expo = int(input("Enter the value for the Exponent: "))

answer = calcpow(base,expo)

print(base,"raised to the power",expo,"is",answer)

Output:
Enter the value for the Base: 5
Enter the value for the Exponent: 4
5 raised to the power 4 is 625
```



```python
def calcpow(number,power):
    result = 1
    for i in range(1,power+1):
        result = result * number
    return result

base = int(input("Enter the value for the Base: "))
expo = int(input("Enter the value for the Exponent: "))

answer = calcpow(base,expo)

print(base,"raised to the power",expo,"is",answer)
```

```
Enter the value for the Base: 5
Enter the value for the Exponent: 4
5 raised to the power 4 is 625

=== Code Execution Successful ===
```

Flow of execution in a function call

The flow of execution refers to the order in which statements are executed during a program run. The program's opening statement is where execution always starts. The statements are carried out one at a time, in ascending order. Although the order in which a programme runs is unaffected by function declarations, keep in mind that statements inside a function are not performed until the function is called.

Example –

Defining function

def sum(x, y) :

Calling function

sum(a,b)

Where a, b are the values being passed to the function sum().

Scope of a Variable

An internal function variable can't be accessed from the outside. There is a well defined accessibility for each variable. The scope of a variable is the area of the programme that the variable is accessible from. A variable may fall under either of the two scopes listed below:

Global Variable – A variable that is defined in Python outside of any function or block is referred to as a global variable. It is accessible from any functions defined afterward.

Local Variable – A local variable is one that is declared inside any function or block. Only the function or block where it is defined can access it.