Name : Taufeeq ahmad

ID : 6856

Subject : Data science

Programe : BS (SE)

Date : 26 june 2020

Q1(A): Why Functions are used discuss in detail?

Ans: Functions are used because of following reasons –
a) To improve the readability of code.
b) Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.
c) Debugging of the code would be easier if you use functions, as errors are easy to be traced.
d) Reduces the size of the code, duplicate set of statements are replaced by function calls.

Syntax of a function

```
return_type function_name (argument list)
{
    Set of statements – Block of code
}
```

return_type: Return type can be of any data type such as int, double, char, void, short etc. Don't worry you will understand these terms better once you go through the examples below.

function_name: It can be anything, however it is advised to have a meaningful name for the functions so that it would be easy to understand the purpose of function just by seeing it's name.

argument list: Argument list contains variables names along with their data types. These arguments are kind of inputs for the function. For example – A

function which is used to add two integer variables, will be having two integer argument.

Block of code: Set of C statements, which will be executed whenever a call will be made to the function.

Do you find above terms confusing? – Do not worry I'm not gonna end this guide until you learn all of them :)
Lets take an example – Suppose you want to create a function to add two integer variables.

Let's split the problem so that it would be easy to understand –
Function will add the two numbers so it should have some meaningful name like sum, addition, etc. For example lets take the name addition for this function.

```
return_type addition(argument list)
```

This function addition adds two integer variables, which means I need two integer variable as input, lets provide two integer parameters in the function signature. The function signature would be –

```
return_type addition(int num1, int num2)
```

The result of the sum of two integers would be integer only. Hence function should return an integer value – **I got my return type** – It would be integer –

```
int addition(int num1, int num2);
```

So you got your function prototype or signature. Now you can implement the logic in C program like this:

How to call a function in C?

Consider the following C program

Example1: Creating a user defined function addition()

```
#include <stdio.h>
int addition(int num1, int num2)
{
     int sum;
     /* Arguments are used here*/
     sum = num1+num2;

     /* Function return type is integer so we are returning
      * an integer value, the sum of the passed numbers.
```

```
     */
    return sum;
}

int main()
{
    int var1, var2;
    printf("Enter number 1: ");
    scanf("%d",&var1);
    printf("Enter number 2: ");
    scanf("%d",&var2);

    /* Calling the function here, the function return type
     * is integer so we need an integer variable to hold the
     * returned value of this function.
     */
    int res = addition(var1, var2);
    printf ("Output: %d", res);

    return 0;
}
```

Output:

```
Enter number 1: 100
Enter number 2: 120
Output: 220
```

Q1(b): . How arguments are used in function , write a simple program in Python?

Ans: A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.

- The code block within every function starts with a colon (:) and is indented.
- The statement return [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]
```

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.

Example

The following function takes a string as input parameter and prints it on standard screen.

```
def printme( str ):
   "This prints a passed string into this function"
   print str
   return
```

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function −

Live Demo

```
#!/usr/bin/python

# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;

# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

When the above code is executed, it produces the following result −

```
I'm first call to user defined function!
Again second call to the same function
```

Pass by reference vs value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example −

Q2(a): Why .upper(),.lower(),capitalize() and .swapcase() function are used ?

Ans: In Python, isupper() is a built-in method used for string handling.
The isupper() methods returns “True” if all characters in the string are uppercase, Otherwise, It returns “False”.
This function is used to check if the argument contains any uppercase characters such as :

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Syntax :

```
string.isupper()
Parameters:
isupper() does not take any parameters
Returns :
1.True- If all characters in the string are uppercase.
2.False- If the string contains 1 or more non-uppercase characters.
```

Examples:

```
Input : string = 'GEEKSFORGEEKS'

Output : True


Input : string = 'GeeksforGeeks'

Output : False
```

(b): Write a program in which the discussed functions are used.

Ans:

brightness_4

```
# Python code for implementation of upper()
# Given string and new string
```

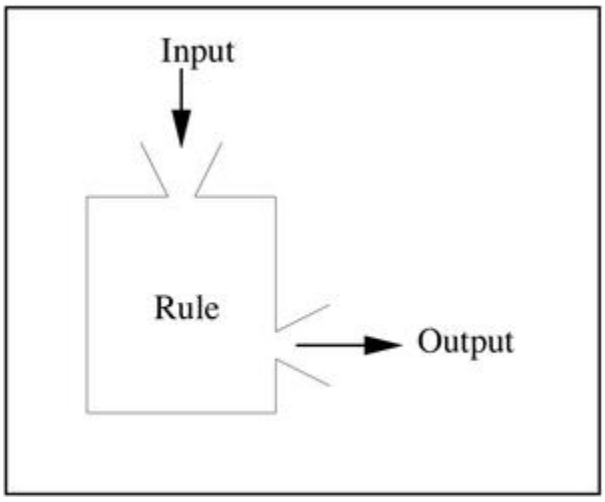
```
string ='GeeksforGeeks is a computer Science portal for Geeks'
newstring =''
count1 = 0
count2 = 0
count3 = 0

for a in string:
# Checking for lowercase letter and converting to uppercase.
    if (a.isupper()) == True:
        count1+= 1
        newstring+=(a.lower())
# Checking for uppercase letter and converting to lowercase.
    elif (a.islower()) == True:
        count2+= 1
        newstring+=(a.upper())
# Checking for whitespace letter and adding it to the new string as it is.
    elif (a.isspace()) == True:
        count3+= 1
        newstring+= a
print("In original String : ")
print("Uppercase -", count1)
print("Lowercase -", count2)
print("Spaces -", count3)

print("After changing cases:")
print(newstring)
```

Output:

```
In original String :
Uppercase - 4
Lowercase - 41
Spaces - 7
After changing cases:
gEEKSFORgEEKS IS A COMPUTER sCIENCE PORTAL FOR gEEKS
```

Q3(a): What are the rules for defining the function?

Ans: A function machine takes an input, and based on some rule produces an output.



The tables below show some input-output pairs for different functions. For each table, describe a function rule in words that would produce the given outputs from the corresponding inputs. Then fill in the rest of the table values as inputs and outputs which are consistent with that rule.

a. Input values can be any English word. Output values are letters from the English alphabet.

input	cat	house	you	stem			
output	t	e	u	Â	z	Â	Â

b. Input values can be any rational number. Output values can be any rational number.

input	2	5	−1.53	0	−4		
output	7	10	3.47	5	Â	8	Â

c. Input values can be any whole number. Output values can be any whole number.

input	1	2	3	4	5	6	7
output	2	1	4		Â	Â	Â

d. Input values can be any whole number between 1 and 365. Output values can be any month of the year.

input	25	365	35	95	330	66	Â
output	January	December	February	April	November		October

For at least one of the tables, describe a second rule which fits the given pairs but ultimately produces different pairs than the first rule for the rest of the table.

(b): Write a suitable program of our defined function in Python?

Ans: Functions that we define ourselves to do certain specific task are referred as user-defined functions. The way in which we define and call functions in Python are already discussed.
Functions that readily come with Python are called built-in functions. If we use functions written by others in the form of library, it can be termed as library functions.

All the other functions that we write on our own fall under user-defined functions. So, our user-defined function could be a library function to someone else.

Advantages of user-defined functions

1. User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
3. Programmars working on large project can divide the workload by making different functions.

Example of a user-defined function

```
# Program to illustrate
# the use of user-defined functions

def add_numbers(x,y):
   sum = x + y
   return sum

num1 = 5
num2 = 6

print("The sum is", add_numbers(num1, num2))
```

Run Code

Output

```
Enter a number: 2.4
Enter another number: 6.5
The sum is 8.9
```

Q4(a): . What are the rules for defining the function and Parameter passing to the function?

Ans: Functions that we define ourselves to do certain specific task are referred as user-defined functions. The way in which we define and call functions in Python are already discussed.

Functions that readily come with Python are called built-in functions. If we use functions written by others in the form of library, it can be termed as library functions.

All the other functions that we write on our own fall under user-defined functions. So, our user-defined function could be a library function to someone else.

Advantages of user-defined functions

1. User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
3. Programmars working on large project can divide the workload by making different functions.

Example of a user-defined function

```
# Program to illustrate
# the use of user-defined functions

def add_numbers(x,y):
   sum = x + y
   return sum

num1 = 5
num2 = 6

print("The sum is", add_numbers(num1, num2))
```

Run Code

Output

```
Enter a number: 2.4
Enter another number: 6.5
```

```
The sum is 8.9
```

Here, we have defined the function `my_addition()` which adds two numbers and returns the result.

(b): . Write a suitable program of our defined function by parameter passing in Python?

Ans: Functions that we define ourselves to do certain specific task are referred as user-defined functions. The way in which we define and call functions in Python are already discussed.

Functions that readily come with Python are called built-in functions. If we use functions written by others in the form of library, it can be termed as library functions.

All the other functions that we write on our own fall under user-defined functions. So, our user-defined function could be a library function to someone else.

Advantages of user-defined functions

1. User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
3. Programmars working on large project can divide the workload by making different functions.

Example of a user-defined function

```
# Program to illustrate
# the use of user-defined functions

def add_numbers(x,y):
```

```
    sum = x + y
    return sum

num1 = 5
num2 = 6

print("The sum is", add_numbers(num1, num2))
```

Run Code

Output

```
Enter a number: 2.4
Enter another number: 6.5
The sum is 8.9
```

Here, we have defined the function `my_addition()` which adds two numbers and returns the result.

This is our user-defined function. We could have multiplied the two numbers inside our function (it's all up to us). But this operation would not be consistent with the name of the function. It would create ambiguity.

It is always a good idea to name functions according to the task they perform.

In the above example, `print()` is a built-in function in Python.

Q5(a): What are return values to a Function discuss in detail?

Ans: To return a value from a custom function, you need to use the return keyword. We saw this in action recently in our random-canvas-circles.html example.
Our `draw()` function draws 100 random circles somewhere on an HTML `<canvas>`:

```
function draw() {
  ctx.clearRect(0, 0, WIDTH, HEIGHT)
  for (let i = 0; i < 100; i++) {
    ctx.beginPath()
    ctx.fillStyle = 'rgba(255,0,0,0.5)'
    ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI)
    ctx.fill()
  }
}
```

Inside each loop iteration, three calls are made to the `random()` function, to generate a random value for the current circle's *x-coordinate*, *y-coordinate*, and *radius*, respectively. The `random()` function takes one parameter — a whole number — and it returns a whole random number between `0` and that number. It looks like this:

```
function random(number) {
  return Math.floor(Math.random() * number)
}
```

This could be written as follows:

```
function random(number) {
  const result = Math.floor(Math.random() * number)
  return result
}
```

But the first version is quicker to write, and more compact.

We are returning the result of the calculation `Math.floor(Math.random() * number)` each time the function is called. This return value appears at the point the function was called, and the code continues.
So when you execute the following:

```
ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI)
```

If the three `random()` calls returned the values `500`, `200`, and `35`, respectively, the line would actually be run as if it were this:

```
ctx.arc(500, 200, 35, 0, 2 * Math.PI)
```

The function calls on the line are run first, and their return values substituted for the function calls, before the line itself is then executed.

(b): Write a suitable program of a Function with returning value?

Let's have a go at writing our own functions featuring return values.

sFirst of all, make a local copy of the function-library.html file from GitHub. This is a simple HTML page containing a text `<input>` field and a paragraph. There's also a `<script>` element, in which we have stored a reference to both HTML elements in two

variables. This little page will allow you to enter a number into the text box, and display different numbers related to it in the paragraph below.

1. Let's add some useful functions to this `<script>` element. Below the two existing lines of JavaScript, add the following function definitions:

```
function squared(num) {
  return num * num;
}

function cubed(num) {
  return num * num * num;
}

function factorial(num) {
  let x = num;
  while (x > 1) {
    num *= x-1;
    x--;
  }
  return num;

}
```

The `squared()` and `cubed()` functions are fairly obvious — they return the square or cube of the number that was given as a parameter. The `factorial()` function returns the factorial of the given number.

17. Next, we're going to include a way to print out information about the number entered into the text input. Enter the following event handler below the existing functions:

```
input.onchange = function() {
  const num = input.value;
  if (isNaN(num)) {
    para.textContent = 'You need to enter a number!';
  } else {
    para.textContent = num + ' squared is ' + squared(num) + '. ' +
                       num + ' cubed is ' + cubed(num) + '. ' +
                       num + ' factorial is ' + factorial(num) + '.';
  }

}
```

Here we are creating an `onchange` event handler. It runs whenever the `change` event fires on the text input — that is, when a new value is entered into the text `input`, and submitted (e.g., enter a value, then press tab). When this anonymous function runs, the value in the `input` is stored in the `num` constant.

Next, we do a conditional test. If the entered value is not a number, an error message is printed to the paragraph. The test looks at whether the expression `isNaN(num)` returns `true`. The `isNaN()` function to test whether the `num` value is not a number — if so, it returns `true`, and if not, it returns `false`.

If the test returns `false`, the `num` value is a number. Therefore, a sentence is printed out inside the paragraph element that states the square, cube, and factorial values of the number. The sentence calls the `squared()`, `cubed()`, and `factorial()` functions to calculate the required values.

27. Save your code, load it in a browser, and try it out.