Course No. : EE207

EE207-Computer Programming

Module 4 note

Module 4: Functions & Storage classes

Syllabus :-

Functions

- Functions – declaring
- defining and accessing functions
- parameter passing methods
- passing arrays to functions
- Recursion

Storage classes

- extern, auto
- register and static
- Example programs

- ✓ 7 Hours for this module(for 1st teaching)
- ✓ In semester exam 15% marks from this module
- ✓ In this module we studying user defined functions in C
- ✓ Reference ANSI C- Second edition

Functions and Recursion

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. A function is a named sequence of statements
that performs a desired operation.

Advantages of Function:

- It provides abstraction mechanism (hide details from the user)
- It eliminate redudunt (repetitious) code
- It hide complexity
- It support division of labour (top-down modular programming)

User Defined and Built-in functions:
The functions like print, type( ), help( ), float( ), int( ), str( ), etc., which is already available in Python library are called built-in functions But you can also create your own functions. These functions are called ***user-defined functions.***

Function Call
Using a function in our program is called function call. Eg:

```
>>> type ("32")
```

Here, *type( )* is the function name, by which we call (invoke) the function. "32" is called argument of the function. The result of the function call is the return value.

So function will take a value as input, process it and return the result.

Math Module
Python has a math module that provides most of the familiar mathematical functions. A module is a file that contains a collection of related functions grouped together.

Before we can use the functions from a module, we have to import them:

```
>>>import math
```

To call one of the functions, we have to specify the name of the module and the name of the function, separated by a dot, also known as a period. This format is called dot notation.

```
>>> math.sqrt(4)
2
```

Math Function	Description

math.log10 (x)	Return logarithm base 10
math.sin(x)	Return sine value of x
math.cos(x)	Return cos value of x
math.ln(x)	Return natural logarithm
math.sqrt(x)	Return square root of x

The functions can be combined to evaluate expressions:

```
>>> print tanx=math.sin(x)/math.cos(x)
```

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 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 body
        return [expression]
```

There can be any number of statements inside the function, but they have to be indented from the left margin. In the examples in this book, we will use an indentation of two spaces.

The list of parameters can be empty.

Example:
The following function has no parameters.

```
def printLine( ):
        print "_ _ _ _ _ _ _ _ _ _ _ _ _"
```

The next example shows a parameterized function.

```
def add(a,b):
        c=a+b
        return c
```

Actual and Formal Parameters

Global variable -- a variable defined in the "main" program. A variable is *defined* the first time it appears on the left-hand-side of an assignment statement.
Local variable -- a variable defined within a function. Local variables are temporary, scratch memory. When a function completes, its local variables are **de-allocated** (they die!). Consider the code below:

```
def func():
        x = 33
        return 99

x = 88
y = func()
```

The *x* defined within *func* is a local variable. The *x* and *y* defined with main are *global* variables.

Parameter -- Data sent to one function from another.
Formal parameter -- The parameter defined as part of the function definition, e.g., x in:

```
def cube(x):
        return x*x*x
```

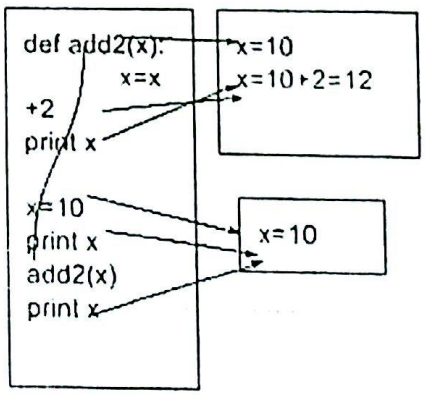
Actual Parameter -- The actual data sent to a function. It's found in the function call, e.g., 7 in

```
result = cube(7)    or y in:
y=5
res = cube(y)
```

When function is called, the value of actual parameter is copied to formal parameter, in the position order.
Changes in local variables does not affect actuals paramers

```
def  xChange (a,b):
        a,b=b,a


a=10
b=20
xChange(a,b)
print a.b
```



Output: 10,20 (because the variables a,b in xChange function are local to the function definition.

Consider this program:

```
def xChange(a,b):
        a,b=b,a
        return a,b
```

```
a=10
b=20
a,b=xChange(a,b)
print a,b
```

Output: 20,10 (because the updated values are returned to the variables.

Functions can be called inside another function. For example:

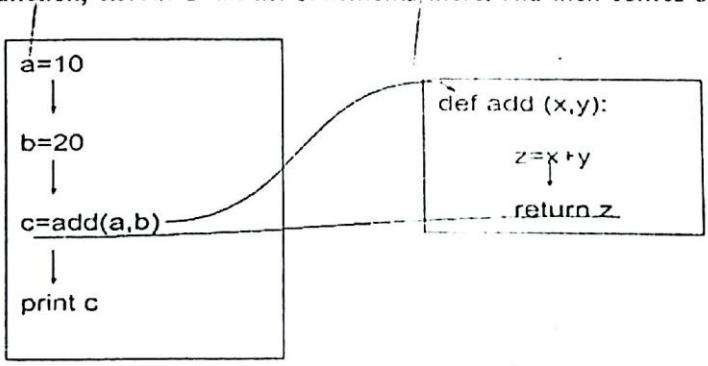
```
def printL( ):
        print "_____"


def nLine(n):
        i=0
        while i<n:
                printL( )
                i+=1
```

Flow of Execution

In order to ensure that a function is defined before its first use, you have to know the order in which statements are executed, which is called the flow of execution. Execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom.

When function is called, instead of going to the next statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.



Function call with Default argument

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 gives an idea on default arguments, it prints default age if it is not passed. Eg:

```
def add(a, b=10):
        c=a+b
        print c
```

add (10,20) will print 30 as normal. But,
add(10) will return 20. as it take default value for second arguament.

Exersices

1. Write a function that will check whether a number is odd or even
2. Write a function that will return largest among 2 numbers
3. Write a function that will check whether a number is prime or not
4. Write a function that will reverse the number (1234 -->4321)
5. Write a function that will return the sum of digits of a number
6. Write a menu driven calculator
7. Write a function to find the factorial of a number.

Solutions with Explanation

```
1. def oddEven(x):
        if x%2==0:
                return True
        else:
                return False
2. def largest(x,y):
        if x>y:
                return x
        else:
                return y
3. def isPrime(n):
        i=2
        while i<n:
                if n%i= = 0:
                        return False
        return True
4. def reverse(x):
        r=0
        while n>0:
                d=n%10
                r=r*10+d
                n=n/10
        return r
    5. def sumOf Digits(x):
s=0
while n>0:
d=n%10
        s=s+d
n=n/10
         return s
```

Recursion

A problem can be solved by decomposing problem into a set of simpler problems and solve these with different functions. In some cases, you can decompose a complex problem into smaller problems of exactly the same form. In these cases, the subproblems can all be solved by using the same function. This design strategy is called recursive design, and the resulting functions are called recursive functions.

A recursive function is a function that calls itself. To prevent a function from repeating itself indefinitely, it must contain at least one selection statement. This statement examines a condition called a base case to determine whether to stop or to continue with another recursive step.

```
def recursive(n):
        if n==0:
                return
        print "Recursion of ", n
        recursive(n-1)

recursive(5)
```

Output:
Recursion of 5
Recursion of 4
Recursion of 3
Recursion of 2
Recursion of 1

Most recursive functions expect at least one argument. This data value is used to test for the base case that ends the recursive process, and also is modified in some way before each recursive step. The modification of the data value should produce a new data value that allows the function to reach the base case eventually.

Factorial using recursion

We know that

n!= n *(n-1) * (n-2)*.....*1
  =n* (n-1)!

And we konw that 1!=1

So, we can define recursion as:

n!= 1 if n==1 otherwise, n* (n-1)!

So the function will look like:

```
def fact(n):
        if n==1:
```