PYTHON

TAJENDAR ARORA

FUNCTIONS 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.

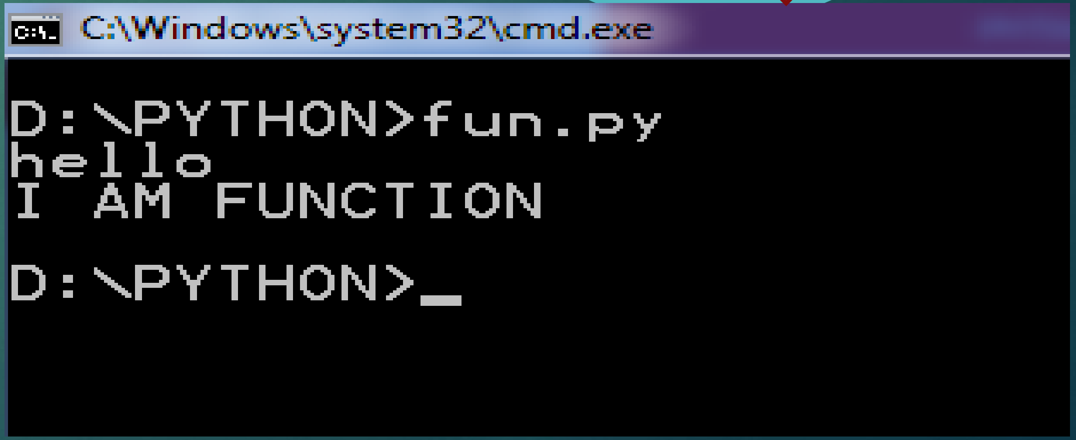
The code block within every function starts with a colon (:) and is indented.

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

FUNCTION WITH (NO PASS NO RETURN)

```
def hello():
    print ('hello‘)
    print ('I AM FUNCTION‘)
    return;
hello()
```

OUTPUT


C:\Windows\system32\cmd.exe

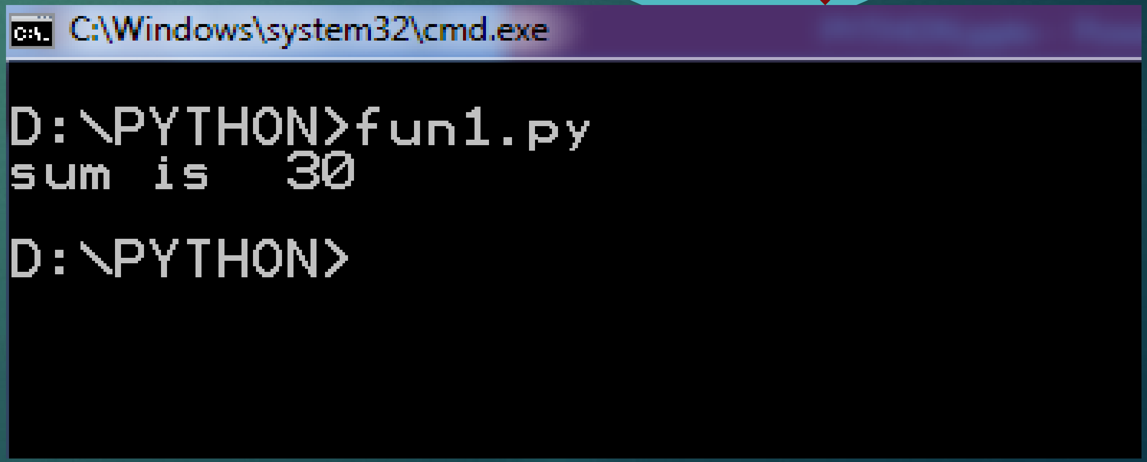
```
D:\PYTHON>fun.py
hello
I AM FUNCTION

D:\PYTHON>_
```

FUNCTION WITH (VALUE PASS BUT NO RETURN)

```
def sum(a,b):
    c=a+b
    print ("sum is ",c)
    return;
sum(10,20)
```

OUTPUT


C:\Windows\system32\cmd.exe

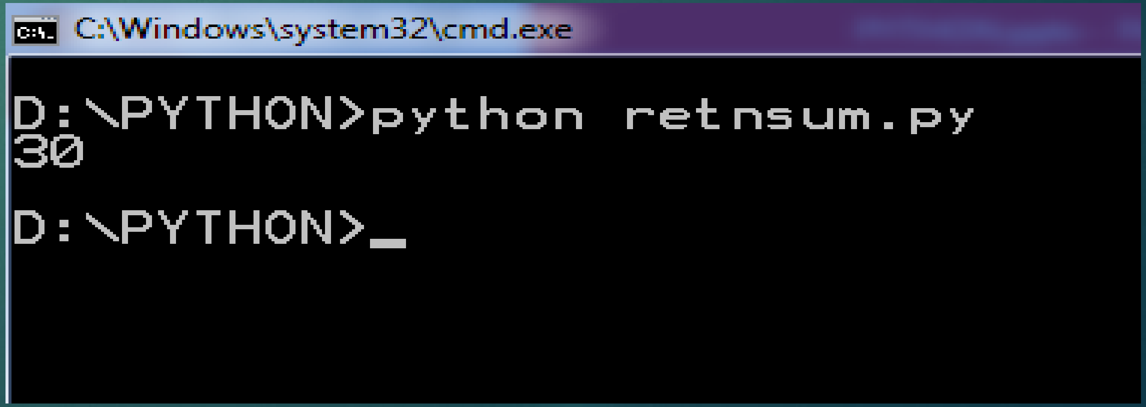
```
D:\PYTHON>fun1.py
sum is  30

D:\PYTHON>
```

FUNCTION WITH (VALUE PASS WITH VALUE RETURN)

```
def sum(a,b):
    c=a+b
    return c;
x=sum(10,20)
print x;
```

OUTPUT



```
D:\PYTHON>python retnsum.py
30

D:\PYTHON>_
```

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

```
def printinfo( name, age = 35 ):
print ("Name: ", name;)
print ("Age ", age;)
return;
printinfo( age=50, name="miki" );
printinfo( name="miki" );
```

FUNCTION VARIABLE ARGUMENTS

An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

```
def printinfo(*vartuple):
      sum=0
      for var in vartuple:
        sum=sum+var
        print (var)
      Print (sum)
Return

printinfo(10);
printinfo(70,60,50);
```

Lambda FUNCTION

- *Lambda operator or lambda function* is used for creating small, one-time and anonymous function objects in Python.
- Basic syntax
- lambda arguments : expression
- lambda arguments : expression *lambda* operator can have any number of arguments, but it can have only one expression. It cannot contain any statements and it returns a function object which can be assigned to any variable.

- add = lambda x, y : x + y
- print add(2, 3) # Output: 5

- a = [1, 2, 3, 4, 5, 6]
- print (filter(lambda x : x % 2 == 0, a)) # Output: [2, 4, 6]

Local Vs. Global variable

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

```
total = 0; # This is global variable.
def sum( arg1, arg2 ):
total = arg1 + arg2; # Here total is local variable.
print ("Inside the function local total : ", total )
return total;

# Now you can call sum function
sum( 10, 20 );
print ("Outside the function global total : ", total )
```

MODULES

we can define the function in modules which give the flexiblity to reuse in other programs.

Creating Module

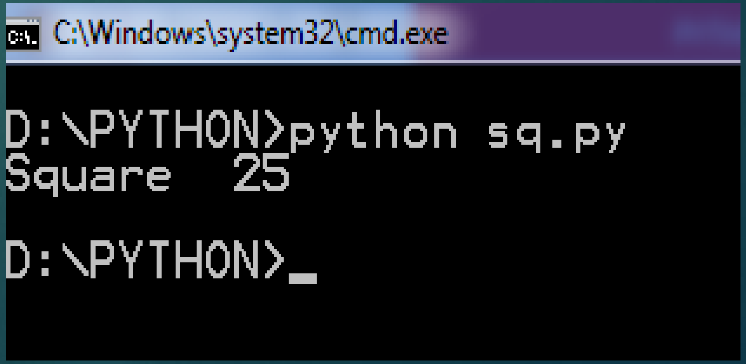
```
def square(a):
    c=a*a;
    print ("Square ",c)


    return;
```

It is used in

```
import mymodule
mymodule.square(5);
```

OUTPUT


C:\Windows\system32\cmd.exe

```
D:\PYTHON>python sq.py
Square  25

D:\PYTHON>_
```

Command Line Arguments

- The Python sys module provides access to any command-line arguments via the sys.argv. This serves two purposes:
- sys.argv is the list of command-line arguments.
- len(sys.argv) is the number of command-line arguments.
- Here sys.argv[0] is the program i.e. script name.

Command Line Arguments

- **Example**
- import sys
- print ('Number of arguments:', len(sys.argv), 'arguments.')
- print ('Argument List:', str(sys.argv))

- **$ python test.py arg1 arg2 arg3**
- **This produces the following result:**
- **Number of arguments: 4 arguments.**
- **Argument List: ['test.py', 'arg1', 'arg2', 'arg3']**

OS Module

- Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files.
- To use this module you need to import it first and then you can call any related functions.

The rename() Method

The *rename() method takes two arguments, the current filename and the new filename.*

Syntax

```
os.rename(current_file_name, new_file_name)
```

OS Module

- You can use the *remove() method to delete files by supplying the name of the file to be deleted as the argument.*

```
os.remove(file_name)
```

OS Module

- The mkdir()Method

You can use the *mkdir() method of the os module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.*

Syntax

```
os.mkdir("newdir")
```

OS Module

- The *rmdir() Method*
- The *rmdir() method deletes the directory, which is passed as an argument in the method.*
- Before removing a directory, all the contents in it should be removed.
- Syntax
- os.rmdir('dirname')