Object Oriented Programming using Python (I)

Lecture(2)

Functions

Prepared by: ***Dr. Rula Amjed Hamid***
University of Information Technology and Communications
College of Business Informatics

Writing Function

- There are two aspects to every Python function:

Function definition: contains the code that determines the function's behavior

```
def name ( parameter_list ) :
    block
```

- **Function invocation**: A function is used within a program via a function invocation

Writing Function

function definition consists of three parts:

• **Name** The name is an identifier . As with variable names, the name chosen for a function should describe its functionality.

• **Parameters** every function definition specifies the parameters that it accepts from callers. The parameters appear in a parenthesized comma-separated list. The list of parameters is empty if the function requires no information from code that calls the function.

• **Body** The body contains the code to execute when clients invoke the function. The code within the body is responsible for producing the result, and may return value to the client.

Writing Function

example

```
def table7():
    n = 1
   while n <11 :
          print(n * 7, end =' ')
          n = n +1
```

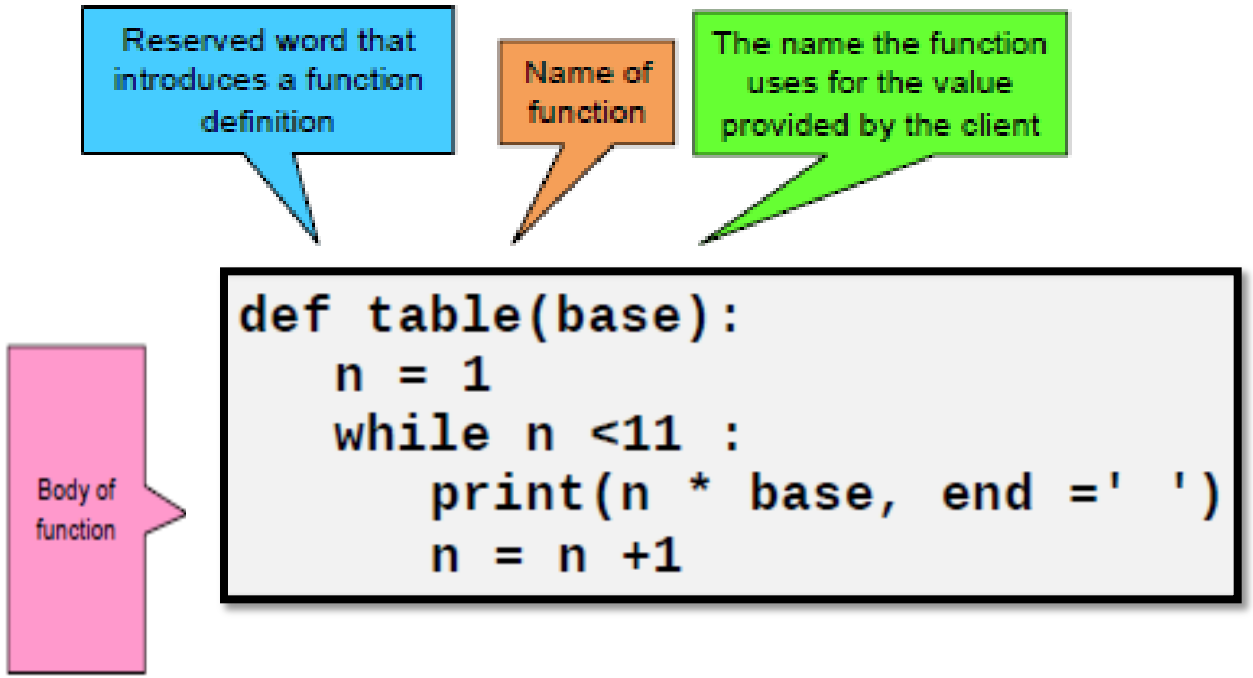
Function definition

```
table7()
```

Function invocation

```
7 14 21 28 35 42 49 56 63 70
```

examples




```
>>> table(13)
13 26 39 52 65 78 91 104 117 130

>>> table(9)
9 18 27 36 45 54 63 72 81 90
```

examples

function that return value

```
#  Definition of the prompt function
def prompt():
    value = int(input("Please enter an integer value: ")
    return value

print("This program adds together two integers.")
value1 = prompt()     #  Call the function
value2 = prompt()     #  Call the function again
sum = value1 + value2
print(value1, "+", value2, "=", sum)
```

examples

function that passing parameter and return value

```
# Definition of the prompt function
def prompt(n):
    value = int(input("Please enter integer" ))
    return value

print("This program adds together two integers.")
value1 =prompt(1) # Call the function
value2 =prompt(2) # Call the function again
sum = value1 + value2
print(value1, "+", value2, "=", sum)
```

practical program

- This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

```
Emil Refsnes
```

- This function expects 2 arguments, but gets only 1:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

```
Traceback (most recent call last):
  File "demo_function_args_error.py", line 4, in <module>
    my_function("Emil")
TypeError: my_function() missing 1 required positional argument: 'lname'
```

- If the number of arguments is unknown, add a * before the parameter name:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
The youngest child is Linus
```

- You can Passing a List as an Argument
- You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
- E.g. if you send a List as an argument, it will still be a List when it reaches the function:

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

→

```
apple
banana
cherry
```

Using Local Variables

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

Global variables

Local variables

Global vs. Local variables

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

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

Using Local Variables

```
def fun1(x):
    print("x is " ,x)
    x=2
    print("change local x to ",x)
x=50
fun1(x)
print("x is still",x)
```

X in main program is differ than x that define within a function

```
>>>
x is  50
change local x to  2
x is still 50
>>>
```

Using Local Variables

```
def increment(x):
    print("Beginning execution of increment, x =", x)
    x += 1 # Increment x
    print("Ending execution of increment, x =", x)

def main():
    x = 5
    print("Before increment, x =", x)
    increment(x)
    print("After increment, x =", x)

main()
```

The result

```
>>>
Before increment, x = 5
Beginning execution of increment, x = 5
Ending execution of increment, x = 6
After increment, x = 5
>>>
```

Using the global statement

Is used to declare a global variable when we give the value of the variable in the function of this change it appears in the main program

```
def fun1():
    global x
    print("x is " ,x)
    x=2
    print("change global x to ",x)
x=50
fun1()
print("value of x is ",x)
```

```
>>>
x is  50
change global x to  2
value of x is  2
>>>
```