Lecture 09: User Defined Functions - II

Swakkhar Shatabda

B.Sc. in Data Science
Department of Computer Science and Engineering
United International University

January 17, 2024

Local Variables

- A function's parameters and variables defined in its block are all local variables—they can be used only inside the function and exist only while the function is executing.
- Trying to access a local variable outside its function's block causes a NameError, indicating that the variable is not defined.

```
x = 6 # global variable - accessible from anywhere

def helloFunc(n):
    m = x  * n # local variable
    return m

print(helloFunc(5))
print(m) # this line will produce a name error
```

Find Output - I

```python
x = 30

def funcu(x):
    x = 2
    return 2

print(funcu(x))
print(x)
```

Find Output - II

```
x = 30

def funcX():
    global x
    x = 2

funcX()
print(x)
```

What is a tuple?

- An immutable list is called a tuple.

```
teamScore = (282,7)
print(teamScore[0])
print(teamScore[1])
print(len(teamScore))
```

We can not change values in a tuple.

```
teamScore = (282,7)
teamScore[0]=306
```

```
TypeError: 'tuple' object does not support item assignment
```

However, we can redefine the tuple.

```
teamScore = (282,7)
teamScore = (306,7)
```

Arbitrary Argument Lists

- Functions with arbitrary argument lists, such as built-in functions min and max, can receive any number of arguments.
- The * before the parameter name tells Python to pack any remaining arguments into a tuple that's passed to the args parameter.

```
def varFunc(*x):
    return sum(x)

print(varFunc(1,2,3,4,5))
print(varFunc(3,4,5))
```

Returning a list

```python
def funcList():
    myList = []
    for i in range(5):
        myList.append(i)
    return myList

x = funcList()
print(x)
```

```
[0, 1, 2, 3, 4]
```

Passing Arguments

- Python arguments are always passed by reference.
- When a function call provides an argument, Python copies the argument object's reference—not the object itself—into the corresponding parameter.

```
def simpleFunc(x):
    print(id(x))

n = 5
print(id(n))
simpleFunc(n)
```

```
4353916456
4353916456
```

Call by Object Reference - Immutable

```python
def simpleFunc(x):
    print("Inside function x:",x)
    print("Inside function  id(x):",id(x))
    x = 9
    print("Inside function after change x:",x)
    print("Inside function after change id(x):",id(x))

n = 5
print("Before function call n:",n)
print("Before function call id(n):",id(n))

simpleFunc(n)

print("After function call n:",n)
print("After function call id(n):",id(n))
```

Call by Object Reference - Mutable

- When a reference to a mutable object like a list is passed to a function, the function can modify the original object in the caller.

```
def funcMutable(x):
    x.append(50)
listA =[10,20,30,40]
print(listA)
funcMutable(listA)
print(listA)
```

If you dont want the function to change it call like the following.

```
funcMutable(listA[:])
```

This passes the function a copy of the list, not the original. Any changes the function makes to the list will affect only the copy, leaving the original list intact.

Modules

- Storing your functions in a separate file called a module and then importing that module into your main program.
- An import statement tells Python to make the code in a module available in the currently running program file.
- Storing your functions in a separate file allows you to hide the details of your program's code and focus on its higher-level logic.

```
import module_name
```

You can store all your functions in a file called module_name.py and import all functions in another working file as above and call them as following.

```
module_name.function_name()
```

Modules

- Or alternatively import specific function / functions only

```
from module_name import function_name
from module_name import function_0, function_1, function_2
```

- If the name of a function you're importing might conflict with an exist- ing name in your program, or if the function name is long, you can use a short, unique alias—an alternate name similar to a nickname for the function.
- You can also provide an alias for a module name.

```
from module_name import function_name as fn
import module_name as mn
```

Modules

- You can tell Python to import every function in a module by using the asterisk (*) operator.

```python
from module_name import *
```