NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Week 6
Course Review & Advanced Topics

Functions

Instructor: Asst. Prof. LIN Shang-Wei
Email: shang-wei.lin@ntu.edu.sg

Abstraction in Different Aspects

Abstraction in Data: **Data Structures**

Programs = Algorithms + Data Structures

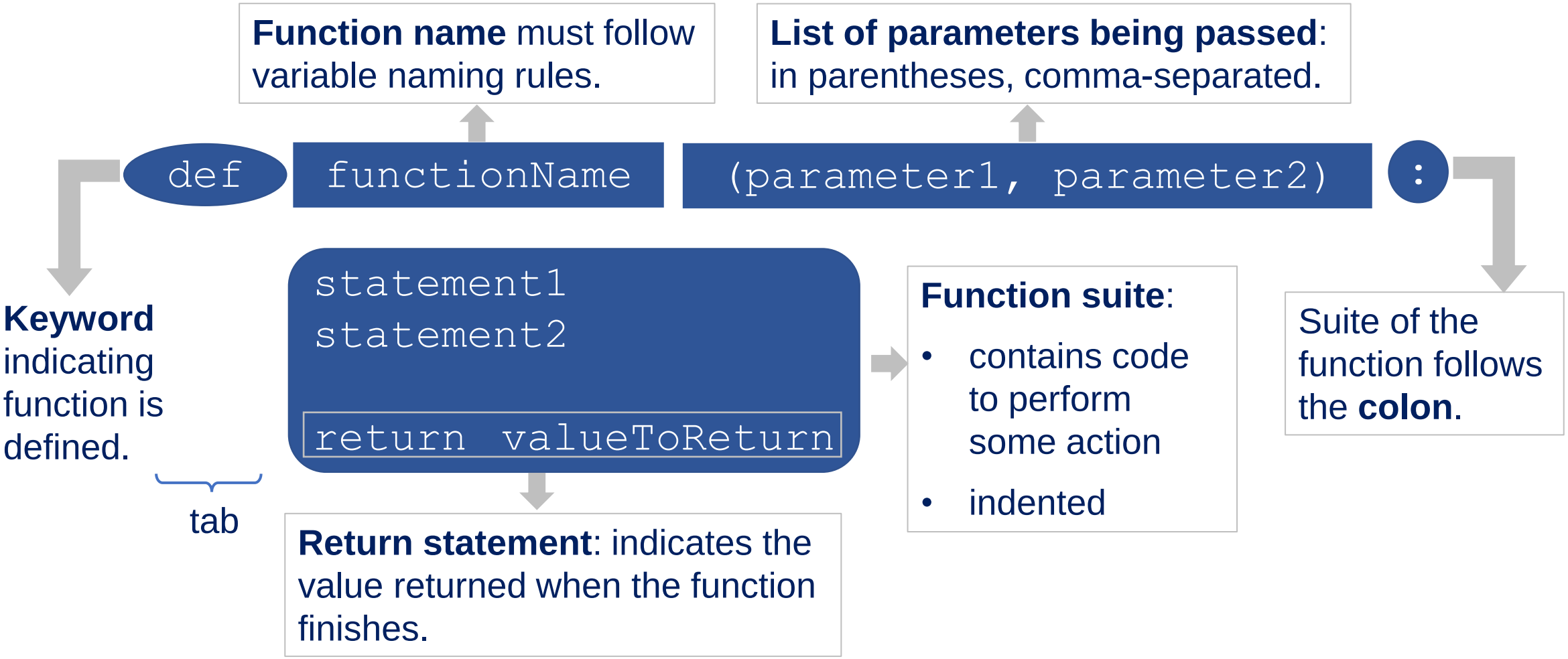
(1976, Niklaus Wirth)

Abstraction in Algorithms:

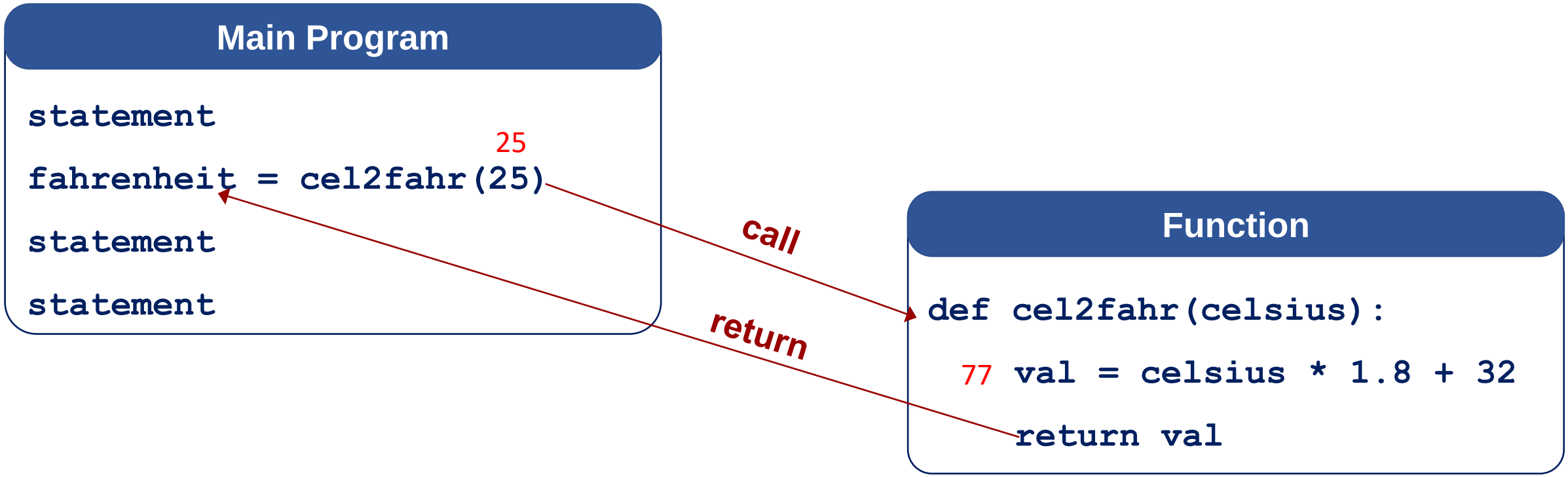
Functions

Function Basics

Function Definition in Python



Dynamics of Function Calls



Q1: What is the output of the following Python program?

```
charList = ['a', 'e', 'i', 'o', 'u']
myStr = "This is a string!"

def funcA(content, target):
    num = 0

    for char in content:
        if char in target:
            num += 1

    return num


result = funcA(myStr, charList)
print(result)
```

A. 0
B. 1
C. 2
D. 3
✔ E. 4
F. 5

Q2: What is the output of the following Python program?

```
myStr = "Hello"
result = 0

def funcA(content):
    num = 0

    for char in content:
        num += 1

    return num


result = funcA(myStr)
print(result)
```

5

- ✔ A.0
- B.1
- C.2
- D.3
- E.4
- F.5

Advanced Topics

Scope and Namespace

Scope

"The set of program statements over which a variable exists, i.e. can be referred to."

It is about understanding, for any variable, what its associated value is.

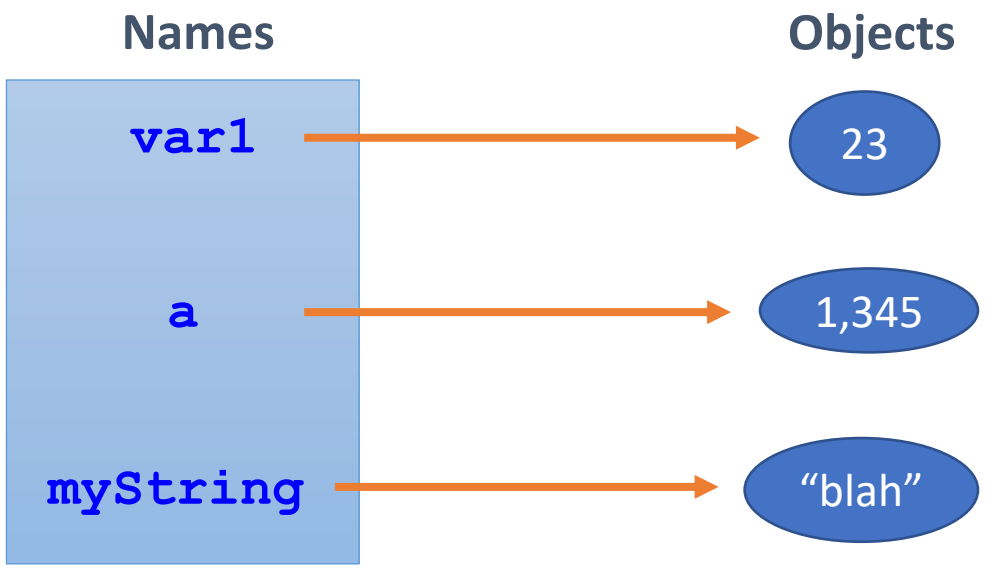
A function has its own scope: its function suite (body)

```
def myFunc(param1, param2):
    statement1
    statement2
    …
```

Namespace

Namespace is an association of name and objects

It looks like a dictionary, and for the most part it is (at least for modules and classes).



Scope vs. Namespace

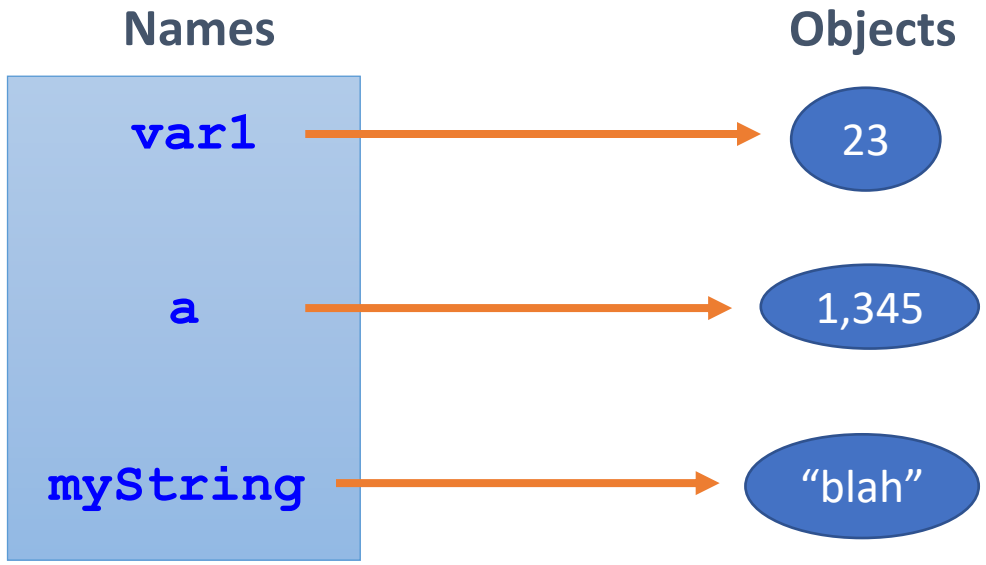
The same thing, but from different point views

Physical point of view vs. **Logical point of view**

```
def myFunc(param1, param2):
```

```
var1 = 23
a = 1.345
myString = 'blah'
```

vs.



LEGB Rule

For Python, there are potentially multiple namespaces that could be used to determine the object associated with a variable name.

Which one to use?

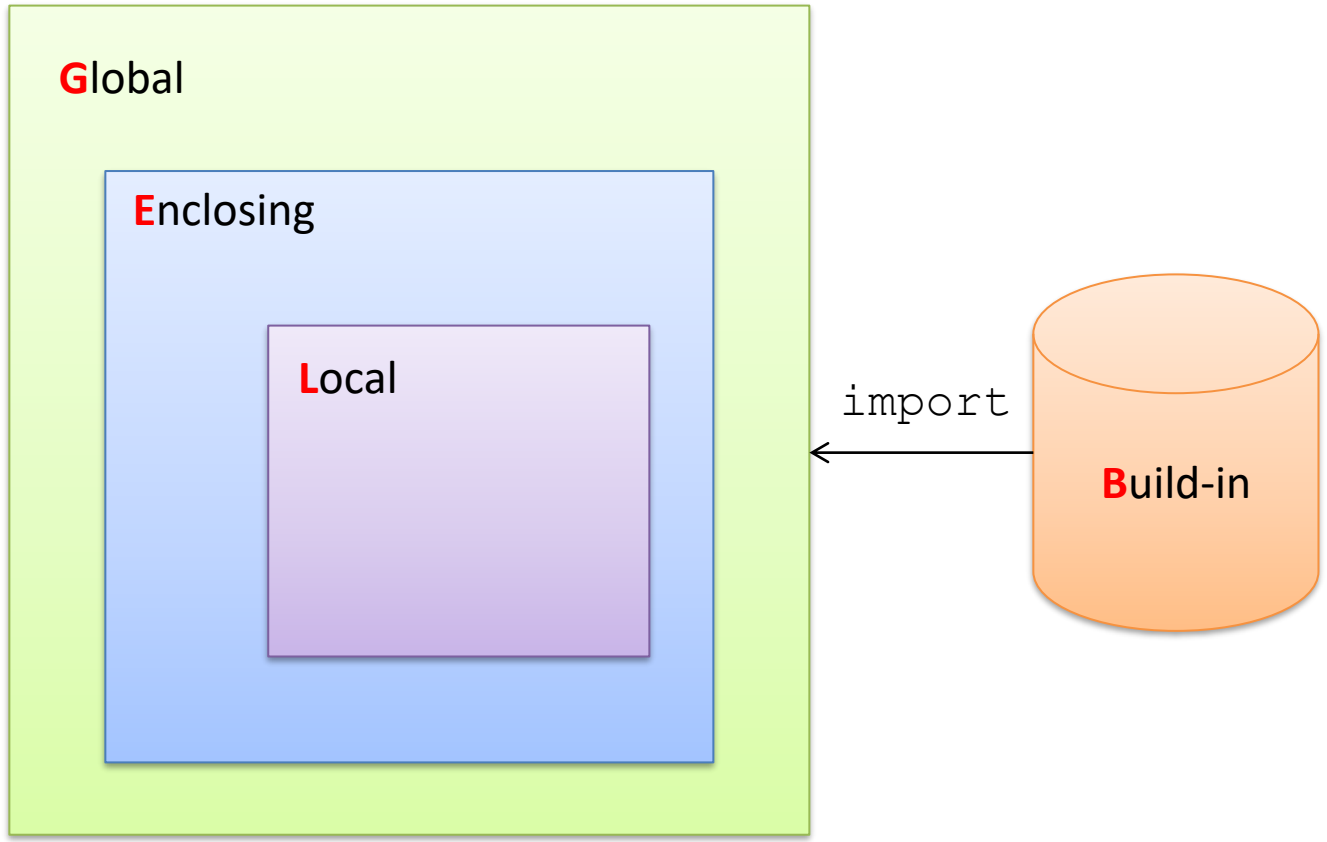
The search order of namespaces for a name:

- **Local : inside the function in which it was defined.**
- **Enclosing : If not there, enclosing/encompassing. Is it defined in an enclosing function?**
- **Global : If not there, is it defined in the global namespace?**
- **Built-in : Finally, check the built-in, defined as part of the special built-ins scope.**
- **Else, ERROR.**

LEGB Rule (cont.)

search order:

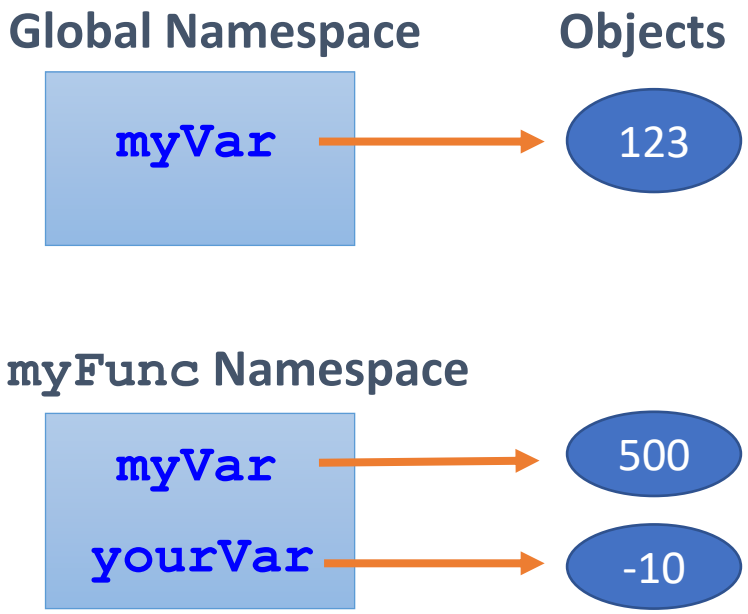
Local
↓
Enclosing
↓
Global
↓
Build-in
↓
ERROR



Local and Global

If a reference is assigned in a function, then that reference is only available within that function.

If a reference with the same name is provided outside the function, the reference is reassigned




```
myVar = 123                          # global

def myFunc():
    myVar = 500                      # local
    yourVar = -10                    # local
    print(myVar, yourVar)

myFunc()                             # prints 500 -10
print(myVar)                         # prints 123
print(yourVar)                       # ERROR
```

Enclosing

```
def enclosing():
    myVariable = 'defined by enclosing'

    def enclosed():
        print('scope: ' + myVariable)

    enclosed()

enclosing()
```

```
scope: defined by enclosing    # output
```

The `Global` Keyword

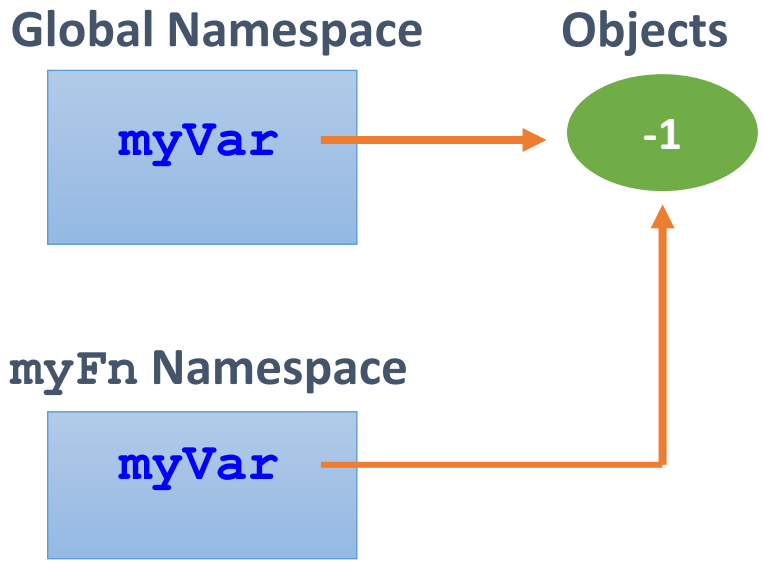
What if you really want to use or modify the global variables inside a function?

Ans: Using the `global` keyword in a function allows you to access global objects.

```
myVar = 100

def myFn():
    global myVar
    myVar = -1

myFn()
print(myVar)          # prints -1
```



Functions and Scope

Function's Scope

Each function maintains a namespace for names defined locally within the function.

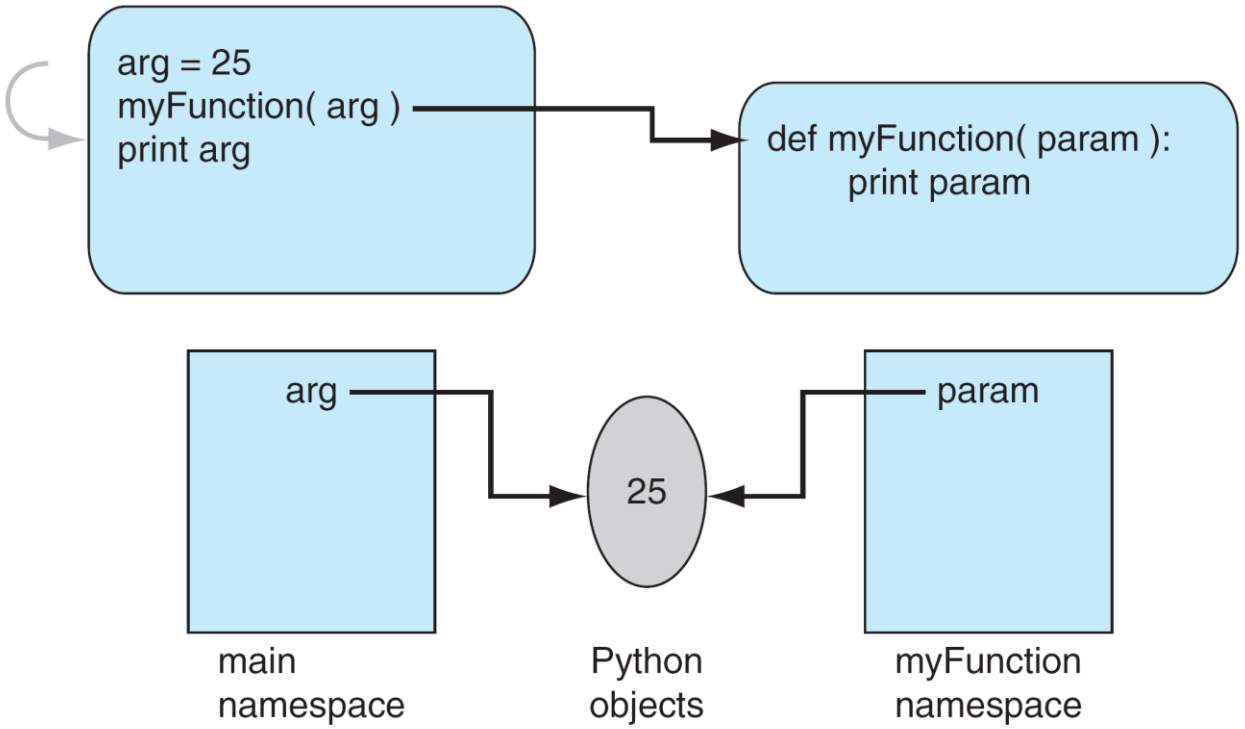
"Locally" means one of two things:

- a name assigned something within the function
- an argument received by invocation of the function

Parameters vs. Arguments

For each argument in the function invocation, the argument's associated object is passed to the corresponding parameter in the function.

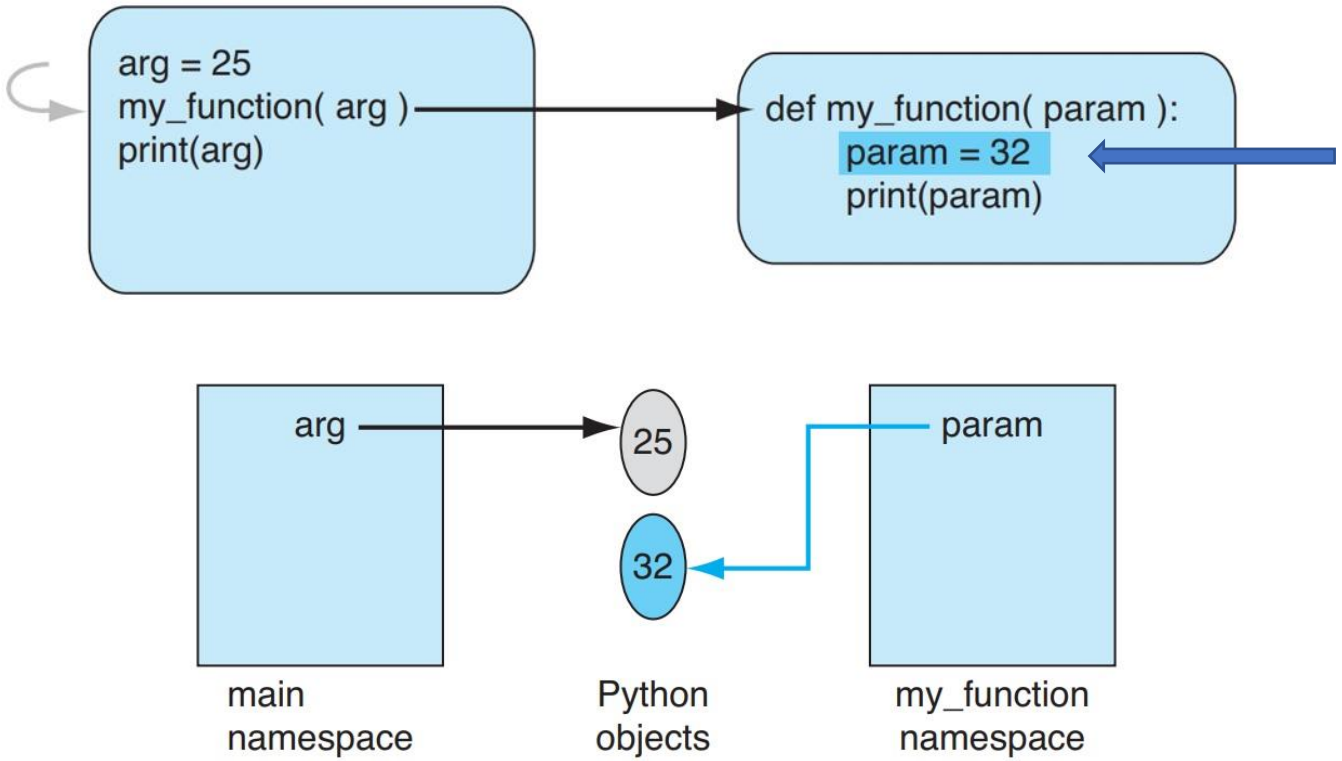
The argument and the parameter share an association with the same object.



Parameters vs. Arguments (cont.)

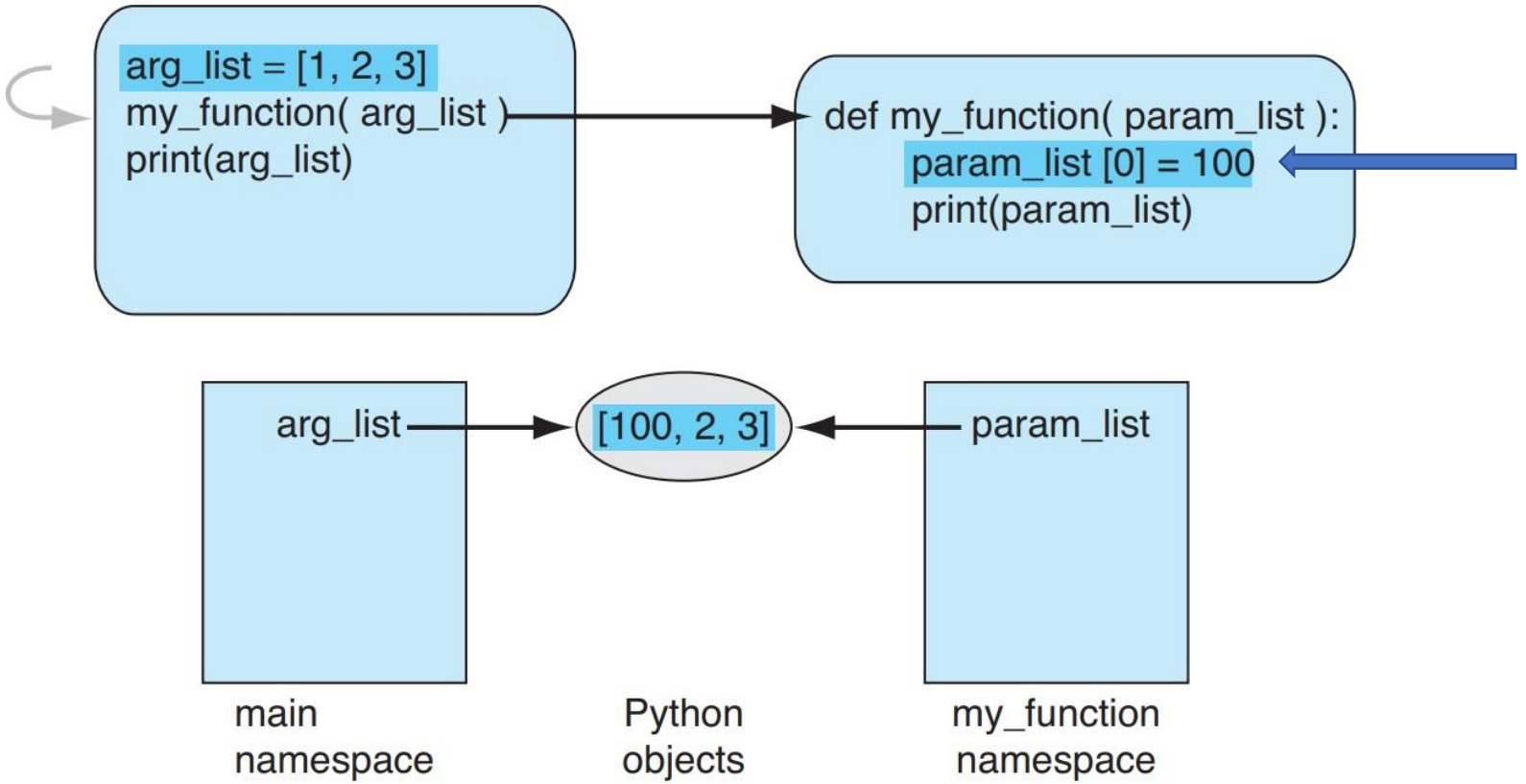
If a parameter is assigned to a new value, then a new association is created.

This assignment does not affect the object associated with the argument.



Parameters vs. Arguments (cont.)

When passing a mutable data structure, if the shared object is modified, both the parameter and the argument will reflect that change.



Q3: Are the results of the two program the same?

```
def myFun (param):
    param.append(4)
    return param


myList = [1,2,3]
newList = myFun(myList)
print(myList,newList)
```

Program A

```
def myFun (param):
    param=[1,2,3]
    param.append(4)
    return param

myList = [1,2,3]
newList = myFun(myList)
print(myList,newList)
```

Program B

A. Yes

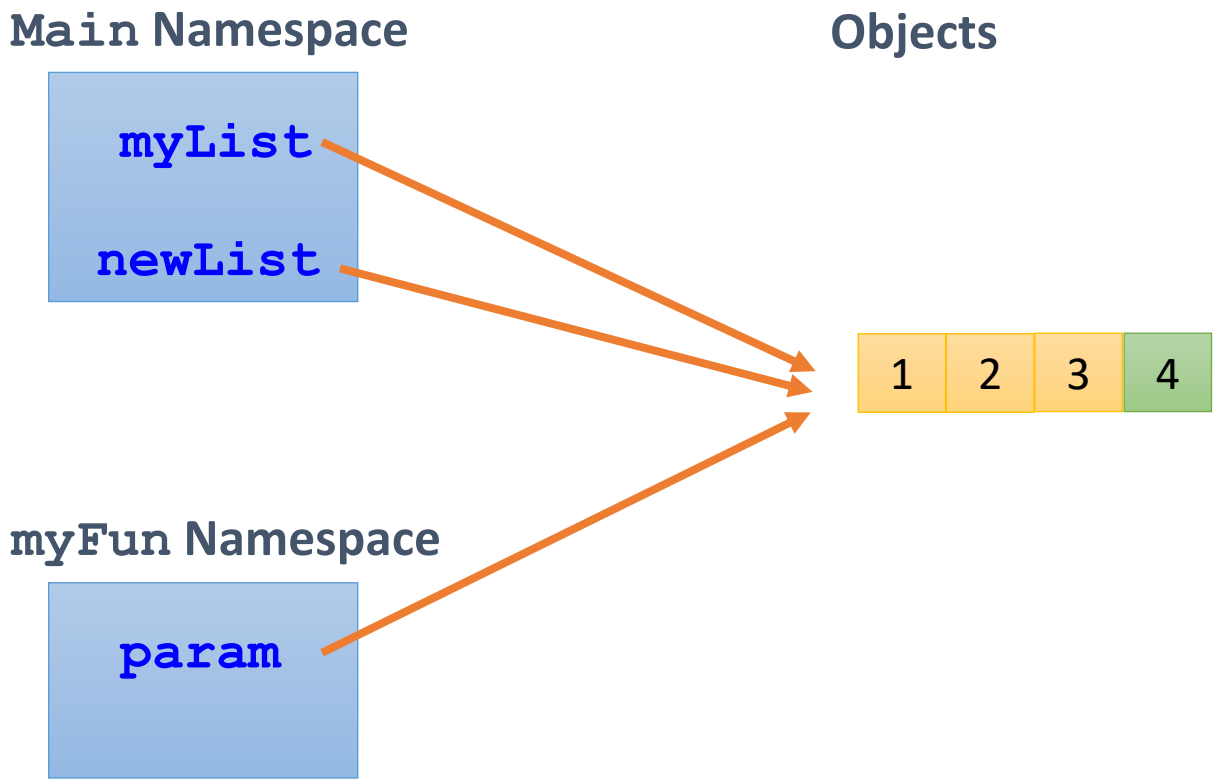
✔ B. No

Program A

```
def myFun (param):
    param.append(4)
    return param


myList = [1,2,3]
newList = myFun(myList)
print(myList,newList)
```

Program A



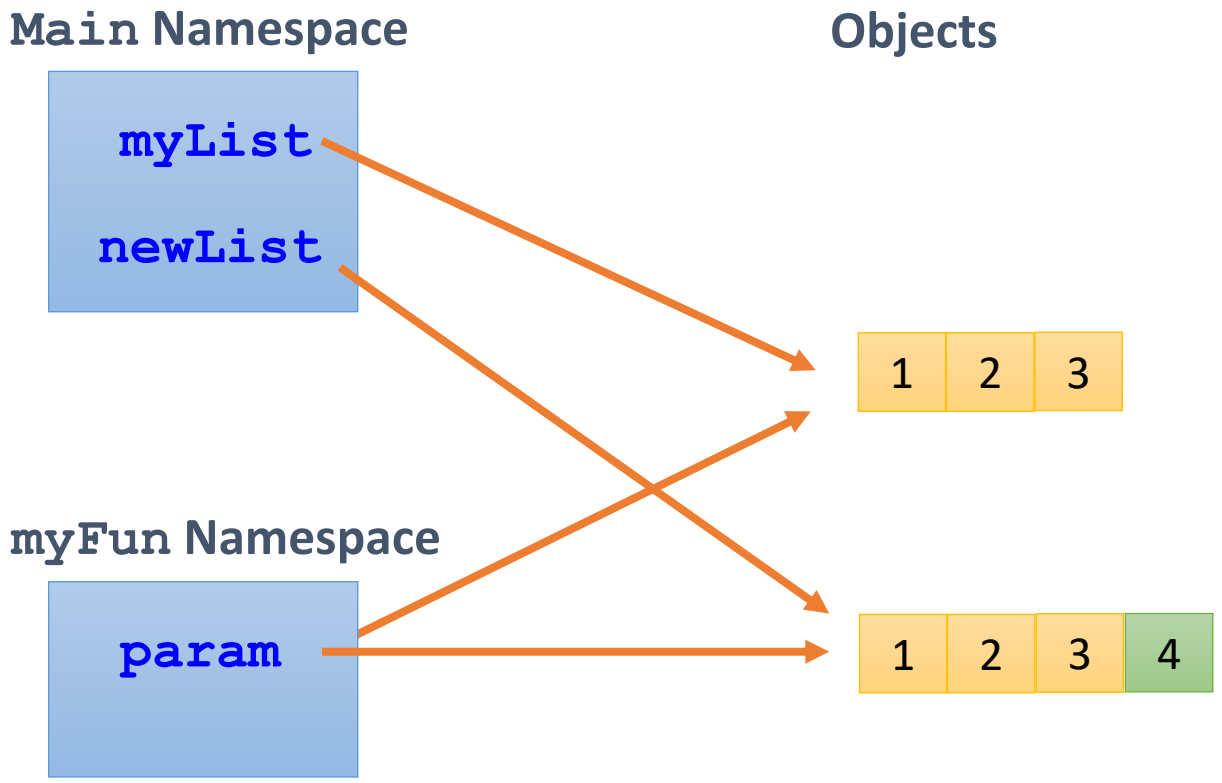

```
[1,2,3,4] [1,2,3,4]      # output
```

Program B

```
def myFun (param):
    param=[1,2,3]
    param.append(4)
    return param

myList = [1,2,3]
newList = myFun(myList)
print(myList,newList)
```

Program B




```
[1,2,3] [1,2,3,4] # output
```

Q4: What is the output of the following Python program?

```
myVar = 127

def myFun (myVar):
    myVar = 7
    print('myVar: ', myVar)

myFun(myVar)
print('myVar: ', myVar)
```

A. myVar: 7
   myVar: 7

B. myVar: 127
   myVar: 127

✔ C. myVar: 7
   myVar: 127

D. myVar: 127
   myVar: 7

Q5: What is the output of the following Python program?

```
myVar = 127

def myFun ():
    a = myVar + 1
    print('a: ', a)

myFun()
```

✔ A.a: 128
B.a: 127
C.ERROR

Q6: What is the output of the following Python program?

```
myVar = 127

def myFun ():
        myVar = myVar + 1

myFun()
print(myVar)
```

How to fix it?

```
myVar = 127

def myFun ():
    global myVar
    myVar = myVar + 1

myFun()
print(myVar)
```

A. 128
B. 127
✔ C. ERROR

local variable 'myVar'
referenced before assignment