Lecture 5

CORE ELEMENTS PART V: FUNCTIONS PARAMETERS & VARIABLES SCOPE

Overview

- Function Parameters
- Function: reading the small prints
- Variable Scope

Passing Parameters

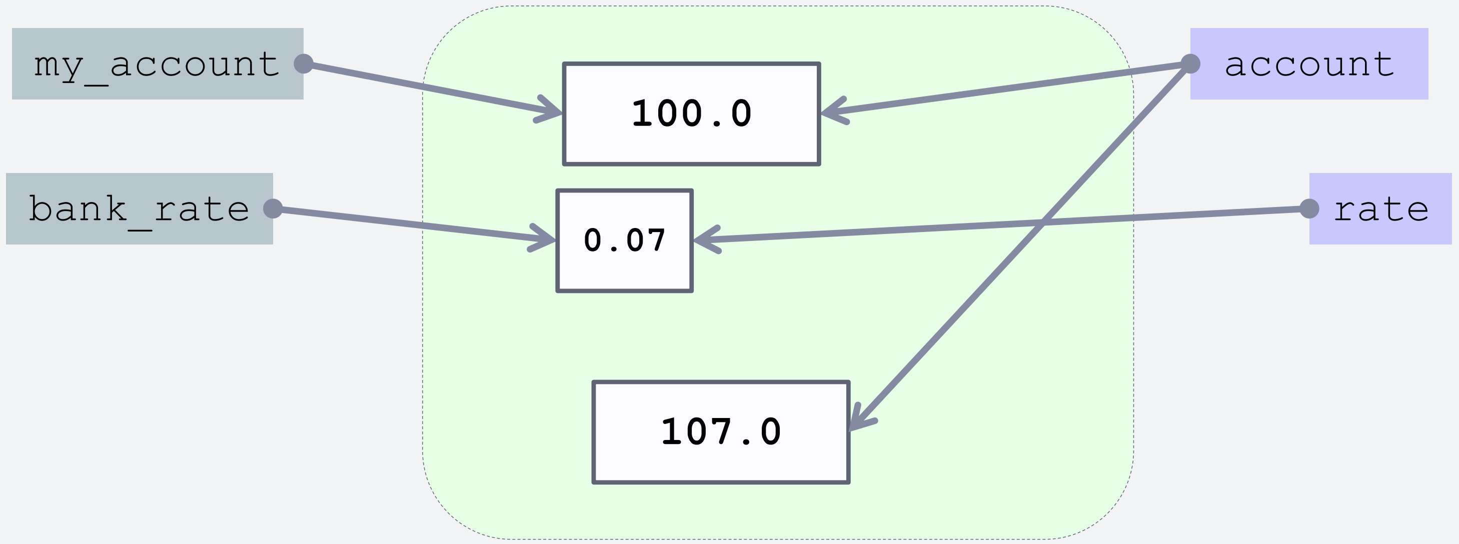
- Code in Python interpreter
  - File: TPOP_2014_15_Lecture5_parameterPassing.py

Passing Parameters

Code

```
def addInterestOne(account, rate):
        account =  account * (1+rate)

my_account = 100.0
bank_rate = 0.07
addInterestOne(my_account, bank_rate)
print "new accounts balance:", my_account
```

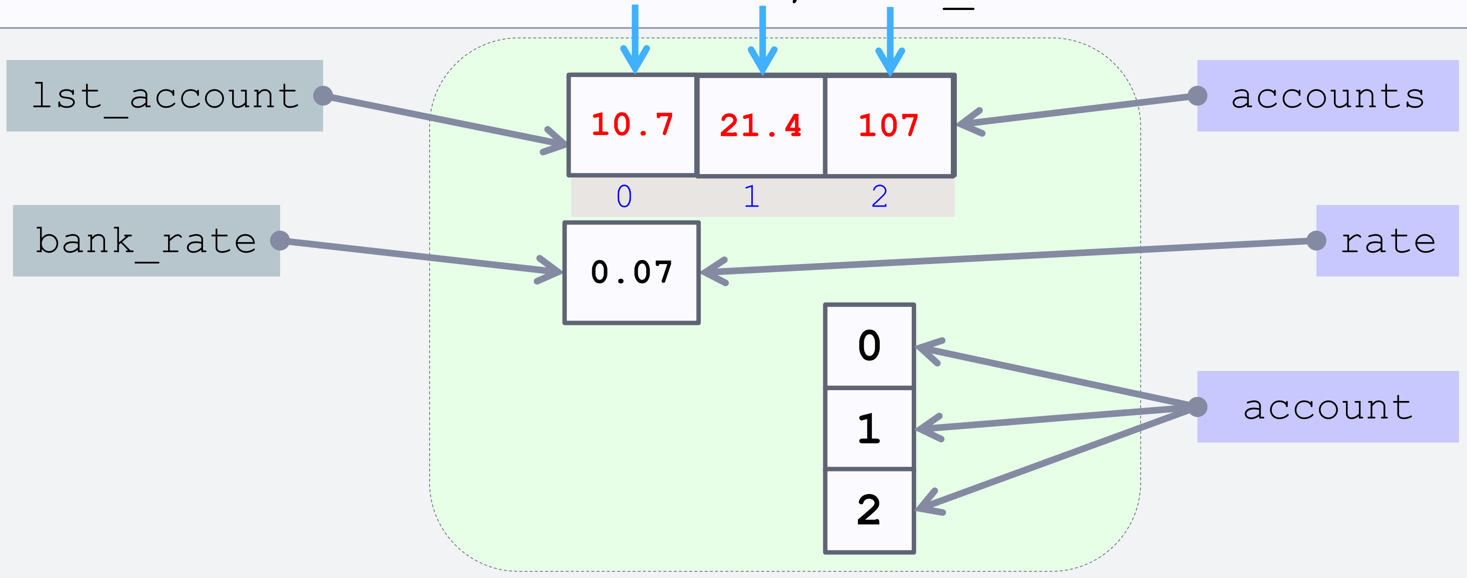


Passing Parameters

Code

```
def addInterestAll(accounts, rate):
    for account in range(len(accounts)):
        accounts[account] *= (1+rate)

lst_accounts = [10, 20, 100]
bank_rate = 0.07
addInterestAll(lst_accounts, bank_rate)
print "new accounts balance:", lst_account
```

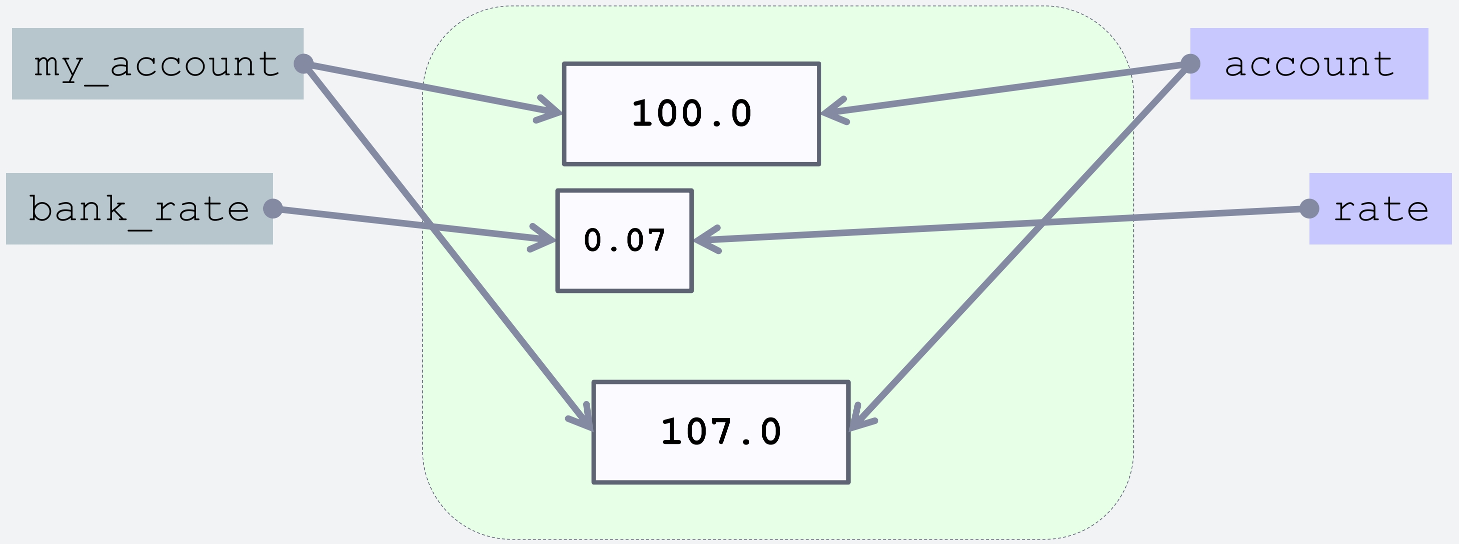


Passing Parameters

Code

```
def addInterestOne(account, rate):
        account =  account * (1+rate)
        return account

my_account = 100.0
bank_rate = 0.07
my_account = addInterestOne(my_account, bank_rate)
print "new accounts balance:", my_account
```



Function Design Concepts

- Use arguments for inputs and return for outputs
- Use global variables only when absolutely necessary
- Do NOT change mutable arguments unless the caller expect it

Functions

THE SMALL PRINTS

Variable Scope

- Code in Python interpreter
  - File: TPOP_2014_15_Lecture5_python_scope.py

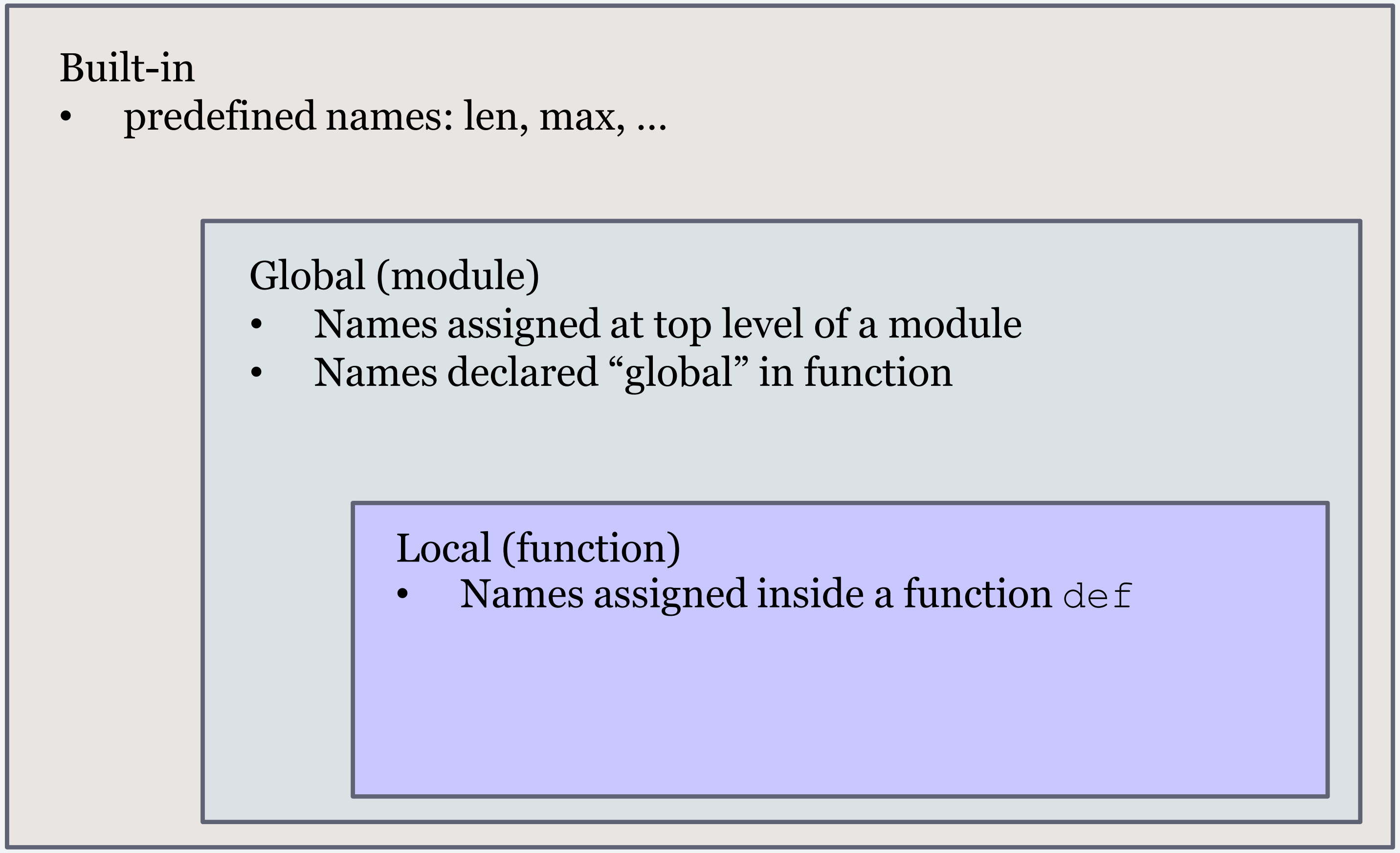
Namespace

In order to avoid clashes between names (variables) inside the function and outside the function, function define a nested namespace

- Functions define **local** scopes
- Modules define **global** scopes
- Each call to a function is a **new** local scope
- **Assigned** names in a function are **local**, unless declared **global**
- Names **not assigned** a value in the definition function are assumed to be **global**

Namespace & Variable Scope

- Python’s three scopes



Built-in
- predefined names: len, max, …

Global (module)
- Names assigned at top level of a module
- Names declared “global” in function

Local (function)
- Names assigned inside a function `def`

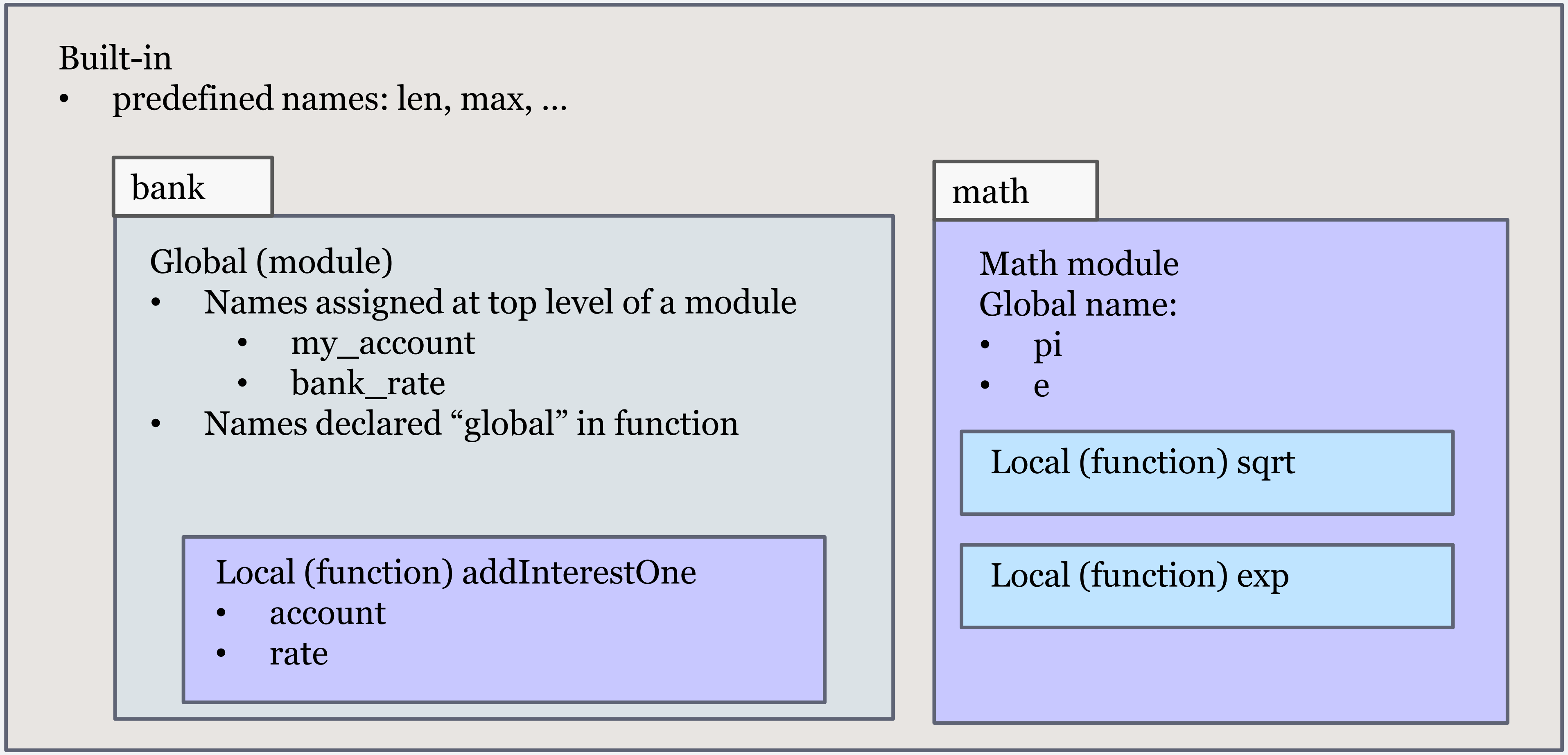
Name Resolution: LGB rule

Name/variable references search at most 3 scopes:

- **Local**
- **Global**
- **Built-in**

Namespace & Variable Scope

- Python's three scopes


Built-in
• predefined names: len, max, …
bank
Global (module)
• Names assigned at top level of a module
• my_account
• bank_rate
• Names declared "global" in function
Local (function) addInterestOne
• account
• rate
math
Math module
Global name:
• pi
• e
Local (function) sqrt
Local (function) exp

Name Resolution: LGB rule

When you use an unqualified name inside a function, Python searches the local (L), then the global (G), and then the built-in (B) scopes and stop at the first place the name is found

see **`change_global1(x)`** and **`change_global2(x)`** in **`python_scope.py`**

Name Resolution: LGB rule

When you assign a name in a function (e.g. `result = 1.0`), Python always creates or change the name in the **local** scope, unless it's declared to be **global** in that function.

see **`change_global4(x)`** and **`change_global3(x)`** in **`python_scope.py`**

Name Resolution: LGB rule

When outside a function, the local scope is the same as the global, e.g. the module’s namespace.

Name Resolution: LGB rule

- To summarise:

Code

```
x, y, v = 1, 3, 7

def the_global_thing(z):
    global u
    v =  5
    u = x + y * (z - v)
```

  - Global names are ???
  - Local names are ???

Summary

- Parameter passing
  - Immutable object passed by value
  - Mutable object passed by reference

- Namespace and scopes
  - Three scopes
  - **Local**
  - **Global**
  - **Built-in**

Exercises

Try to understand the scope of each object and what is really happening in the code provided in the file:

- TPOP_2014_15_Lecture5_python_scope_exercises.py

Compare and Discuss your findings with one of your peers.

More on Function

Warning

You do not need to know or understand this right now.
You may not even use it this year

READ the following slide ONLY if you are confident with what we have seen so far on function

Special Argument-Matching Modes

- Positional: matched left to right
  - What we have seen so far

- Keywords: matched by argument name

Code

```
def my_func(name, age = 18, nationality = ‘French’):
    print name, age, nationality

my_func(‘Lilian Blot’, age = 21)
my_func(name = ‘toto’, nationality = ‘UK’)
my_func(‘titi’, 5, ‘US’)
my_func(‘nono’, 30)
```

Special Argument-Matching Modes

- Defaults: specify values for arguments that are not passed

- varargs: catch unmatched positional or keyword arguments

Code

```
def my_args_func(*args): #unmatched positional argument
    print args

def my_args2_func(**args): # unmatched keyword argument
    print args
```