

Scopes

- ❖ Python creates, changes, or looks up the name in what is known as a *namespace*—a place where names live
- ❖ Python uses the location of the assignment of a name to associate it with (i.e., *bind* it to) a particular namespace
- ❖ If a variable is assigned inside a def, it is *local* to that function
- ❖ If a variable is assigned in an enclosing def, it is *nonlocal* to nested functions
- ❖ If a variable is assigned outside all defs, it is *global* to the entire file

1

Scope Details

- ❖ The enclosing module is a global scope
- ❖ The global scope spans a single file only
- ❖ Assigned names are local unless declared global or nonlocal
- ❖ All other names are enclosing function locals, globals, or built-ins
- ❖ Each call to a function creates a new local scope

- ❖ Python's name-resolution scheme is sometimes called the *LEGB rule*:
 - Local
 - Enclosing and lambdas
 - Global
 - Built-ins

2

The global Statement

- ❖ `global` allows us to *change* names that live outside a `def` at the top level of a module file

```
X = 88                                # Global X

def func():
    global X
    X = 99                            # Global X: outside def

func()
print(X)                              # Prints 99
```

3

Scopes and Nested Functions

- ❖ **A reference** (`X`) looks for the name `X` first in the current local scope (function); then in the local scopes of any lexically enclosing functions in your source code, from inner to outer; then in the current global scope (the module file); and finally in the built-in scope (the module builtins).

```
X = 99                                # Global scope name: not used

def f1():
    X = 88                            # Enclosing def local
    def f2():
        print(X)                    # Reference made in nested def
        f2()
    f1()

f1()                                  # Prints 88: enclosing def local
```

4

The nonlocal Statement in 3.X

- ❖ nonlocal both allows assignment to names in enclosing function scopes and limits scope lookups for such names to enclosing defs.

```
>>> def tester(start):
    state = start          # Referencing nonlocals works normally
    def nested(label):
        print(label, state) # Remembers state in enclosing scope
    return nested

>>> F = tester(0)
>>> F('spam')
spam 0
>>> F('ham')
ham 0
```

5

The nonlocal Statement in 3.X

```
>>> def tester(start):
    state = start
    def nested(label):
        print(label, state)
        state += 1          # Cannot change by default (never in 2.X)
    return nested

>>> F = tester(0)
>>> F('spam')
UnboundLocalError: local variable 'state' referenced before assignment

>>> def tester(start):
    state = start          # Each call gets its own state
    def nested(label):
        nonlocal state     # Remembers state in enclosing scope
        print(label, state)
        state += 1        # Allowed to change it if nonlocal
    return nested

>>> F = tester(0)
>>> F('spam')            # Increments state on each call
spam 0
>>> F('ham')
ham 1
>>> F('eggs')
eggs 2
```

6