Python Programming

Lecture 7 Variable Scope, Advanced Functions

7.1 Variable Scope

Variable Scope (变量的作用域)

In Python, module, class, def, lambda can introduce new variable scope, if/elif/else/, try/except, for/while will not introduce new variable scope.

```
if True:
    msg = 'I am from Shanghai'
print(msg) # We can use msg.
```

```
def test():
    msg = 'I am from Shanghai'
print(msg) # error
```

Global and Local（全局和局部）

```
msg_loc = "Shanghai" # Global
def test():
    msg = 'I am from Shanghai' # Local
```

New Assignment

```
1 msg = 'I am from Shanghai'
2 def test():
3     msg = 'I am from Beijing'
4     print(msg)
5 test()
6 print(msg)
```

```
I am from Beijing
I am from Shanghai
```

Reference（引用）

```
1 msg = 'I am from Shanghai'
2 def test():
3     print(msg)
4 test()
```

```
I am from Shanghai
```

Referenced before assignment

```
1 msg = 'I am from Shanghai'
2 def test():
3     print(msg)
4     msg = 'I am from Beijing'
5 test()
```

```
UnboundLocalError: local variable 'msg'
referenced before assignment
```

- How to modify the variable outside? The global keyword

```
num = 1
def fun():
    global num
    num = 123
    print(num)
fun()
print(num)
```

```
123
123
```

```
num = 1
def fun():
    print(num)
    global num
    num = 123
    print(num)
fun()
```

```
SyntaxError: name 'num' is used
prior to global declaration
```

```python
a = 10
def test():
    a = a + 1
    print(a)
test()
```

```
UnboundLocalError: local variable 'a'
referenced before assignment
```

```python
a = 10
def test():
    global a
    a = a + 1
    print(a)
test()
```

```
11
```

```python
a = 10
def test():
    a = 10
    a = a + 1
    print(a)
test()
print(a)
```

```
11
10
```

mutable vs. immutable

```
1 a = [1,2,3]
2 def test():
3     print(a)
4     a = [1,2,3,4]
5 test()
```

```
UnboundLocalError: local variable 'a'
referenced before assignment
```

```
1 a = [1,2,3]
2 def test():
3     print(a)
4     a.append(4)
5 test()
6 print(a)
```

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

```
1 a = {"color": "green"}
2 def test():
3     print(a)
4     a["color"] = "red"
5     a["position"] = "left"
6 test()
7 print(a)
```

```
{'color': 'green'}
{'color': 'red', 'position': 'left'}
```

Parameter and Argument

- For immutable objects (number, string, tuple), the assignment inside a function cannot change the value of the variable outside the function even if the variable name inside is the same with the variable name outside.

```
1 def ChangeInt(a):
2     a = 10
3     return a
4 
5 b = 2
6 print(ChangeInt(b))
7 print(b)
```

```
10
2
```

```
1 def ChangeInt(b):
2     b = 10
3     return b
4 
5 b = 2
6 print(ChangeInt(b))
7 print(b)
```

```
10
2
```

```
def ChangeInt(a):
    a = 10
    print(a)
    return a

b = 2
ChangeInt(b)
print(b)
print(ChangeInt(b))
```

```
10
2
10
10
```

```
def ChangeInt(a):
    a = 10
    print(a)

b = 2
ChangeInt(b)
print(b)
print(ChangeInt(b))
```

```
10
2
10
None
```

- For mutable objects (like lists and dictionaries), assigning a new value to a variable inside a function will not change the original variable outside the function. (the same with immutable objects!)

```
1 def changeme(mylist):
2     mylist = [1,2]
3     print("inside: ", mylist)
4
5 x = [10,20]
6 changeme(x)
7 print("outside: ", x)
```

```
inside: [1, 2]
outside: [10, 20]
```

- However, modifying the contents of the mutable object (e.g., adding elements to a list or updating a dictionary) can change the value of the variable outside the function.

```
1 def changeme(mylist):
2     mylist.extend([1,2])
3     print ("inside: ", mylist)
4
5 x = [10,20]
6 changeme(x)
7 print ("outside: ", x)
```

```
inside: [10, 20, 1, 2]
outside: [10, 20, 1, 2]
```

7.2 Nested Functions（嵌套函数）

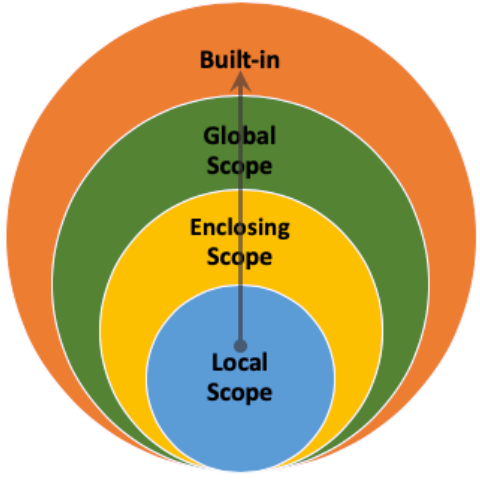
Nested Functions

```
1
2 def outer():
3     def inner():
4         num = 100
5         print(num)
6     inner()
7 outer()
8
9
```

```
1 num = 3             # Global（全局）
2 def outer():
3     num = 10        # Enclosing（外层）
4     def inner():
5         num = 100 # Local（局部）
6         print(num)
7     inner()
8     print(num)
9 outer()
```

- （引用顺序）Local→Enclosing→Global→ Build-in (LEGB principle)

```
1 num = 3
2 def outer():
3     num = 10
4     def inner():
5         print(num)
6     inner()
7     print(num)
8 outer()
9
10 # output?
```



nonlocal keyword

```
def outer():
    num = 10
    def inner():
        nonlocal num
        # nonlocal keyword
        num = 100
        print(num)
    inner()
    print(num)
outer()
```

```
100
100
```

```
def outer():
    num = 10
    def inner():
        global num
        num = 100
        print(num)
    inner()
    print(num)
outer()
```

```
100
10
```

```
def outer():
    global num
    num = 10
    def inner():
        global num
        num = 100
        print(num)
    inner()
    print(num)
outer()
```

```
100
100
```

```
def outer():
    global num
    num = 10
    def inner():
        num = 100
        print(num)
    inner()
    print(num)
outer()
```

```
100
10
```

```python
num = 3
def outer():
    global num
    num = 10
    def inner():
        num = 100
        print(num)
    inner()
    print(num)
outer()
print(num)
```

```
100
10
10
```

```python
num = 3
def outer():
    num = 10
    def inner():
        global num
        num = 100
        print(num)
    inner()
    print(num)
outer()
print(num)
```

```
100
10
100
```

```python
num = 3
def outer():
    num = 10
    def inner():
        nonlocal num
        num = 100
        print(num)
    inner()
    print(num)
outer()
print(num)
```

```
100
100
3
```

```python
num = 3
def outer():
    global num
    num = 10
    def inner():
        global num
        num = 100
        print(num)
    inner()
    print(num)
outer()
print(num)
```

```
100
100
100
```

7.3 Advanced Functions

Arbitrary Arguments（任意数量的实参）

```
1 def make_pizza(toppings):
2     print(toppings)
3 
4 make_pizza('pepperoni')
```

- If we want to pass multiple arguments, we may create a list at first.

```
1 x=['mushrooms', 'green peppers', 'extra cheese']
2 def make_pizza(toppings):
3     for y in toppings:
4         print(y, end=', ')
5 
6 make_pizza(x)
```

```
1 def make_pizza(topping_1, topping_2, topping_3):
2     print(topping_1, topping_2, topping_3)
3 
4 make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

- However, sometimes we are not sure about the exact number of the arguments.

- We can pass multiple arguments at once in the following way.

```
def make_pizza(*toppings):
    print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

- Note that Python packs the arguments into a tuple, even if the function receives only one value.

```
def make_pizza(*toppings):
    for topping in toppings:
        print("- " + topping)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

```
- pepperoni
- mushrooms
- green peppers
- extra cheese
```

- Mixing Positional and Arbitrary Arguments
- If you want a function to accept several different kinds of arguments, the parameter that accepts an arbitrary number of arguments must be placed last in the function definition.

```
def make_pizza(size, *toppings):
    print(str(size) + "-inch pizza:")
    for topping in toppings:
        print("- " + topping)

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

```
16-inch pizza:
- pepperoni
12-inch pizza:
- mushrooms
- green peppers
- extra cheese
```

- Using Arbitrary Keyword Arguments

```
def build_profile(**user_info):
    print(user_info)

build_profile(location='Shanghai',field='Management')
```

```
{'location': 'Shanghai', 'field': 'Management'}
```

- Using Arbitrary Keyword Arguments

```
def build_profile(first, last, **user_info):
    profile = {}
    for key, value in user_info.items():
        profile[key] = value
    return first, last, profile

user_profile = build_profile('albert', 'einstein',
                             location='princeton',
                             field='physics')
print(user_profile)
```

```
('albert', 'einstein',
{'location': 'princeton', 'field': 'physics'})
```

```python
def build_profile(*name, **user_info):
    print(name)
    print(user_info)

build_profile('albert', 'einstein',
              location='princeton',
              field='physics')
```

```
('albert', 'einstein')
{'location': 'princeton', 'field': 'physics'}
```

- As a tradition, we often use *args and **kw

- Example

```
1  def test(x,y=1,*a,**b):
2      print(x,y,a,b)
3
4  test(1)
5  test(1,2)
6  test(1,2,3)
7  test(1,2,3,4)
8  test(x=1,y=2)
9  test(1,a=2)
10 test(1,2,3,a=4)
11 test(1,2,3,y=4)
```

- Result

```
1 1 () {}
1 2 () {}
1 2 (3,) {}
1 2 (3, 4) {}
1 2 () {}
1 1 () {'a': 2}
1 2 (3,) {'a': 4}
TypeError: test() got multiple values
```

Map（映射函数）

```
def f(x):
    return x * x

y = map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])
print(list(y))
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
print(list(map(str, [1, 2, 3, 4, 5, 6, 7, 8, 9])))
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9']
```

filter（过滤函数）

```
def k(x):
    return x%2 == 0

y = list(filter(k, [1, 2, 3, 4, 5, 6, 7, 8, 9]))
print(y)
```

```
[2, 4, 6, 8]
```

Anonymous function: lambda（匿名函数）

```
def add( x, y ):
    return x + y

lambda x, y: x + y

lambda x, y = 2: x + y
lambda *z: z
```

- Sometimes the anonymous function is convenient.
- It has only one expression. (You do not have to use return)

```
print(list(map(lambda x: x * x, [1, 2, 3, 4, 5, 6, 7, 8, 9])))
```

```
print(list(filter(lambda x: x%2==0, [1, 2, 3, 4, 5, 6, 7, 8, 9])))
```

List Comprehension（列表生成式）

- [expression for value in iterable if condition]

```
>>> [x*x for x in range(1, 11)]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
>>> [x*x for x in range(1, 11) if x%2==0]
[4, 16, 36, 64, 100]
```

```
#if-else和只有if的顺序不同
>>> [x*x if x%2==0 else x**3 for x in range(1, 11)]
[1, 4, 27, 16, 125, 36, 343, 64, 729, 100]
```

```
>>> [m+n for m in 'ABC' for n in 'XYZ']
['AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'CX', 'CY', 'CZ']
```

```
>>> d = {'x': 'A', 'y': 'B', 'z': 'C' }
>>> [k + '=' + v for k, v in d.items()]
['y=B', 'x=A', 'z=C']
```

```
>>> L = ['Hello', 'World', 'IBM', 'Apple']
>>> [s.lower() for s in L]
['hello', 'world', 'ibm', 'apple']
```

Summary

- Functions
  - Reading: Python for Everybody, Chapter 4
  - Reading: Python Crash Course, Chapter 8