2017-05-26 Fri 15:48, by 田宇伟

else块、nonlocal声明及复制

目录

- 1. if 语句之外的 else 块
- 2. nonlocal 声明
- 3. 浅复制与深复制
  - 3.1. 为任意对象做深复制和浅复制

1 if 语句之外的 else 块

else 子句不仅能在 if 语句中使用，还能在 for 、 while 、 try 语句中使用。

else 子句的行为：

for

仅当 for 循环正常运行完毕时（即 for 循环没有被 break 语句终止）才运行 else 块。

```
from random import randrange


def insertion_sort(seq):
    if len(seq) <= 1:
        return seq

    _sorted = seq[:1]
    for i in seq[1:]:
        inserted = False
        for j in range(len(_sorted)):
            if i < _sorted[j]:
                _sorted = [*_sorted[:j], i, *_sorted[j:]]
                inserted = True
                break
        if not inserted:
            _sorted.append(i)
    return _sorted


print(insertion_sort([randrange(1, 100) for i in range(10)]))
```

```
[10, 24, 30, 43, 54, 64, 66, 76, 83, 99]
```

```
from random import randrange


def insertion_sort(seq):
    if len(seq) <= 1:
        return seq
    _sorted = seq[:1]
    for i in seq[1:]:
        for j in range(len(_sorted)):
            if i < _sorted[j]:
                _sorted = [*_sorted[:j], i, *_sorted[j:]]
                break
        else:
            _sorted.append(i)
    return _sorted


print(insertion_sort([randrange(1, 100) for i in range(10)]))
```

```
[3, 15, 16, 16, 18, 52, 57, 62, 63, 68]
```

while

仅当 `while` 循环因为条件为 **假值** 而退出时（即 `while` 循环没有被 `break` 语句终止）才运行 `else` 语句。

```
while False:
    print('Will never print!')
else:
    print('Loop failed!')
```

```
Loop failed!
```

try

仅当 `try` 块中没有异常抛出时才运行 `else` 块。

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by 0!")
    else:
        print("result = {}".format(result))
    finally:
        print("divide finished!")


divide(2, 1)
print('-' * 16)
divide(2, 0)
```

```
result = 2.0
divide finished!
----------------
division by 0!
divide finished!
```

在所有的情况下，如果异常或者 `return` 、 `break` 或 `continue` 语句导致控制权跳到了复合语句的主块之外， `else` 语句也会被跳过。

2 nonlocal 声明

```
def make_averager():
    series = []

    def averager(new_value):
        series.append(new_value)
        total = sum(series)
        return total/len(series)

    return averager


avg = make_averager()
print(avg.__code__.co_varnames)
print(avg.__code__.co_freevars)
print(avg.__closure__)
print([avg.__closure__[i].cell_contents for i in range(len(avg.__closure__))])
print(avg(10))
print(avg.__closure__)
print([avg.__closure__[i].cell_contents for i in range(len(avg.__closure__))])
print(avg(11))
print(avg(12))
```

```
('new_value', 'total')
('series',)
(<cell at 0x10770d648: list object at 0x107915948>,)
[[]]
10.0
(<cell at 0x10770d648: list object at 0x107915948>,)
[[10]]
10.5
11.0
```

闭包是一种函数，它会保留定义函数时存在的自由变量的绑定。

```
def make_averager():
    count = 0
    total = 0

    def averager(new_value):
        count += 1
        total += new_value
        return total / count

    return averager


avg = make_averager()
print(avg.__code__.co_varnames)
print(avg.__code__.co_freevars)
# print(avg(10))  # UnboundLocalError: local variable 'count' referenced before assignment
```

```
('new_value', 'count', 'total')
()
```

Python 3 引入了 `nonlocal` 声明，其作用是把变量标记为自由变量，即使在函数中为变量重新赋予新值了，也会变成自由变量，闭包中保存的绑定会更新。

```
def make_averager():
    count = 0
    total = 0

    def averager(new_value):
        nonlocal count, total
        count += 1
        total += new_value
        return total / count

    return averager


avg = make_averager()
print(avg.__code__.co_varnames)
print(avg.__code__.co_freevars)
print(avg.__closure__)
print([avg.__closure__[i].cell_contents for i in range(len(avg.__closure__))])
print(avg(10))
print(avg.__closure__)
print([avg.__closure__[i].cell_contents for i in range(len(avg.__closure__))])
```

```
('new_value',)
('count', 'total')
(<cell at 0x10ae9c648: int object at 0x10ad56a90>, <cell at 0x10af0c438: int object at 0x10ad56a90>)
[0, 0]
10.0
(<cell at 0x10ae9c648: int object at 0x10ad56ab0>, <cell at 0x10af0c438: int object at 0x10ad56bd0>)
[1, 10]
```

3 浅复制与深复制

复制列表（或多数内置的可变集合）最简单的方式是使用内置的类型构造方法。

```
l1 = [3, [55, 44], (7, 8, 9)]
l2 = list(l1)  # l2 = l1[:]
print(l2)
print(l2 == l1)
print(l2 is l1)
```

```
[3, [55, 44], (7, 8, 9)]
True
False
```

构造方法或 `[:]` 做的是 **浅复制** （即复制了最外层容器，副本中的元素是源容器中元素的引用）。

```
l1 = [3, [55, 44], (7, 8, 9)]
l2 = list(l1)
l1.append(100)
l1[1].remove(55)
```

```
print('l1:', l1)
print('l2:', l2)
l2[1] += [33, 22]
l2[2] += (10, 11)
print('l1:', l1)
print('l2:', l2)
```

```
l1: [3, [44], (7, 8, 9), 100]
l2: [3, [44], (7, 8, 9)]
l1: [3, [44, 33, 22], (7, 8, 9), 100]
l2: [3, [44, 33, 22], (7, 8, 9, 10, 11)]
```

(visual demo)

3.1 为任意对象做深复制和浅复制

有时我们需要深复制（即副本不共享内部对象的引用。） copy 模块提供的 deepcopy 和 copy 函数能为任意对象做深复制和浅复制。

```
class Bus:

    def __init__(self, passengers=None):
        if passengers is None:
            self.passengers = []
        else:
            self.passengers = list(passengers)

    def pick(self, name):
        self.passengers.append(name)

    def drop(self, name):
        self.passengers.remove(name)
```

```
import copy

bus1 = Bus(['Alice', 'Bill', 'Claire', 'David'])
bus2 = copy.copy(bus1)
bus3 = copy.deepcopy(bus1)
print(id(bus1), id(bus2), id(bus3))
bus1.drop('Bill')
print(bus2.passengers)
print(id(bus1.passengers), id(bus2.passengers), id(bus3.passengers))
print(bus3.passengers)
```

```
4506370456 4506371240 4506370568
['Alice', 'Claire', 'David']
4506335240 4506335240 4506278728
['Alice', 'Bill', 'Claire', 'David']
```