学习Python Doc第二天:控制语句和函数

张朝龙

目录

今天是学习Python的第二天，主要内容涉及控制语句，函数定义以及代码风格。

1 控制语句

除了之前学习过的 while ,典型的控制语句还有 if,else for,break,continue

2 if 语句

看代码：

```
x = 23
if x<0:
    x = 0
    print('Negative changed to zero')
elif x==0:
    print('zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

可以有0个 elif ， 也可以有多个 elif ， 最后的 =else= 也是可选的。

3 for 语句

Python 的 for 语句和C语言中的 for 稍有不同。Python的 for 语句可以在任何 list 或者 string 之间循环。看代码：

```
words = ['cat','window','defenestrate']
for w in words:
```

```
    print(w,len(w))
```

输出为：

```
cat 3
window 6
defenestrate 12
```

如果你需要在循环的过程中改变循环对象，建议首先创建一份寻欢对象的副本，看代码：

```
words = ['cat','window','defenestrate']
for w in words[:]:
    if len(w) > 6:
        words.insert(0,w)
```

输出

```
['defenestrate', 'cat', 'window', 'defenestrate']
```

从代码中可以看出，在 `for w in words[:]:` 这一样代码中，Python隐含的创建了 `words[:]` 的一份copy。

4 range() 函数

如果你需要在一个整数数组里循环， `range()` 函数使其易如反掌，看代码：

```
for i in range(5):
    print(i)
```

输出:

```
0
1
2
3
4
```

可以看到， `range(n)` 返回的是 `[0,n-1]` . `range()` 函数非常灵活，比如：

1. `range(5,9)` 生成从5到9，包含5不包含9的整数列表
2. `range(0,10,3)` 生成从0到10，包含0不包含10，间隔为3的整数列表
3. `range(-10,-100,-30)` 生成从-10到-100，间隔30的整数列表

为了索引字符串构成的list，我们可以结合 `range()` 和 `len()` ，看代码：

```
a = ['Mary','had','a' ,'little','lamb']
for i in range(len(a)):
    print(i,a[i])
```

输出：

```
0 Mary
1 had
2 a
3 little
4 lamb
```

山外有山，人外有人，函数外有函数，大多数情况下 enumerate() 是更方便的选择。关于 enumerate() 就不展开了。

关于 range() 有个非常奇怪的现象，大多数时候 range() 表现的像个 list，但是它不是。 range() 的返回是一个对象，这个对象可以用来迭代（这样的对象叫做迭代器，这个迭代器并不占用空间存储内容，因此比较节省空间）。 list() 函数也是迭代器(iterator) .

5 循环中的 break continue 和 else

break continue 的表象和 C 语言中类似。不同的是 for ， for 可以有一个 else 。这个 else 在 for 循环结束后执行（ break 跳出除外， break 跳出不会执行 for 的 else ）。看代码：

```
for n in range(2,10):
    for x in range(2,n):
        if n%x == 0:
            print(n, 'equals', x ,'*', n//x)
            break
    else:
        # loop fell through without finding a factor
        print(n,'is␣a␣prime␣number')
```

输出为：

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

请再次注意： else= 属于 =for 不属于 if .

continue 和C中的 continue 一样, 看代码：

```
for num in range(2,10):
    if n%2 == 0:
        print('find␣an␣even␣number',num)
        continue
    print('find␣a␣number',num)
```

输出为：

```
find an even number 2
find a number 3
find an even number 4
find a number 5
find an even number 6
find a number 7
find an even number 8
find a number 9
```

6 什么也不做的 pass

pass 语句什么也不做，常用来做语法占位，比如：

```
while True:
    pass
```

或者,创建一个最小的 class

```
class MyEmptyClass:
    pass
```

另外 pass 也可以用来占位，稍后再回来实现相关功能。比如：

```
def initlog(*args):
    pass % Remember to come back and fill the pass
```

7 定义函数

在Python中使用 def 来定义函数，看代码：

```
def fib(n):
    """Pring␣a␣Fibonacci␣series␣up␣to␣n"""
    a,b = 0,1
    while a<n:
        print(a,end='␣')
        a,b = b,a+b
    print()
```

调用 fib(2000) 输出：

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

函数的第一行往往是这个函数的文档，一句话说明这个函数的功能。Python有相应的工具可以把用这些注释生成文档供用户查看。执行函数过程中，会为函数中的变量生成一个本地符号表。系统检索变量值的顺序是：1. 本地符号表; 2. 全局符号表；3. 内置变量符号表。所以全局变量不能在一个函数中直接赋值(因为函数内创建的是本地符号表)，但是全局变量在函数中确实可以被引用的。

一个函数的定义在当前符号表中引入了这个函数的名字。解释器把函数名字识别为用户定义的函数。这个值可以赋给另外的名字。看代码：

```
def fib(n):
    """Pring␣a␣Fibonacci␣series␣up␣to␣n"""
    a,b = 0,1
    while a<n:
        print(a,end='␣')
        a,b = b,a+b
    print()

f = fib
f(1000)
```

输出为：

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

如果有其他计算机语言的基础，你会疑惑： fib 只能是一个代码段，不能算一个函数，因为根本没有返回值。事实上，即使没有 return 语句的函数也有返回值，只不过这个返回值无关紧要。这个值是 None 。如果你一定要看到这个 None ，看代码：

```
>>>print(fib(0))

None
```

写一个返回菲波那切数列的函数很容易，看代码：

```
def fib(n):
    """Pring␣a␣Fibonacci␣series␣up␣to␣n"""
    result = []
    a,b = 0,1
    while a<n:
        result.append(a)
        a,b = b,a+b
    return result
```

调用：

```
>>> f100 = fib(100)
>>> f100
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

从这个例子，我们得知：

1. return 语句返回一个值。如果没有 return 语句，返回 None 。
2. result.append(a) 调用了 result 对象的一个方法 append 。不同类型的对象有不同的方法，尽管这些方法可能名字相同，但是由于类型不同，所以在使用过程中也不会造成歧义。