第八章 条件和循环

8.1 if 语句

```
if expression:
    expr_true_suite
```

多个条件的可以使用`and`，`or`，`not`

8.2 else 语句

```
if expression:
    expr_true_suite
else:
    expr_false_suite
```

由于使用了缩进作为代码块的边界，避免`else`与`if`的单语句匹配问题。
举个例子

8.3 elif (else-if 语句)

```
if expression1:
    expr1_true_suite
elif expression2:
    expr2_true_suite
elif expressionN:
    exprN_true_suite
else:
    none_of_the_above_suite
```

也可以使用成员关系操作符`in/not in`和序列或者字典的配合来满足我们的需求

8.4 条件表达式(三元操作符)

可以使用`C and [X] or [Y])[0]`来模拟条件表达式(完成条件判断和赋值操作),社区有不同的方案，参见**PEP 308**

Python 2.5 中加入的语法确定为: `X if C else Y`——
`true_part if condition else false_part`

```
#条件表达式
def condition_expression(condition, true_part, false_part):
    return (condition and [true_part] or [false_part])[0]

def min(x, y):
    smaller = (x < y and [x] or [y])[0]
    return smaller

#Python 自带的条件表达式
def min2(x, y):
    smaller = x if x < y else y
    return smaller
```

8.5 while 语句

```
while expression:
    suite_to_repeat
```

8.6 for 语句

```
for iter_var in iterable:
    suite_to_repeat
```

每次循环`iter_var`迭代变量狐会被设置成可迭代对象(`list`,`tuple`,`dict`,`str`)的当前元素

- **序列类型的迭代**
- 通过序列项迭代
- 通过索引迭代
- 通过项和索引来迭代

```python
#通过序列项迭代
nameList = ['Walter','Nicole','Steven','Henry']
for eachName in nameList:
    print eachName
#通过索引迭代--通过索引取元素
for index in range(len(nameList)):
    print nameList[index]

#通过项和索引来迭代
seasons = ['Spring', 'Summer', 'Fall', 'Winter']

for index, value in enumerate(seasons):
    print index,"--",value

for index, value in enumerate(seasons, start=1):
    print index,"--",value

for index, value in enumerate(seasons, start=5):
    print index,"--",value


#output
#start 的默认值为0
0 -- Spring
1 -- Summer
2 -- Fall
3 -- Winte

#start=1
```

```
1 -- Spring
2 -- Summer
3 -- Fall
4 -- Winter

#start=5
5 -- Spring
6 -- Summer
7 -- Fall
8 -- Winter
```

从性能方面考虑，直接迭代序列要比通过索引迭代快。(参见8-13)

迭代器对象有一个 `next()` 方法, 调用后返回下一个条目. 所有条目迭代完后, 迭代器引发一个 `StopIteration` 异常告诉程序循环结束. for 语句在内部调用 `next()` 并捕获异常。

- **`range()` 内建函数**

range() 完整语法

`range(start, end, step =1)`

`range()` 返回的list
`xrange()` 返回xrange object–The xrange type is an immutable sequence which is commonly used for looping.

区间为 `[start, end)` (左开右闭)每次递增step,并且step不可以为零,否则将会引发异常。

如果只给start和end,则step默认为1

```
print range(3, 7)
[3, 4, 5, 6]
```

```
for eachVal in range(2, 19, 3):
        print " ",eachVal,
  2   5   8   11   14   17

#loop in C
for (eachVal = 2; eachVal < 19; eachVal += 3)
{
    printf("value is: %d\n", eachVal);
}
```

range() 简略语法

`range(end)` –此时start默认为0,step默认为1
`range(start, end)` –此时step默认为1

xrange() 内建函数

当有一个很大的范围列表时，`xrange()`更为合适。
`xrange()`只被用在`for`循环中，在`for`循环外使用是没用的。

`range()`函数会在内存里创建列表的完整拷贝，而`xrange()`函数不会。

与序列相关的内建函数

`sorted()`, `zip()`-返回一个序列(列表)
`reversed()`, `enumerate()`-返回迭代器(`Iterator`–Java, `ContainerType::iterator`-C++)

```
albums = ('Poe','Gaudi','Freud','Poe2')
years = (1976, 1987, 1990, 2003)

for year, album in zip(years, albums):
    print year, album

1976 Poe
1987 Gaudi
1990 Freud
2003 Poe2
```

8.7 break 语句

结束当前循环然后跳转到下面的语句

```
#the largest divisor of a given number num--数字 num 的最大约数
def largestDivisor(num):
    divisor = num / 2
    while divisor > 0:
        if num % divisor == 0:
            print divisor,'is the largest factor of ', num
            break
        divisor -= 1

for div in range(100):
        largestDivisor(div)

21 is the largest factor of  42
```

```python
#greatest common divisor--最大公约数
#greatest common divisor=GCD
def GCD(num1, num2):
    if num1 % num2 == 0:
        print num2,"is greatest common divisor",num1
        return
    if num2 % num1 == 0:
        print num1,"is greatest common divisor",num2
        return

    factor1 = num1 / 2
    factor2 = num2 / 2
    while True:
        #factor1
        while factor1 > 0:
            if num1 % factor1 == 0 :
                break
            factor1 -= 1
        #fator2
        while factor2 > 0:
            if num2 % factor2 == 0 :
                break
            factor2 -= 1
        if num1 % factor2 == 0:
            print factor2,"is greatest common divisor ",num1, " and ", num2
            break
        if num2 % factor1 == 0:
            print factor1,"is greatest common divisor ",num1, " and ", num2
            break
```

```
GCD(6, 12)
GCD(12, 12)
GCD(27, 36)

6 is greatest common divisor 6  and  12
12 is greatest common divisor 12  and  12
9 is greatest common divisor  27  and  36
```

8.8 continue 语句

遇到 continue 语句时, 程序会终止当前循环,并忽略剩余的语句, 然后回到循环的顶端。

在开始下一次迭代前

- 如果是条件循环, 将验证条件表达式是否成立
- 如果是迭代循环,将验证是否还有元素可以迭代
  只有在条件为真或者有元素可迭代时才开始下一次迭代

```
#用户最多输入3次密码
def checkPassword():
    valid = False
    count = 3
    while count > 0:
        # check for valid passwd
        for eachPasswd in ('123','1234','12345'):
            input = raw_input("enter password:  ")
            if input == eachPasswd:
                valid = True
                break
            if not valid:
                print "invalid input"
                count -= 1
                print "input password left ",count," times "
                continue
            else: #login success
                break
        if valid:
            print "login success"
            break;

#输入成功的
enter password:  123
login success

#输入3次不成的
enter password:  12
invalid input
```

```
input password left  2  times
enter password:  1q2w
invalid input
input password left  1  times
enter password:  1q2w3e
invalid input
input password left  0  times
```

8.9 pass 语句

如果你在需要子语句块的地方不写任何语句, 解释器会提示你语法错误。
`pass`语句表示NOP=No OPeration,这在开发中可以作为一个标识，表示将来要完成的代码。

8.10 再谈else 语句(for/while 的 else)

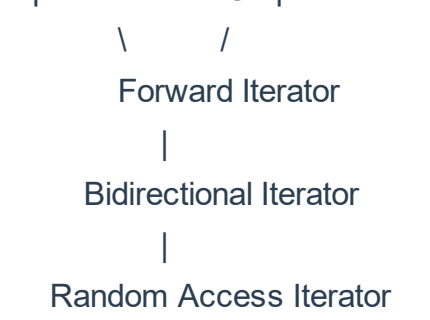
在循环中使用时, else子句只在循环完成后执行, 也就是说 break 语句也会跳过 else 块。

```
#largest divisor or prime 8.10 --for/while-else
def largestDivisorOrPrime(num):
    divisor = num / 2
    while divisor > 0 and divisor != 1:
        if num % divisor == 0:
            print divisor,'is the largest factor of ', num
            break
        divisor -= 1
    else:
        print num,"is prime"

#output
5 is the largest factor of  10
11 is prime
6 is the largest factor of  12
13 is prime
7 is the largest factor of  14
5 is the largest factor of  15
8 is the largest factor of  16
17 is prime
9 is the largest factor of  18
19 is prime
10 is the largest factor of  20
```

8.11 迭代器和 iter() 函数

为类序列对象提供了一个类序列的接口。可以理解为"指向"(其实C/C++中指针就是最原始的迭代器类型
C++当中的5中Iterator



Python 还提供了一整个 itertools 模块, 它包含各种有用的迭代器。(到模块 `itertools` 一看结果全是 `pass`)

如何迭代

迭代器就是有一个 next() 方法的对象。

当一个循环机制需要下一个项时, 调用迭代器的 `next()` 方法就可以获得它。条目全部取出后, 会引发一个 `StopIteration` 异常, 这并不表示错误发生, 只是告诉外部调用者, 迭代完成.

使用迭代器

序列

```
myTuple = (123, 'xyz', 45.67)
it = iter(myTuple)
print it.next()

123
```

需要把上面的代码放到 `try-except` 中否则会引发异常。

字典

字典的迭代器会遍历它的键(keys)。
`for eachKey in myDict.keys()` 可以缩写为 `for eachKey in myDict`

另外字典还可以用
`myDict.iterkeys()`, `myDict.itervalues()`, `myDicit.iteritems()` 这三种方法。

文件

文件对象生成的迭代器会自动调用 `readline()`

可以把 `for eachLine in myFile.readlines()` 缩写为
`for eachLine in myFile`

可变对象和迭代器

在迭代字典的 key 时, 你绝对不能改变这个字典。

使用字典的 keys() 方法是可以的, 因为keys() 返回一个独立于字典的列表

迭代器是与实际对象绑定在一起的, 它将不会继续执行下去。

```
myDict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
for eachKey in myDict:
    print eachKey, myDict[eachKey]
    del myDict[eachKey]
print myDict.get('a')


a 1
Traceback (most recent call last):
  File "F:/AllExercises/Python/src/Chapter8/InContext.py", line 151, in <module>
    for eachKey in myDict:
RuntimeError: dictionary changed size during iteration
```

为避免这样的错误，可参见**PEP 234**

创建迭代器

两种方法

1. 对一个对象调用 `iter()` 就可以得到它的迭代器
2. 创建迭代器的方法是使用类,需要实现 `__iter__()` 和 `next()` 方法

`iter(func, sentinel)`
如果传递两个参数给 `iter()` ,它会重复调用func,直到迭代器的下一个值为sentinel

8.12 列表解析

列表解析List comprehensions, 或缩略为 list comps

```
[expression for iter_var in iterable]
```

最后的结果值是该表达式产生的列表。即列表解析会将产生的结果放入到创建的列表中。如果产生的结果量很大的话，性能会很低。

迭代变量并不需要是表达式的一部分。

```
six = [6 for i in range(6)]
print six

[6, 6, 6, 6, 6, 6]
```

列表解析的表达式可以取代内建的 map() 函数以及 lambda , 而且效率更高。

结合if 语句还可以有另外一个版本：
`[expression for iter_var in iterable if cond_expr]`
这个语法在迭代时会过滤/捕获满足条件表达式cond_expr 的序列成员

矩阵例子

```
matrix = [(x+1,y+1) for x in range(3) for y in range(5)]
    print matrix

[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (3, 1), (3, 2), (3,
3), (3, 4), (3, 5)]
```

计算文件中字符个数

```
#计算单词总个数
f = open('test.txt', 'r')
sumWords = len([word for line in f for word in line.split()])

f.seek(0) #回到文件的开头
sumCharacter = sum([len(word) for line in f for word in line.split()])
```

列表解析支持多重嵌套`for`循环以及多个`if`子句。更多信息可以参考**PEP 202**

8.13 生成器表达式-Generator Expressions

生成器表达式是列表解析的一个扩展。2.2版本加入生成器-generator。
生成器是特定的函数, 允许返回一个值，然后"暂停"代码的执行, 稍后恢复。

列表解析的一个不足就是必要生成所有的数据, 用以创建整个列表。
这对大量数据的迭代器有负面影响，生成器表达式通过结合列表解析和生成器解决了这个问题

2.4版本加入生成器表达式generator expressions 。可以参考**PEP 289**。
列表解析返回的是一个列表。
生成器表达式并不真正创建列表, 而是返回一个生成器，这个生成器在每次计算出一个条目后，把这个条目"产生" (yield)出来. 生成器表达式使用了"延迟计算"(lazy evaluation)(类似于懒加载–用到的时候加载), 所以它在使用内存上更有效。

列表解析和生成器表达式语法:

```
#列表解析语法
[expr for iter_var in iterable if cond_expr]
#生成器表达式语法
(expr for inter_var in iterable if cond_expr)
```

为了避免创建庞大的列表, 我们可以使用生成器表达式来完成求和操作

求文件中的全部字符个数

```
allCharacters = sum(len(word) for line in file for word in line.split())
```

代码的演变

通过寻找文件中最长的行来看代码的改进

```
f = open('/ect/motd','r')
longest = 0
while True:
    linelength = len(f.readline().strip())
    if not linelength:
        break;
    if linelength > longest:
        longest = linelen
f.close()
return longest
```

如果读取了所有的行，那么应该尽早释放文件资源

```
f = open('/ect/motd','r')
longest = 0
allLines = f.readlines() #readlines()会读取文件所有的行，在读大文件时不合适
for line in allLines:
    linelen = len(line.strip())
    if linelen > longest:
        longest = linelen
return longest
```

列表解析可以允许在得到行的集合前做一定的处理。
在下面的代码中，在读取行时还调用了`strip()`方法。

```
f = open('/ect/motd','r')
longest = 0
allLines = [x.strip() for x in f.readlines()]
f.close()
for line in allLines:
    linelen = len(line)
    if linelen > longest:
        longest = linelen
return longest
```

之前得到的是文件中行的集合，以下代码获得的是文件中行长度的集合，这样可以直接使用`max()`内建函数了

```
f = open('/ect/motd','r')
allLinesLens = [len(x.strip()) for x in f]
f.close()
return max(allLinesLens)
```

这段代码存在文件是在一行一行读取文件的时候，列表解析会将文件的所有行读到内存中然后生成列表，这时候可以使用声称其表达式取代列表解析，从而提高性能(“延迟计算”(lazy evaluation)获得的性能提升)。

可以再次将代码简化

```
#因为`open()`的第二个参数mode默认为读取文件'r'
return max(len(x.strip()) for x in open('/etc/motd'))
```

8.14 相关模块

Python 2.2 引进了迭代器。Python 2.3中引入了itertools模块

8.15 习题

8-1 条件语句

```
(a)
statement A
statement C
(b)
statement A
statement D
statement E
(c)
statement A
statement B
```

8-2 循环

```
def showMenu():
    prompt = """
    f-from
    t-to
    i-increment
    Enter f,t and i:
    """
    while True:
        try:
            f = int(raw_input(prompt))
            t = int(raw_input(prompt))
            i = int(raw_input(prompt))
            loop(f, t, i)
        except(EOFError,KeyboardInterrupt,IndexError):
            choice = 'q'
def loop(f, t, i):
    end = t + 1
    for var in range(f, end, i):
        print var,' ',

showMenu()
```

8-3 `range()`

```
(a) end = 9, range(9)
(b) start = 3, end = 18 + 1, step = 3
(c) start = -20, end = 860 + 1, step = 220
```

8-4 素数

```
def isprime(num):
    for factor in range(2, num):
        if num % factor == 0:
            print num, "not prime"," factor--",factor
            return False
    else: #正常退出循环
        print num, "is prime"
        return True

num = int(raw_input("Input a number: "))
isprime(num)
```

8-5 约数

```
def getfactors(num):
    factors = [1]
    divisor = num / 2
    while divisor > 0 and divisor != 1:
        if num % divisor == 0:
            print divisor,'is factor of ', num
            factors.insert(1,divisor)
        divisor -= 1
    factors.append(num)
    return factors

print 'factor of 20 is : ', getfactors(20)
```

8-6 素因子分解

```
def PrimeFactorization(num):
    primeFactors = []
    for factor in getfactors(num):
        if isprime(factor):
            primeFactors.append(factor)
    return primeFactors

print PrimeFactorization(20)
```

8-7 全数

```
def isperfect(num):
    allFactors = getfactors(num)
    sumOfFactors = 0
    allFactors.remove(allFactors[len(allFactors) - 1])
    for f in allFactors:
        sumOfFactors += f
    if sumOfFactors == num:
        print num,"is perfect"
        return 1
    else:
        print num,"is not perfect"
        return 0

print isperfect(6)
```

8-8 阶乘

```
def factorial(n):
    multiplied = 1
    for m in range(1, n + 1):
        multiplied *= m
    return multiplied

print factorial(6)
```

8-9 Fibonacci 数列

```
def F(n): #Fibnacci
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return F(n-1) + F(n-2)

print F(6)
```

8-10 文本处理-统计元音辅音以及单词个数

```
def wordcount(line):
    vowel = 'aeiou'
    consonantCount = 0
    cnt = 0
    vowelCount = 0
    for word in line.split(' '):
        cnt += 1
        for c in word:
            if c.isalpha():
                if c in vowel:
                    vowelCount += 1
                else:
                    consonantCount += 1
            else:
                print '非字符'
    print "元音有: %d个，辅音有: %d个，单词数为: %d" % (vowelCount,consonantCount,cnt)

print wordcount("The quick brown fox jumps over a lazy dog.")
```

8-11 文本处理-按名字排序

```
def sortedName():
    totalNames = int(raw_input("Enter total number of names:"))
    i = 0
    errorCount = 0
    valid = False
    names = {}
    prompt = "Last Name, First Name"
    while i != totalNames:
        i += 1
        name = raw_input(prompt)
        if name.find(',') == -1:
            print "Wrong format... should be Last, First"
            errorCount += 1
            print "You have done this %d time(s) already. Fixing input..." % (errorCount)
            l = name.split(' ')[0]
            f = name.split(' ')[1]
            if l in names:
                names[l] = f
            names.setdefault(l, f)
        else:
            lastName = name.split(',')[0]
            firstName = name.split(',')[1]
            if names.has_key(lastName):
                names[lastName] = firstName
            names.setdefault(lastName, firstName)
    for key in sorted(names.keys()):
        print '%s, %s' % (key, names[key])

sortedName()
```

8-12 (整数)位操作-ASCII

```
def printASCII(f, t):
    end = t + 1
    print 'DEC\t\t\tBIN\t\t\t\t\tOCT\t\t\t\tHEX\t\t\tASCII'
    print '-------------------------------------------------------------------------'
    for c in range(f, end):
        ch = chr(c)
        if ch in string.printable:
            print c, '\t\t\t', str(bin(c))[2:].rjust(8,'0'),
             '\t\t\t\t',str(oct(c)), '\t\t\t', str(hex(c))[2:], '\t\t\t\t',ch
        else:
            print c, '\t\t\t', str(bin(c))[2:].rjust(8,'0'), '\t\t\t\t',str(oct(c)),'\t\t\t\t', st
r(hex(c))[2:]

printASCII(0, 150)
```

8-13 性能

为什么在数据量大式序列项迭代比索引迭代的性能好？
python中一切皆对象，产生下标要构造整数对象。