第五章 条件、循环和其它语句

5.1 print和import

5.2 赋值

5.3 语句块

5.4 条件和条件语句

5.5 循环

5.6 列表推导式

5.7 其它语句

print函数

- 在print函数中如果需要打印多个表达式也是可以的,这时只需使用逗号将它们隔开然后输出

```
>>> print('prices =', 20, 30, 40)
prices = 20 30 40          # 从第2个参数开始每个参数前添加了一个空格

>>> 20, 30, 40                    # 打印元组
(20, 30, 40)

>>> print(20, 30, 40)             # 打印多个表达式
20 30 40

>>> print((20, 30, 40))           # 打印元组的另外一种方法
(20, 30, 40)
```

print函数

- 在print函数中如果需要同时打印文本和变量的值,但是却不希望使用字符串格式化的方法,这个特性就非常有用

```
>>> name = 'Gumby'
>>> salutation = 'Mr.'
>>> greeting = 'Hello,'
>>> print(greeting, salutation, name)
Hello, Mr. Gumby
>>> print('Hello, ' + 'Mr. ' + 'Gumby')      #使用字符串连接操作时必须自
                                             # 行加入所需的空格
Hello, Mr. Gumby
```

import语句

- 从模块（module）导入函数时，通常可以使用import语句

```
import somemodule
```

- 或者

```
from somemodule import somefunction
```

- 或者

```
from somemodule import function1, function2, function3
```

- 或者（当需要从给定模块somemodule中导入所有函数）

```
from somemodule import *
```

import语句

- 当多个模块module1和module2中有同名函数somefunction时，通常可以先使用import语句

```
import module1
import module2
```

- 当需要使用同名函数时，需要这样

```
module1.somefunction()
module2.somefunction()
```

import语句

- 此外还可以为模块或函数提供别名，例如

```
>>> import math as foobar          # 为模块提供别名
>>> foobar.sqrt(9)
3.0

>>> from math import sqrt as foobar          # 为函数提供别名
>>> foobar(9)
3.0
```

- 因此，当需要使用来自不同模块的同名函数时可以

```
from module1 import somefunction as f1
from module2 import somefunction as f2
```

第五章 条件、循环和其它语句

5.1 print和import

5.2 赋值

5.3 语句块

5.4 条件和条件语句

5.5 循环

5.6 列表推导式

5.7 其它语句

赋值语句的特殊技巧

- 赋值操作不会拷贝数据，只是把名字和对象做一个绑定，也就是说赋值语句是起一个绑定或重绑定的作用
- **序列解包**是指将多个值的序列解开，然后放到变量的序列中。利用序列解包可以同时对多个变量和数据结构成员进行赋值

```
>>> x, y , z = 1, 2, 3
>>> print(x, y, z)
1 2 3
>>> x, y = y, x
>>> print(x, y)
2 1
>>> values = (1, 2, 3)
>>> a, b, c = values
>>> print(a, b, c)
1 2 3
```

序列解包

- 当函数或者方法返回元组（或者其它序列或可迭代对象）时，序列解包功能特别有用

```
>>> scoundrel = {'name':'Robin', 'girlfriend':'Marion'}
>>> key, value = scoundrel.popitem()
>>> key
'girlfriend'
>>> value
'Marion‘
```

序列解包

- 当使用序列解包时，也可以在赋值符号（=）左边的变量参数表中使用星号运算符，表明此处应该有一个列表

```
>>> par1, par2, *rest = [1, 2, 3, 4, (5, 6)]
>>> print('par1 =', par1, ' par2 =', par2, ' rest =', rest)
par1 = 1  par2 = 2  rest = [3, 4, (5, 6)]

>>> *rest, par1, par2 = [1, 2, 3, 4, (5, 6)]
>>> print('rest =', rest, ' par1 =', par1, ' par2 =', par2)
rest = [1, 2, 3]  par1 = 4  par2 = (5, 6)

>>> *rest, par1, par2 = [1, 2]
>>> print('rest =', rest, ' par1 =', par1, ' par2 =', par2)
rest = []  par1 = 1  par2 = 2
```

链式赋值

- 链式赋值是将同一个值赋给多个变量的捷径

```
>>> x = y = z = 5
>>> print(x, y, z)
5 5 5

>>> x is y                  # 判断同一性
True
>>> y is z
True

>>> x == y                  # 判断相等性
True
>>> x == z
True
```

增量赋值

- 增量赋值对+、-、*、/、%等标准运算符都适用

```
>>> x = 3
>>> x += 1
>>> x
4
>>> x -= 2
>>> x
2
>>> x *= 3
>>> x
6
>>> x /= 2
>>> x
3.0
>>> x //= 3
>>> x
1.0
```

第五章 条件、循环和其它语句

5.1 print和import

5.2 赋值

5.3 语句块

5.4 条件和条件语句

5.5 循环

5.6 列表推导式

5.7 其它语句

语句块

- 语句块是在条件为真时执行一次或多次的一组语句，在代码前放置空格来缩进语句即可创建语句块

- **块中每行都应该缩进同样的量（量的大小可以自行确定，只要在块内一致即可）！**缩进的方式如下，

这里是一行

这里是另外一行：　　**# 冒号（:）表示语句块的开始**

　　这是另外一个语句块（内部语句块）

　　仍旧在同一块中

　　这是本块中的最后一行

这里跳出了上面内部语句块

第五章 条件、循环和其它语句

5.1 print和import

5.2 赋值

5.3 语句块

5.4 条件和条件语句

5.5 循环

5.6 列表推导式

5.7 其它语句

条件和条件语句

- 布尔变量可以让程序选择是否执行语句块，下面的值在作为布尔表达式时会被解释器看作假（False）：

  False　None　0　“”　()　[]　{}

  除此以外，其它的一切均可以被解释为真，包括True

- False和True属于布尔类型

布尔值和bool函数

- bool函数可以用来将其它值转换成布尔类型的值

```
>>> bool('Hello, world!')
True

>>> bool(42)
True

>>> bool([])
False

>>> bool(‘ ’)        # 此处参数为一个空格
True
>>> bool(‘’)         # 此处参数为空字符串
False

>>> bool(None)
False
```

条件执行和if语句

- if语句可以实现条件执行：如果条件（在if和冒号之间的表达式）判定为真，则执行后面的语句块；反之则不执行

```
>>> name = input('What is your name? :')
What is your name? :Gumby
>>> if name.startswith('Gum'): print('Hello, Mr.', name)

Hello, Mr. Gumby
```

else子句

- else子句可以用来增加一种选择

```
>>> if name.startswith('Gum'): print('Hello, Mr.', name)
else:
        print('Hello, stranger')


Hello, Mr. Gumby
```

elif子句

- 如果需要检查多个条件，就可以使用elif，它是else if的缩写，有时和if、else子句联合使用，可以看作是具有条件的else子句

```
num = input('Please enter a number: ')
number = int(num)
if number > 0:
    print('This is a positive number.')
elif number < 0:
    print('This is a negative number.')
else:
    print('This is zero.')
```

嵌套代码块

- 语句里面还可以嵌套语句，以if语句为例

```
name = input('What is your name? ')
if name.endswith('Gumby'):
    if name.startswith('Mr.'):
        print('Hello, Mr. Gumby')
    elif name.startswith('Mrs.'):
        print('Hello, Mrs. Gumby')
    else:
        print('Hello, Gumby')
else:
    print('Hello, stranger')
```

复杂的条件

表达式	描述
x==y	x等于y
x < y	x小于y
x > y	x大于y
x >= y	x大于等于y
x <= y	x小于等于y
x != y	x不等于y
x is y	x和y是同一对象
x is not y	x和y是不同的对象
x in y	x是容器（例如序列）y的成员
x not in y	x不是容器（例如序列）y的成员

Python中的比较运算符

相等运算符

- 如果需要判断两个对象是否相等，应该使用相等运算符（==）

```
>>> 'foo' == 'foo'
True
>>> 'foo' == 'bar'
False
>>> 'foo' = 'foo'
SyntaxError: can't assign to literal
```

- 单个等号（=）是赋值运算符，是用来改变值的，不能用来进行比较

同一性运算符

- is运算符是判定同一性（是否为同一对象）而不是相等性的

```
>>> x = y = [1, 2, 3]
>>> z = [1, 2, 3]
>>> x == y
True
>>> x == z
True
>>> x is y
True
>>> x is z
False
>>> id(x), id(y), id(z)
(46671432, 46671432, 46671872)
```

字符串、序列和元组的比较

- 字符串可以按照字母顺序排列进行比较

```
>>> 'Hello, world!' > 'Hello, Mr. Gumby!'
True
>>> 'Hello, World!' < 'Hello, Mr. Gumby!'
False
```

- 序列和元组也可以进行比较

```
>>> [1, 2, 3] < [1, 3]
True
>>> [1, 2, 3] < [1, 2, 2]
False
>>> [(1, 2), (3, 4)] < [(3, 4)]
>>> (1, 2, 3, 4) < (5,)
True
```

布尔运算符

- and、or和not都是布尔运算符，使用这三个运算符可以任意结合用于条件判定

```
num = input('Enter a number :')
number = int(num)
if number <= 10 and number>= 0: print('We have got the right number!')
```

- 上面的条件判定也可以用以下语句替换：

```
if 0 <= number <= 10: print('We have got the right number!')
```

断言

- assert是用来检查条件，如果该条件为真，就什么都不做；如果为假，则会抛出AssertError，并且包含错误信息

```
>>> age = -20
>>> assert 0 < age < 100, 'An age must be realistic.'
Traceback (most recent call last):
  File "<pyshell#149>", line 1, in <module>
    assert 0 < age < 100, 'An age must be realistic.'
AssertionError: An age must be realistic.
```

- assert语句在程序中放置检查点，确保条件成立时才能让程序正常工作。断言中的条件后可以跟字符串，用来解释断言

第五章 条件、循环和其它语句

5.1 print和import

5.2 赋值

5.3 语句块

5.4 条件和条件语句

5.5 循环

5.6 列表推导式

5.7 其它语句

循环

- while循环：在任何条件为真的情况下重复执行一个代码块

```
x = 0
while (x < 10):
    print(x)
    x += 1
```

- for循环：主要用于为某个可迭代对象iterable（可以按次序迭代的对象，如序列）的每个元素都执行一个代码块

```
words = ['this', 'is', 'an', 'ex', 'parrot']
for word in words:
    print(word)
```

for循环

- 因为迭代某一范围内的数字很常见的，所以可以使用内建函数range，其工作方式类似分片，包含下限但不包括上限

```
>>> list(range(10))                # range的参数中可以只有上限
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(2, 7, 2))           # 参数中包含下限、上限和步长
[2, 4, 6]
>>> list(range(8, 2, -2))          # 可以使用负数作为步长
[8, 6, 4]
```

- 使用range函数和for循环较之使用while循环完成同一任务更为简洁

```
>>> for x in range(10): print(x)
```

for循环

- for循环可以用于遍历字典的所有键（或值），就像遍历访问序列一样

```
>>> d = {'a':1, 'b':2, 'c':3}
>>> for key in d.keys():
        print(key, 'corresponds to', d[key])
```

```
b corresponds to 2
c corresponds to 3
a corresponds to 1
```

- 因为字典中的元素是无序排列的，所以迭代时处理元素的顺序是不确定的

for循环

- 同样，for循环还可以用于遍历字典的所有元素，同时使用序列解包

```
>>> d = {'a':1, 'b':2, 'c':3}
>>> for key, value in d.items():
        print(key, 'corresponds to', value)


b corresponds to 2
c corresponds to 3
a corresponds to 1
```

迭代工具

- 并行迭代：程序可以同时迭代两个序列

```
>>> names = ['anne', 'beth', 'george', 'damon']
>>> ages = [12, 45, 32, 102]
>>> for i in range(len(names)):  # 循环次数与列表中包含的元素个数相等
        print(names[i], 'is', ages[i], 'years old.')
```

```
anne is 12 years old.
beth is 45 years old.
george is 32 years old.
damon is 102 years old.
```

- i是循环索引的标准变量名

zip函数

- 内建函数zip可以用来进行并行迭代，把多个序列压缩在一起，然后返回元组的列表；还可以在循环中解包元组

```
>>> names = ['anne', 'beth', 'george', 'damon']
>>> ages = [12, 45, 32, 102]
>>> ids = [1, 2, 3, 4]
>>> for no, name, age in zip(ids, names, ages):
        print('No.', no, 'with name', name, 'is', age, 'years old.')
```

```
No. 1 with name anne is 12 years old.
No. 2 with name beth is 45 years old.
No. 3 with name george is 32 years old.
No. 4 with name damon is 102 years old.
```

zip函数

- 内建函数zip的一个重要特点是可以处理不等长的序列，当其中最短的序列“用完”时即停止处理

```
>>> names = ['anne', 'beth', 'george']
>>> ages = [12, 45]
>>> for name, age in zip(names, ages):
        print(name, 'is', age, 'years old.')
```

```
anne is 12 years old.
beth is 45 years old.
```

enumerate函数

- 内建函数enumerate可以在提供索引的地方迭代索引-值对

```
>>> strings = ['to', 'be', 'or', 'not', 'to', 'be']
>>> for index, string in enumerate(strings):
        if ‘be’ in string:       # 将’be’换成’think’
                strings[index] = 'think'


>>> strings
['to', 'think', 'or', 'not', 'to', 'think']
```

翻转和排序迭代

- 函数reversed和sorted可以用于任何序列或可迭代的对象上，不是原地修改对象，而是返回翻转和排序后的版本

```
>>> a = sorted([4, 2, 7, 3])
>>> a
[2, 3, 4, 7]
>>> b = reversed([4, 2, 7, 3])
>>> b
<list_reverseiterator object at 0x02D0D2B0>    # b为迭代器对象
>>> print(list(b))
[3, 7, 2, 4]
 # ‘’.join(iterable) 是构造新的字符串对象的一种较好的方法
>>> ‘’.join(reversed(‘Hello, world!’))  # 翻转原字符串
'!dlrow ,olleH'
```

跳出循环

- break语句：跳出循环可以使用该语句

```
from math import sqrt
for n in range(99, 0, -1):
    root = sqrt(n)
    if root == int(root):
        print(n)
        break
```

- continue语句：表示跳过剩余的循环体，进行下一次循环的开始

```
for x in seq:
    if condition1:continue
    if condition2:continue
    do_something()
```

while True/break

- while True部分可以实现一个永远不会停止的循环，这时可以在循环内部的if语句部分加入条件，在条件满足时调用break语句跳出循环

```
while True:
    word = input('Please enter a word : ')
    if not word: break
    # 处理word
    print('The word is ' + word)
```

第五章 条件、循环和其它语句

5.1 print和import

5.2 赋值

5.3 语句块

5.4 条件和条件语句

5.5 循环

5.6 列表推导式

5.7 其它语句

列表推导式

- 列表推导式是利用其它列表创建新列表的方法，类似于for循环

```
>>> [x * x for x in range(4)]
[0, 1, 4, 9]

>>> [x * x for x in range(10) if x % 3 == 0]
[0, 9, 36, 81]

>>> [(x, y) for x in range(2) for y in range(3)]
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

列表推导式

- 假如需要将名字首字母的男孩子和女孩子的名字配对，可以按如下步骤操作

```
>>> boys = ['chris', 'arnold', 'bob']
>>> girls = ['alice', 'bernice', 'clarice']
>>> [b+'+'+g for b in boys for g in girls if b[0]==g[0]]
['chris+clarice', 'arnold+alice', 'bob+bernice']
```

- 上面做法的效率不高，因为它会检查男孩子和女孩子的名字的所有配对

列表推导式

- 更优的方案会建立一个字典，然后以女孩子名字的首字母作为键，以与之对应的女孩子名字作为值；然后用列表推导式遍历整个男孩子名字的集合，并查找那些与当前男孩子名字首字母相匹配的女孩子名字的集合

```
>>> boys = ['chris', 'arnold', 'bob']
>>> girls = ['alice', 'agnes', 'bernice', 'clarice']

>>> letterGirls = {}
>>> for girl in girls:
        letterGirls.setdefault(girl[0], []).append(girl)  # 因此处缺省值设为列表类型，所以后面可以调用列表的append方法
>>> [b+'+'+g for b in boys for g in letterGirls[b[0]]
['chris+clarice', 'arnold+alice', 'bob+bernice']
```

第五章 条件、循环和其它语句

5.1 print和import

5.2 赋值

5.3 语句块

5.4 条件和条件语句

5.5 循环

5.6 列表推导式

5.7 其它语句

其它语句

- 有的时候程序段什么都不做，或者说还没有为之确定具体的细节，这时就需要pass语句，它在代码中作为占位符使用

```
if name == 'Ralph Auldus Melish':
        print('Welcome!')
elif name == 'Enid':
        # 还没结束...
        pass
elif name == 'Bill Gates':
        print('Access Denied.')
```

- 因为Python代码中不允许出现空代码块，解决方法就是在语句块中加上pass语句

删除语句del

- 一般来说，Python会删除那些不再使用的对象：当某个对象的引用计数降为0时，说明没有任何引用指向该对象，该对象将作为垃圾被回收
- 另外一个方法就是使用del语句，该语句不仅移除一个对象的引用，也会移除那个名字本身

```
>>> x
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    x
NameError: name 'x' is not defined
```

删除语句del

- del语句不仅移除一个对象的引用，也会移除那个名字

```
>>> x = ['Hello', 'world']
>>> y = x
>>> y[1] = 'Python'
>>> del x
>>> x
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    x
NameError: name 'x' is not defined
>>> y    # 删除x并不影响y，因为删除的只是名字，而非列表本身
['Hello', 'Python']
```

命名空间

- 命名空间（namespace），即名字和对象的映射，可以将命名空间理解为一个字典

- 各个命名空间是独立的，没有任何关系的，所以一个命名空间中不能有重名，但不同的命名空间是可以重名而没有任何影响

命名空间

- 命名空间都是有创建时间和生存期的
  - Python built-in names（包括内置函数，内置常量，内置类型）组成的命名空间，在Python解释器启动的时候被创建，在解释器退出的时候被删除
  - Python模块的global names（这个模块定义的函数，类，变量）组成的命名空间，在这个module被import的时候创建，在解释器退出的时候退出
  - 函数的local names组成的命名空间，在函数被调用的时候创建，函数返回的时候被删除

执行字符串函数exec

- 有时候需要动态地创造Python代码，然后将其作为函数执行，其中执行存储在字符串中Python代码的函数是exec，例如

```
>>> exec("print('Hello, world!')")
Hello, world!
```

- 很多情况下会为exec函数提供命名空间，从而阻止代码干扰命名空间，这时可以在函数的参数中增加<scope>。<scope>是一个字典，用于放置代码字符串的命名空间

执行字符串函数exec

```
>>> from math import sqrt
>>> scope = {}
>>> exec('sqrt = 1', scope)
>>> sqrt(4)
2.0
>>> scope['sqrt']
1
```

- 由此可见，潜在的破坏性代码并不会覆盖sqrt函数，该函数仍然能正确工作，而通过exec赋值的变量只在自己的作用域内有效

请尝试打印scope的内容！

求值字符串函数eval

- eval是Python的内建函数，用于计算以字符串形式书写的表达式，并返回计算结果值

```
>>> eval(input('Please enter an arithmetic expression : '))
Please enter an arithmetic expression : 9+12*3
45
 # 可以从字符串中提取数据生成列表、元组等对象
>>> a = '[1, 2, 3, 4]'
>>> b = eval(a)
>>> b
[1, 2, 3, 4]
>>> type(b)
<class 'list'>
```

小结

- print语句可以用来打印由逗号隔开的多个值
- 可以使用import … as…语句进行函数的局部重命名
- 通过序列解包和链式赋值功能，多个变量赋值可以一次性完成，通过增量复制则可以原地改变变量
- 块是通过缩排使语句成组的一种方法，可以在条件以及循环语句中使用
- 条件语句根据条件执行或者不执行一个语句块，几个条件可以串联使用if elif else
- 简单来说，断言就是肯定某事件为真，如果为假断言就会在程序中引发异常

小结

- 循环语句在条件为真时继续执行同一语句块，可以使用continue语句跳过块中某个语句直接进入下一次循环，也可以使用break语句跳出当前循环
- 通过列表推导式，可以从旧的列表中产生新的列表、对元素应用函数、过滤掉不需要的元素等等
- pass语句什么都不做，可以充当占位符使用；del语句用来删除变量（即对象的引用），但是不能删除值；exec函数执行存储在字符串中的Python代码；eval函数用于计算以字符串形式书写的表达式，并返回计算结果值

本章的新函数

eval(source[, globals[, locals]])	将字符串作为表达式计算并返回值
enumerate(seq)	产生用于迭代的（索引，值）对
range([start,] stop[, step])	创建整数列表
reversed(seq)	产生seq中值的反向版本
sorted(seq,[, cmp][,key][,reverse])	返回seq中值排序后的列表
zip(seq1, seq2,…)	创造用于并行迭代的新序列

题目1：有一分数序列：2/1，3/2，5/3，8/5，13/8，21/13...求出这个数列的前20项之和。

题目2：给定一个字符串，请判断它是不是回文（palindrome）。例如，12321是回文，此时顺读和倒读得到的都是一样的文字序列，而abcca则不是。

题目3: 请在三行中打印出前30个质数（即2、3、5...），其中每行10个。