



Chap5 Program Control Structure

第5章 程序控制结构

Department of Computer Science and Technology
Department of University Basic Computer Teaching
Nanjing University

程序控制结构



顺序

sequence
structure



选择

selection
structure



循环

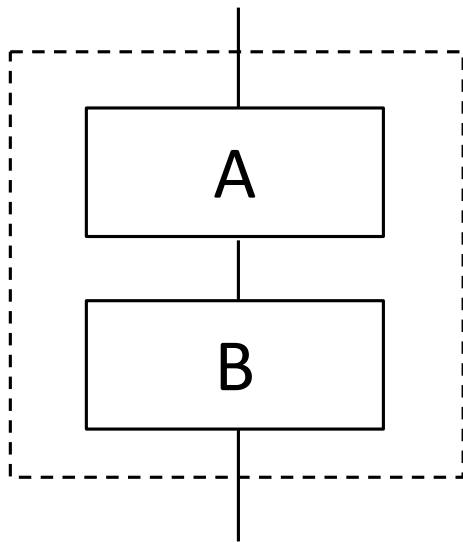
repetition
structure

5.1

顺序结构

顺序结构

4



一个入口
一个出口



Filename: seq.py

```
mystring = 'Hello, World!'  
print(mystring)
```

5.1.1 赋值语句

赋值语句

6

普通
赋值

$r = 2$

增量
赋值

$m /= 5$

链式
赋值


$b = a = a + 1$

多重
赋值

$p, r = 3, 5$

- 多重赋值的基本形式

变量1, 变量2, ..., 变量n = 表达式1, 表达式2, ..., 表达式n



```
>>> name, age = 'Niuyun', 18
>>> name
'Niuyun'
>>> age
18
```

多重赋值的本质

8

S_{ource}

```
>>> name, age = 'Niuyun', 18
```

```
>>> temp = 'Niuyun', 18
```

```
>>> temp
```

```
('Niuyun', 18)
```

```
>>> name, age = temp
```

```
>>> name
```

```
'Niuyun'
```

```
>>> age
```

```
18
```

元组打包
Tuple packing

序列解包
Sequence unpacking

多重赋值

9



```
>>> x = 3
```

```
>>> y = 5
```

```
>>> x, y = y, x
```

```
>>> x
```

```
5
```

```
>>> y
```

```
3
```

语法糖
syntactic sugar

5.1.2 基本输入和输出语句

输入/输出

11



输入

input()



输出

print()

输入函数input()

12

输入语句的一般形式:

```
x = input(['输入提示'])
```

返回值类型是str



输入input()函数

返回值类型: str

int()函数

float()函数



```
>>> x = input('Enter an integer between 0 and 10: ')
```

```
Enter an integer between 0 and 10: 3
```

```
>>> x
```

```
'3'
```

```
>>> x = int(input('Enter an integer between 0 and 10: '))
```

```
Enter an integer between 0 and 10: 3
```

```
>>> x
```

```
3
```

```
>>> y = float(input('Enter the price of the apples: '))
```

```
Enter the price of the apples: 4.5
```


```
>>> y
```

```
4.5
```

输入input()函数

返回值类型: str

eval()函数



```
>>> z = eval(input('Enter a number: '))
Enter a number: 3
>>> z
3
>>> z = eval(input('Enter the price of every apple: '))
Enter the price of every apple: 3.5
>>> z
3.5
>>> z = eval(input('Enter the price of the apples: '))
Enter the price of the apples: 6 + 3 * 4
>>> z
18
```

输入input()函数

15

返回值类型: str

eval()函数




```
>>> a, b, c = eval(input("Please enter the numbers(a,b,c): "))
Please enter the numbers(a,b,c): 12,34,567
>>> a
12
>>> b
34
>>> c
567
```

输入input()函数

16

返回值类型: str

eval()函数



```
>>> a, b, c = eval(input("Please enter the numbers(a,b,c): "))
Please enter the numbers(a,b,c): 12,34,567
>>> nums
(12, 34, 567)
>>> lst = eval(input("Please enter a list: "))
Please enter a list: [12, 34, 567]
>>> lst
[12, 34, 567]
```


输出函数print()

输出语句的一般形式:

输出到标准输出设备

```
print(对象1, 对象2, ..., 对象n, sep = ' ', end = '\n' )
```

- sep表示输出对象之间的分隔符, 默认为空格
- 参数end的默认值为'\n', 表示print()函数输出完成后自动换行



输出函数print()

18

改变sep的值

Source

```
>>> print('1', '2', '3')
```

```
1 2 3
```

```
>>> print('1', '2', '3', sep = ',')
```

```
1,2,3
```

```
>>> a, b = 454, 213
```

```
>>> print(a, '*', b, '=', sep = '', end = ''); print(a*b)
```

```
454*213=96702
```

格式化输出形式:

- `print('格式字符串' % (对象1, 对象2, ..., 对象n))`
- `print('格式化模板'.format(对象1, 对象2, ..., 对象n))`
- `print(f'...{对象1}...{对象2}...')`

输出函数print()——格式化模板

{参数的位置:[对齐说明符][符号说明符][最小宽度说明符][.精度说明符][类型说明符]}

Source

```
>>> "{0} is taller than {1}.".format("Xiaoma", "Xiaowang")
'Xiaoma is taller than Xiaowang.'
>>> age, height = 21, 1.758
>>> print("Age:{0:<5d}, Height:{1:5.2f}".format(age, height))
Age:21  , Height: 1.76
>>> print("{:,}".format(2**100))
1,267,650,600,228,229,401,496,703,205,376
```

符号	描述
b	二进制，以2为基数输出数字
o	八进制，以8为基数输出数字
x	十六进制，以16为基数输出数字，9以上的数字用小写字母（类型符为X时用大写字母）表示
c	字符，将整数转换成对应的Unicode字符输出
d	十进制整数，以10为基数输出数字
f	定点数，以定点数输出数字
e	指数记法，以科学计数法输出数字，用e（类型符是E时用大写E）表示幂
[+]m.nf	输出带符号（若格式说明符中显式使用了符号"+", 则输出大于或等于0的数时带"+"号）的数，保留n位小数，整个输出占m列（若实际宽度超过m则突破m的限制）
0>5d	右对齐，>左边的0表示用0填充左边，>右边的数字5表示输出项宽度为5
<	左对齐，默认用空格填充右边，<前后类似上述右对齐可以加填充字符和宽度
^	居中对齐
{}	输出一个{}

f-string

22

```
>>> x, y = 3, 5.678
>>> print('x={0},y={1:.2f}'.format(x, y))
>>> print(f'x={x}, y={y:.2f}')    # F
x=3, y=5.68
>>> x = 3
>>> print(f'{x*4+5}')
17
>>> print(f'PI={math.pi}')
PI=3.141592653589793
```

5.2

选择结构

5.2.1 if语句

语 法

if 表达式(条件):
语句序列

表达式 (条件)

- 简单的数字或字符
- 条件表达式:
 - 关系运算符
 - 成员运算符
 - 逻辑运算符
- True 或 False

语句序列

- 条件为True时执行的代码块
- 同一语句序列必须在同一列上进行相同的缩进 (通常为4个空格)

F_{ile}

Filename: ifpro.py

sd1 = 3

sd2 = 3

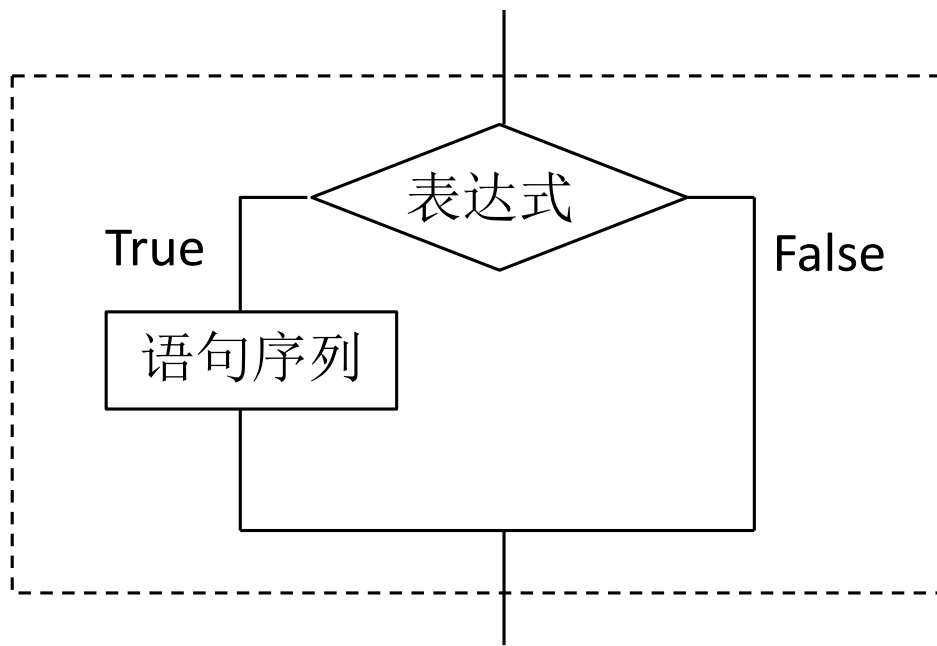
if sd1 == sd2:

print("the square's area is", sd1*sd2)

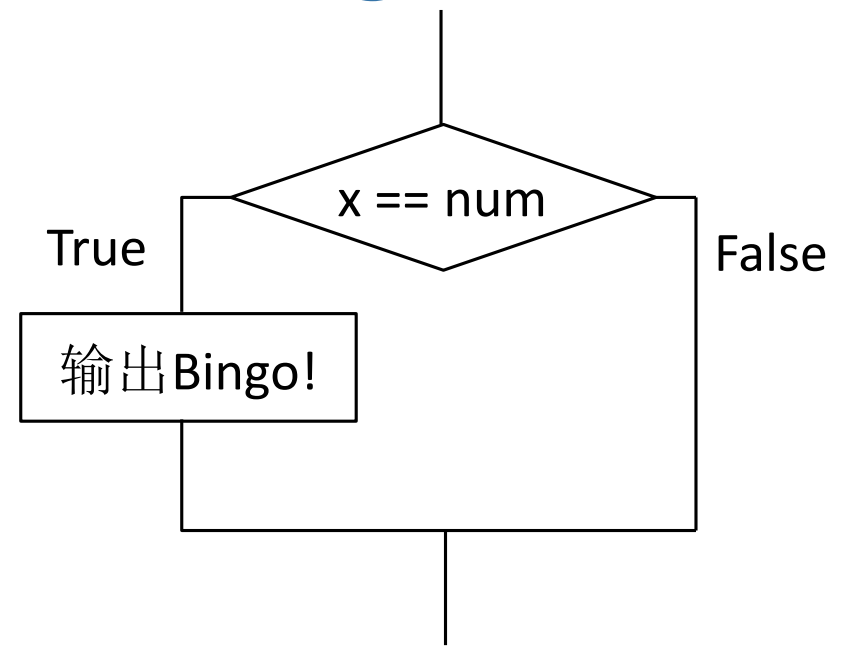
I_nput and O_utput

the square's area is 9

例5.1 程序随机产生一个0~300之间的整数， 玩家竞猜，若猜中则提示Bingo



单分支结构流程图



猜数字流程图

例5.1 程序随机产生一个0~300之间的整数， 玩家竞猜，若猜中则提示Bingo

F_{ile}

```
# prog5-1.py
```

```
from random import randint
```

```
x = randint(0, 300)
```

```
num = int(input('Please enter a number between 0~300: '))
```

```
if num == x:
```

```
    print('Bingo!')
```

很难一次性猜对！
而且毫无反馈！

5.2.2 else子句

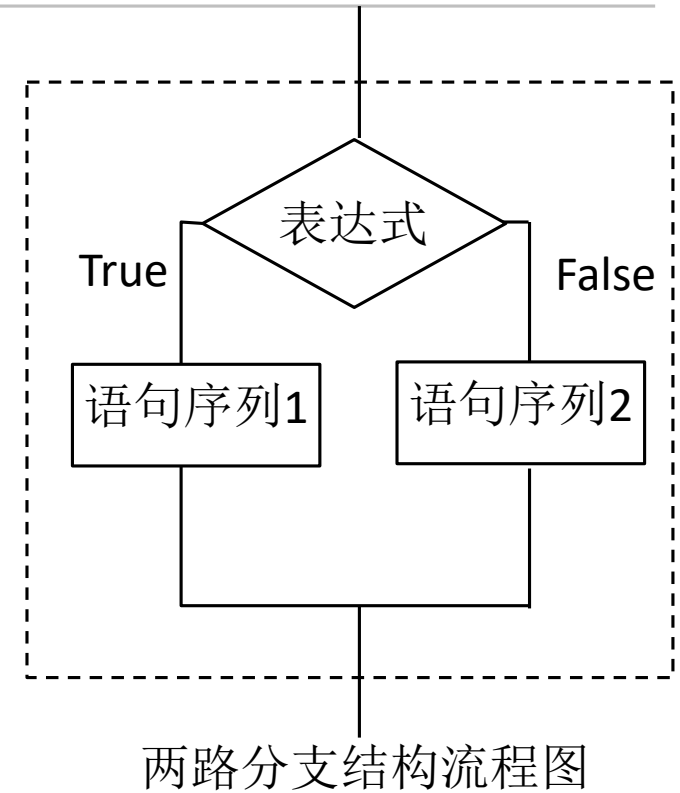
else 语句

语 法

```
if 表达式:  
    语句序列1  
else:  
    语句序列2
```

语句序列2

- 表达式条件为 False 时执行的代码块
- 代码块必须缩进
- else 语句不缩进



例5.2 程序随机产生一个0~300之间的整数，玩家竞猜，若猜中则提示Bingo，否则提示Wrong

F_{ile}

prog5-2.py

```
from random import randint
```

```
x = randint(0, 300)
```

```
num = int(input('Please enter a number between 0~300: '))
```

```
if num == x:
```

```
    print('Bingo!')
```

```
else:
```

```
    print('Fail!')
```

I_{nput and} O_{utput}

Please enter a number between 0~300: 123
Fail!

else 语句

32

File

Filename: elsepro-2.py

```
x = eval(input('Please enter the first number: '))
y = eval(input('Please enter the second number: '))
if x >= y:
    t = x
else:
    t = y
```

检查条件“ $x \geq y$ ”是否满足，若满足则取 x 否则取 y 赋给变量 t

else 语句——三元运算符

33

条件表达式（也称三元运算符）的常见形式如下所述：

$x \text{ if } C \text{ else } y$

F_{ile}

Filename: elsepro-2.py

```
x = eval(input('Please enter the first number: '))
```

```
y = eval(input('Please enter the second number: '))
```

```
if x >= y:
```

```
    t = x
```

```
else:
```

```
    t = y
```

```
t = x if x >= y else y
```

5.2.3 elif子句

语 法

```
if 表达式1:  
    语句序列1  
elif 表达式2:  
    语句序列2  
...  
elif 表达式N-1:  
    语句序列N-1  
else:  
    语句序列N
```

语句序列2

- 表达式2为True时执行的代码块

语句序列N-1

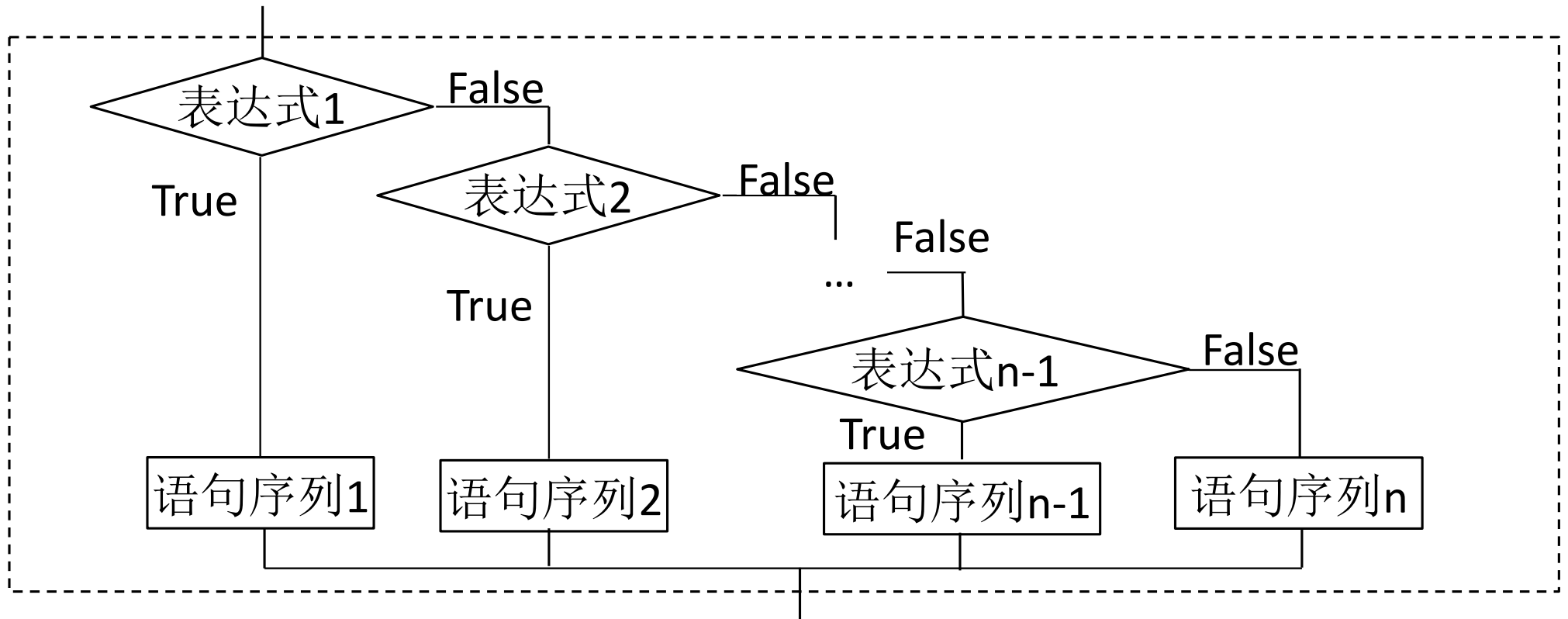
- 表达式N为True时执行的代码块

语句序列N

- 语句序列N是以上所有条件都不满足时执行的代码块

elif 语句

36



多分支结构流程图

例5.3 猜数字游戏

- 程序随机产生一个0~300之间的整数，玩家竞猜，若猜中则提示Bingo，若猜大了提示Too large，否则提示Too small



Filename: 5-3-1.py

```
from random import randint
```

```
x = randint(0, 300)
```

```
num = int(input('Please input a number between 0~300: '))
```

```
if num == x:
```

```
    print('Bingo!')
```

```
elif num > x:
```

```
    print('Too large, please try again.')
```

```
else:
```

```
    print('Too small, please try again.')
```

5.2.4 嵌套的if语句

嵌套的if语句

语 法



```
1 : if 表达式1:
2 :     if 表达式2:
3 :         语句序列1
4 :     else:
5 :         语句序列2
6 : else:
7 :     if 表达式3:
8 :         语句序列3
9 :     else:
10:         语句序列4
```

例5.3 猜数字游戏——改写代码



```
# Filename: 5-3-1.py
from random import randint
x = randint(0, 300)
num = int(input('Please input a number
between 0~300: '))
if num == x:
    print('Bingo!')
elif num > x:
    print('Too large, please try again.')
else:
    print('Too small, please try again.')
```



```
# Filename: 5-3-2.py
from random import randint
x = randint(0, 300)
num = int(input('Please input a number between 0~300: '))
if num == x:
    print('Bingo!')
else:
    if num > x:
        print('Too large, please try again.')
    else:
        print('Too small, please try again.')
```


例5.4 符号函数 (sign function)

41

- 请分别用if-elif-else结构和嵌套的if结构实现符号函数 (sign function) , 符号函数的定义:

$$\text{sgn}(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases}$$

例5.4 符号函数

F_{ile}

```
# prog5-4-1.py
x = eval(input('Enter a number: '))
if x < 0:
    sgn = -1
elif x == 0:
    sgn = 0
else:
    sgn = 1
print('sgn = {:.0f}'.format(sgn))
```

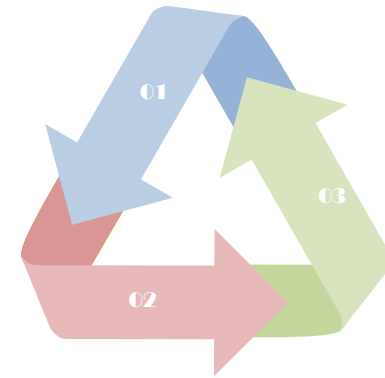
F_{ile}

```
# prog5-4-2.py
x = eval(input('Enter a number: '))
if x != 0:
    if x < 0:
        sgn = -1
    else:
        sgn = 1
else:
    sgn = 0
print('sgn = {:.0f}'.format(sgn))
```

5.3

循环结构

- 循环结构是满足一个指定的条件，每次使用不同的数据对算法中的计算或处理步骤完全相同的部分**重复**计算若干次的算法结构，也称为重复结构



5.3.1 while语句

while 循环

语法

While 表达式:
语句序列 (循环体)

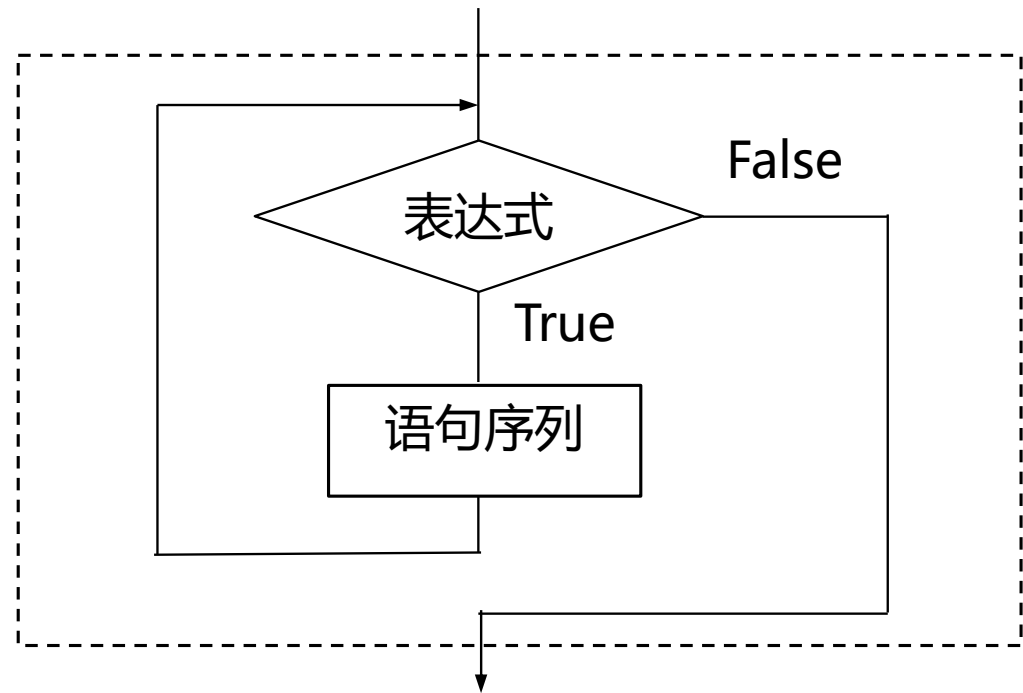
表达式

- 当表达式值为True时执行语句序列代码块
- 继续判断表达式的值是否为True,
- 若是则继续执行循环体,
- 如此周而复始, 直到表达式的值为False或发生异常时停止循环的执行
- 若循环体什么都不执行, 用pass语句表示

while 语句

有几点要注意：

- while语句是先判断再执行，所以循环体有可能一次也不执行；
- 循环体中需要包含能改变循环变量值的语句，否则表达式的结果始终是True的话会造成死循环；
- 要注意语句序列的对齐，while语句只执行其后的一条或一组同一层次的语句。



while语句流程图

例5.5 计算 $1+2+\dots+100$ 的值



```
# prog5-5.py
```

```
s = 0
```

```
i = 1
```

```
while i <= 100:
```

```
    s += i
```

```
    i += 1
```

```
print('1+2+...+100 = {:d}'.format(s))
```



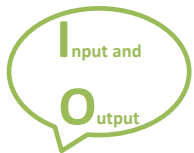
$1+2+\dots+100 = 5050$

经典累加问题

例5.6 求两个正整数的最大公约数和最小公倍数

S1: 判断 x 除以 y 的余数 r 是否为0。
若 r 为0则 y 是 x 、 y 的最大公约数，
继续执行后续操作；否则 $y \rightarrow x$ ，
 $r \rightarrow y$ 重复执行第S1步。

S2: 输出（或返回） y 。



Enter the first number: 18
Enter the second number: 24
最大公约数 = 6
最小公倍数 = 72



```
# prog5-6.py
# -*- coding: gb2312 -*-
x = eval(input("Enter the first number: "))
y = eval(input("Enter the second number: "))
z = x * y
while x % y != 0:
    x, y = y, x%y
print("最大公约数 = ", y)
print("最小公倍数 = ", z // y)
```

例5.7 计算 π

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} \dots$$

通项的绝对值小于等于 10^{-8} 时
停止计算

Input and Output

pi = 3.141592633590251

math模块中pi值等于
3.141592653589793

Source

```
# prog5-7.py
```

```
import math
```

```
x, s = 1, 0
```

```
sign = 1
```

```
k = 1
```

```
while math.fabs(x) > 1e-8:
```

```
    s += x
```

```
    k += 2
```

```
    sign *= -1
```

```
    x = sign / k
```

```
s *= 4
```

```
print("pi = {:.15f}".format(s))
```

5.3.2 for语句

for 循环

52

语 法

for 变量 in 可迭代对象:
语句序列

可以明确循环的次数

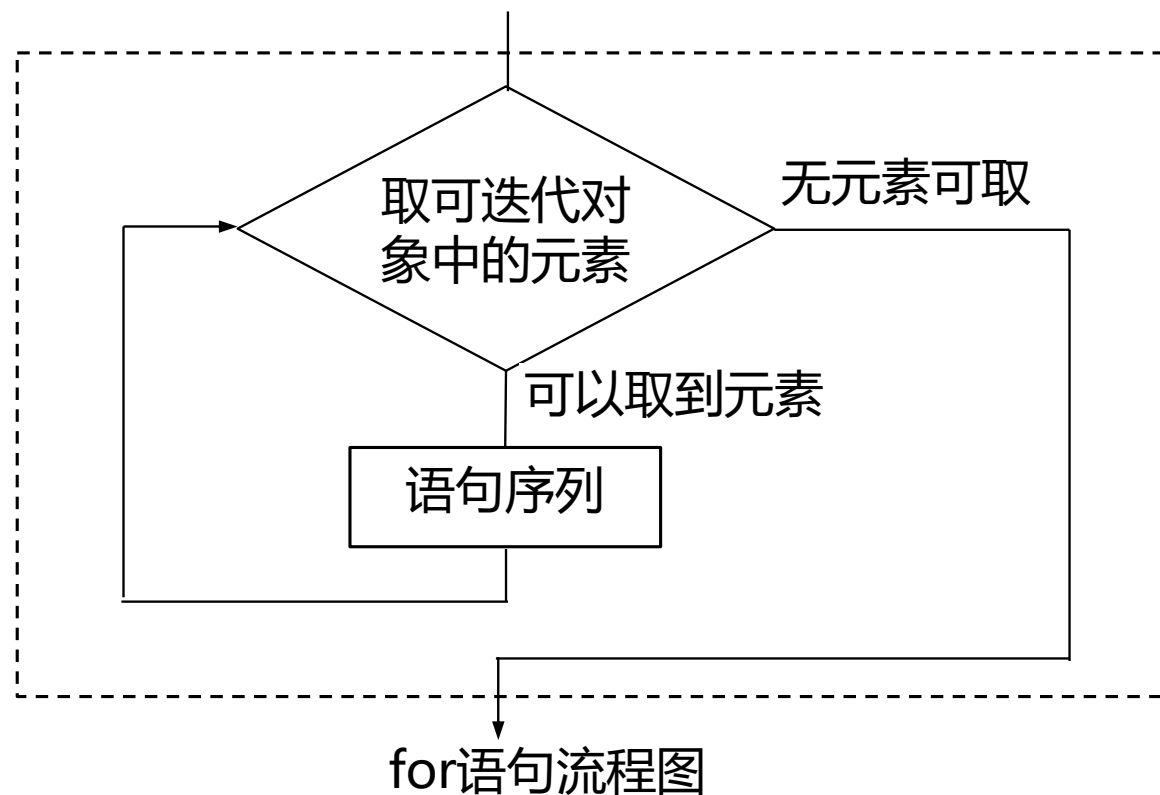
- 遍历一个数据集内的成员
- 在列表解析中使用
- 生成器表达式中使用

可迭代对象

- 序列
- 迭代器
- 其他可迭代对象（字典的键、文件的行）


可迭代对象指可以按次序迭代（循环）的对象，包括序列、迭代器（iterator）以及其他可以迭代的对象如字典的键和文件的行等。

执行时变量取可迭代对象中的一个值，执行语句序列，再取下一个值，执行语句序列



for 循环

54



```
>>> aList = [1, 2, 3]
>>> for item in aList:
    print(item)
1
2
3
>>> for item in enumerate(['a', 'b', 'c']):
    print(item)
(0, 'a')
(1, 'b')
(2, 'c')
```

猜数字游戏

- 程序随机产生一个0~300间的整数，玩家竞猜，允许猜多次，系统给出“猜中”、“太大了”或“太小了”的提示。



```
# Filename: guessnum2.py
from random import randint
x = randint(0, 300)
for count in range(5):
    digit = int(input('Please input a number between 0~300: '))
    if digit == x:
        print('Bingo!')
    elif digit > x:
        print('Too large, please try again.')
    else:
        print('Too small, please try again.')
```

可迭代对象 和 迭代器

可迭代对象

- 能用for循环遍历的对象都可被称为可迭代对象
- 字符串、列表等

迭代器

- 属于可迭代对象，for语句会通过`__iter__()`方法获得对象的迭代器，并通过`__next__()`获取下一个元素

可迭代对象 和 迭代器

S_{ource}

```
>>> aList = [1, 2, 3]
>>> for item in aList:
    print(item)
```

```
1
2
3
```

S_{ource}

```
>>> i = iter([1, 2, 3])
```

```
>>> next(i)
```

```
1
```

```
>>> next(i)
```

```
2
```

```
>>> next(i)
```

```
3
```

```
>>> next(i)
```

Traceback (most recent call last):

File "<pyshell#16>", line 1, in <module>

i.__next__()

StopIteration

可迭代对象 和 迭代器


58



```
>>> from collections import Iterable, Iterator # 导入Iterable和Iterator类
>>> isinstance(aList, Iterable)
True
>>> isinstance(aList, Iterator)
False
>>> isinstance(iter(aList), Iterator)
True
>>> isinstance(enumerate('abc'), Iterator)
True
```

可迭代对象 和 迭代器

- 迭代器转换成可迭代对象



```
>>> enumerate('abc')
<enumerate object at 0x000001F56219EDC8>
>>> list(enumerate('abc'))
[(0, 'a'), (1, 'b'), (2, 'c')]
```

for语句的迭代

60

序列项

01

02

序列索引

序列项
和索引

03

04

字典的键

文件的行

05

for 语句迭代——序列项迭代

61

S_{source}

```
>>> s = ['I', 'love', 'Python']
>>> for word in s:
    print(word, end = ' ')
I love Python
>>> for i in range(1, 5):
    print(i * i)

1
4
9
16
```

for 语句迭代——序列索引迭代

62

Source

```
>>> s = ['I', 'love', 'Python']  
>>> for i in range(len(s):  
        print(s[i], end = ' ')  
I love Python
```

for 语句迭代——迭代器迭代

63

Source

```
>>> courses = ['Maths', 'English', 'Python']
```

```
>>> scores = [88, 92, 95]
```

```
>>> for c, s in zip(courses, scores):  
    print('{0} - {1:d}'.format(c, s))
```

Maths - 88

English - 92

Python - 95

for 语句迭代——其他迭代

64

Source

```
>>> d_stock = {'AXP': '78.51', 'BA': '184.76', 'CAT': '96.39'}
```

```
>>> for k, v in d_stock.items():  
        print('{0:>3}: {1}'.format(k, v))
```

```
AXP: 78.51
```

```
BA: 184.76
```

```
CAT: 96.39
```

```
>>> for k in d_stock.keys():  
        print(k, d_stock[k])
```

```
AXP 78.51
```

```
BA 184.76
```

```
CAT 96.39
```


迭代中修改迭代对象



```
lst = [1, 2, 4, 3, 5]
for item in lst:
    if item % 2 == 0:
        lst.remove(item)
print(lst)
```

在遍历可变对象的同时删除了列表元素，影响了迭代器的迭代机制，要避免使用，类似的还有pop、insert和append等操作

例5.8 求斐波纳契 (Fibonacci) 数列前20项

66

斐波纳契数列:

0, 1, 1, 2, 3, 5, 8, 13, 21,
34, 55, 89, 144, ...

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_{I+1} = F_{I-1} + F_I \end{cases}$$

F_{ile}

```
# prog5-8.py  
f = [0] * 20  
f[0], f[1] = 0, 1  
for i in range(2, 20):  
    f[i] = f[i-1] + f[i-2]  
print(f)
```

I_{input} and O_{output}

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]

例5.9 统计英语句子中单词词频

从键盘输入一个英文句子，除单词和空格外句子中只包含“,”、“.”、“'”、“'”和“!”这几个标点符号，统计句子中包括的每个单词（将句中大写全部转换成小写）的词频并将结果存入字典中并输出。

例5.9 统计英语句子中单词词频

F_{ile}

prog5-9.py

s = input("Enter an English sentence: ")

s = s.lower()

sList = s.split()

sDict = {}

for item in sList:

if item[-1] in ',.\\"!':

item = item[:-1]

if item not in sDict:

sDict[item] = 1

else:

sDict[item] += 1

print(sDict)

I_{nput} and O_{utput}

Enter an English sentence: A friend in need is a friend indeed.

{'a': 2, 'friend': 2, 'in': 1, 'need': 1, 'is': 1, 'indeed': 1}

- 利用字典作词频统计的程序写法在Python中很常用

```
for item in sList:  
    处理标点  
    sDict[item] = sDict.get(item, 0) + 1
```

也可用collections模块的Counter()函数

例5.10 输出公司代码和股票价格

假设已有若干道琼斯工业指数成分股公司某个时期的财经数据，包括公司代码、公司名称和股票价格：

```
>>> stockList = [('AXP', 'American Express Company', '78.51'),  
                  ('BA', 'The Boeing Company', '184.76'),  
                  ('CAT', 'Caterpillar Inc.', '96.39')]
```

从数据中获取公司代码和股票价格对并输出。

例5.10 输出公司代码和股票价格

用序列索引迭代

File

prog5-10-1.py

```
stockList = [('AXP', 'American Express Company',
               '78.51'), ('BA', 'The Boeing Company',
               '184.76'), ('CAT', 'Caterpillar Inc.', '96.39')]
```

```
aList = []
```

```
bList = []
```

```
for i in len(stockList):
```

```
    aStr = stockList[i][0]
```

```
    bStr = stockList[i][2]
```

```
    aList.append(aStr)
```

```
    bList.append(bStr)
```

```
stockDict = dict(zip(aList, bList))
```

```
print(stockDict)
```

Iinput and **O**utput

```
{'CAT': '96.39', 'BA': '184.76', 'AXP': '78.51'}
```

例5.10 输出公司代码和股票价格

用序列迭代

Input and Output

```
{'CAT': '96.39', 'BA': '184.76', 'AXP': '78.51'}
```



```
# prog5-10-2.py
```

```
stockList = [('AXP', 'American Express Company',  
              '78.51'), ('BA', 'The Boeing Company',  
                        '184.76'), ('CAT', 'Caterpillar Inc.', '96.39')]
```

```
stockDict = {}
```

```
for data in stockList:
```

```
    stockDict[data[0]] = data[2]
```

```
print(stockDict)
```


5.3.3 嵌套循环

- while语句和for语句可以嵌套自身语句结构，也可以相互嵌套，可以呈现各种复杂的形式。

例5.11 编写程序统计一元人民币换成一分、两分和五分的所有兑换方案个数

75

File

```
# prog5-11.py
```

```
i, j, k = 0, 0, 0 # i, j, k分别代表五分、两分和一分的数量
```

```
count = 0
```

```
for i in range(21):
```

```
    for j in range(51):
```

```
        k = 100 - 5 * i - 2 * j
```

```
        if k >= 0: #k是一分，可以是任意个
```

```
            count += 1
```

```
print('count = {:d}'.format(count))
```

Input and Output

count = 541

可以用while替换for

例5.12 两个列表的新组合

- 从两个列表中分别选出一个元素，组成一个元组放到一个新列表中，要求新列表中包含所有的组合

Input and Output

```
[('C++', 2), ('C++', 3), ('C++', 4),  
('Java', 2), ('Java', 3), ('Java', 4),  
('Python', 2), ('Python', 3),  
('Python', 4)]
```

File

```
# prog5-12.py
```

```
result = []
```

```
pdList = ['C++', 'Java', 'Python']
```

```
creditList = [2, 3, 4]
```

```
for pdl in pdList:
```

```
    for credit in creditList:
```

```
        result.append((pdl, credit))
```

```
print(result)
```

例5.13 杨辉三角的输出

根据输入的行数 n ，输出 n 行杨辉三角形，杨辉三角形的排列规律是每一行两端都是1，其余各数都是上一行中与此数最相邻的两数之和，它最早由北宋数学家贾宪发现，收录在南宋数学家杨辉的著作中。5行杨辉三角形如下所示：

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

例5.13 杨辉三角的输出

F_{ile}

```
# prog5-13.py
n = int(input("enter the number of layers: "))
lstPre = [1]
print(lstPre)
for i in range(n-1):
    lstPre.insert(0, 0)
    lstPre.append(0)
    lstNext = []
    for i in range(len(lstPre) - 1):
        lstNext.append(lstPre[i] + lstPre[i+1])
    lstPre = lstNext[:]
    print(lstNext)
```

I_{nput and} O_{utput}

enter the number of layers: 7

[1]

[1, 1]

[1, 2, 1]

[1, 3, 3, 1]

[1, 4, 6, 4, 1]

[1, 5, 10, 10, 5, 1]

[1, 6, 15, 20, 15, 6, 1]

例5.14 编程计算 $1+2!+3!+\dots+50!$ 的和

79

File

```
# prog5-14-1.py
s = 0
for i in range(1, 51):
    product = 1
    for j in range(2, i+1):
        product *= j
    s += product
print(s)
```

File

```
# prog5-14-2.py
item, s = 1, 0
for i in range(1, 51):
    item *= i
    s += item
print(s)
```

递推法

Input and Output

31035053229546199656252032972759319953190362094566672920420940313

运行时间结果对比

用time模块中的clock()函数测试这两个程序的执行时间，将以下两条语句分别加到prog5_14_1和prog5_14_2的前后：

```
t0 = time.perf_counter()  
print(time.perf_counter() - t0, "seconds process time")
```

某次测试的运行时间结果对比为：

0.0004589000018313527

3.220001235604286e-05

可以看出，用一重循环的递推法效率要高于二重循环的非递推法

5.3.4 break, continue语句

break 语句

- break语句终止当前循环，转而执行循环之后的语句

循环非正常结束



Filename: breakpro.py

```
s = 0
```

```
i = 1
```

```
while i < 10:
```

```
    s += i
```

```
    if s > 10:
```

```
        break
```

```
    i += 1
```

```
print('i = {0:d}, sum = {1:d}'.format(i, s))
```



i=5, s=15

s 是 1+2+3+... 不断累加的和，当 i 等于 5 时 s 第一次大于 10，执行 break 语句跳出 while 循环语句，继而去执行 print 语句

break 语句

“while i < 10:” 意义不大，常会将此条语句替换成另一种在Python中常与break语句一起使用的循环控制语句 “while True:”



Filename: breakpro.py

```
s = 0
```

```
i = 1
```

```
while True:
```

```
    s += i
```

```
    if s > 10:
```

```
        break
```

```
    i += 1
```

```
print('i = {0:d}, sum = {1:d}'.format(i, s))
```

while i < 10:




while True:

break 语句

i 和 j 分别等于4和9时 i 和 j 的乘积第一次等于36，如果break语句可以跳出两重循环的话则输出结果应该是“4 9”，而程序的实际输出结果是“10 4”，所以break只能跳出紧包层即break所在层次的循环。

两重循环



```
for i in range(11):  
    for j in range(11):  
        if i * j >= 36:  
            break  
    print(i, j)
```

例5.15 输出2~100之间的素数，每行显示5个

85

素数 (prime) : 只能被1和n自身整除的正整数n (13是素数, 6不是素数)。

素数判断算法:

- 若n不能被2~n-1范围内的任一个整数整除n就是素数, 否则n不是素数
- 如果发现n能被某个整数整除可立即停止继续判断n是否能被范围内其他整数整除。

I nput and O utput

```
2 3 5 7 11
13 17 19 23 29
31 37 41 43 47
53 59 61 67 71
73 79 83 89 97
```

$2 \sim n/2$

or

$2 \sim \sqrt{n}$

例5.15 输出2~100之间的素数，每行显示5个

86

for 循环和break

I nput and O utput

```
2 3 5 7 11
13 17 19 23 29
31 37 41 43 47
53 59 61 67 71
73 79 83 89 97
```

F_{ile}

```
# Filename: 5-15.py
from math import sqrt

count = 0
for i in range(2, 101):
    flag = True
    k = int(sqrt(i))
    for j in range(2, k+1):
        if i % j == 0:
            flag = False
            break
    if flag:
        count += 1
        if count % 5 == 0:
            print(i, end = '\n')
        else:
            print(i, end = ' ')
```

continue 语句

- 在while和for循环中，continue语句的作用：
 - 跳过循环体内continue后面的语句，并开始新一轮循环
 - while循环则判断循环条件是否满足
 - for循环则判断迭代是否已经结束

continue语句

程序的功能是对于在1~20之间的数，当它是3的倍数时执行print(i, end = ' ')函数调用语句，当它不是3的倍数时执行continue语句，跳过其后的print()函数调用语句继续执行下一轮循环

File

```
for i in range(1,21):  
    if i % 3 != 0:  
        continue  
    print(i, end = ' ')
```


Input and Output

3 6 9 12 15 18

continue语句


89

循环中的break:



```
for i in range(1,21):  
    if i % 3 != 0:  
        break  
    print(i, end = ' ')
```

循环中的continue:




```
for i in range(1,21):  
    if i % 3 != 0:  
        continue  
    print(i, end = ' ')
```

break	continue
break语句跳出所有轮循环	continue语句则是跳出本轮循环
没有任何输出	输出1-20之间所有3的倍数"3 6 9 12 15 18"

continue语句


90

循环中的continue:



```
for i in range(1,21):  
    if i % 3 != 0:  
        continue  
    print(i, end = ' ')
```

循环中的替代continue:



```
for i in range(1,21):  
    if i % 3 == 0:  
        print(i, end = ' ')
```

5.3.5 循环结构中的else子句

- 循环中的else子句：
 - 如果循环代码从break处终止，跳出循环
 - 正常结束循环，则执行else中代码

例5.16 输入一个整数，并判断其是否为素数

93

如果输入的整数有有效因子（除了1和它本身的因子）则可以直接输出非素数的结论并继而执行break语句跳出循环，如果没有执行过break语句则表示循环正常结束，也就是整数并无有效因子则表明它是一个素数，则执行else子句的输出语句。



Filename: 5-16.py

```
from math import sqrt
```

```
num = int(input('Please enter a number: '))
```

```
for j in range(2, int(sqrt(num)+1)):
```

```
    if num % j == 0:
```

```
        print('{:d} is not a prime.'.format(num))
```

```
        break
```

```
else:
```

```
    print('{:d} is a prime.'.format(num))
```

例5.17 猜数字游戏（想停就停，非固定次数）94

程序随机产生一个0~300间的整数，玩家竞猜，允许玩家自己控制游戏次数，如果猜中系统给出提示并退出程序，如果猜错给出“太大了”或“太小了”的提示，如果不想继续玩可以退出并说再见。

例5.17 猜数字游戏 (想停就停, 非固定次数)

95

I nput and O utput

Please enter a number between 0~300: 123
Too small, please try again.
Enter y if you want to continue: y
Please enter a number between 0~300: 200
Too large, please try again.
Enter y if you want to continue: y
Please enter a number between 0~300: 150
Too small, please try again.
Enter y if you want to continue: n
I quit and byebye!

F ile

```
# Filename: prog5-17.py
from random import randint
x = randint(0, 300)
go = 'y'
while (go == 'y'):
    digit = int(input('Please enter a number between 0~300: '))
    if digit == x:
        print('Bingo!')
        break
    elif digit > x:
        print('Too large, please try again.')
    else:
        print('Too small, please try again.')
    print('Enter y if you want to continue.')
    go = input()
    print(go)
else:
    print('I quit and byebye!')
```

5.3.6 特殊的循环——列表解析

- Python中有一种特殊的循环，通过for语句结合if语句，生成新列表，这种特殊的轻量级循环称为列表解析（list comprehension，也译作列表推导式）。

- 列表解析中的多个for语句相当于是for结构的嵌套使用

```
[ 表达式 for 表达式1 in 序列1  
    for 表达式2 in 序列2  
    ...  
    for 表达式N in 序列N  
    if 条件 ]
```

创建一个从0到9的简单的整数序列;



```
>>> [x for x in range(10)]  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

对range(10)中每一个值求平方数；

A speech bubble icon containing the word "Source" in orange.

```
>>> [x ** 2 for x in range(10)]  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

对range(10)中的每一个值加入if语句“if x ** 2 < 50”，只生成比50小的平方数；

Source

```
>>> [x ** 2 for x in range(10) if x ** 2 < 50]  
[0, 1, 4, 9, 16, 25, 36, 49]
```

使用嵌套的for语句。(x + 1, y + 1)的所有组合就包含了x从0-1, y也从0-1的变化。

A small orange speech bubble icon containing the word "Source" in a stylized font.

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

对一个包含多个元素（也是列表）的列表中的每一个元素
line取其第0个元素，即1,4,7。

Source

```
>>> [line[0] for line in [[1,2,3], [4,5,6], [7,8,9]]]  
[1, 4, 7]
```

例5.12 列表解析方法

104

Source

```
>>> pdlList = ['C++', 'Java', 'Python']
>>> creditList = [2, 3, 4]
>>> [(pdl, credit) for pdl in pdlList for credit in creditList]
[('C++', 2), ('C++', 3), ('C++', 4), ('Java', 2), ('Java', 3), ('Java', 4), ('Python', 2), ('Python', 3), ('Python', 4)]
```


列表解析将数字字符串转换成全部整数

105



```
>>> data = [int(x) for x in input("enter the nums: ").split(',')]
enter the nums: 1,2,3,4,5
>>> data
[1, 2, 3, 4, 5]
```

字典解析&集合解析

106

```
>>> dt = {'A':1,'B':2}
```

```
>>> {v:k for k, v in dt.items()}
```

```
{1: 'A', 2: 'B'}
```

```
>>> s = {1, 10, 12, 32, 100}
```

```
>>> {x%5 for x in s}
```

```
{0, 1, 2}
```

```
>>> {chr(x): 0 for x in range(97, 123)}
```

```
>>> s = 'abcbxccc'
```

```
>>> {w: s.count(w) for w in s}
```

- 语法形式上与列表解析很相似的生成器表达式，只是把 “[...]” 换成 “(...)” 。
- 生成器是一个返回迭代器的函数，是一种特殊的迭代器，它在内部实现了迭代器协议，每次计算出一个条目后把这个条目“产生” (yield) 出来，使用 “惰性计算” (lazy evaluation) 机制，只有在检索时才被赋值，因此在大数据量处理时有优势。

Source

```
>>> data = (x for x in range(10))
```

```
>>> data
```

```
<generator object <genexpr> at 0x000002B9FAD10360>
```

```
>>> sum(x for x in range(10))
```

```
45
```

```
>>> for i in data:
```

```
    print(i, end = ' ')
```

```
0 1 2 3 4 5 6 7 8 9
```

- 顺序结构
- 选择结构
- 循环结构
- 列表解析

