

《Python程序设计》

Python语句控制

刘潇

机械科学与工程学院

2023年10月19日

2023秋季

本节要点

- 掌握bool表达式的运算和条件语句结构
- 掌握循环语句的迭代过程和break、continue等语句
- 了解os模块，结合条件和循环语句操作文件

主要内容

1. Python语句

2. 条件语句

3. 循环语句

4. 文件操作

Python语句

Python语句是Python程序执行的基本单元，已经学习过的语句有print语句、赋值语句、import语句、del语句、表达式语句等

Python语句

```
1 import math as m
2 a = 2 + 2
3 b = a
4 print(m.sqrt(a))
5 del a
6 print(b)
```

2.0
4

import语句，导入模块

- import math
- from math import sqrt
- from math import *
- import math as m
- from math import sqrt as s

print语句在Python3.0中为函数

`print(*objects, sep=' ', end='\n', file=sys.stdout)`

赋值语句

- **并行赋值**：等号左右两边目标个数需相同
- **链式赋值**：多变量引用，需注意Python内存管理
- **运算符赋值**：将表达式替换成赋值运算符，代码简洁

赋值语句

```
1  #并行赋值
2  x, y, z = 1, 2, 3
3  print(x)
4  x, *y, z = 1, 2, 3, 4
5  print(y)
6
7  #链式赋值
8  x = y = "hello world!"
9
10 #运算符赋值
11 x = "hello"
12 x += " world!"
13 x *= 2
14 print(x)
```

变量名前加*，利用列表收集多余的值

等价于 $x = \text{"hello world!"}$
 $y = x$

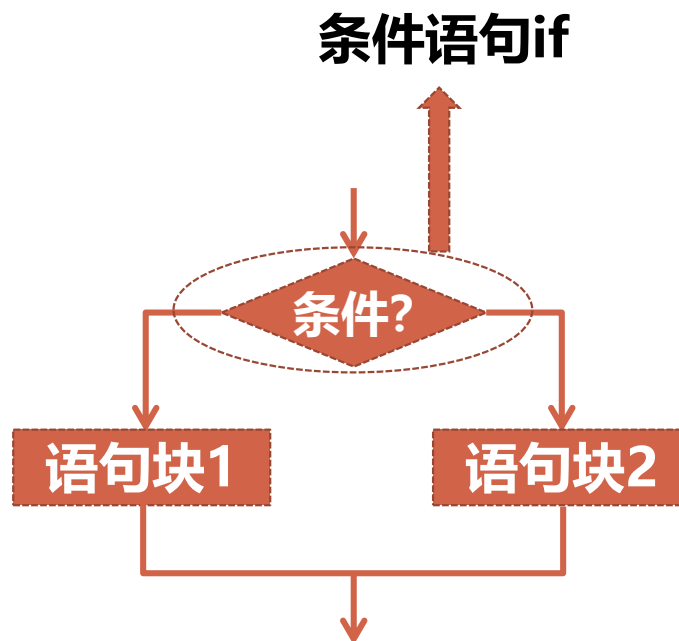
赋值运算符可用于多种数据类型

```
1
[2, 3]
hello world!hello world!
```

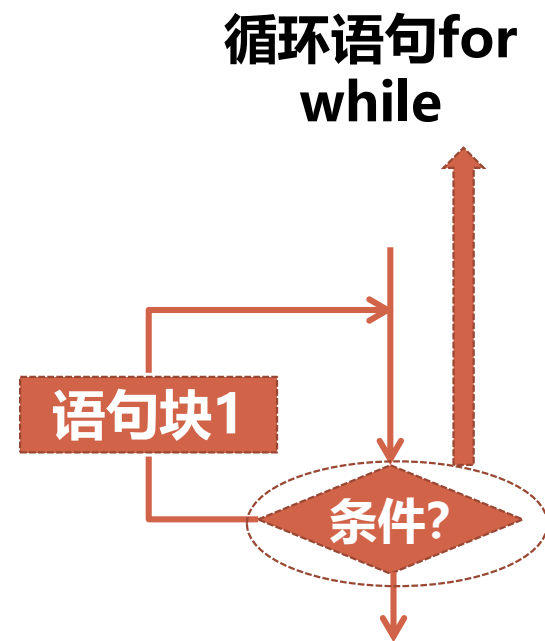
Python程序控制结构



顺序结构



分支结构



循环结构

条件语句

□ 条件语句是用来判断给定的条件是否满足，并根据判断的结果（True或False）决定是否执行或如何执行后续流程

条件语句

```
1 age = float(input("input your age: "))
2 if age < 1:
3     print("he is a baby!")
4
5 print(0.5 < 1)
```

条件语句：if和冒号之间的表达式为条件

0.5 < 1表达式的
返回值为True

**注意代码块严格缩进
(四个空格!!!)**

```
input your age: 0.5
he is a baby!
True
```

Python中False、None、0、""、()、[]、{}作为if的条件表达式时为假（False），其他值都为真（True）

Python的真假差别为“有些东西”和“没有东西”的差别

bool类型

□ True和False属于bool类型，bool()函数与list()、str()、tuple()等函数一样可以用来转换其他的值

bool类型

```
1 # True和False分别是1和0的别名
2 print(True == 1)
3 print(False == 0)
4 print(True + False + 40)
5
6 age = float(input("input your age: "))
7 if age + 0.5:
8     print("he is a baby!")
9
10 # 转换数据类型
11 print(bool("hello world"))
12 print(bool("40"))
13 print(bool(""))
```

True和False分别是1和0的别名

“有些东西” 都为True

```
True
True
41
input your age: -0.5
True
True
False
```


复杂的条件

比较运算符 (返回True或者False)

`x == y`

x等于y

`x != y`

x不等于y

`x > y`

x大于y

`x < y`

x小于y

`x >= y`

x大于等于y

`x <= y`

x小于等于y

`x is y`

x和y是同一个对象

`x is not y`

x和y是不同的对象

`x in y`

x是容器y的成员

`x not in y`

x不是容器y的成员

逻辑运算符 (返回True或者False, 或者对应的对象)

`x and y`

如果两个语句都为真, 则返回 True

`x or y`

如果其中一个语句为真, 则返回 True

`not x`

反转结果, 如果结果为 true, 则返回 False

bool表达式

bool表达式

```
1 # 比较运算符返回True或者False
2 print(1 < 2)
3 a, b = [1, 2], [1, 2]
4 print(a is b)
5
6 # 逻辑运算符
7 print(True and False)
8
9 print(1 and 2)
10 print(1 or 2)
11 print(1 and 0)
12 print(not 2)
```

返回True或者False

返回对象

返回True或者False

```
True
False
False
2
1
0
False
```

1 or 2: 先判断1为真，后面是or则表示整个表达式为真，那么就不用看后面的值了

复杂的条件

复杂的条件

```
1 name = input("Input you name: ")
2 if "h" in name:
3     print("your name contains the letter 'h'.")
4
5 number = float(input("Input a number:"))
6 if number < 10 and number > 1:
7     print("Great!")
8
9 if 1 < number < 10:
10    print("Great!")
```

1 < number < 10

链式比较

```
Input you name: hello
your name contains the letter 'h'.
Input a number:5
Great!
Great!
```

else和elif子句

利用else子句给if语句**增加一种选择**，elif是包含条件的else子句。else子句不是独立的语句，是if语句的一部分

else和elif子句

```
1 number = float(input("Input a number: "))
2
3 if number > 0:
4     print("The number is positive.")
5 elif number < 0:
6     print("The number is negative.")
7 else:
8     print("The number is zero.")
9
10 print("The number is positive." if number > 0 else "The number is negative.")
```

注意if, elif和else
后面的冒号

Input a number: 2

The number is positive.

The number is positive.

条件成立
的值

条件

条件不成立的值

更简洁的表达 (**三目运算**)

三目运算符 (if else)

exp1 if condition else exp2

condition为判断条件，**exp1**和**exp2**为表达式，
条件为真，执行**exp1**并把结果作为整个表达式的
结果，条件为假，执行**exp2**并把结果作为整个表
达式的结果

C语言中 a?b:c
if (a) {
 return b
} **else** {
 return c
}

三目运算

```
1 a = 3 < 2 and 2 or 3
2 print(a)
3
4 a = 2 if 3 < 2 else 3
5 print(a)
6
7 b = 5 if False and 1 else 6
8 print(b)
```

3
3
6

可以用elif吗?

多分支结构

if 表达式1:

语句块1

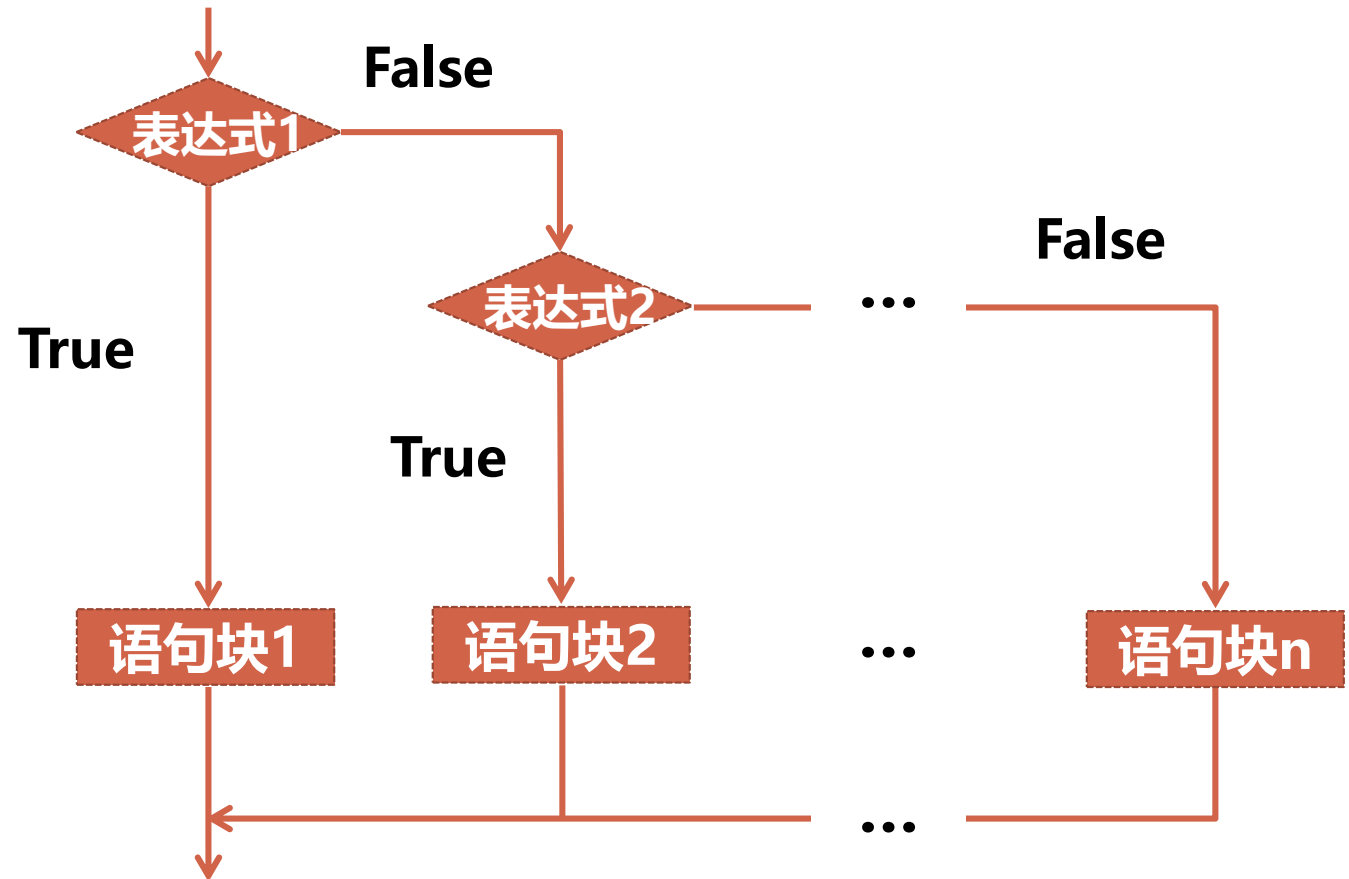
elif 表达式2:

语句块2

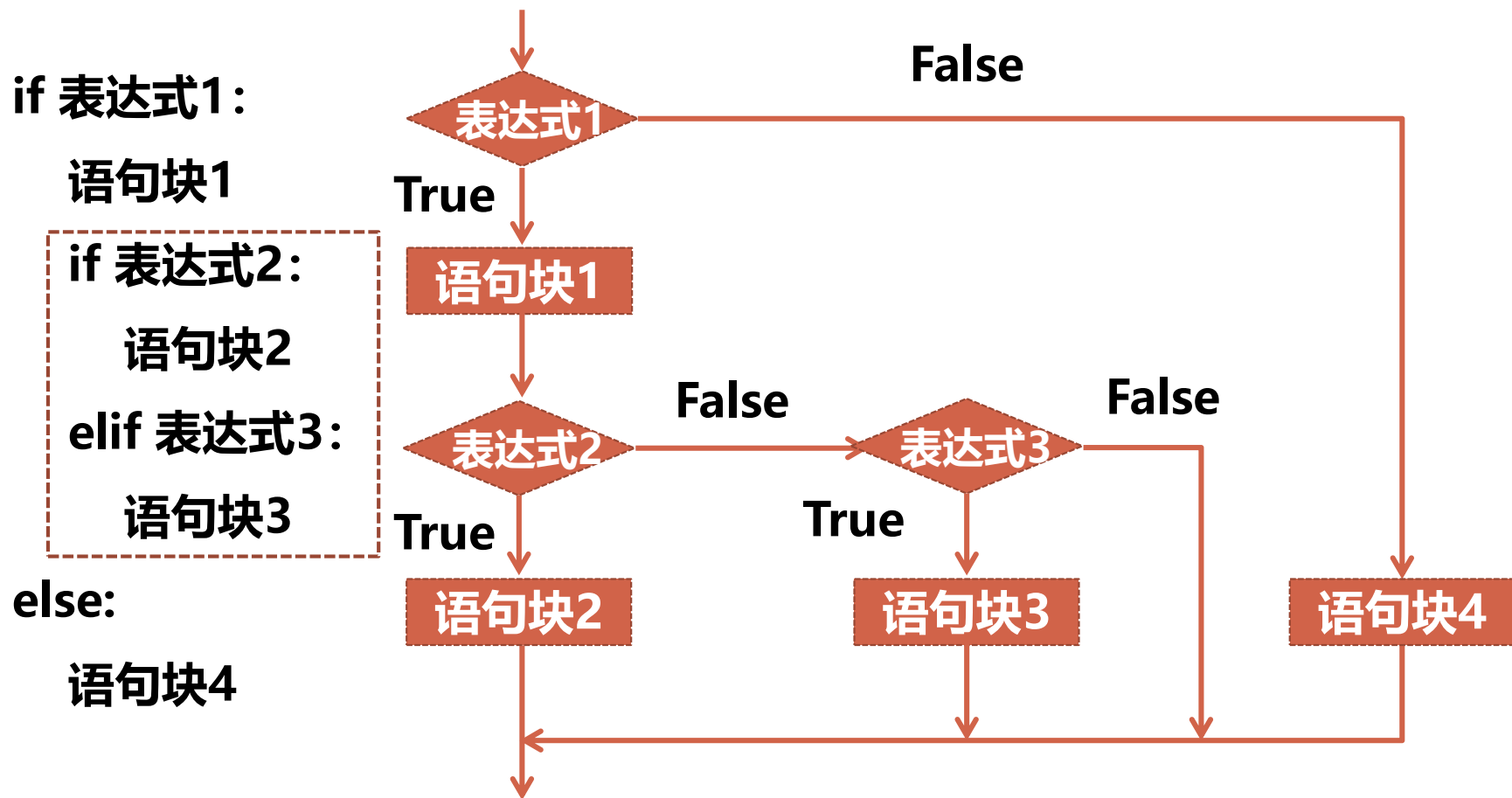
...

else:

语句块n



嵌套结构



嵌套if语句

条件语句示例

输入一个整数，判断它是否能同时被2和3整除

条件语句示例

```
1 number = int(input("Input a number: "))
2 if number % 2 == 0:
3     if number % 3 == 0:
4         print(str(number) + "能同时被2和3整除")
5     else:
6         print(str(number) + "能被2整除，不能被3整除")
7 elif number % 3 == 0:
8     print(str(number) + "能被3整除，不能被2整除")
9 else:
10    print(str(number) + "不能被2和3整除")
```

Input a number: 50
50能被2整除，不能被3整除

断言语句assert

断言可以在条件不满足程序运行的情况下直接返回错误，而不必等待程序运行后出现崩溃的情况

断言assert

```
1 import sys
2 #print(dir(sys))
3 #print(sys.__doc__)
4
5 assert ('linux' in sys.platform), "该代码只能在 Linux 下执行"
```

条件为false时触发异常

```
-----
AssertionError                                Traceback (most recent call last)
<ipython-input-14-9a27a863c9ce> in <module>
      3 #print(sys.__doc__)
      4
----> 5 assert ('linux' in sys.platform), "该代码只能在 Linux 下执行"

AssertionError: 该代码只能在 Linux 下执行
```

pass语句

pass语句是不执行任何操作的语句，相当于语句中的占位符，可以跟数据类型中的None类比

pass语句

```
1 number = int(input("Input a number: "))
2 if number % 2 == 0:
3     if number % 3 == 0:
4         print(str(number) + "能同时被2和3整除")
5     else:
6         #还不知道写啥
7 else:
8     print(str(number) + "不能被2和3整除")
```

用pass让程序继续执行

File "<ipython-input-87-e92bc170d710>", line 7

```
else:
    ~
```

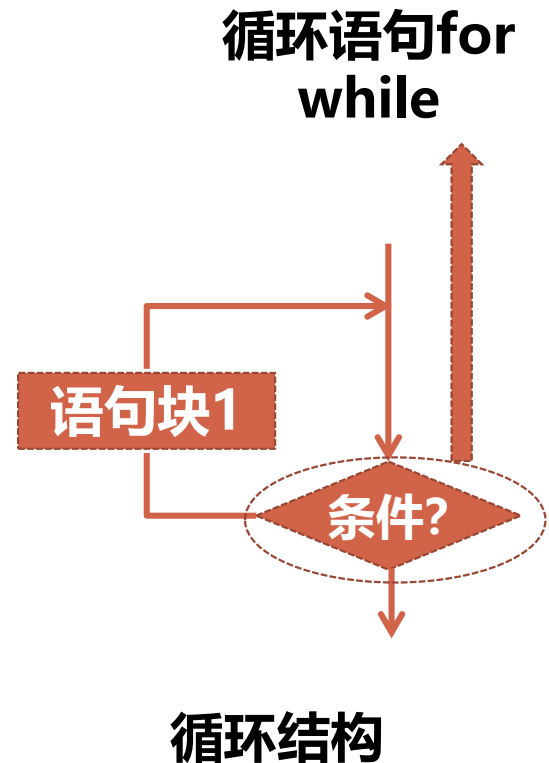
IndentationError: expected an indented block

循环语句

□ 循环语句是用来重复多次操作语句块，Python提供**while**和**for**两种循环语句

while语句：用于编写通用循环的方法，条件为真是一直循环

for语句：用于遍历序列对象内的元素，通常用于已知的循环次数



while循环语句

□ while循环：当表达式的值为真时，执行相应的语句块。然后再判断表达式的值，如果为真，则继续执行语句块循环语句是用来重复多次操作语句块，**直到表达式的值为假**

while语句

```
1 x, i = 0, 1
2 while i <= 100:
3     x += i
4     i += 1
5 print(x)
```

5050

while和: 之间为表达式

i递增，当i大于100时，表达式的值为假

求1到100的累加和

打印9*9乘法口诀表

打印9 * 9乘法口诀表

```
1 row = 1
2 while row <= 9:
3     col = 1
4     while col <= row:
5         print("{}*{}={:<2}".format(row, col, row*col), end = " ")
6         col += 1
7     print("\n", end = "")
8     row += 1
```

print函数默认会换一行

```
1*1=1
2*1=2  2*2=4
3*1=3  3*2=6  3*3=9
4*1=4  4*2=8  4*3=12  4*4=16
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25
6*1=6  6*2=12  6*3=18  6*4=24  6*5=30  6*6=36
7*1=7  7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
8*1=8  8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64
9*1=9  9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

break和continue语句

break用于结束或跳出循环，**continue**只是结束当前迭代，然后跳到下一次迭代开头

break和continue语句

```
1 while True:
2     name = input("input your name: ")
3     if not name:
4         break # 当输入name后跳出循环
5     print("your name is", name)
6
7 row = 1
8 while row <= 9:
9     col = 1
10    while col <= row:
11        if col > 5: break
12        print("{}*{}={:<2}".format(row, col, row*col), end = " ")
13        col += 1
14    print("\n", end = "")
15    row += 1
```

```
1*1=1
2*1=2  2*2=4
3*1=3  3*2=6  3*3=9
4*1=4  4*2=8  4*3=12  4*4=16
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25
6*1=6  6*2=12  6*3=18  6*4=24  6*5=30
7*1=7  7*2=14  7*3=21  7*4=28  7*5=35
8*1=8  8*2=16  8*3=24  8*4=32  8*5=40
9*1=9  9*2=18  9*3=27  9*4=36  9*5=45
```

仅结束内层循环

```
input your name: John
your name is John
input your name:
```

else子句

else子句在程序循环正常结束后运行

else子句

```
1 number = int(input("number is: "))
2 x, i = 0, 1
3 while i <= number:
4     x += i
5     i += 1
6     if x > 500:
7         print("break while循环")
8         break
9 else:
10    print(x)
11    print("while循环结束!")
```

```
number is: 50
break while循环
```

else子句后的语句仅在没有调用break时才执行

for循环语句

□ for循环：用于**可迭代对象**（列表、元组、字典、字符串等），可以为每个元素执行代码块，不需要设置索引变量

for语句

```
1 numbers = [0, 1, 2, 3, 4, 5, 6]
2 for number in numbers:
3     print(number, end = " ")
4 print("\n", end = "")
5
6 # range() 创建范围的内置函数
7 for number in range(2, 10):
8     print(number, end = " ")
9 print("\n", end = "")
10
11 for number in range(10):
12     print(number, end = " ")
13 print("\n", end = "")
```

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

range函数默认从0开始，
range(0,10,2)第三个参数定义步长

break、continue和else适用于for循环

break、continue和else

```
1 for i in range(1,10):
2     for j in range(1,10):
3         if j <= i:
4             print("{}*{}={:<2}".format(i,j,i*j), end = " ")
5         print("\n", end = "")
6     else:
7         print("for循环结束")
```



循环正常结束后执行

```
1*1=1
2*1=2  2*2=4
3*1=3  3*2=6  3*3=9
4*1=4  4*2=8  4*3=12  4*4=16
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25
6*1=6  6*2=12  6*3=18  6*4=24  6*5=30  6*6=36
7*1=7  7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
8*1=8  8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64
9*1=9  9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
for循环结束
```

break、continue和else适用于for循环

```
1 for i in range(1,10):
2     for j in range(1,10):
3         if j > 5: break
4         if j <= i:
5             print("{}*{}={:<2}".format(i, j, i*j), end = " ")
6         print("\n", end = "")
7 else:
8     print("for循环结束")
```

仅结束内层循环

外层循环正常结束后执行

```
1*1=1
2*1=2  2*2=4
3*1=3  3*2=6  3*3=9
4*1=4  4*2=8  4*3=12  4*4=16
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25
6*1=6  6*2=12  6*3=18  6*4=24  6*5=30
7*1=7  7*2=14  7*3=21  7*4=28  7*5=35
8*1=8  8*2=16  8*3=24  8*4=32  8*5=40
9*1=9  9*2=18  9*3=27  9*4=36  9*5=45
for循环结束
```

```
1 for i in range(1,10):
2     for j in range(1,10):
3         if j <= i:
4             if j > 5: continue
5             print("{}*{}={:<2}".format(i, j, i*j), end = " ")
6         print("\n", end = "")
7 else:
8     print("for循环结束")
```

仅结束当前迭代步

```
1*1=1
2*1=2  2*2=4
3*1=3  3*2=6  3*3=9
4*1=4  4*2=8  4*3=12  4*4=16
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25
6*1=6  6*2=12  6*3=18  6*4=24  6*5=30
7*1=7  7*2=14  7*3=21  7*4=28  7*5=35
8*1=8  8*2=16  8*3=24  8*4=32  8*5=40
9*1=9  9*2=18  9*3=27  9*4=36  9*5=45
for循环结束
```

那段代码运行速度更快？

break、continue和else适用于for循环

打印出10以内的前三个偶数

```
1 i = 1
2 for number in range(10):
3     if i > 3:
4         break
5     elif number%2 == 0:
6         print(number, end = " ")
7         i += 1
```

0 2 4

可以在break和continue
语句前打印提示信息



打印出10以内除了2的偶数

```
1 for number in range(10):
2     if number%2 != 0 or number == 2:
3         continue
4     print(number, end = " ")
```

0 4 6 8

for循环示例

打印出2到20内的斐波那契数列 $F(1) = 1, F(2) = 1, F(n) = F(n-1) + F(n-2) \ (n \geq 2, n \in \mathbb{N}^*)$

```
1 list1 = [1, 1]
2 i = len(list1)
3 for number in range(2, 20):
4     if number == list1[i-1] + list1[i-2]:
5         list1.append(number)
6         i += 1
7 print(list1)
```

[1, 1, 2, 3, 5, 8, 13]

利用莱布尼茨公式计算 π

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \dots = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

```
1 i = 0
2 for n in range(10**6):
3     i += (-1)**n/(2*n+1)
4 print(4*i)
```

3.1415916535897743

```
1 i = 0
2 n = 0
3 while True:
4     i += (-1)**n/(2*n+1)
5     n += 1
6     if n > 10**6: break
7 print(4*i)
```

3.1415936535887745

用while实现

迭代字典

迭代字典

```
1 dt = {"a":1, "c":3, "b":2}
2 for key in dt:
3     print(key, "corresponds to", dt[key])
4 for key in dt.keys():
5     print(key, "corresponds to", dt[key])
6
7 keylist = list(dt.keys())
8 keylist.sort()
9 for key in keylist:
10     print(key, "corresponds to", dt[key])
```

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

默认是遍历字典的键，但不确定顺序

通过sort()方法排序后迭代

对字典进行排序

对字典进行排序

```
1 dt1 = {"a":1, "c":3, "b":2}
2 dt2 = {}
3 keylist = list(dt1.keys())
4 keylist.sort()
5 for key in keylist:
6     dt2[key] = dt1[key]
7 print(dt2)
8
9 dt3 = {"a":1, "c":3, "b":2, "d":1}
10 dt4 = {}
11 valueslist = list(dt3.values())
12 valueslist.sort()
13 for value in valueslist:
14     for key in dt3:
15         if dt3[key] == value:
16             dt4[key] = value
17             break
18     del dt3[key]
19 print(dt4)
```

按键排序

按值排序

```
{ 'a': 1, 'b': 2, 'c': 3}
{ 'a': 1, 'd': 1, 'b': 2, 'c': 3}
```

sorted函数-排序

sorted()相比于列表的**sort**方法可适用于任何序列或可迭代对象，而且不就地修改对象，而是返回排序后的版本

sorted函数-排序

```
1  # sort方法原地修改
2  list1 = [1, 3, 2, 5, 4]
3  Slist1 = list1.sort()
4  print(list1, Slist1)
5
6  #sorted函数返回修改的列表
7  list2 = [1, 3, 2, 5, 4]
8  Slist2 = sorted(list2)
9  print(list2, Slist2)
10
11 #sorted用于字典对象
12 dt1 = {"a":1, "c":3, "b":2, "d":1}
13 dt2 = sorted(dt1)
14 print(dt1, dt2)
```

sorted函数排序字典，返回排序后的键列表

```
[1, 2, 3, 4, 5] None
[1, 3, 2, 5, 4] [1, 2, 3, 4, 5]
{'a': 1, 'c': 3, 'b': 2, 'd': 1} ['a', 'b', 'c', 'd']
```

sorted函数-排序

`sorted(可迭代对象, key = , reverse =)`

可迭代对象：字符串、列表、元组、字典等（可以用for进行迭代）

key：带一个参数的函数，实参取自于可迭代对象中的元素

reverse：排列规则，默认为False（升序排列）

```
1 list3 = [-1, 3, -4, 5, 2]
2 #按list3中元素的绝对值降序排列
3 print(sorted(list3, key = lambda x:abs(x), reverse = True))
4
5 #按dt1中的键排序
6 print(sorted(dt1.items(),key = lambda x:x[0]))
7 #按dt1中的值排序
8 print(sorted(dt1.items(),key = lambda x:x[1]))
```

```
[5, -4, 3, 2, -1]
[('a', 1), ('b', 2), ('c', 3), ('d', 1)]
[('a', 1), ('d', 1), ('b', 2), ('c', 3)]
```


zip函数-并行迭代

zip()用于将可迭代对象中对应的元素打包成一个个的元组，然后返回由这些元组组成的可迭代对象

zip函数-并行迭代

```
1 list1 = [1, 2, 3, 4]
2 list2 = [5, 6, 7, 8]
3 print(list(zip(list1, list2)))
4
5 list3 = range(10)
6 print(list(zip(list1, list2, list3)))
7
8 list4 = []
9 for x, y in zip(list1, list2):
10     list4.append(x+y)
11 print(list4)
12
13 #创建字典
14 print(dict(zip(list1, list2)))
```

当长度不同时，zip在最短的序列用完后停止

可迭代对象

创建字典

```
[(1, 5), (2, 6), (3, 7), (4, 8)]
[(1, 5, 0), (2, 6, 1), (3, 7, 2), (4, 8, 3)]
[6, 8, 10, 12]
{1: 5, 2: 6, 3: 7, 4: 8}
```

enumerate函数

enumerate()用于将可迭代对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据索引位置和数据

enumerate函数

```
1 list1 = ["title1", "title2", "title3", "title4"]
2 print(enumerate(list1))
3 for x, y in enumerate(list1):
4     if "2" in y:
5         list1[x] = "titleB"
6 print(list1)
7
8 #for y in list1:
9 #    if "2" in y:
10 #        list1[list1.index(y)] = "titleB"
11 #print(list1)
12
13 print(sorted(enumerate(list1), key = lambda x:x[0]*(-1)))
```

list1[list1.index(y)]

```
<enumerate object at 0x0000000005C25280>
['title1', 'titleB', 'title3', 'title4']
[(3, 'title4'), (2, 'title3'), (1, 'titleB'), (0, 'title1')]
```

列表推导

一种从可迭代对象创建列表的方式，具有代码简洁的特点

列表推导

```
1 list1 = [i for i in range(10)]
2 print(list1)
3
4 list2 = [i**2 for i in range(10)]
5 print(list2)
6
7 list3 = [i**2 for i in range(10) if i%3 == 0]
8 print(list3)
9
10 list4 = [(i, j) for i in range(3) for j in range(3)]
11 print(list4)
12
13 list5 = [x*y for x in range(5) if x > 2 for y in range(4) if y < 3]
14 print(list5)
```

各语句之间是**嵌套关系**，
表达式后面的语句是最外层，依次往右进一层，左边第一条语句是最后一层

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

[0, 9, 36, 81]

[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]

[0, 3, 6, 0, 4, 8]

字典推导

将列表推导中的[]改为{}, 从可迭代对象创建字典

键的表达式: 值的表达式

字典推导

```
1 squares = {i: "{} suqared is {}".format(i, i**2) for i in range(10)}  
2 print(squares[5])  
3  
4 import random  
5 randomdict = {i: random.randint(10, 100) for i in range(1, 5)}  
6 print(randomdict)
```

5 suqared is 25

{1: 46, 2: 44, 3: 22, 4: 80}

文件操作

□ **os模块 (operate system) : 用于访问操作系统功能的模块，提供了多数操作系统的功能接口函数**

os模块

```
1 import os
2 #系统操作
3 print(os.name) #获取操作系统名
4 print(os.sep) #获取当前操作系统的路径分隔符
5 print(os.getenv("PATH")) #获取环境变量名的值
6 CurrentDir = os.getcwd() #获取当前工作目录
7 print(CurrentDir)
8
9 #目录操作
10 print(os.listdir(os.getcwd())) #返回指定目录下的所有文件和目录名
11 os.mkdir("test") #创建一个目录(指定目录名字符串)
12 os.rmdir("test") #删除一个空目录(指定目录名字符串)
13 os.makedirs(r"test\01") #创建一个多层递归目录(指定目录绝对或相对路径)
14 #os.removedirs(r"test\01") #删除一个多层递归的空目录(指定目录绝对或相对路径)
15 os.chdir(r"test") #改变当前工作目录到指定目录中
16 print(os.getcwd())
17 os.rename("01", "02") #重命名目录名或者文件名
18 os.chdir(CurrentDir)
19 list(os.walk(CurrentDir)) #遍历指定目录下的所有文件夹和文件
```

导入OS模块后自动识别操作系统

os.path()模块

□ os.path()模块：用于返回文件和路径（目录）的属性

os.path模块

```
1  #os.makedirs(r"test\01")
2  print(os.path.exists(r"test\01")) #判断文件或者目录是否存在
3  print(os.path.isfile(r"test\01")) #判断是否为文件
4  print(os.path.isdir(r"test\01")) #判断是否为目录
5  print(os.path.abspath(r"test\01")) #返回绝对路径
6  print(os.path.basename(r"test\01")) #返回文件或目录名
7  print(os.path.dirname(r"test\01")) #返回文件或目录所在的目录路径
8  print(os.path.split(r"test\01")) #拆分为dirname和basename
9  print(os.path.join("test", "01")) #将dirname和basename合成为路径
```

创建和删除文件

利用open()函数创建或打开文件

`file = open("文件名" , "参数")`
file为文件引用变量

- "r" - 读取 - **默认值**。打开文件进行读取，如果文件不存在则报错
- "a" - 追加 - 打开供追加的文件，如果不存在则创建该文件
- "w" - 写入 - 打开文件进行写入，如果文件不存在则创建该文件
- "x" - 创建 - 创建指定的文件，如果文件存在则返回错误
- "t" - 文本 - **默认值**。文本模式
- "b" - 二进制 - 二进制模式（例如图像）

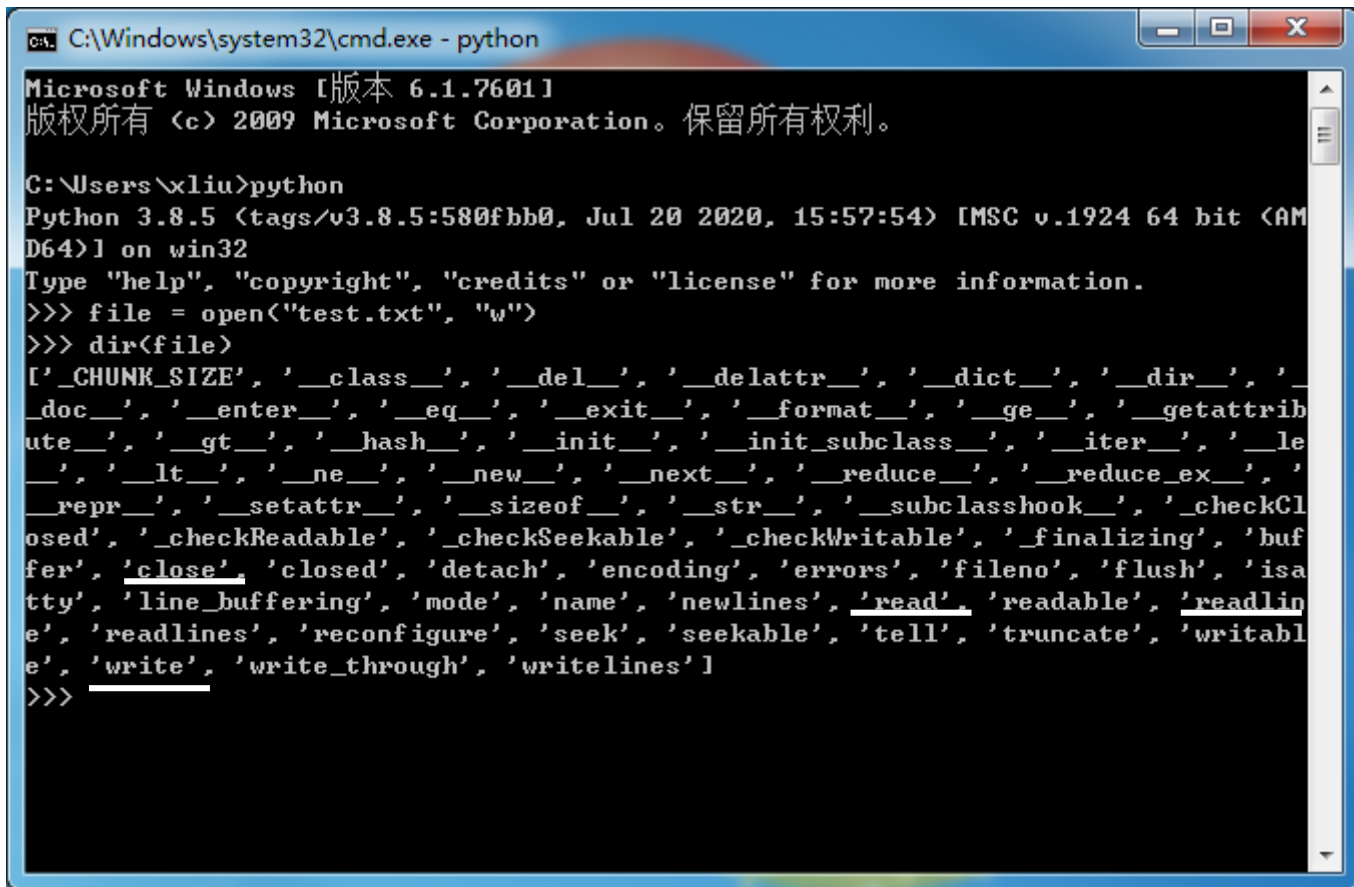
```
1 # os.mknod("test.txt") # linux环境可用
2 file = open("test.txt", "w")
3 file.close() #关闭文件
4 os.remove("test.txt") #删除文件
```

文件操作示例

在当前工作目录下创建含有"a01", "b02", "c03", "d04", "e05", "f06"子目录的"calculation目录", 并在每个子目录下创建名字为POSCAR的文件

```
1 import os
2 CurrentDir1 = os.getcwd()
3 # 创建calculation目录
4 os.mkdir("calculation")
5 os.chdir("calculation")
6 CurrentDir2 = os.getcwd()
7 # 创建子目录
8 stringname = [chr(i) for i in range(97, 103)]
9 numbername = [i for i in range(1, 7)]
10 for i in range(6):
11     os.mkdir(stringname[i]+"0"+str(numbername[i]))
12     os.chdir(stringname[i]+"0"+str(numbername[i]))
13     file = open("POSCAR", "w")
14     file.close()
15     os.chdir(CurrentDir2)
16 os.chdir(CurrentDir1)
```


修改文件



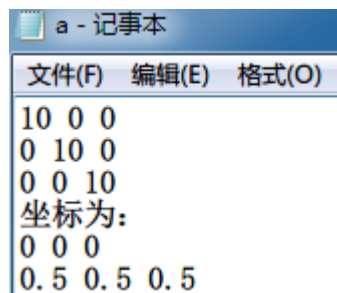
```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\xliu>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> file = open("test.txt", "w")
>>> dir(file)
['_CHUNK_SIZE', '__class__', '__del__', '__delattr__', '__dict__', '__dir__', '__doc__', '__enter__', '__eq__', '__exit__', '__format__', '__ge__', '__getattr__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__lt__', '__ne__', '__new__', '__next__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_checkClosed', '_checkReadable', '_checkSeekable', '_checkWritable', '_finalizing', 'buffer', 'close', 'closed', 'detach', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'line_buffering', 'mode', 'name', 'newlines', 'read', 'readable', 'readline', 'readlines', 'reconfigure', 'seek', 'seekable', 'tell', 'truncate', 'writable', 'write', 'write_through', 'writelines']
>>>
```

file.close()： 关闭文件； **file.readline()：** 读一行； **file.write(“文本”)：** 将文本写入文件，默认不换行

文件修改示例

```
1 # 创建文件a.txt
2 file_a = open("a.txt", "w")
3
4
5 # 在文件a.txt中写入文本
6 file_a.write("10 0 0")
7 file_a.write("\n")
8 file_a.write("0 10 0")
9 file_a.write("\n")
10 print("0 0 10", file = file_a)
11 print("坐标为: ", file = file_a)
12 print("0 0 0", file = file_a)
13 print("0.5 0.5 0.5", file = file_a)
14
15 # 关闭文件a.txt
16 file_a.close()
```



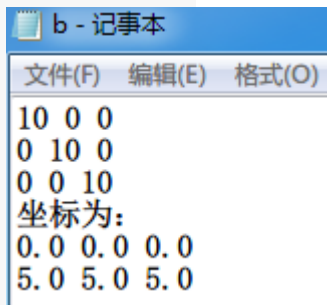
a - 记事本

文件(F) 编辑(E) 格式(O)

10 0 0
0 10 0
0 0 10
坐标为:
0 0 0
0.5 0.5 0.5

文件修改示例

```
1  # 创建文件b.txt
2  file_b = open("b.txt", "w")
3
4  # 利用文件a.txt中的内容编写b.txt
5  file_a = open("a.txt")
6  Cellvector = []
7  for i in range(3):
8      stringline = file_a.readline().strip()
9      print(stringline, file = file_b)
10     Cellvector.append(stringline.strip().split(" "))
11     #print(Cellvector)
12     stringline = file_a.readline().strip()
13     print(stringline, file = file_b)
14     for i in range(2):
15         stringline = file_a.readline().strip().split(" ")
16         Convertline = [0]*3
17         for j in range(3):
18             for k in range(3):
19                 Convertline[j] += float(Cellvector[j][k]) * float(stringline[k])
20             Convertline[j] = str(Convertline[j])
21         print(" ".join(Convertline), file = file_b)
22
23 # 关闭文件a.txt和b.txt
24 file_a.close()
25 file_b.close()
```



b - 记事本

文件(F) 编辑(E) 格式(O)

10 0 0
0 10 0
0 0 10
坐标为:
0.0 0.0 0.0
5.0 5.0 5.0

课后作业

课后作业

判断某一年是否会闰年（符合条件之一为闰年：1能被4整除而不能被100整除，2能被400整除）

In []:

1

求1000以内最大的20个质数的和

In []:

1

求自然对数e的近似值

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

In []:

1

保留字

保留字是 Python 中一些已经被赋予特定意义的单词，**不能用保留字作为标识符**给变量、函数、类、模板以及其他对象命名。

and	as	assert	break	class	continue
def	del	elif	else	except	finally
for	from	False	global	if	import
in	is	lambda	nonlocal	not	None
or	pass	raise	return	try	True
while	with	yield			

区分大小写

小结

- 赋值类语句：print、import、赋值、表达式等
 - 条件语句：bool表达式、多分支、嵌套
 - 循环语句：迭代、嵌套、continue、break
 - 文件操作：os模块和常用函数，文件对象的方法
- 非赋值类语句
(流控语句)

下一节：函数