Python 使用随笔

suyi

2021 年 11 月 10 日

目录

1 Python 语言基础

1.1 type()

$type(x)$ 能查询变量 x 的类型。

1.2 id()

$id(x)$ 能返回对象 x 的内存地址。

Python 解释器会为每个出现的对象分配内存单元，即使它们的值相等，也会这样。

例如，执行 a=2.0，b=2.0 这两个语句时，会先后为 2.0 这个 float 类型对象分配内存单元，然后将 a 与 b 分别指向这两个对象。所以 a 与 b 指向的不是同一对象。

为了提高内存利用效率，对于一些简单的对象，如一些数值较小（-256 256）的整型（int）对象，Python 采取重用对象内存的办法。

例：执行 a=2，b=2 时，由于 2 作为简单的 int 类型且数值小,Python 不会两次为其分配内存单元，而是只分配一次，然后将 a 与 b 同时指向已分配的对象。

1.3 x.real and x.imag

$x.real$ 和 $x.imag$ 能分别获取复数 $x = a + bj(J)$ 的实部 a 和虚部 b。

1.4 eval()

与字符串有关的一个重要函数是 eval，其调用格式为：eval(字符串)

$eval()$ 函数的作用就是删去字符串最外面那层引号。

```
1  >>> c='23+45'
2  >>> eval(c)
3  68
```

1.5 len()

$len()$ 函数返回字符串的长度，即字符串中所包含的字符个数，其调用格式为：len(字符串)

1.6 list,tup,dict and set

list 是列表，$list = ['abc', 1, 1.0, 1+1j]$，元素可以改变，可以用位置编号（索引）来访问列表元素。

tuple 是元组，$tup = ('abc', 1, 1.0, 1+1j)$，元素不可以改变，可以用位置编号（索引）来访问元组元素。

dictionary 是字典，$dict = \{'name' :' brenden','code' : 410012,'dept' :' sales'\}$，字典是一种映射类型（mapping type），它是一个无序的“关键字:值”对集合。关键字必须使用不可变类型，也就是说列表和包含可变类型的元组不能做索引关键字。在同一个字典中，关键字还必须互不相同。只能通过关键字来访问对应的值。

```
1  >>> dict={'name':'brenden','code':410012,'dept':'
       sales'}
2  >>> print(dict) #输出完整的字典
3  {'name': 'brenden', 'dept': 'sales', 'code':
       410012}
4  >>> print(dict['code']) #输出关键字为“code”的值
       410012
5  >>> dict['payment']=4500 #在字典中添加一个“关键字
       :值”对
6  >>> print(dict) #输出完整的字典
7  {'name': 'brenden', 'dept': 'sales', 'code':
       410012, 'payment':4500}
```

set 是集合，$set = \{'Tom','Jim','Mary','Jack','Rose'\}$，集（set）是一个无序且包含不重复元素的数据类型。基本功能是进行成员关系测试和消除重复元素。可以使用大括号或者 set() 函数创建集合类型。注意：创建一个空集合必须用 set() 而不是，因为是用来创建一个空字典。

1.7 常用系统函数

数学库模块 (*math*) 提供了很多数学运算函数；

复数模块 (*cmath*) 提供了用于复数运算的函数；

随机数模块 (*random*) 提供了用来生成随机数的函数；

时间 (*time*) 和日历 (*calendar*) 模块提供了能处理日期和时间的函数

在调用系统函数之前，先要使用 import 语句导入相应的模块，格式如下：

import 模块名

该语句将模块中定义的函数代码复制到自己的程序中，然后就可以访问模块中的任何函数，其方法是在函数名前面加上“模块名.”。

例：调用数学模块 math 中的平方根函数 sqrt()，语句如下：

```
>>>import math #导入math模块
>>>math.sqrt(2) #调用sqrt()函数
1.4142135623730951
```

另一种导入模块的方法，格式如下：

from 模块名 *import* 函数名

该语句从指定模块中导入指定函数的定义，这样调用模块中的函数时，不需要在前面加上“模块名.”。

```
>>> from math import sqrt
>>> sqrt(2)
1.4142135623730951
```

如果希望导入模块中的所有函数定义，则函数名用“*”。格式如下：

from 模块名 *import* *

这样调用指定模块中的任意函数时，都不需要在前面加“模块名.”。

使用这种方法固然省事方便，但当多个模块有同名的函数时，会引起混乱，使用时要注意。

1.7.1 math 模块函数

math 模块主要处理数学相关的运算。

(1) 数学常量

e：返回自然常数 e（自然对数的底）。

pi：返回圆周率 π 的值。

(2) 绝对值和平方根函数

fabs(x)：返回 x 的绝对值（返回值为浮点数）。

例：fabs(-10) 返回 10.0。

sqrt(x)：返回 x 的平方根（x>0）。

例：sqrt(4) 返回 2.0。

（3）幂函数和对数函数

pow(x,y)：返回 x 的 y 次幂。

例：pow(2,3) 返回 8.0

exp(x)：返回自然常数 e 的 x 次幂。

例：exp(1) 返回 2.718281828459045。

log(x[,base])：返回 x 的自然对数。

例：log(e) 返回 1.0。

可以使用 base 参数来改变对数的底。

例：log(100,10) 返回 2.0。

log10(x)：返回 x 的常用对数。

例：log10(100) 返回 2.0。

（4）取整和求余函数

ceil(x)：对 x 向上取整。

例：ceil(4.1) 返回 5。

floor(x)：对 x 向下取整。

例：floor(4.9) 返回 4。

fmod(x,y)：返回求 x/y 的余数（返回值为浮点数）。

例：fmod(7,4) 返回 3.0。

（5）弧度角度转换函数

degrees(x)：将弧度转换为角度。

例：degrees(pi) 返回 180.0。

radians(x)：将角度转换为弧度。

例：radians(90) 返回 1.5707963267948966。

（6）三角函数和反三角函数

sin(x)：返回 x 的正弦值（x 为弧度）。

例：sin(pi/2) 返回 1.0。

cos(x)：返回 x 的余弦值（x 为弧度）。

例：cos(pi) 返回-1.0。

tan(x)：返回 x 的正切值（x 为弧度）。

例：tan(pi/4) 返回 0.9999999999999999（数学上为 1）。

asin(x)：返回 x 的反正弦值（返回值为弧度）。

例：degrees(asin(1)) 返回 90.0。

acos(x)：返回 x 的反余弦值（返回值为弧度）。

例：degrees(acos(-1)) 返回 180.0。

atan(x)：返回 x 的反正切值（返回值为弧度）。

例：degrees(atan(1)) 返回 45.0

1.7.2 cmath 模块函数

cmath 模块函数与 math 模块函数基本一致。包括圆周率 pi、自然常数 e 等常量。复数的幂指数、对数函数、平方根函数、三角函数等。

cmath 模块函数名和 math 模块函数名相同。只是 math 模块对实数运算，cmath 模块对复数运算。

例：

```
>>> import cmath
>>> cmath.pi
3.141592653589793
>>> cmath.sqrt(-1)
1j
>>> cmath.sin(1)
(0.8414709848078965+0j)
>>> cmath.log10(100)
(2+0j)
>>> cmath.exp(100+10j)
(-2.255522560520288e+43-1.4623924736915717e+43j)
```

cmath 模块包括复数运算特有的函数。复数 x=a+bi，phase(x) 函数返回复数 x 的幅角，即 atan(b/a)。

例：

```
>>> from cmath import *
>>> phase(1+1j)
0.7853981633974483
>>> phase(1+2j)
1.1071487177940904
```

cmath 模块的 polar() 函数和 rect() 函数可以对复数进行极坐标表示和笛卡儿表示方法的转换。

polar(x) 函数将复数的笛卡儿坐标表示转换为极坐标表示，输出为一个二元组 (r,p)，复数的模 r=abs(x)，幅角 p=phase(x)。

rect(r, p) 函数将复数的极坐标表示转换为笛卡儿坐标表示，输出为 r*cos(p)+r*sin(p)*1j。

例：

```
>>> c=3+4j
>>> r,p=polar(c)
>>> print(r,p)
5.0 0.9272952180016122
>>> rect(r,p)
(3.0000000000000004+3.9999999999999996j)
```

1.7.3 random 模块函数

（1）随机数种子

使用 seed(x) 函数可以设置随机数生成器的种子，通常在调用其他随机模块函数之前调用此函数。

对于相同的种子，每次调用随机函数生成的随机数是相同的。

默认将系统时间作为种子值，使得每次产生的随机数都不一样。

（2）随机挑选和排序

choice(seq)：从序列的元素中随机挑选一个元素。

sample(seq,k)：从序列中随机挑选 k 个元素。

shuffle(seq)：将序列的所有元素随机排序。例：choice([0,1,2,3,4,5,6,7,8,9])，从 0 到 9 中随机挑选一个整数。

```
>>> from random import *
>>> choice([0,1,2,3,4,5,6,7,8,9])
6
>>> choice([0,1,2,3,4,5,6,7,8,9])
0
```

（3）生成随机数

下面生成的随机数符合均匀分布（uniform distribution），即范围内每个数字出现的概率相等。

random()：随机生成一个 [0,1) 范围内的实数。

uniform(a,b)：随机生成一个 [a,b] 范围内的实数。

randrange(a,b,c)：随机生成一个 [a,b) 范围内以 c 递增的整数，省略 c 时以 1 递增，省略 a 时初值为 0。

randint(a,b)：随机生成一个 [a,b] 范围内的整数，相当于 randrange(a,b+1)

1.7.4 time 模块函数

time()：返回当前时间的时间戳 [chuō]。时间戳是从 Epoch（1970 年 1 月 1 日 00:00:00 UTC）开始所经过的秒数，不考虑闰秒。要注意的是，在中国时区，是 UTC+8。

localtime([secs])：接收从 Epoch 开始的秒数，并返回一个时间元组。时间元组包含 9 个元素，相当于 *struct_time* 结构。省略秒数时，返回当前时间戳对应的时间元组。

```
1  >>> from time import *
2  >>> localtime()
3  time.struct_time(tm_year=2019, tm_mon=3,tm_mday
       =10, tm_hour=23, tm_min=33, tm_sec=43,tm_wday
       =6, tm_yday=69, tm_isdst=0)
```

其中 *tm_wday* 表示一周的第几日，0 到 6，0 是周一；*tm_yday* 表示一年的第几日，1 到 365；*tm_isdst* 表示夏令时，-1, 0, 1, -1 是决定是否为夏令时的旗帜。

asctime([tupletime])：接收一个时间元组，并返回一个日期时间字符串。时间元组省略时，返回当前系统日期和时间。

例：

```
1  >>> asctime()
2   'Sun Mar 10 23:37:06 2019'
3  >>> asctime(localtime(time()))
4   'Sun Mar 10 23:37:43 2019'
```

ctime([secs])：类似于 asctime(localtime([secs]))，不带参数时与 asctime() 功能相同。

例：

```
>>> ctime(time())
'Sun Mar 10 23:39:22 2019'
```

strftime(日期格式)：按指定的日期格式返回当前日期。

例：

```
>>> strftime("%Y-%m-%d %H:%M:%S")
'2019-03-10 23:41:13'
```

Python 时间日期格式化符号有：

```
%y：表示两位数的年份（00~99）；
%Y：表示4位数的年份（000~9999）；
%m：表示月份（01~12）；
%d：表示月中的一天（0-31）；
%H：表示24小时制小时数（0-23）；
%I：表示12小时制小时数（01~12）；
%M：表示分钟数（00~59）；
%S：表示秒（00-59）。
```

1.7.5 calendar 模块函数

日历（calendar）模块提供与日历相关的功能。在默认情况下，日历把星期一作为一周的第一天，星期日为最后一天。要改变这种设置，可以调用 setfirstweekday() 函数。

setfirstweekday(weekday)：设置每个星期的开始工作日代码。星期代码是 0-6，代表星期一-星期日。

firstweekday()：返回当前设置的每个星期开始工作日。默认是 0，即星期一。

isleap(year)：如果指定年份是闰年返回 True，否则为 False。

leapdays(y1,y2)：返回在 [y1,y2) 范围内的闰年数。

calendar(year)：返回指定年份的日历。

例：

```
>>> from calendar import *
>>> c=calendar(2019)
```

```
3    >>> print(c)
```

将输出 2019 年的日历（略）。

month(year,month)：返回指定年份和月份的日历。

例：

```
1    >>> c=month(2019,3)
2    >>> print(c)
```

将输出 2019 年 3 月的日历（略）。

monthcalendar(year,month)：返回整数列表，每个子列表表示一个星期(从星期一到星期日)。

例：

```
1    >>> c=monthcalendar(2019,3)
2    >>> print(c)
3    [[0, 0, 0, 0, 1, 2, 3], [4, 5, 6, 7, 8, 9, 10],
        [11, 12, 13,14, 15, 16, 17], [18, 19, 20, 21,
        22, 23, 24], [25, 26, 27,28, 29, 30, 31]]
```

monthrange(year,month)：返回两个整数，第 1 个数代表指定年和月的第一天是星期几，第二个数代表所指定月份的天数。

例:

```
1    >>>monthrange(2016,1)
2    (4,31)
```

表明 2016 年 1 月的第 1 天是星期五，该月有 31 天。

weekday(year,month,day)：返回给定日期的星期代码。

1.8 built-in function

Python 内置函数包含在模块 builtins 中，该模块在启动 Python 解释器时自动装入内存，而其他的模块函数都要等使用 import 语句导入时才会装入内存。内置函数随着 Python 解释器的运行而创建，在程序中 可以随时调用这些函数。

例：print()；type()；id()

1.8.1 range() 函数

迭代器（iterator）和生成器（generator）.

range() 函数返回的是可迭代对象，迭代时产生指定范围的数字序列。迭代器可以看成一个特殊的对象，每次调用该对象时会返回自身的下一个元素。生成器是能够返回一个迭代器的函数。迭代器不要求事先准备好整个迭代过程中所有的元素。迭代器仅仅在迭代到某个元素时才计算该元素，而在这之前或之后，元素可以不存在。这个特点使得迭代器能节省内存空间，特别适合用于遍历一些很大的或无限的集合。

range() 函数的调用格式：

$$range([start,]end[,step])$$

range() 函数产生的数字序列从 start 开始，默认是从 0 开始；序列到 end 结束，但不包含 end；如果指定了可选的步长 step，则序列按步长增加，默认为 1。

使用内置函数 “iter(可迭代对象)” 可以获取可迭代对象的迭代器。

使用内置函数 “next(迭代器对象)” 可以得到迭代器对象的下一个元素，如果迭代器对象没有新的元素，则抛出 StopIteration 异常。

例：

```
>>> range(2)
range(0, 2)
>>> s=range(2) #产生可迭代对象
>>> t=iter(s) #产生迭代器对象
>>> t
<range_iterator object at 0x02178968>
>>> next(t) #产生迭代器对象的下一个元素
0
>>> next(t)
1
>>> next(t)
#迭代器对象没有新的元素时导致StopIteration异常
```

可以利用 range() 函数和 list() 函数产生一个列表。

例：

```
>>> list(range(2,15,3))
[2, 5, 8, 11, 14]
>>> list(range(5))
[0, 1, 2, 3, 4]
```

可以利用 range() 函数和 tuple() 函数产生一个元组。

例:

```
>>> tuple(range(2,15,3))
(2, 5, 8, 11, 14)
>>> tuple(range(5))
(0, 1, 2, 3, 4)
```

1.8.2 数值运算函数

Python 有些内置函数用于数值运算。

abs(x): 返回 x 的绝对值，结果保持 x 的类型。x 为复数时返回复数的模。

pow(x,y[,z]): 其中的 x、y 是必选参数，z 是可选参数。省略 z 时，返回 x 的 y 次幂，结果保持 x 或 y 的类型。如果使用了参数 z，其结果是 x 的 y 次方再对 z 求余数。

例:

```
>>> pow(2,3)
8
>>> pow(2,3,3)
2
```

round(x[,n]): 用于对浮点数进行四舍五入运算，返回值为浮点数。它有一个可选的小数位数参数。如果不提供小数位参数，它返回与第一个参数最接近的整数，~~但仍然是浮点类型~~。第二个参数表示将结果精确到小数点后指定位数。

例:

```
>>> round(3.46)
3
```

```
>>> round(3.14159,3)
 3.142
>>> x=round(3.46)
>>> type(x)
<class 'int'>
```

divmod(x,y)：把除法和取余运算结合起来，返回一个包含商和余数的元组。对整数来说，它的返回值就是 x/y 取商和 x/y 取余数的结果。

例:

```
>>> divmod(7,4)
 (1,3)
```

1.8.3 Python 系统的帮助信息

查看 Python 帮助信息可以使用内置函数 dir() 和 help()。

dir() 函数的调用方法很简单，只需把想要查询的对象加到括号中就可以了，它返回一个列表，其中包含要查询对象的所有属性和方法。查看某个对象的帮助信息可以用 help() 函数。

例:

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__
    ', '__spec__', 'acos', 'acosh', 'asin', 'asinh'
    , 'atan', 'atan2', 'atanh', 'ceil', 'comb', '
    copysign', 'cos', 'cosh', 'degrees', 'dist', 'e
    ', 'erf', 'erfc', 'exp', 'expm1', 'fabs', '
    factorial', 'floor', 'fmod', 'frexp', 'fsum', '
    gamma', 'gcd', 'hypot', 'inf', 'isclose', '
    isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', '
    ldexp', 'lgamma', 'log', 'log10', 'log1p', '
    log2', 'modf', 'nan', 'nextafter', 'perm', 'pi'
    , 'pow', 'prod', 'radians', 'remainder', 'sin',
     'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc',
     'ulp']
```

```
>>> help(math.cos)
Help on built-in function cos in module math:
cos(x, /)
    Return the cosine of x (measured in radians).
```

在 Python 解释器提示符下输入“help()”命令，可以进入联机帮助环境。“help>”是帮助系统提示符，在该提示符下输入想了解的主题，Python 就会给出有关主题的信息。

1.9 基本运算

1.9.1 算术运算

算术运算符：+(加)、-(减)、*(乘)、/(除)、//(整除)、%(求余)、**(乘方)

“/”：浮点数除法，其运算结果是一个浮点数，即使被除数和除数都是整型，也返回一个浮点数。

“//”：整数除，除法运算后返回商的整数部分。如果结果为正数，可将其视为朝向小数位取整（注意：不是四舍五入）。当整数除以负数，“//”运算符将结果朝着最近的整数“向上”四舍五入。“//”运算符并非总是返回整数结果。如果分子或者分母是浮点型，它返回的值将会是浮点类型。

例：

```
>>> 5/3
1.6666666666666667
>>> 5/3.0
1.6666666666666667
>>> 5//3
1
>>> 5//3.0
1.0
>>> -5//3
-2
>>> 5%3
2
>>> 5%3.0
```

```
14      2.0
```

“**”：乘方运算符。实现乘方运算，其优先级高于乘除运算。

例：

```
1       >>>2**10
2       1024
3       >>>4*5/2**3
4       2.5
```

1.9.2 浮点数的计算误差

Python 中能表示浮点数的有效数字是有限的，而在实际应用中数据的有效位数并无限制，这种矛盾，势必带来计算时的微小误差。

例:

```
1       >>>x=2.2
2       >>>x-1.2
3       1.0000000000000002
```

尽管在很多情况下这种误差不至于影响数值计算结果的实际应用，但对浮点数进行“等于”判断时就会得到截然不同的结果。

例:

```
1       >>>x=2.2
2       >>>x-1.2==1
3       False
```

结论：对浮点数判断是否相等要慎用“==”运算符。

解决方案：

判断它们是否“约等于”，只要在允许的误差范围内，这种判断仍是有意义的。

所谓“约等于”是指两个浮点数非常接近，即它们的差足够小（具体误差可以根据实际情况进行调整）。

例：

```
1       >>>x=2.2
2       >>>abs((x-1.2)-1)<1e-6
```

```
3       True
```

1.9.3 数据类型的转换

当算术表达式中需要违反自动类型转换规则，或者说自动类型转换规则达不到目的时，可以使用类型转换函数，将数据从一种类型强制转换到另一个类型，以满足运算要求。

常用类型转换函数：

int(x)：将 x 转换为整型。int() 取整不是四舍五入，是“向零取整”，是真正的“截尾取整”。与 ceil() 函数向上取整、floor() 函数向下取整不同。

float(x)：将 x 为转换浮点型。

complex(x)：将 x 转换为复数，其中复数的实部为 x 和虚部为 0。

complex(x,y)：将 x 和 y 转换成一个复数，其中实部为 x 和虚部 y。

提示：使用取整、求余等运算可以进行整除的判断，可以分离整数的各位数字。

例：

```
1       >>> float(2+3.0)
2       5.0
3       >>> int(2+3.0)
4       5
5       >>> int(-2.546)
6       -2
7       >>> complex(3)
8       (3+0j)
9       >>> complex(3.5,5.0)
10      (3.5+5j)
11      >>> 5-int(5/3)*3
12      2
13      >>> m=1234
14      >>> (m//10)%10
15      3
```

可以用 int() 函数将整数字符串转换成对应的整数；

可以用 float() 函数将浮点数字符串转换成对应的浮点数；

可以用 complex() 函数将复数字符串转换成对应的复数;
可以用 str() 函数将数值型数据转换为字符串。

1.10 位运算

位运算就是直接对整数按二进制位进行操作，其运算符主要有：&，|，~，^，>>和<<。

在计算机内部常用补码来表示数：运算量是补码，运算结果也是补码，人机界面使用原码。

1.10.1 按位与运算

按位与运算符是 &。

例：

```
>>> -5 & 3
3
正数
0 0 0 0 0 1 0 0 = 4
0 0 0 0 0 1 0 1 = 5
————————————
0 0 0 0 0 1 0 0 = 4 & 5 = 4

正数负数
1 1 1 1 1 1 0 0 = -4补
0 0 0 0 0 1 0 1 = 5
————————————
0 0 0 0 0 1 0 0 = -4 & 5 = 4

负数
1 1 1 1 1 1 0 0 = -4补
1 1 1 1 1 0 1 1 = -5补
————————————
1 1 1 1 1 0 0 0 = [-4 & -5]补
-4 & -5=[[-4 & -5]补]补=1 0001000 = -8
```

1.10.2 按位或运算

按位或运算符是 |。

例:

```
>>> -5 | 3
-5
正数
0 0 0 0 0 1 0 0 = 4
0 0 0 0 0 1 0 1 = 5
————————————
0 0 0 0 0 1 0 1 = 4 | 5 = 5

正数负数
1 1 1 1 1 1 0 0 = -4补
0 0 0 0 0 1 0 1 = 5补
————————————
1 1 1 1 1 1 0 1 = [-4 | -5]补;
-4 | -5=[[-4 | -5]补]补 =1 0000011 = -3

负数
1 1 1 1 1 1 0 0 = -4补
1 1 1 1 1 0 1 1 = -5补
————————————
1 1 1 1 1 1 1 1 = [-4 | -5]补;
-4 | -5=[[-4 | -5]补]补=1 0000001 = -1
```

1.10.3 按位异或运算

按位异或运算符是 ^。

0 ^ 0=0, 0 ^ 1=1, 1 ^ 0=1, 1 ^ 1=0

例:

```
>>> -5 ^ 3
-8
```

1.10.4 按位取反运算

按位取反运算符是 ~。

~ 0=1，~ 1=0

例：

```
>>> ~ 7
-8
```

1.10.5 左移运算

左移运算符是 < <。(左移 1 位等效为乘 2)

例：

```
>>> 3 << 2
12
```

将 3 左移 2 位，右边（最低位）补 0。

1.10.6 右移运算

右移运算符是 > >。(右移 1 位等效为除 2 取商) 最高位用符号位填充：移动对象为正数时，高位补 0。移动对象为负数时，高位补 1。

例:

```
>>> -3 >> 2
-1
```

例：

```
x=2**10
y=pow(2,10)
z=2<<9
a=3/5
b=3//5
c=3%5
print(x,y,z)
print(a,b,c)
程序输出结果为：
```

```
10      1024  1024  1024
11      0.6  0  3
```

2 顺序结构

2.1 lambda()

Python 支持定义单行函数，称为 lambda 函数，可以用在任何需要函数的地方。lambda 函数是一个可以接收任意多个参数并且返回单个表达式值的函数。

例：已知 f(x,y)=x2+y2，输入 x,y 的值，求出对应的函数值。

```
f=lambda x,y:x**2+y**2
print("f(3,4)=",f(3,4))
程序运行结果:
f(3,4)=25
```

2.2 Python 语句行与注释

在 Python 中，语句行从解释器提示符后的第一列开始，前面不能有任何空格，否则会产生语法错误。

每个语句行以回车符结束。

可以在同一行中使用多条语句，语句之间使用分号分隔。

如果语句行太长，可以使用反斜杠将一行语句分为多行显示。

例：

```
>>>t=1+2+3+4\
+5+6
>>>t
21
```

如果在语句中包含小括号、中括号或大括号，则不需要使用多行续行符。

例：

```
>>>def f(
):return 120
#提示：回车，空一行表示结束输入。
>>>f()
120
```

注释对程序的执行没有任何影响，目的是对程序做解释说明，以增强程序的可读性。在程序调试阶段，有时需要暂时不执行某些语句，这时可以给这些语句加注释符号，相当于对这些语句做逻辑删除，需要执行时，再去掉注释符号即可。

程序中的单行注释采用 # 开头。注释可以从任意位置开始，可以在语句行末尾，也可以独立成行。对于多行注释，一般推荐使用多个 # 开头的多行注释，也可采用三引号（实际上是用三引号括起来的一个多行字符串，起到注释的作用）。

注意：注释行是不能使用反斜杠续行的

2.3 赋值语句

赋值语句是没有返回值的。

2.3.1 复合赋值语句

在程序设计中，经常遇到在变量已有值的基础上做某种修正的运算。例：x=x+5.0。

这类运算的特点：变量既是运算对象，又是赋值对象。

为避免对同一存储对象的地址重复计算，Python 提供了 12 种复合赋值运算符：

+=、-=、*=、/=、//=、%=、**=、< <=、> >=、&=、|=、^=

例:

```
1    x+=5.0 #等价于 x=x+5.0
2    x*=u+v #等价于 x=x*(u+v)
```

2.3.2 链式赋值

链式赋值语句的一般形式：变量 1= 变量 2=……= 变量 n= 表达式，等价于：

变量 n= 表达式

……

变量 1= 变量 2

链式赋值用于为多个变量赋同一个值。

例:

```
>>>a=b=10
>>>id(a)
1471472544
>>>id(b)
1471472544
>>>a
10
>>>b
10
#创建一个值为10的整型对象，将该对象的同一个引用赋值给a和b，即a和b均指向数据对象10
```

2.3.3 同步赋值

同步赋值的一般形式为：变量 1, 变量 2,……, 变量 n= 表达式 1, 表达式 2,……, 表达式 n。

赋值号左边变量的个数与右边表达式的个数要一致。首先计算右边 n 个表达式的值，然后同时将表达式的值赋值给左边的 n 个变量。这并非等同于简单地将多个单一赋值语句进行组合。

例:

```
>>>a,b,c=10,20,30
>>>a
10
>>>b
20
>>>c
30
```

例:

```
>>>x,x=11,22
>>>x
 22
```

将右边表达式的各值从左到右依次赋值给左边各变量（先做 x=11，后做 x=22）。

同步赋值有先后顺序，但不是传统意义上的单一赋值语句的先后执行。

例：

```
>>>x=34
>>>x,y=12,x
>>>x
 12
>>>y
 34
```

先计算右边表达式的各值，再将它们从左到右依次赋值给左边各变量。

要交换 a，b 两个变量的值，一般需要一个中间变量：

```
>>>t=a
>>>a=b
>>>b=t
```

如果采用同步赋值，一个语句即可完成：

例：

```
>>>a,b=10,20
>>>a,b=b,a
>>>a
 20
>>>b
 10
```

不需中间变量即可交换两个变量的值, 优雅、简洁。

2.4 标准输入

Python 用内置函数 input() 实现标准输入，其调用格式：input([提示字符串])。如果有 “提示字符串”，则原样显示，提示用户输入数据。input() 函

数从标准输入设备（键盘）读取一行数据，并返回一个字符串（去掉结尾的换行符）。

input() 函数把输入的内容当成字符串，如果要输入数值数据，可以使用类型转换函数将字符串转换为数值。

例：

```
>>>x=input()
 12
>>>x
 '12'
>>>x=int(input())
 12
>>>x
 12
```

使用 input() 函数可以给多个变量赋值。

例:

```
>>>x,y=eval(input())
 3,4
>>>x
 3
>>>x+y
 7
#从键盘输入“3,4”，input()函数返回字符串“3,4”。
#Eval()函数将去掉字符串最外侧的引号。
#经过eval()函数处理，字符串“3,4”变成由3和4组成的元组。
```

2.5 标准输出

有两种输出方式：使用表达式和使用 print() 函数。

直接使用表达式可以输出该表达式的值。使用表达式语句输出，输出形式简单，一般用于检查变量的值。

常用输出方法是用 print() 函数，其调用格式为：

print([输出项 1, 输出项 2,……, 输出项 n][,sep= 分隔符][,end= 结束符])

其中，输出项之间以逗号分隔，没有输出项时输出一个空行。

sep 表示输出时各输出项之间的分隔符（默认以空格分隔）。（提示：separator，分隔符）end 表示结束符（默认以回车换行结束）。

print() 函数从左至右求每一个输出项的值，并将各输出项的值依次显示在屏幕的同一行上。

例：

```
>>>print(10,20)
10 20
>>>print(10,20,sep=',')
10,20
>>>print(10,20,sep=',',end='*')
10,20*
```

第三次 print() 函数调用时。以 “*” 作为结束符，并且不换行。print() 函数输出文本时默认会在最后增加一个换行，使下一条命令提示符和语句显示在下一行。如果不希望在本行输出最后增加这个换行，或者希望本行输出文本后增加其他内容，可以对 print() 函数的 end 参数进行赋值。

例：

```
print(10,20,sep=',',end='*')
print(30,40,end="#")
print(50,60,end="")
print(70,80,90,sep=":",end="Goodbye!")
输出结果：
10,20*30 40#50 6070:80:90Goodbye!
#每次执行的print()函数并没有产生换行
```

2.6 格式化输出

例：

```
>>>7.80
7.8
```

末尾的 0 没有输出，在很多情况下没有太大问题，但有时就必须输出。

例：在财务系统中，表示七元八角不应显示成 7.8，而应显示为 7.80，甚至在前面还要加货币符号，即 ¥7.80。

为了解决这个问题，可以采用格式化输出。Python 的格式化输出方法有三种：

（1）利用字符串格式化运算符。（不建议使用，这种旧式的格式化方法最终会从 Python 中移除）

（2）利用 format() 内置函数。

（3）利用字符串的 format() 方法。

例：

```
>>>print('t=¥%.2f'%7.8)
t=¥7.80
>>>print('t=¥',format(7.8,'.2f'))
t=¥7.80
>>>print('t=¥{0:.2f}'.format(7.8))
t=¥7.80
```

2.6.1 字符串格式化运算符%

（不建议使用，这种旧式的格式化方法最终会从 Python 中移除）

用运算符% 分隔格式字符串与输出项，一般格式为：

格式字符串%(输出项 1, 输出项 2,……, 输出项 n)

格式字符串由普通字符和格式说明符组成。普通字符原样输出，格式说明符决定所对应输出项的输出格式。

格式说明符以百分号% 开头，后接格式标志符。

例：

```
>>>'V=%s,%s,%s'%(1,2.3,['AA','BB','CC'])
"V=1,2.3,['AA','BB','CC']"
>>>print('V=%s,%s,%s'%(1,2.3,['AA','BB','CC']))
V=1,2.3,['AA','BB','CC']
```

在格式化运算符% 后面的括号内有三个输出项，即 1、2.3 和 ['AA','BB','CC']，都使用格式说明符%s 将值转换为字符串。

常用格式说明符	
格式说明符	**格式化结果**
%%	百分号
%c	字符
%s	字符串
%d	带符号整数（十进制）
%o	带符号整数（八进制）
%x 或%X	带符号整数（十六进制）
%e 或%E	浮点数字（科学计数法）
%f 或%F	浮点数字（用小数点符号）
%g 或%G	浮点数字（根据值的大小，采用%e 或%f）

一般情况下，如果没有什么特殊要求，不管输出项的类型如何，都可使用格式符%s。

例:

```
1  >>> '%6.2f'%1.235
2  '␣␣1.24'
```

总共输出的长度为 6 个字符，其中小数部分占 2 位。

例:

```
1  >>> '%06.2f'%1.235
2  '001.24'
```

如果输出的位数不足 6 位就用 0 补足 6 位。(小数点也占用 1 位)

类似于这里 0 的标记还有-、+。其中，-表示左对齐，+ 表示在正数前面也标上 + 号（默认是不加的)。

例:

```
1  >>> '%(name)s:%(score)06.1f'%{'score':9.5,'name':'
       Lucy'}
2  'Lucy:0009.5'
```

这种形式只用在要输出的内容为字典类型时。每个格式说明符对应哪个输出项由圆括号中的键来指定。

例:

```
>>>'%0*.*f'%(6,2,2.345)
'002.35 '
```

相当于'%06.2f'%2.345

例:

```
>>>print("%+3d,%0.2f"%(25,123.567))
+25,123.57
>>>print("N:%-10s,A:%-8d,S:%-.2e"%("Aviad"
    ,25,1839.8))
N:Aviad      ,A:25      ,S:1.84e+03
>>>nH=0xFF
>>>print("nH=%x,nD=%d,nO=%o"%(nH,nH,nH))
nH=ff,nD=255,nO=377
```

2.6.2 format() 内置函数

format() 内置函数可以将一个输出项单独进行格式化，一般格式为:

format(输出项 [, 格式字符串])

当省略格式字符串时，该函数等价于函数“str(输出项)”的功能。

format() 函数解释格式字符串是根据输出项的类型来决定的，不同的类型有不同的格式化解释。基本的格式控制符有:

d、b、o、x 或 X: 分别按十进制、二进制、八进制、十六进制输出一个整数;

f 或 F、e 或 E、g 或 G: 按小数形式或科学计数形式输出一个整数或浮点数;

c: 输出以整数为编码的字符;

% 输出百分号。

例:

```
>>>print(format(15,'X'),format(65,'c'),format
    (3.145,'f'))
F A 3.145000
```

例：

```
>>>print(format(3.145,'6.2f'))
   3.15
```

格式字符串还可以指定填充字符、对齐方式（其中，< 表示左对齐、> 表示右对齐、^ 表示居中对齐、= 表示填充字符位于符号和数字之间）、符号（其中，+ 表示正号，- 表示负号）。

例：

```
>>>print(format(3.145,'0=+10'),format(3.14159,'
    05.3'))
+00003.145  03.14
>>>print(format(3.145,'0+10'),format(3.14159,'05.3
    '))
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: Invalid format specifier
#显然“0+10”处不加‘=’时输出错误
>>>print(format('test','<20'))
test
>>>print(format('test','^20'))
         test
```

比较：

```
>>> print(format(3.145,'6.2f'))
   3.15
>>> print(format(3.145,'6.2'))
    3.1
```

显然，当“.2”后面有‘f’时，输出为两位小数，三位数字；而当“.2”后面没有‘f’时，输出仅为两位数字。

2.6.3 字符串的 format() 方法

调用格式为：

格式字符串.format(输出项 1, 输出项 2,……, 输出项 n)

格式字符串中可以包括普通字符和格式说明符。普通字符原样输出，格式说明符决定所对应输出项的转换格式。

```
>>>a,b=2,3
>>>print("a={},b={},a*b={}".format(a,b,a*b))
a=2,b=3,a*b=6
```

大括号表示一个槽位置，大括号中的内容由后面紧跟的 format() 方法中的参数按从左到右的次序顺序填充（也可以按照序号或键指定）。

格式说明符使用大括号括起来，一般形式如下：

[序号或键]: 格式说明符

可选的序号对应于要格式化的输出项的位置，从 0 开始。0 表示第一个输出项，1 表示第二个输出项，以后依此类推。序号全部省略则按输出项的自然顺序输出；

可选的键对应于要格式化的输出项的名字或字典的键值；

格式说明符同 format() 内置函数。格式说明符用冒号 (:) 开头。

例：

```
>>>'{0:.2f},{1}'.format(3.145,500)
'3.15,500'
```

格式说明符“0:.2f”包含了两方面的含义：“0”表示该输出项是 format 括号中的第一个输出项（序号为 0）；格式符“:.2f”用于描述该输出项如何被格式化，即小数部分占 2 位，按输出项实际位数输出。“1”表示该输出项是 format 括号中的第二个输出项（序号为 1）。它采用默认格式输出。

（1）使用大括号“{}”格式说明符，大括号及其里面的字符（称为格式化字符）将会被 format() 中的参数替换。

例:

```
>>>print('a={},b={}'.format(1,2))
a=1,b=2
>>>import math
print("PI={}.".format(math.pi))
```

```
5    PI=3.141592653589793.
```

（2）使用“{序号}”形式的格式说明符，在大括号中的数字用于指向输出对象在 format() 函数中的位置。

例:

```
1    >>>print('a={0},b={1},c={2}'.format(11,22,33))
2    a=11,b=22,c=33
```

例:

```
1    >>>print('a={1},b={0},c={2}'.format(11,22,33))
2    a=22,b=11,c=33
```

（3）使用“{键}”形式的格式说明符，大括号中是一个标识符，该标识符会指向使用该名字的参数。

例:

```
1    >>>print('a={k},b={m},c={n}'.format(m='AA',n='BB',
         k='CC'))
2    a=CC,b=CC,c=BB
```

（4）混合使用“{序号}”、“{键}”形式的格式说明符。

例:

```
1    >>>print('a={1},b={n},c={0},d={m}'.format('AA','BB
         ',m='CC',n='DD'))
2    a=BB,b=DD,c=AA,d=CC
```

（5）输出项的格式控制在 {} 中序号或键后面可跟一个冒号: 和格式符。{0:8} 表示 format 中的第一个参数（序号为 0）占 8 个字符宽度，如果输出位数大于该宽度，就按实际位数输出；如果输出位数小于此宽度，默认右对齐，左边补空格，补足 8 位。

{1:.3} 表示第二个参数（序号为 1）除小数点外的输出位数是 3 位。

{1:.3f} 表示浮点数的小数位保留 3 位，其中 f 表示浮点型（d 表示整型）。

例:

```
>>>print('PI={0:.3f}.'.format(math.pi))
PI=3.142.
>>>print('PI={0:.3}.'.format(math.pi))
PI=3.14.
```

{0:.3} 表示第一个参数（序号为 0）除小数点外的输出位数是 3 位。
{0:.3f} 表示浮点数的小数位保留 3 位，其中 f 表示浮点型。
例：

```
print('{0:<15}'.format(1234567890)) #左对齐
print('{0:>15}'.format(1234567890)) #右对齐
print('{0:*^15}'.format(1234567890)) #居中，用*填充
print('{0:10b}'.format(65)) #二进制，默认右对齐
print('{0:10o}'.format(65)) #八进制
print('{0:10x}'.format(65)) #十六进制，字母小写
1234567890
      1234567890
**1234567890***
   1000001
       101
        41
>>> print('{0:<15}\n{0:>15}\n{0:*^15}\n{1:10b}\n{1:10o}\n{1:10x}'.format(1234567890,65))
1234567890
      1234567890
**1234567890***
   1000001
       101
        41
```

2.7 顺序结构程序举例

一个 Python 程序不需要变量定义，可直接描述程序功能。程序结构：

（1）输入（I, input）：输入原始数据。

（2）计算（P, process）：对原始数据进行处理。

（3）输出（O, output）：输出处理结果。

例：已知 $x=5+3i$，$y=e^{\frac{\sqrt{\pi}}{2}}$，求 $z=\frac{2\sin 56^o}{x+\cos|x+y|}$ 的值。

```
import math
x=5+3J; #x是一个复数
y=math.exp(math.sqrt(math.pi)/2);
z=2*math.sin(math.radians(56)) #z的分子
z/=(x+math.cos(abs(x+y))) #求z
print("z=",z)
z= (0.24736586291983403-0.1531502292295295j)
```

例：从键盘输入一个 3 位整数 n，输出其逆序数 m。

```
n=int(input("n="))
a=n%10; #求n的个位数字
b=n//10%10; #求n的十位数字
c=n//100; #求n的百位数字
m=a*100+b*10+c
print("{0:3}的逆序数是{1:3}".format(n,m))
127的逆序数是721
```

例：求一元二次方程 $ax^2+bx+c=0$ 的根。

```
from cmath import sqrt
a=float(input('a='))
b=float(input('b='))
c=float(input('c='))
d=b*b-4*a*c
x1=(-b+sqrt(d))/(2*a)
x2=(-b-sqrt(d))/(2*a)
print("x1={0:.5f},x2={1:.5f}".format(x1,x2)
```

3 选择结构

3.1 条件的描述

关系运算（>、<、!= 等）
逻辑运算（not、and、or）
成员运算（in、not in）
身份运算（is、is not ）

3.1.1 关系运算

关系运算符有：<(小于)、<=(小于等于)、>(大于)、>=(大于等于)、==(等于)、!=(不等于)

关系表达式：表达式 1 关系运算符 表达式 2

关系表达式的值是一个逻辑值，即结果为真（True）或假（False ）。

注意：布尔常量 True 和 False 首字母必须大写!

数字运算时，True 可以看成 “1”，False 可以看成 “0”。

关系运算符的优先级相同，但关系运算符的优先级低于算术运算符的优先级。

例：“a<b+c” 等价于 “a<(b+c)”。

```
1  >>>2+True>2
2  True
3  >>>2+(True>2)
4  2
```

3.1.2 逻辑运算

逻辑运算符有：and（与）、or（或）、not（非）

not 的优先级最高，其次是 and，or 的优先级最低。

逻辑表达式：逻辑量 1 逻辑运算符 逻辑量 2

Python 把非 0 当成真，0 当成假：运算量或运算结果非零即为 True，否则就是 False。

例：

```
>>>not 0
True
>>>not "AA"
False
>>>3+(not False)
4
```

逻辑运算的重要规则：短路运算

(1) a and b：当 a 为 False 时，则不再计算后续的与运算分量 b。

(2) a or b：当 a 为 True 时，则不再计算后续的或运算分量 b。

3.1.3 测试运算

1. 成员测试

成员运算符：in 和 not in

格式：x in y 和 x not in y

in：在指定序列 y 中查找某个值 x 是否存在，若存在就返回 True，否则返回 False

not in：在序列 y 中没有找到值 x 时返回 True。

例：

```
>>>3 in (20,15,3,14,5) #元组
True
>>>3 not in (20,15,3,14,5)
False
```

2. 身份测试

身份运算符：is 和 is not

格式：x is y 和 x is not y

用于测试两个变量是否指向同一个对象。

例：

```
>>>a=20
>>>b=20
>>>a is b
```

```
4       True
5       >>>a is not b
6       False
```

例:

```
1       >>>a=1234
2       >>>b=1234
3       >>>a is b
4       False
5       >>>a=b=1234
6       a is b
7       True
```

涉及到 Python 解释器对内存的分配问题，详见 1.2 节 “id()”。

3.2 选择结构的实现

3.2.1 单分支选择结构

if 语句格式:

if 表达式:

语句块

注意:

(1) 在 if 语句的条件表达式后面必须加冒号。

(2) 条件表达式可以是任意表达式。

(3) if 语句中的语句块必须向右缩进，语句块可以是单个语句，也可以是多个语句。当语句块包含两个或两个以上的语句时，语句必须缩进一致，即语句块中的语句必须上下对齐。

(4) 如果语句块中只有一条语句，if 语句也可以写在同一行上。

例:

```
1       >>>if 'A':
2           print('BBB')
3       BBB
4       >>>if 3>2:print("BBB")
5       BBB
```

```
6       if 3>2:
7           x=4
8           y=5
9           print(x,y)
10      4 5
```

例:

```
1       x=1
2       if x>2:print("AAA")
3       print("Goodbye!")
4       运行结果:
5       Goodbye!
6
7       x=1
8       if x>2:
9           print("AAA")
10          print("Goodbye!")
11      运行结果:
12      (无)
```

例: 输入两个整数 a 和 b，先输出较大数，再输出较小数。

```
1       a,b=eval(input("输入a,b:"))
2       if a<b: #若a<b，交换a和b，否则不交换
3           a,b=b,a
4       print("{0},{1}".format(a,b))
```

3.2.2 双分支选择结构

一般格式为:

if 表达式:

    语句块 1

else:

    语句块 2

注意：if 语句表达式后面或 else 后面的语句块应缩进对齐。

例：

```
x=1
if x>2:
    y=3
    print(y)
else:
    y=4
    print(y)
运行结果:
4
```

例：输入三角形的三个边长，求三角形的面积。

```
from math import *

a,b,c=eval(input("a, b, c="))
if a+b>c and a+c>b and b+c>a:
    p=(a+b+c)/2
    s=sqrt(p*(p-a)*(p-b)*(p-c))
    print("a={0},b={1},c={2}".format(a,b,c))
    print("area={}".format(s))
else:
    print("a={0},b={1},c={2}".format(a,b,c))
    print("input data error")
```

3.2.3 多分支选择结构

一般格式为:

if 表达式 1:

语句块 1

elif 表达式 2:

语句块 2

elif 表达式 3:

语句块 3

……

elif 表达式 m:

语句块 m

else:

语句块 n

3.2.4 选择结构的嵌套

语句一:

```
if 表达式1:
    if 表达式2:
        语句块1
    else:
        语句块2
```

语句二:

```
if 表达式1:
    if 表达式2:
        语句块1
else:
    语句块2
```

根据对齐格式来确定 if 语句之间的逻辑关系。

例：输入的三个数，找出其中最大数。

```
x,y,z=eval(input("x,y,z="))
max=x
if z>y:
    if z>x:
        max=z
elif y>x:
    max=y
print("The max is:{}".format(max))
```

3.3 条件运算

条件运算有三个运算量，其一般格式为：

表达式 1 if 表达式 else 表达式 2

先求中间条件表达式的值，如果其值为 true，则以左边表达式 1 的值为结果；否则，则以右边表达式 2 的值为结果。

例：生成 3 个 2 位随机整数，输出其中最大的数。

```
1  import random
2  x=random.randint(10,99)
3  y=random.randint(10,99)
4  z=random.randint(10,99)
5  max=x if x>y else y
6  max=max if max>z else z
7  print("x={},y={},z={}".format(x,y,z))
8  print("max=",max)
```

4 循环结构

循环结构由循环体及循环条件两部分组成，被重复执行的语句称为循环体，决定是否继续重复的表达式称为循环条件。

Python 提供了 while 语句和 for 语句来实现循环结构。

4.1 while 循环结构

当型循环：先判断循环条件，条件满足时执行循环体。

4.1.1 While 语句的一般格式

while 表达式:
语句块

注意：在循环体内必须有修改循环条件表达式值的语句，使其值趋向 False，让循环趋于结束，避免无限循环（死循环）。

当程序死循环时，可以按 Ctrl+C 快捷键来中断循环。

如果循环体中只有一条语句，可以将该语句与 while 写在同一行中。

例:

```
1    >>>i=1
2    >>>while i in range(5):print(i,end=' ');i+=1
3    1 2 3 4
```

4.1.2 在 while 语句中使用 else 子句

在 Python 中，可以在循环语句中使用 else 子句，else 中的语句会在循环正常执行完的情况下执行（不管是否执行循环体）。

但当通过 break 语句跳出循环体而中断循环时，else 部分将不会被执行。

例:

```
1    count=int(input())
2    while count<5:
3        print(count,"is less than 5")
4        count=count+1
5    else:
```

```
6           print(count,"is not less than 5")
```

4.1.3 while 循环的应用

例：while 循环的应用

```
1   s=0
2   n=1
3   while n<=100: #循环条件
4       s+=n #实现累加求和
5       n+=1 #n增1
6   print("1+2+3+……+9+100=",s)
```

也可以利用列表的求和函数来实现：

```
1   t=list(range(1,101))
2   s=sum(t)
3   print(s)
```

例：求

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

，直到最后一项的绝对值小于 10^{-6} 时停止计算。其中 x 为弧度，但从键盘输入时以角度为单位。

```
1   from math import *
2   x = eval(input("请输入一个度数："))
3   x = radians(x)
4   i = 1  #判断正负号
5   sum = x  #总和
6   tp = x  #临时值
7   while abs(tp) >= 1e-6:
8       tp *= -x * x / (2 * i * (2 * i + 1))
9       i += 1
10      sum += tp
11  print("x={0:.3f},sin(x)={1:.3f}".format(x, sum))
```

4.2 for 循环结构

计数循环：已知重复执行次数的循环。

while 语句和 for 语句都可以实现计数循环。Python 中的 for 循环也是一个通用的序列迭代器，可以遍历任何有序的序列对象的元素。for 语句可用于字符串、列表、元组以及其他内置可迭代对象。

4.2.1 for 语句的一般格式

```
for 目标变量 in 序列对象:
    语句块
```

注意：

(1) for 语句是通过遍历任意序列的元素进行来建立循环的，针对序列的每一个元素执行一次循环体。列表、元组、字符串都是序列，可以利用它们来建立循环。

遍历列表:

```
s=['AA','BB','CC']
for i in s:
    print('s=',i)
程序运行结果:
s= AA
s= BB
s= CC
```

遍历元组：

```
s=('AA','BB','CC')
for i in s:
    print('s=',i)
程序运行结果:
s= AA
s= BB
s= CC
```

遍历字符串:

```
    for i in "uestc":
        print(i,end='-')
    程序运行结果:
    u-e-s-t-c-
```

例:

```
    ch='c'
    for ch in "abcde":
        print(ch,end='-')
    程序运行结果:
    a-b-c-d-e
```

遍历集合:

```
    s={'AA','BB','CC'}
    for i in s:
        print('s=',i)
    运行结果可能每次都不相同（无序）
```

(2) for 循环的循环次数显然就是序列中的元素个数，即序列的长度。可以利用序列长度来控制循环次数，这时关注的不是序列元素的值，而是元素的个数。

例：读入 5 个数，求其和。

```
    s=0
    for i in [5,2,4,1,3]: #与列表元素的值无关
        x=int(input())
        s+=x
    print("s=",s)
```

(3) 可以在 for 循环体中修改目标变量的值，但当程序执行流程再次回到循环开始时，就会自动被设成序列的下一个元素。退出循环之后，该变量的值就是序列中最后的元素。

例:

```
    s=[1,2,3,4,5]
    for i in s:
```

```
3           print(i,end="")
4           i=i+3
5       print(",i=",i)
6       运行结果:
7       12345,i= 8
```

(4) for 语句也支持一个可选的 else 块，它的功能就像在 while 循环中一样，如果循环离开时没有遇到 break 语句，就会执行 else 块。即序列所有元素都被访问过了之后，执行 else 块。

例:

```
1       for i in "abcd":
2           print(i,end="")
3       else:
4           print("␣Goodbye!")
5       运行结果:
6       abcd Goodbye!
```

例:

```
1       for i in "1234":
2           print(i,end="")
3           if i=='3':break
4           print(i,end="")
5       else:
6           print("␣Goodbye!")
7       运行结果:
8       11223
9       退出本层循环
```

例:

```
1       for i in "1234":
2           print(i,end="")
3           if i=='3':continue
4           print(i,end="")
5       else:
```

```
6           print("␣Goodbye!")
7       运行结果:
8       1122344  Goodbye!
9       退出本次循环
```

4.2.2 range 对象在 for 循环中的应用

range() 函数返回的是可迭代对象。

在 for 循环中，Python 将自动调用内置函数 iter() 获得迭代器，自动调用内置函数 next() 获取元素，还完成了检查 stopIteration 异常的工作。如果需要遍历一个数字序列，可以使用 range 对象。

例:

```
1       for i in range(5):
2           print(i,end='␣')
3       运行结果:
4       0 1 2 3 4
```

首先 Python 对关键字 in 后的对象调用 iter() 函数获得迭代器，然后调用 next() 函数获得迭代器的元素，直到抛出 stopIteration 异常。

例:

```
1       i=1
2       for i in range(1,7,2):
3           print(i,end='␣')
4           i+=1
5       运行结果: 1 3 5
6       i=2
7       for i in range(7,1,-2):
8           print(i,end='␣')
9           i+=1
10      运行结果: 7 5 3
11      i=1
12      while i in range(1,7,2):
13          print(i,end='␣')
14          i+=1
```

```
15      运行结果:  1
16      i=7
17      while i in range(7,1,-2):
18          print(i,end='␣')
19          i-=2
20      运行结果:  7 5 3
```

4.3 循环控制语句

循环控制语句可以改变循环的执行路径。Python 支持以下循环控制语句: break 语句、continue 语句和 pass 语句。

4.3.1 break 语句

break 语句用在循环体内，迫使所在循环立即终止，即跳出所在循环体，继续执行循环结构后面的语句。

提前结束本层循环。

4.3.2 continue 语句

在循环结构中执行 continue 语句时，并不会退出循环结构，而是立即结束本次循环，重新开始下一轮循环。跳过循环体中在 continue 语句之后的所有语句，继续下一轮循环。

提前结束本次循环

4.3.3 pass 语句

pass 语句是一个空语句，它不做任何操作，代表一个空操作。

pass 语句用于在某些场合下语法上需要一个语句但实际却什么都不做的情况，就相当于一个占位符。

例:

```
1   x,y=2,3
2   if x>y:
3       pass
4   else:
5       x,y=y,x
```

```
6     print("max=",x)
```

subparagraph 标题

算术运算符：+(加)、-(减)、*(乘)、/(除)、//(整除)、%(求余)、**(乘方)

位运算就是直接对整数按二进制位进行操作，其运算符主要有：&，|，~，^，>>和<<。

表达式 1 if 表达式 else 表达式 2

```
1     from tkinter import *
```