Python小清单：

变量： 创建变量后，可以存储、检索或修改与该变量关联的数据。

          请求对象ID：

            >>>id(aint)

对象：python中的对象有：

        \*一个标识；

        \*一些属性；

        \*零个或多个名称。

 字符串：str类型。

        只有字符串类型（没有单个字符与字符串的不同）

列表：list类型。         [4,3.57,'abc']

        列表也是序列类型，但它的序列中不仅包含字符，还能包含元素。列表也是 一个数据集合。

字典：dist类型。 映射类型（集合类型）

        元素中的第一个元素是键，第二个元素是值。键可用于值得搜索，与字典或电话簿类似。

            {"Jones":3456858,"Tom":56984}

集合:     set类型。

        集合是含有不重复元素的数据集。  set([1,3,5])

字符串

**python字符串对象的构建**通过两种方法进行：一种构造方法是利用字符串构造函数str，另一种更便捷的方式是用两个单引号（')或两个双引号（")括起来。

    三重引号字符串（''')可以保留所有字符串的格式信息：如果字符串跨越多行，行与行之间的那些回车符也会保存，引号、制表符也会保存。

**字符序列 ，**字符在字符串中的位置称为”索引“。 python使用索引用用算符”[]“，查看字符序列的单个字符。

    例如”hello world"[0]在索引0位置的字符指的是字符‘h'。

>>>hellostring="hello world"

>>>hellostring

'hello world'

>>>hellostrint[5]

' '

>>>hellostring[-1]

'd'

**索引和分片**：不仅可以对字符建立索引，还可以对字符串中的子序列建立索引。

    例如：>>>hellostring[6:10]

               'worl'

注意;python采用的是半开区间，包含起始值，不包含结束值。

    >>>hellostring[6:]

    'world'    #从第六位开始

    >>>hellostring[:5]

    'hello'    #从起始位（0）开始，到第五位结束

    >>>hellostring[-1]

    'd'    # 负一（-1）位即为最后一位。python的解释是负的步长意味着向后进行。

    >>>hellostring[3:-2]

    'lo wor'

拓展分片：步长                                            【起始值：结束值：步长】

    >>>hellostring[::2]

    'hlowrd'

    >>>hellostring[::3]

    'hlwl'

复制分片：不提供分片的其起始值和结束值，那么得到的是原字符的副本。

    >>>nameone='monty'

    >>>nametwo=nameone[:]

    >>>nametwo

    'nametwo'

**字符串的操作**

连接符号（+）和重复符号（\*）

单个字符字符串的比较符号：有==，>,<

ord(a)的值是97 ord(A)的值是65

多字符字符串的比较与C语言一样：从第一个向后比较。

**in**运算符：           in运算符用于检查集合的成员。例如：'a' in 'abcd'，运算符有两个参数：测试字符串和可能包含测试字符串的字符串。返回bool值

>>>vowels='aeiou'

>>>'a'in vowels

true

>>>'x'in vowels

false

**字符串集合是不可改变的**

**字符串函数**

    len函数在其圆括号中只需要一个参数，即要确定其长度的字符串。

>>>mystr='hello world'

>>>len(mystr)

11

>>>strlength=len(mystr)

>>>print strlength

11

**字符串方法：**调用采用点号（.）来表示。

**upper**方法:调用方法是 A string.upper( )     ’A string'对象调用方法upper来创建一个新的字符串，调用的对象本身转换为大写的形式。

>>>mystring='python rules'

>>>mystring.upper()

'PYTHON RULES'

**find**方法：返回子串在原字符中的第一次出现的索引位置(如果出现多次），如果没有找到该子串，则返回-1.

**方法链**：可以链接方法和函数，使用一系列的“点标记”来实现链接，如'A String'.upper().find('S')

>>>mystring='python rules'

>>>mystring.upper()

'PYTHON RULES'

>>>mystring.upper().find('O')

4

**可选参数：**默认情况下，find从索引为0处开始，但如果提供了第二个参数，就从该参数指定的位置开始查找过程。

>>>astring='he had the bat'

>>>astring.find('t')

7

>>>astring.find('t',8)

13

#如果提供第三个参数，find将在该索引所指的位置停下来

>>>astring.find('t',1,6)

false

**回文：**回文是一类字符串，这种字符串向后读取与向前读取都得到相同的内容。例如一个典型的例子："Madam,I'm Adam"或"A man, a plan,  a cancal, Panama"

    字符反转是分片操作，从字符串的结束为位置开始，开始位置结束，步长为-1.

几个方法

**upper方法**:调用方法是 A string.upper( )     ’A string'对象调用方法upper来创建一个新的字符串，调用的对象本身转换为大写的形式。

>>>mystring='python rules'

>>>mystring.upper()

'PYTHON RULES'

**lower方法**：和**upper作用**相反

**find方法**：返回子串在原字符中的第一次出现的索引位置(如果出现多次），如果没有找到该子串，则返回-1.

**方法链**：可以链接方法和函数，使用一系列的“点标记”来实现链接，如'A String'.upper().find('S')

>>>mystring='python rules'

>>>mystring.upper()

'PYTHON RULES'

>>>mystring.upper().find('O')

可选参数：默认情况下，find从索引为0处开始，但如果提供了第二个参数，就从该参数指定的位置开始查找过程。

>>>astring='he had the bat'

>>>astring.find('t')

7

>>>astring.find('t',8)

13

#如果提供第三个参数，find将在该索引所指的位置停下来

>>>astring.find('t',1,6)

false

**split方法：**split方法创建主调字符串的子串，字符串被指定的字符分割。在空格处的分割调用方法是.split(' ")。在每个逗号处分割的方法调用是.split(', ')。

>>>name='john marwood cleese'

>>>first,middle,last=name.split()

>>>transformed=last+', '+first+' '+middle

>>>print transformed

cleese, john marwood

    split方法可以用于用逗号分隔的数据，例如电子表格和数据库产生的数据。此外，该方法不仅可以用手变量，还可以用于字符串常量。最后，如果未指定分隔符，分隔符默认为空白，即空格、制表符或回车符。

>>>line='bob,carol,ted,alice'

>>>first,second,third,fourth=line.split(',')

>>>print fisrt,second,third,fourth

bob carol ted alice

>>>op1,op2="A+B".split('+')

>>>print op1,op2

A B

**replace方法：**需要两个参数，第一个参数是正在查找的字符串，第二个参数是如果找到第一个字符串时要替换成的字符串。例如

>>>'abbc'.replace('b','z')

'azzc'

**strip方法**：消除回车符，和一些不可见字符。如果strip方法没有给出参数，则将从字符串的两端移除空白。如果提供了字符串参数，则只有那些在参数中的字符将被删除。例如 astring.strip('., ')，它消除字符串中的逗号、句号，以及开头和结尾处的空格字符。如', this. '.strip('., ')和'this,,.',.strip('., ')都将得到string对象'this'。

    strip方法只适用于字符串的开头和结尾。例如， 'this string'.strip()将得到字符串'this string'。因为没有提供参数，所以默认情况是消除空白（制表符，回车，空格等），但字符串中间的空白不会收到影响。

    注意：所有开头和结尾处的空白字符都将被删除。

函数

python函数有两部分：定义和调用。

    函数定义由关键字def开始，执行def语句，在命名空间中创建新的名字和与名字相关联的新对象。

    所有东西都是对象，函数也不例外。

    def是复合语句，这意味着在函数的一些部分中包含了其他的python语句和表达式。

    函数中用到的特殊关键字是return语句，一个函数可以有多个return语句，执行的第一条return语句将结束函数。

函数示例：猜词。

# Find a word with a single example of the vowels a,e,i,o,u in that order

datafile=open("dictionary.txt","r")#如果是w的话就是写

def cleanword(word):

"""return word in lower case striped of whitespace."""

return word.strip().lower()

def getvowelsinword(word):

'''return voowles in string word-jinclude repeats.'''

vowelstr="aeiou"

vowelsinword=""

for char in word:

if char in vowelstr:

vowelsinword+=char

return vowelsinword

# mian program

print "Find words containing vowels 'aeiou' in that order:"

for word in datafile:

word=cleanword(word)

if len(word)<=6:

continue

vowelstr=getvowelsinword(word)

if vowlestr=="aeious":

print word

列表与元组

python的内置列表类型也是数据集类型，与字符串一样，也是序列类型。

    与字符串的不同点： 1，列表不仅可以包含字符，还可以包含其他元素；

                                 2，列表是可变类型。

**创建列表：**使用构造函数list创建列表，或者采用方括号( [ ] ) 来构造列表。分辨是列表还是索引操作符的方法是查看是否有逗号，列表中的每个元素由逗号进行分隔。

>>>alist=[1,2,'a',3.14159]

>>>listoflists=[[1,2,3],['a','b','c']]

>>>listfromcollection=list('hello') #使用构造函数list构造的列表，有参数，参数必须是另一个集合

>>>alist

[1,2,'a',3.141589999999999]

>>>listoflists

[[1,2,3],['a','b','c']]

>>>listfromcollection

['h','e','l','l','o']

>>>[]

[]

>>>

    注意：非集合类型（整数，浮点数，布尔值等）不能用来做作为list构造函数的参数。

**二维列表：......**

**索引和分片：**

    注意：索引运算符是方括号及其内的整数或分片。如果方括号包含单个整数，则整数为列表中元素的索引值。

            【起始索引：结束索引：步长】

>>>[1,2,3,4,5,6][2]

3 #将列表的创建与索引操作放在一起

**列表可变性**

>>> mylist=[1,2,'a','z']

>>> mylist[0]

1

>>> mylist[0]='true'

>>> mylist

['true', 2, 'a', 'z']

>>> mylist[-1]=7

>>> mylist

['true', 2, 'a', 7]

>>> mylist[:2]=[27]

>>> mylist

[27, 'a', 7]

>>> mylist[:]=[1,2,3,4]

>>> mylist

[1, 2, 3, 4]

>>> mylist[2:]='abc'

>>> mylist

[1, 2, 'a', 'b', 'c']

    注意：带有索引运算符的列表可以出现在赋值语句的左侧。意味着，完成赋值后，索引位置的值变成了赋值语句右边的值。

                可以改变单个元素也可以一次性改变整个列表分片。唯一需要注意的是，所添加的值必须是集合类型。集合中的每个元素将作为元素添加到列表中。

**range 、split和多重赋值**

    当字符串调用split时，需要提供该方法的参数（没有参数就是任何空白），来创建由参数“分裂”得到的字符串元素。

    当调用range时，它创建参数范围内的整数列表。如果知道列表中元素的数量，可以使用多重赋值，将列表中每个元素分别赋给变量。

>>> result='this is a test'.split()

>>> result

['this', 'is', 'a', 'test']

>>> result='field1,field2,field3,field4'.split(',')

>>> result

['field1', 'field2', 'field3', 'field4']

>>> rangelist=range(-5,5)

>>> rangelist

[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]

**使用join在列表与字符串之间转换**

    jion方法以字符串列表作为参数，并作为分隔符使用主调对象，将每个元素放置到新的字符串中。

    例如调用 ':'.join( 'a','b','c'])，用冒号（：）作为分隔符，连接所有字符，生成的字符串 'a:b:c'

**可变对象及其引用：**

    赋值分片创建列表的副本（[ ： ]） ，创建副本意味着python将创建一个新的对象副本，因此这种修改不会影响其他对象，同样，is运算符显示这些名字没有引用同一对象。

>>> list1=[1,2,3]

>>> list

<type 'list'>

>>> list1

[1, 2, 3]

>>> list2=list1[:]

>>> list1 is list2

False

>>> list2

[1, 2, 3]

>>> list1.append(27)

>>> list1

[1, 2, 3, 27]

>>> list2

[1, 2, 3]

    当列表自身作为一个元素进行追加时，将出现无穷递归的情况：

>>> list1=[1,2,3]

>>> list1.append(list1)

>>> list1

[1, 2, 3, [...]] # [...] 表示无穷递归

>>> #因为该列表是可变的，所以对自身的引用置于列表末尾的，这意味着该列表无限次重复引用自身

**深拷贝与浅拷贝**

    python中，拷贝就是为对象添加新的名字。

>>> list1=[1,2,3]

>>> list2=['a','b','c']

>>> listjoin1=list1+list2 # 封装

>>> listjoin1

[1, 2, 3, 'a', 'b', 'c']

>>> listjoin2=[list1]+[list2]

>>> listjoin2

[[1, 2, 3], ['a', 'b', 'c']]

>>> list1[0]=100 #仅影响第二次添加,

>>> listjoin1

[1, 2, 3, 'a', 'b', 'c']

>>> listjoin2 #listjoin2引用最初的list1

[[100, 2, 3], ['a', 'b', 'c']]

>>>

    只复制引用而不复制对象本身的情况，称为浅拷贝。要复制内容，而不是简单的复制引用，称为浅拷贝。

    在copy模块中，有deepcopy函数，在需要的时候可以进行强制深拷贝。

**元组**

    元组基本上都是不可改变的列表。元组几乎具有列表所有的特征。

    mytuple=(1,2,3)

    要重点注意,的是：逗号创建元组的运算符。没有圆括号的表达式1，2，3，同样产生元组。要创建单个元组不能使用表达式（1），因为它只在解释器中产生1，表达式（1，)产生单个元组。

*元组可以做字典的键，但列表不行。*

列表方法

方法是作用于python中特定类型对象的函数。采用点调用法。

列表类型ye有只作用于列表的方法，分两类：

（1）**改变列表的方法**：这些方法不改变列表本身，但是由于其处理过程他们将返回值

    index(x) 返回列表中与x值相等的第一个元素的索引。如果在列表中没有找到这样的元素，python将报错。

    count(x) 返回列表中x出现的次数。如果列表中不含x，返回0。

（2）**不改变列表的方法**

    注意：a，调用方法的列表会在调用后发生改变；

                b, 除了pop方法，其他方法都不返回值。                     *而，字符串方法都有返回值（一个新的字符串),字符串是不可以改变的，因此必须通过返回值来产生新的字符串对象。*

    append(x)在列表的末尾添加元素。列表的长度增加1，集合也可作为单个元素进行添加。

    pop(x) 删除列表末尾的元素，并返回次元素。 如果指定索引值，则a.pop(anindex）将删除该索引位置的元素并返回该项。

    extend(C) 需要一个集合作为参数。将此参数集合中的每个元素添加到列表的末尾，从而扩展列表。

    insert(i,x) 在指定位置插入元素。第一个参数是元素的索引(元素插入前的索引），此后的每个元素一次位移。

    remove(x) 删除列表中第一个值x的元素。如果没有对应的项将出错。

    sort( ) 将列表中的元素进行排序。与sorted函数进行比较，sorted函数返回拍好序的列表，但不改变原列表。如果只对列表的列表进行排序，则只考虑每个元素的首元素。      sorted函数往往是更好的选择，因为它可以作用于任何集合，而sort方法只能用于列表。sorted不管参数的类型都将返回一个列表。

    rever( ) 将列表的元素反排序。

======================= RESTART: Shell =======================

>>> alist=[1,12,5,8]

>>> alist

[1, 12, 5, 8]

>>> alist.append(17)

>>> alist

[1, 12, 5, 8, 17]

>>> alist.append([40,50,60])

>>> alist

[1, 12, 5, 8, 17, [40, 50, 60]]

>>> anotherlist=[20,2]

>>> alist.insert(3,'a')

>>> alist

[1, 12, 5, 'a', 8, 17, [40, 50, 60]]

>>> alist.extend(anotherlist)

>>> alist

[1, 12, 5, 'a', 8, 17, [40, 50, 60], 20, 2]

>>> alist

[1, 12, 5, 'a', 8, 17, [40, 50, 60], 20, 2]

>>> alist.remove(8)

>>> alist

[1, 12, 5, 'a', 17, [40, 50, 60], 20, 2]

>>> alist.pop()

2

>>> alist

[1, 12, 5, 'a', 17, [40, 50, 60], 20]

>>> alist.sort

<built-in method sort of list object at 0x02A355A8>

>>> alist.sort()

>>> alist

[1, 5, 12, 17, 20, [40, 50, 60], 'a']

>>>

字典

    字典上的所有操作都通过键实现；

    键没有按顺序排列；

    值可以是任何对象，但键只能是不可改变的对象（如整数，字符或元组）。

    在所搜字典时，首先查找键，然后才是关联的值。

**字典索引和赋值**

字典是可变的：在键中可以对索引赋值，是第二个可变数据结构（第一个是列表）

字典中不同的键类型：

    优势1：索引速度快；

    优势2：值可以是任何类型的数据结构（甚至是另一个字典集合）。

**运算符**

集合操作：

    [ ]：在方括号内使用键的索引

    len()：“长度”指字典中键值对的个数。

    in ：成员测试，得到布尔值。键是否在字典中（而不是值）。

    for ：通过键在字典中进行循环。

>>> mydict={'a':2,3:['x','y'],'joe':'smith'}

>>> mydict

{'a': 2, 3: ['x', 'y'], 'joe': 'smith'}

>>> mydict['a']

2

>>> len(mydict)

3

>>> 'a'in mydict

True

>>> 2 in mydict

False

>>> for key in mydict:

print key,mydict[key]

a 2

3 ['x', 'y']

joe smith

**字典方法：**

    items()：所有键值对的元组列表。

    key() ：所有键的列表。

    values() ：所有值的列表。

    copy() ：浅拷贝的字典。

迭代器 与 生成器（这个是网上的资料复制粘贴的）

**迭代器**

      迭代器是访问集合元素的一种方式。迭代器对象从集合的第一个元素开始访问，知道所有的元素被访问完结束。迭代器只能往前不会后退，不过这也没什么，因为人们很少在迭代途中往后退。

**使用迭代器的优点**

      对于原生支持随机访问的数据结构（如tuple、list），迭代器和经典for循环的索引访问相比并无优势，反而丢失了索引值（可以使用内建函数enumerate()找回这个索引值）。但对于无法随机访问的数据结构（比如set）而言，迭代器是唯一的访问元素的方式。

      另外，迭代器的一大优点是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，元素可以不存在或者被销毁。这个特点使得它特别适合用于遍历一些巨大的或是无限的集合，比如几个G的文件，或是斐波那契数列等等。

      迭代器更大的功劳是提供了一个统一的访问集合的接口，只要定义了\_\_iter\_\_()方法对象，就可以使用迭代器访问。

迭代器有两个基本的方法

* next方法：返回迭代器的下一个元素
* \_\_iter\_\_方法：返回迭代器对象本身

下面用生成斐波那契数列为例子，说明为何用迭代器

def fab(max):

n, a, b = 0, 0, 1

while n < max:

print b

a, b = b, a + b

n = n + 1

复制代码

直接在函数fab(max)中用print打印会导致函数的可复用性变差，因为fab返回None。其他函数无法获得fab函数返回的数列。

代码2

复制代码

def fab(max):

L = []

n, a, b = 0, 0, 1

while n < max:

L.append(b)

a, b = b, a + b

n = n + 1

return L

复制代码

代码2满足了可复用性的需求，但是占用了内存空间，最好不要。

代码3

对比

for i in range(1000): pass

for i in xrange(1000): pass

前一个返回1000个元素的列表，而后一个在每次迭代中返回一个元素，因此可以使用迭代器来解决复用可占空间的问题

复制代码

class Fab(object):

def \_\_init\_\_(self, max):

self.max = max

self.n, self.a, self.b = 0, 0, 1

def \_\_iter\_\_(self):

return self

def next(self):

if self.n < self.max:

r = self.b

self.a, self.b = self.b, self.a + self.b

self.n = self.n + 1

return r

raise StopIteration()

复制代码

执行

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | >>> for key in Fabs(5):      print key      1  1  2  3  5 |

Fabs 类通过 next() 不断返回数列的下一个数，内存占用始终为常数

**1.2 使用迭代器**

使用内建的工厂函数iter(iterable)可以获取迭代器对象：

|  |  |
| --- | --- |
| 1  2  3  4 | >>> lst = range(5)  >>> it = iter(lst)  >>> it  <listiterator object at 0x01A63110> |

使用next()方法可以访问下一个元素：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | >>> it.next()  0  >>> it.next()  1  >>> it.next()  2 |

python处理迭代器越界是抛出StopIteration异常

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | >>> it.next()  3  >>> it.next  <method-wrapper 'next' of listiterator object at 0x01A63110>  >>> it.next()  4  >>> it.next()    Traceback (most recent call last):    File "<pyshell#27>", line 1, in <module>      it.next()  StopIteration |

了解了StopIteration，可以使用迭代器进行遍历了

复制代码

lst = range(5)

it = iter(lst)

try:

while True:

val = it.next()

print val

except StopIteration:

pass

复制代码

结果

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | >>>  0  1  2  3  4 |

事实上，因为迭代器如此普遍，python专门为for关键字做了迭代器的语法糖。在for循环中，Python将自动调用工厂函数iter()获得迭代器，自动调用next()获取元素，还完成了检查StopIteration异常的工作。如下

复制代码

>>> a = (1, 2, 3, 4)

>>> for key in a:

print key

1

2

3

4

复制代码

首先python对关键字in后的对象调用iter函数迭代器，然后调用迭代器的next方法获得元素，直到抛出StopIteration异常。

**1.3 定义迭代器**

下面一个例子——斐波那契数列

复制代码

# -\*- coding: cp936 -\*-class Fabs(object):

def \_\_init\_\_(self,max):

self.max = max

self.n, self.a, self.b = 0, 0, 1 #特别指出：第0项是0，第1项是第一个1.整个数列从1开始def \_\_iter\_\_(self):

return self

def next(self):

if self.n < self.max:

r = self.b

self.a, self.b = self.b, self.a + self.b

self.n = self.n + 1

return r

raise StopIteration()

print Fabs(5)

for key in Fabs(5):

print key

复制代码

**结果**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <\_\_main\_\_.Fabs object at 0x01A63090>  1  1  2  3  5 |

[回到顶部](#_labelTop)

**2. 生成器**

      带有 yield 的函数在 Python 中被称之为 generator（生成器），几个例子说明下（还是用生成斐波那契数列说明）

可以看出代码3远没有代码1简洁，生成器（yield）既可以保持代码1的简洁性，又可以保持代码3的效果

代码4

def fab(max):

n, a, b = 0, 0, 1

while n < max:

yield b

a, b = b, a + b

n = n = 1

执行

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | >>> for n in fab(5):      print n      1  1  2  3  5 |

      简单地讲，yield 的作用就是把一个函数变成一个 generator，带有 yield 的函数不再是一个普通函数，Python 解释器会将其视为一个 generator，调用 fab(5) 不会执行 fab 函数，而是返回一个**iterable 对象**！在 for 循环执行时，每次循环都会执行 fab 函数内部的代码，**执行到** yield b 时，fab 函数就**返回一个迭代值**，下次迭代时，代码从 yield b 的下一条语句继续执行，而函数的本地变量看起来和上次中断执行前是完全一样的，于是函数继续执行，直到再次遇到 yield。看起来就好像一个函数在正常执行的过程中被 yield 中断了数次，每次中断都会通过 yield 返回当前的迭代值。

也可以手动调用 fab(5) 的 next() 方法（因为 fab(5) 是一个 generator 对象，该对象具有 next() 方法），这样我们就可以更清楚地看到 fab 的执行流程：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | >>> f = fab(3)  >>> f.next()  1  >>> f.next()  1  >>> f.next()  2  >>> f.next()    Traceback (most recent call last):    File "<pyshell#62>", line 1, in <module>      f.next()  StopIteration |

return作用

在一个生成器中，如果没有return，则默认执行到函数完毕；如果遇到return,如果在执行过程中 return，则直接抛出 StopIteration 终止迭代。例如

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | >>> s = fab(5)  >>> s.next()  1  >>> s.next()    Traceback (most recent call last):    File "<pyshell#66>", line 1, in <module>      s.next()  StopIteration |

代码5  文件读取

复制代码

def read\_file(fpath):

BLOCK\_SIZE = 1024

with open(fpath, 'rb') as f:

while True:

block = f.read(BLOCK\_SIZE)

if block:

yield block

else:

return

复制代码

如果直接对文件对象调用 read() 方法，会导致不可预测的内存占用。好的方法是利用固定长度的缓冲区来不断读取文件内容。通过 yield，我们不再需要编写读文件的迭代类，就可以轻松实现文件读取。