NLP 南京大学
自然语言处理研究组
Natural Language Processing Group

南京大学
NANJING UNIVERSITY

Python流程控制

黄书剑

程序控制流程

- **控制程序的执行顺序**
  - 顺序
  - 分支
  - 循环
- **程序的数据交互**
  - 控制台
  - 文件

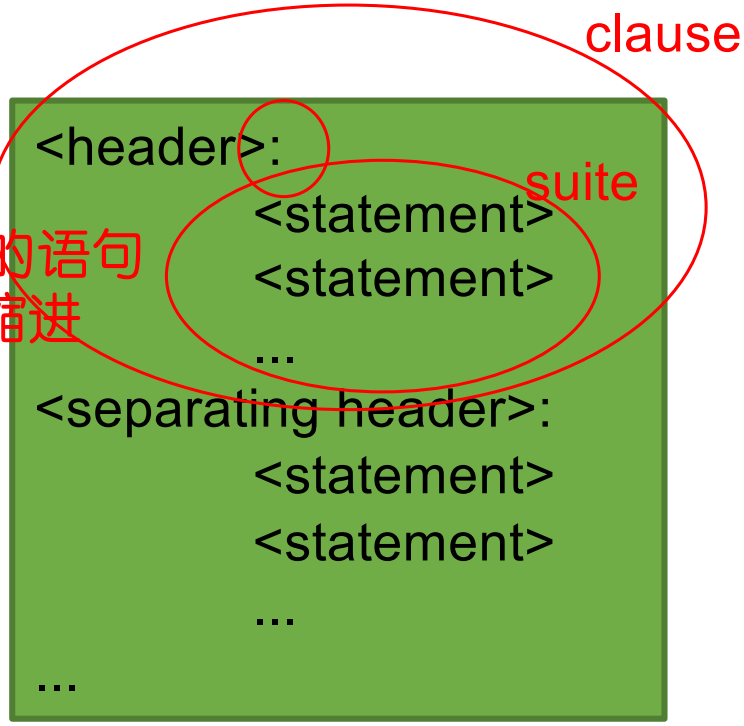
南京大学
NANJING UNIVERSITY

程序执行顺序

Python程序的基本单元

- **值、变量（value、variable）**
- **表达式（expression）**
  - 运算符、函数调用
- **语句（statement）**
  - 单个语句（simple statement）
    - 表达式语句
  - 复合语句（compound statement）
    - 一个或多个分句（clause）

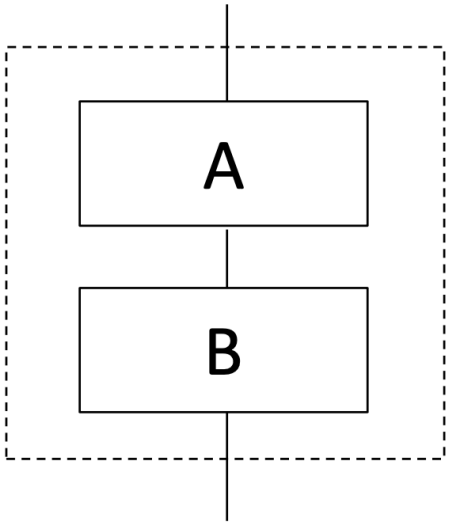
同一个suite的语句
具有相同的缩进



- **一个或多个语句构成一段程序**

顺序执行

- 语句的默认执行顺序为顺序执行
- 改变不同的运行顺序需要依赖特殊的控制规则（复合语句）



分支语句

- **复合语句**
  - clauses
  - header/suite

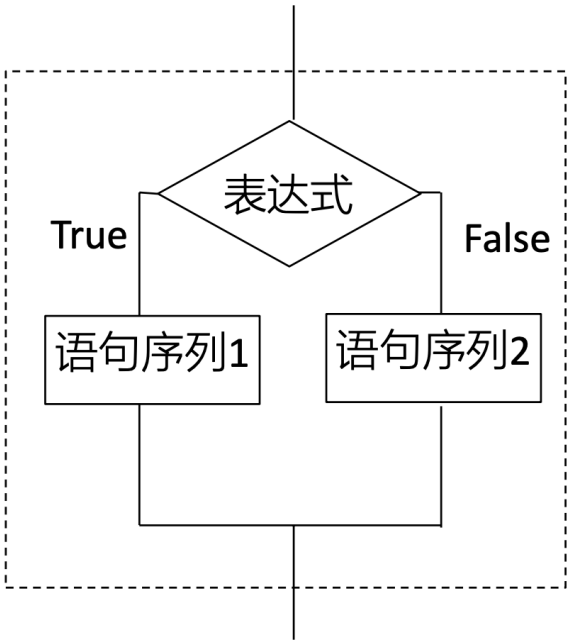
- **通过子句的header控制程序的执行顺序：**
  - 执行当前suite并完成当前语句

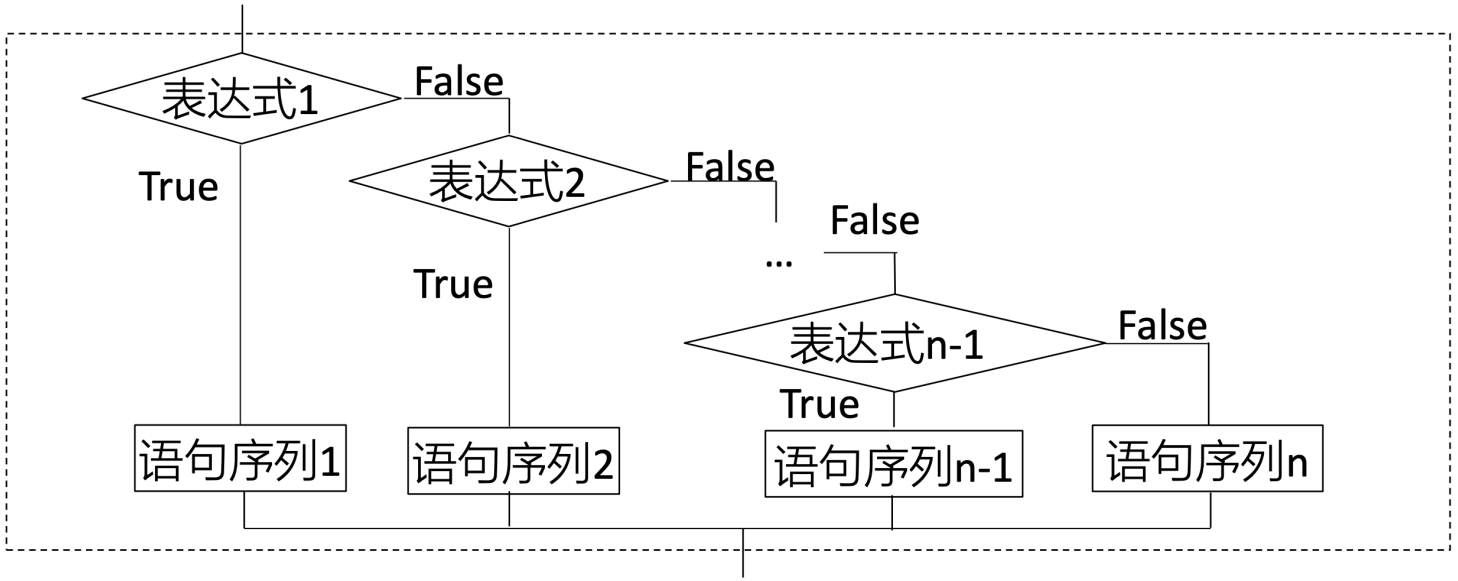
  或
  - 执行下一子句

```
if <expression>:
        <suite>
elif <expression>:
        <suite>
else:
        <suite>
```

多路分支

- 使用一个或多个header可以控制多种执行路径





```
>>> x = eval(input('Enter the final score:'))
Enter the final score:85
>>> if x >= 90:
...     res = "A"
... elif x >= 75:
...     res = "B"
... elif x >= 60:
...     res = "C"
... else:
...     res = "D"
...
>>> res
'B'
```

逻辑值和逻辑值的计算

- **逻辑值**
  - false values: 0, None, False
  - true values: rest
- **关系运算符、成员运算符的运算结果是逻辑值**
  - > < >= <= == != is
- **逻辑运算符可以对逻辑值进行运算**
  - and or not
  - 短路求值 (x < 2) and (x ** 100 > 1000)

条件表达式(conditional expressions)

- **特殊的表达式语句，可以看做一种三元运算符**

<expression> if <expression> else <expression>

– X if condition else Y #如果condition为真则值为X，否则为Y

```
>>> x, y = 0, 20
>>> t = x if x >= y else y
>>> t
20
```

```
c++:
condition ? X : Y;
```

条件表达式(conditional expressions)

- **expression可以也是条件表达式**
  - X if condition1 else Y if condition2 else Z
  - 相当于： X if condition1 else（Y if condition2 else Z）
- **可用于普通语句中，多用于列表解析和lambda表达式等**

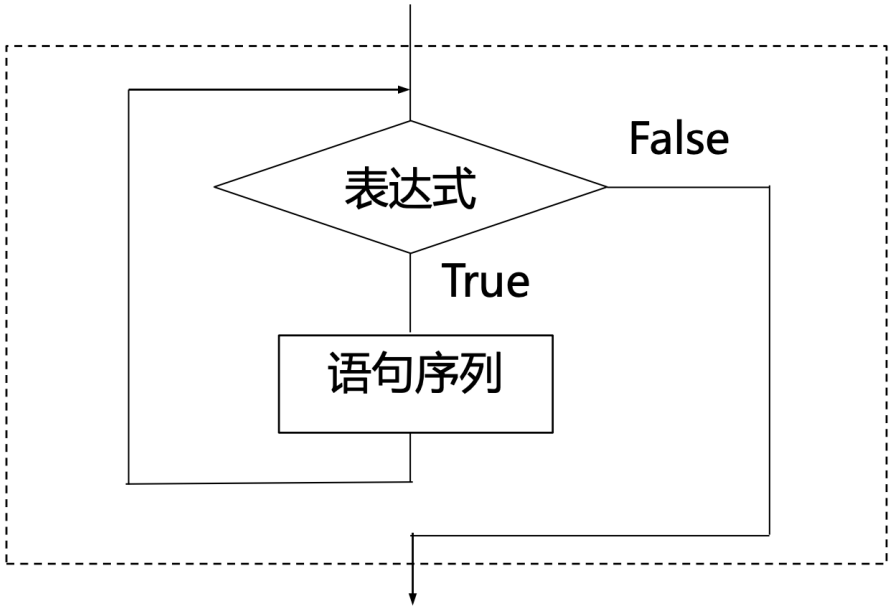
```
>>> ["Even" if i%2==0 else "Odd" for i in range(6)]
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']
```

while循环

- **复合语句**
  - 仅有一个子句
- **通过header控制**
  - 执行当前suite并回到header

  或
  - 完成当前语句

```
while <expression>:
        <suite>
```



- **注意检查循环的初始化和终止条件**
  - 计算斐波那契数列 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55…

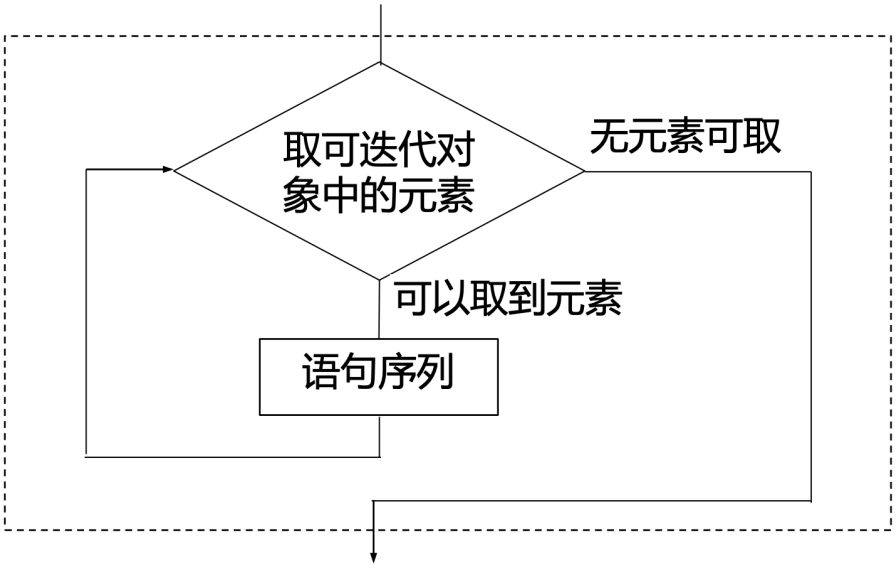
```
>>> n = eval(input("计算第几个斐波那契数（n > 1）？"))
>>> pred, curr = 0, 1   # Fibonacci numbers 1 and 2
>>> index = 2           # Which Fib number is curr?
>>> while index < n:
...     pred, curr = curr, pred + curr
...     index = index + 1
...
```

循环控制语句

- **break**
  - 终止当前循环语句执行（转而执行循环之后的语句）
- **continue**
  - 终止当前子句执行（转而执行循环的header）

for循环

- **遍历可迭代对象**
  - str, list, tuple, dict, file, range, map等




```
for <var> in <iterable>:
    <suite>
```

可迭代对象和迭代器对象

- **可迭代对象**
  - 通过iter() 函数返回一个迭代器
- **迭代器对象**
  - 指向可迭代对象中某个位置
  - 通过next()函数访问其下一个位置的元素
  - 直到无更多元素结束

```
>>> mylist = [1 , 2, 3]
>>> x = iter(mylist)
>>> x
<list_iterator object at
0x7fce9e757950>
>>> next(x)
1
>>> next(x)
2
>>> next(x)
3
>>> next(x)
Traceback (most recent call
last):
  File "<stdin>", line 1, in
<module>
StopIteration
>>>
```

可迭代对象和迭代器对象

- **可迭代对象**
  - 通过iter() 函数返回一个迭代器
- **迭代器对象**
  - 指向可迭代对象中某个位置
  - 通过next()函数访问其下一个位置的元素
  - 直到无更多元素结束

```
>>> mylist = [1 , 2, 3]
>>> tuple(mylist)
(1, 2, 3)
>>> tuple(mylist)
(1, 2, 3)
>>>
```

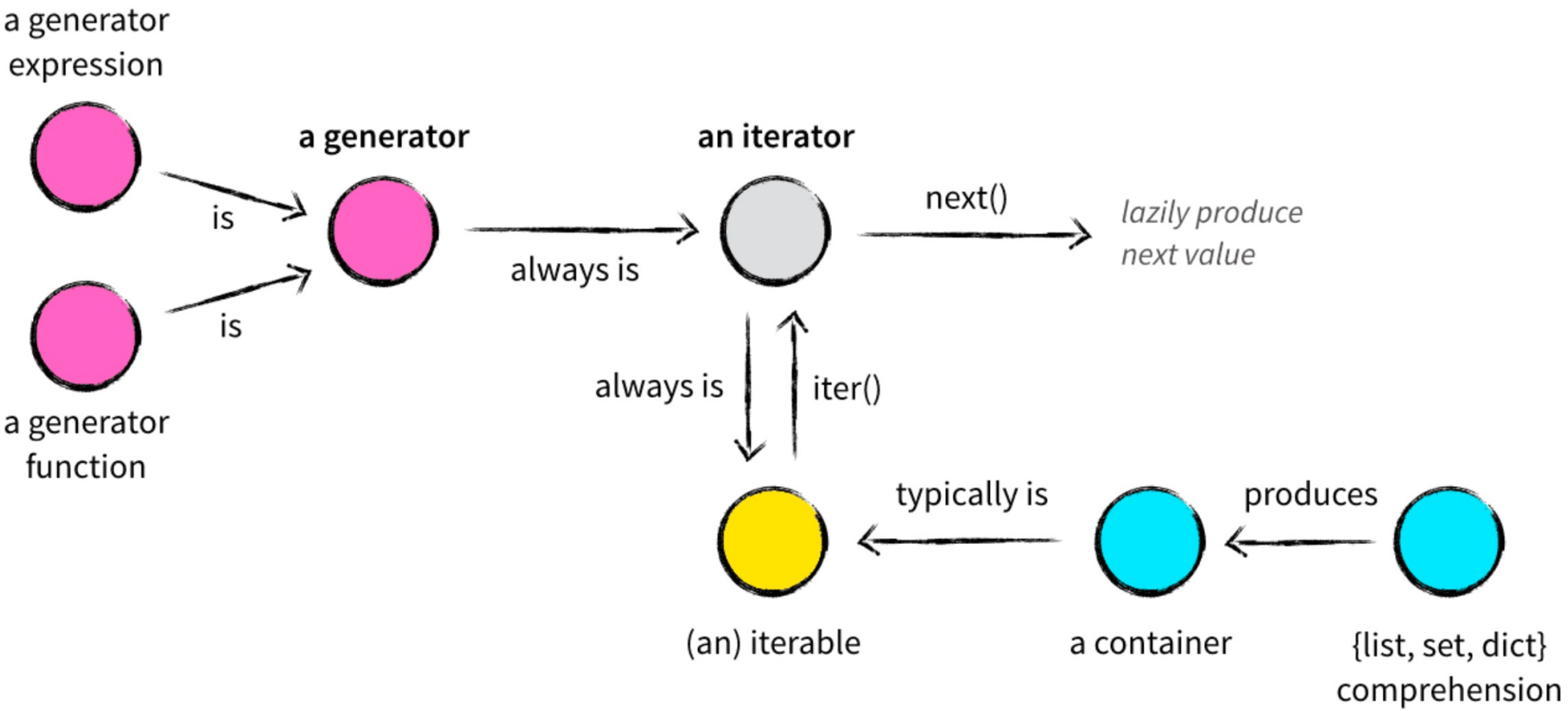
```
>>> x = iter(mylist)
>>> tuple(x)
(1, 2, 3)
>>> tuple(x)
()
```

*生成器generator

- **生成器对象**
  - 可以**惰性生成**需要使用的元素(lazy evaluate)，有利于大规模数据操作
- **可以通过生成器表达式生成**
  - 类似列表解析的语法
  - 使用 ( )，而不是[ ]
- **生成器函数**
  - 包含yield的函数

```
>>> gen = (i**2 for i in range(5))
>>> gen
<generator object <genexpr> at
0x7fce9e20b050>
>>> tuple(gen)
(0, 1, 4, 9, 16)
>>> tuple(gen)
()
```

请感兴趣的同学自行了解：
https://wiki.python.org/moin/Generators


a generator expression
a generator function
is
is
a generator
always is
an iterator
next()
lazily produce next value
always is
iter()
typically is
produces
(an) iterable
a container
{list, set, dict} comprehension

流程控制示例

- **示例：求一元人民币换成一分、 两分和五分的所有兑换方案数**
  - 如何找到所有的可能?

```
>>> count = 0
>>> for i in range(21):
...     for j in range(51):
...             if ( 100 - 5 * i - 2 * j ) >=0:
...                     count += 1
...
>>> count
541
```

  - 为何上述计数过程中没有重复?
  - 如果先搜索一分的使用，代码需要如何变化?

南京大学
NANJING UNIVERSITY

通过控制台进行数据交互

从控制台输入数据

- **input函数，返回值为str类型**

```
input(prompt=None, /)
    Read a string from standard input.  The trailing
newline is stripped.

    The prompt string, if given, is printed to standard
output without a trailing newline before reading input.

    If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-
Z+Return), raise EOFError.
    On *nix systems, readline is used if available.
```

向控制台输出(sys.stdout)

```
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout,
flush=False)

    Prints the values to a stream, or to sys.stdout by
default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the
current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a
newline.
    flush: whether to forcibly flush the stream.
```

- 可以连续输出多个对象（tuple packing）

```
>>> print("3.14", "1.25", "6345", sep = ",\t", end = " end\n")
3.14, 1.25, 6345 end
```

- 配合字符串处理的 format函数，f-string，可以得到不同格式的输出结果

南京大学
NANJING UNIVERSITY

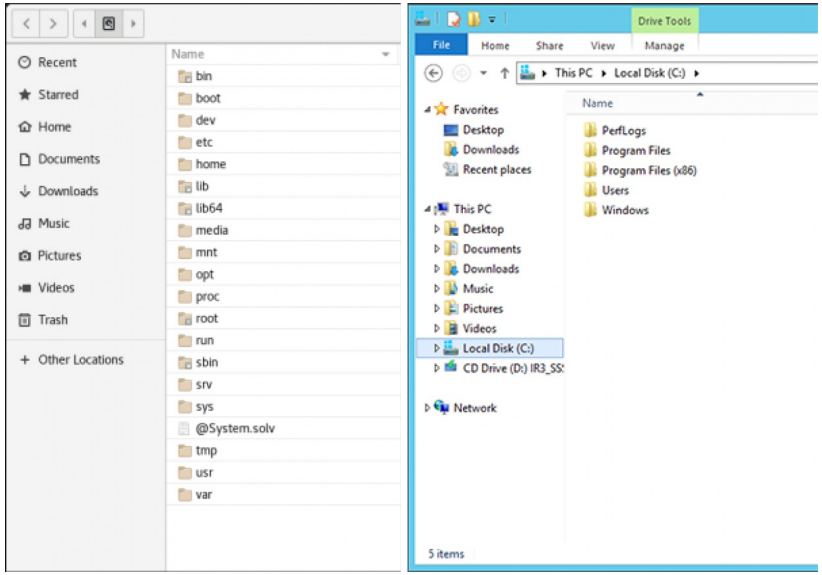
通过文件进行数据交互

面向文件的输入输出

- **需求：**
  - 程序运行结果有时需要永久性地保存起来，以供其它程序或本程序下一次运行时使用。
  - 程序运行所需要的数据也常常要从其它程序或本程序上一次运行所保存的数据中获得。
- **用于永久性保存数据的设备称为外部存储器（外存）**
  - 如：磁盘、磁带、光盘等。
- **在外存中保存数据的方式包括文件和数据库等。**

文件系统

- **操作系统管理磁盘文件的方式**
  - 管理文件的存储和组织、提供文件访问服务
  - FAT、FAT32…
  - NTFS
  - ext2、ext3…
  - HFS、APFS
- **网络文件系统**
- **分布式文件系统**
- **通过缓存等技术加速文件读写**



图片来自于https://www.redhat.com/sysadmin/linux-filesystem-windows

文件数据的存储方式

- **文本方式（text）**
  - 一般只包含可显示的字符和有限的几个控制字符（如：‘\r’、‘\n’、‘\t’等）的编码。
  - 例如，以文本方式存储整数1234567：
    - 依次把1、2、3、4、5、6、7的ASCII码（共7个字节）写入文件。
  - 一般用于存储具有“行”结构的文本数据。（可用记事本等软件打开察看）
- **二进制方式（binary）**
  - 包含任意的二进制字节数据（没有显式含义）。
  - 例如，以二进制方式存储整数1234567：
    - 把1234567按int型机内表示（4个字节：00 12 D6 87）按字节写入文件。
  - 一般用于存储无显式结构的数据。（格式一般只有写文件的人知道）

文件的读写方法

- **打开文件**
  - open()
- **读写操作**
  - 作为可迭代对象
  - read(), write(), seek()
  - readline(), readlines(), writelines()
- **关闭文件**
  - close()

```
>>> f = open("textfile.txt")
>>> f.read()
'This is a test file. \nThe code
is quite simple and
straightforward, but it builds the
full list in memory.\n'
>>> f.read()
''
>>> f.seek(0)
0
>>> [line for line in f]
['This is a test file. \n', 'The
code is quite simple and
straightforward, but it builds the
full list in memory.\n']
>>> f.close(0)
```

上下文管理器和with语句

- **上下文管理器（context manager）**
  - 自动负责对象的执行准备和收尾工作
  - 如：文件的打开和关闭等

```
with <expression> as <var>:
        <suite>
```

```
>>> with open("textfile.txt") as f:
...     [line.strip() for line in f]
...
['This is a test file.', 'The code is quite
simple and straightforward, but it builds the
full list in memory.']
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

部分涉及资源和异常处理的内容，后续再行介绍

https://docs.python.org/3/reference/datamodel.html#context-managers

其他文件操作相关方法

- **import os**
- **目录操作**
  - os.listdir() os.mkdir() os.chdir()
- **执行系统控制台命令**
  - os.system()

```
>>> import os
>>> files = os.listdir()
>>> for f in files:
...     print(f)
...
speedTest.py
textfile.txt

>>> files = os.mkdir("testDIR")
>>> files = os.mkdir("testDIR2")
>>> files = os.listdir()
>>> for f in files:
...     print(f)
...
testDIR2
testDIR
speedTest.py
textfile.txt
```

回顾

- **Python的程序流程控制**
  - 顺序、分支、循环
- **Python的数据交互方法**
  - 控制台、文件

- **部分可能有用的信息**
  - os包
  - with语句