```
>>> import math
>>> print('The value of pi is approximately %5.3f.' % math.pi)
The value of pi is approximately 3.142.
```

可在 old-string-formatting 部分找到更多信息。

7.2 读写文件

`open()` 返回一个*file object*，最常用的有两个参数：`open(filename, mode)`。

```
>>> f = open('workfile', 'w')
```

第一个参数是包含文件名的字符串。第二个参数是另一个字符串，其中包含一些描述文件使用方式的字符。*mode* 可以是 `'r'`，表示文件只能读取，`'w'` 表示只能写入（已存在的同名文件会被删除），还有 `'a'` 表示打开文件以追加内容；任何写入的数据会自动添加到文件的末尾。`'r+'` 表示打开文件进行读写。*mode* 参数是可选的；省略时默认为 `'r'`。

通常文件是以 *text mode* 打开的，这意味着从文件中读取或写入字符串时，都会以指定的编码方式进行编码。如果未指定编码格式，默认值与平台相关 (参见 `open()`)。在 mode 中追加的 `'b'` 则以 *binary mode* 打开文件：现在数据是以字节对象的形式进行读写的。这个模式应该用于所有不包含文本的文件。

在文本模式下读取时，默认会把平台特定的行结束符 (Unix 上的 `\n`, Windows 上的 `\r\n`) 转换为 `\n`。在文本模式下写入时，默认会把出现的 `\n` 转换回平台特定的结束符。这样在幕后修改文件数据对文本文件来说没有问题，但是会破坏二进制数据例如 `JPEG` 或 `EXE` 文件中的数据。请一定要注意在读写此类文件时应使用二进制模式。

在处理文件对象时，最好使用 `with` 关键字。优点是当子句体结束后文件会正确关闭，即使在某个时刻引发了异常。而且使用 `with` 相比等效的 `try-finally` 代码块要简短得多:

```
>>> with open('workfile') as f:
...     read_data = f.read()
>>> f.closed
True
```

如果你没有使用 `with` 关键字，那么你应该调用 `f.close()` 来关闭文件并立即释放它使用的所有系统资源。如果你没有显式地关闭文件，Python 的垃圾回收器最终将销毁该对象并为你关闭打开的文件，但这个文件可能会保持打开状态一段时间。另外一个风险是不同的 Python 实现会在不同的时间进行清理。

通过 `with` 语句或者调用 `f.close()` 关闭文件对象后，尝试使用该文件对象将自动失败。:

```
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

7.2.1 文件对象的方法

本节中剩下的例子将假定你已创建名为 `f` 的文件对象。

要读取文件内容，请调用 `f.read(size)`，它会读取一些数据并将其作为字符串（在文本模式下）或字节对象（在二进制模式下）返回。*size* 是一个可选的数字参数。当 *size* 被省略或者为负的时候，将读取并返回文件的整个内容；如果文件的大小是机器内存的两倍，那么就可能出现问题。否则，最多读取并返回 *size* 字节的内容，如果已到达文件末尾，`f.read()` 将返回一个空字符串 (`''`)。

```
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

`f.readline()` 从文件中读取一行；换行符（`\n`）留在字符串的末尾，如果文件不以换行符结尾，则在文件的最后一行省略。这使得返回值明确无误；如果 `f.readline()` 返回一个空的字符串，则表示已经到达了文件末尾，而空行使用 `'\n'` 表示，该字符串只包含一个换行符。：

```
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

要从文件中读取行，你可以循环遍历文件对象。这是内存高效，快速的，并简化代码:

```
>>> for line in f:
...     print(line, end='')
...
This is the first line of the file.
Second line of the file
```

如果你想以列表的形式读取文件中的所有行，你也可以使用 `list(f)` 或 `f.readlines()`。

`f.write(string)` 会把 *string* 的内容写入到文件中，并返回写入的字符数。：

```
>>> f.write('This is a test\n')
15
```

在写入其他类型的对象之前，需要先把它们转化为字符串（在文本模式下）或者字节对象（在二进制模式下）：

```
>>> value = ('the answer', 42)
>>> s = str(value)  # convert the tuple to string
>>> f.write(s)
18
```

`f.tell()` 返回一个整数，给出文件对象在文件中的当前位置，表示为二进制模式下时从文件开始的字节数，以及文本模式下的不透明数字。

要改变文件对象的位置，请使用 `f.seek(offset, from_what)`。通过向参考点添加 *offset* 来计算位置；参考点由 *from_what* 参数指定。*from_what** 值为 *0* 时，表示从文件开头开始，*1* 表示从当前位置，*2* 表示把文件末尾作为参考点。**from_what* 可以省略，默认为 0，即使用文件开头作为参考点。：

```
>>> f = open('workfile', 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)      # Go to the 6th byte in the file
5
>>> f.read(1)
b'5'
>>> f.seek(-3, 2)  # Go to the 3rd byte before the end
13
>>> f.read(1)
b'd'
```

在文本文件（那些在模式字符串中没有 b 的打开的文件）中，只允许相对于文件开头搜索（使用 seek(0, 2) 搜索到文件末尾是个例外）并且唯一有效的 *offset* 值是那些能从 f.tell() 中返回的或者是零。其他 *offset* 值都会产生未定义的行为。

文件对象有一些额外的方法，例如 isatty() 和 truncate()，它们使用频率较低；有关文件对象的完整指南请参阅库参考。

7.2.2 使用 json 保存结构化数据

字符串可以很轻松地写入文件并从文件中读取出来。数字可能会费点劲，因为 read() 方法只能返回字符串，这些字符串必须传递给类似 int() 的函数，它会接受类似 '123' 这样的字符串并返回其数字值 123。当你想保存诸如嵌套列表和字典这样更复杂的数据类型时，手动解析和序列化会变得复杂。

Python 允许你使用称为 JSON (JavaScript Object Notation) 的流行数据交换格式，而不是让用户不断的编写和调试代码以将复杂的数据类型保存到文件中。名为 json 的标准模块可以采用 Python 数据层次结构，并将它们转化为字符串表示形式；这个过程称为 *serializing*。从字符串表示中重建数据称为 *deserializing*。在序列化和反序列化之间，表示对象的字符串可能已存储在文件或数据中，或通过网络连接发送到某个远程机器。

注解： JSON 格式通常被现代应用程序用于允许数据交换。许多程序员已经熟悉它，这使其成为互操作性的良好选择。

如果你有一个对象 x，你可以用一行简单的代码来查看它的 JSON 字符串表示:

```
>>> import json
>>> json.dumps([1, 'simple', 'list'])
'[1, "simple", "list"]'
```

dumps() 函数的另一个变体叫做 dump()，它只是将对象序列化为 *text file*。因此，如果 f 是一个 *text file* 对象，我们可以这样做:

```
json.dump(x, f)
```

要再次解码对象，如果 f 是一个打开的以供阅读的 *text file* 对象:

```
x = json.load(f)
```

这种简单的序列化技术可以处理列表和字典，但是在 JSON 中序列化任意类的实例需要额外的努力。json 模块的参考包含对此的解释。

参见:

pickle - 封存模块

与 *JSON* 不同，*pickle* 是一种允许对任意复杂 Python 对象进行序列化的协议。因此，它为 Python 所特有，不能用于与其他语言编写的应用程序通信。默认情况下它也是不安全的：如果数据是由熟练的攻击者精心设计的，则反序列化来自不受信任来源的 pickle 数据可以执行任意代码。