目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

23 不简单的输入输出：IO 操作

更新时间：2019-10-16 09:40:20

> 耐心和恒心总会得到报酬的。
>
> ——爱因斯坦

文件读写

打开文件

在进行文件读写之前，有个重要的步骤——将文件打开，同时指定针对文件的读写模式，比如只读、只写、可读可写等等。只有先打开文件才能对文件进行读写操作。

打开文件使用内置函数 open() ：

```
f = open('文件路径', 读写模式)
```

如：

```
f = open('/Users/obsession/text', 'w')
```

其中， 读写模式 有以下常用选项：

- 'r' ：只读，若文件不存在则抛出 FileNotFoundError 异常
- 'w' ：只写，将覆盖所有原有内容，若文件不存在则创建文件
- 'a' ：只写，以追加的形式写入内容，若文件不存在则创建文件
- 'r+' ：可读可写，若文件不存在则抛出 FileNotFoundError 异常
- 'w+' ：可读可写，若文件不存在则创建文件
- 'a+' ：可读可写，写入时使用追加模式，若文件不存在则创建文件

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

`open()` 的返回值为 `file` 对象，也就是这里的变量 `f` 。利用这个对象，我们可以进行文件读写。

上述打开方式默认使用 UTF-8 编码，如果文件内容并非 UTF-8 编码，可以使用 `encoding` 参数指定编码格式，如 `f = open('/Users/obsession/text', 'w', encoding='gbk')` 。

写入文件

写入文件使用：

```
length = f.write('内容')
```

```
>>> f = open(’/Users/obsession/text’，‘w’)
>>> f.write(‘The quick brown fox jumps over the lazy dog’)
43
```

调用 `f.write()` 后将返回写入字符的长度。

读取文件

读取文件使用：

```
content = f.read()
```

```
>>> f = open(’/Users/obsession/text’，‘r’)
>>> f.read()
’The quick brown fox jumps over the lazy dog’
```

上例中将读取文件的所有内容。也可以指定要读取内容的字符长度：

```
>>> f = open(’/Users/obsession/text’，‘r’)
>>> f.read(30)
’The quick brown fox jumps over’
>>> f.read(30)
’ the lazy dog’
>>> f.read(30)
’’
```

此时将根据所指定的长度来读取内容。注意观察示例，每次调用 `f.read(30)` 时都是从上一次读取的结束位置开始，来读取新的内容，直至所有的内容被获取完，之后再调用 `f.read(30)` 只会得到空字符串 `''` 。

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
line = f.readline()
```

例如某文件内容为

> The quick brown fox
> jumps over
> the lazy dog

按行读取文件如下：

> >>> f = open(’/Users/obsession/text’, ‘r’)
> >>> **f.readline()**
> ’The quick brown fox\n’
> >>> **f.readline()**
> ’jumps over\n’
> >>> **f.readline()**
> ’the lazy dog’
> >>> **f.readline()**
> ’’

按行读取文件还可以一次性将所有行读出，然后放进列表里：

```
lines = f.readlines()
```

> >>> f = open(’/Users/obsession/text’, ‘r’)
> >>> **f.readlines()**
> [‘The quick brown fox\n’, ‘jumps over\n’, ‘the lazy dog’]

关闭文件

每次打开文件后，无论进行了多少读写操作，最终都一定要将文件关闭，因为打开文件会消耗相关系统资源（文件描述符），不使用时应及时释放。

关闭文件使用：

```
f.close()
```

还有一种方式能自动关闭打开的文件，那就是使用 with 语句：

```
with open('/Users/obsession/text', 'w') as f:
    f.write('The quick brown fox jumps over the lazy dog')
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

with 语句会在它的代码块执行完毕后，或代码块抛出异常的，自动关闭文件，为我们省却了 f.close() 步骤。

文件系统操作

文件系统操作需要使用内置的 os 模块。

- 创建目录

```
import os

os.mkdir('/Users/obsession/test_dir')
```

- 判断路径是否是一个目录

```
os.path.isdir('/Users/obsession/test_dir')
```

- 列举目录下的内容

```
os.listdir('/Users/obsession')
```

- 删除目录

```
os.rmdir('/Users/obsession/test_dir')
```

- 创建文件
  创建文件可以直接使用之前学过的 open()：

```
f = open('/Users/obsession/test_file', 'w')
f.close()
```

- 判断路径是否是一个文件

```
os.path.isfile('/Users/obsession/test_file')
```

- 删除文件

```
os.remove('/Users/obsession/test_file')
```

- 重命名文件

```
os.rename('/Users/obsession/test_file', 'test_file_02')
```

序列化和反序列化

程序运行时，产生的所有对象都位于内存之中。内存有个特点，那就是它是非持久的，如果程序运行结束或者计算机断电，占用的内存将被清空。

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
class Pair:
    def __init__(self, first, second):
        self.first = first
        self.second = second

pair = Pair(10, 20)
```

这就涉及到序列化和反序列化了。**序列化**是将内存中的对象转换为可被存储或可被传输的形式的过程。**反序列化**是将序列化后的内容恢复回内存中对象的过程。

pickle

Python 中内置的 pickle 模块用作序列化和反序列化。它的序列化结果是二进制形式。

序列化使用：

```
import pickle

some_bytes = pickle.dumps(对象)
```

```
>>> pair = Pair(10, 20)
>>> pickle.dumps(pair)
b'\x80\x03c__main__\nPair\nq\x00)\x81q\x01}q\x02(X\x05\x00\x00\x00firstq\x03K\nX\x06\x00\x00\x00secondq\x04K\x14ub.'
```

上面输出的乱码便是 pair 对象被序列化后的二进制。

对于刚才序列化后的结果，可以使用 pickle.loads() 将其反序列化回对象。如：

```
some_bytes = b'\x80\x03c__main__\nPair\nq\x00)\x81q\x01}q\x02(X\x05\x00\x00\x00firstq

pair = pickle.loads(some_bytes)
```

此 pair 对象可以像之前一样正常被使用：

```
>>> pair.first
10
>>> pair.second
20
```

也可以与 open() 相结合，将序列化的结果保存在文件中，此时使用 pickle.dump()（注意与之前的 pickle.dumps() 不同）：

```
with open('/Users/obsession/dump', 'wb') as f:
    pickle.dump(pair, f)
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
with open('/Users/obsession/dump', 'rb') as f:
    pair = pickle.load(f)
```

JSON

pickle 使用 Python 专用的序列化格式，序列化后的结果无法做到跨语言使用。另外其序列化结果是二进制，不适合阅读。

JSON 相对而言更加通用和流行，并且其结果为文本格式，更具可读性。

同样是刚才的 pair 对象，可以像这样将它序列化为 JSON 字符串：

```
import json

json_string = json.dumps(pair.__dict__)
```

```
>>> json_string
'{"first": 10, "second": 20}'
```

注意上面结果为字符串类型。另外这里使用了 pair.__dict__ 来获取包含所有 pair 属性的字典，因为类对象不能直接用于 json.dumps() 序列化，而字典可以。

或者使用 default 参数，向 json.dumps() 告知如何进行从对象到字典的转换，这样便可以不使用 __dict__ 属性。如下：

```
def pair_to_dict(pair):
    return {
        'first': pair.first,
        'second': pair.second,
    }

json_string = json.dumps(pair, default=pair_to_dict)
```

```
>>> json_string
'{"first": 10, "second": 20}'
```

从 JSON 反序列化为对象：

```
def dict_to_pair(d):
    return Pair(d['first'], d['second'])

pair = json.loads(json_string, object_hook=dict_to_pair)
```

上述反序列化过程中，json.loads() 首先会将 JSON 字符串反序列化为字典，然后使用 object_hook 参数进一步从字典转换出 pair 对象。

与 pickle 相似，json 也可以与 open() 结合使用，将序列化的结果保存在文件中：

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

或从文件中反序列化出对象：

```
with open('/Users/obsession/json', 'r') as f:
    pair = json.load(f, object_hook=dict_to_pair)
```

← 22 Python 的小招数：其它常用语言特性

24 让你的代码更灵活：进程和线程 →

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论

千学不如一看，千看不如一练