**Daily Report**

25th February, 2018

1. **断言**

需要依靠print来检查是否出错的地方，可以使用断言assert来代替，如：

assert n != 0, 'n is zero!'

assert的意思是，表达式n != 0应该是True，否则，根据程序运行的逻辑，后面的代码肯定会出错；如果断言失败，assert语句本身就会抛出AssertionError

但如果在代码中到处使用assert还是觉得繁琐，但在解释器中用（-O）参数关闭assert，具体方式为输入python -O 文件名

1. **Logging**

使用logging代替print，不必抛出错误，还能输出到文件

同时，使用logging可以将记录信息，分为debug，info，warning，error四个等级，并且最后统一输出某个级别信息

使用logging的另一个好处在于通过简单的配置，一条语句可以输出到不同的地方，比如console和文件

1. **Pdb**

启动调试器pdb，让程序以单步运行，随时可以检查运行状态

启动方式为 -m pdb

l：查看代码

n： 单步执行代码

p 变量名：查看变量

q：结束调试，退出程序

1. **pdb.set\_trace（）**

首先import pdb，再在可能出错的地方放一个pdb.set\_trace（），就可以设置一个断点，当程序运行到该处时，会自动暂停

p 变量名：查看变量

c：继续运行

1. **单元测试**

单元测试是用来对一个模块、一个函数或者一个类来进行正确性检验的测试工作

1. **文档测试**

Python内置的“文档测试”（doctest）模块可以直接提取注释中的代码并执行测试

1. **文件读写**

Python读文件使用内置函数open（），需要输入文件名和标示符，当标示符为‘r’时，表示读；若文件不存在，则会抛出一个IOError

使用read（）则可读取文件全部内容

使用close（）关闭文件，若不关闭，则文件对象会占用操作系统资源，并且同一时间能打开的文件数量也是有限的

Read（size）每次最多读取size个字节的内容

Readline（）每次可以读取一行的内容

Readlines（）一次读取所有内容并返回list

当读写文件时，都可能出错；一旦出错，最后将不会执行close（），则文件不能关闭，为了保证文件始终能够关闭，使用try...finally...来实现

但上述方法过于繁琐，可以使用with语句自动帮我们调用close（）函数，如：

with open('/path/to/file', 'r') as f:

print(f.read())

当标示符为‘r’时，都是用于读取文本文件，要读取二进制文件，比如图片、视频等等，用'rb'模式打开文件即可

要读取非UTF-8编码的文本文件，需要给open()函数传入encoding参数，如：

>>> f = open('/Users/michael/gbk.txt', 'r', encoding='gbk')

>>> f.read()

'测试'

遇到有些编码不规范的文件，你可能会遇到UnicodeDecodeError，因为在文本文件中可能夹杂了一些非法编码的字符。遇到这种情况，open()函数还接收一个errors参数，表示如果遇到编码错误后如何处理。最简单的方式是直接忽略：

>>> f = open('/Users/michael/gbk.txt', 'r', encoding='gbk', errors='ignore')

写文件时，传入标识符'w'或者'wb'表示写文本文件或写二进制文件。

使用open（）函数写入时，必须调用close（）关闭文件，不然可能使写入的数据丢失，缺少；也可以使用with语句来完成（这是因为with语句自动调用close（）函数），如：

with open('/Users/michael/test.txt', 'w') as f:

f.write('Hello, world!')

当标示符为‘w’写入文件时，如果文件已存在，会直接覆盖（相当于删除源文件再创建新文件），若想在原文件文末添加，则标示符为‘a’

1. **StringIO**

即为在内存中读写str

getvalue()方法用于获得写入后的str

首先需要创建一个StringIO，然后write各个需要输入的量，每次回返回一个字符量值，最后通过getvalue（）即可获得全部已输入的值

1. **BytesIO**

StringIO操作的只能是str，如果要操作二进制数据，就需要使用BytesIO

1. **序列化**

把变量从内存中变成可存储或传输的过程称之为序列化，在Python中叫pickling，在其他语言中也被称之为serialization，marshalling，flattening等等

序列化之后，就可以把序列化后的内容写入磁盘，或者通过网络传输到别的机器上

反过来，把变量内容从序列化的对象重新读到内存里称之为反序列化，即unpickling

Python提供了pickle模块来实现序列化

pickle.dumps()方法把任意对象序列化成一个bytes，然后，就可以把这个bytes写入文件。或者用另一个方法pickle.dump()直接把对象序列化后写入一个file-like Object：

>>> f = open('dump.txt', 'wb')

>>> pickle.dump(d, f)

>>> f.close()

当我们要把对象从磁盘读到内存时，可以先把内容读到一个bytes，然后用pickle.loads()方法反序列化出对象，也可以直接用pickle.load()方法从一个file-like Object中直接反序列化出对象

1. **JSON**

如果我们要在不同的编程语言之间传递对象，就必须把对象序列化为标准格式，比如XML，但更好的方法是序列化为JSON，因为JSON表示出来就是一个字符串，可以被所有语言读取，也可以方便地存储到磁盘或者通过网络传输。JSON不仅是标准格式，并且比XML更快，而且可以直接在Web页面中读取，非常方便

序列化方法：

>>> import json

>>> d = dict(name='Bob', age=20, score=88)

>>> json.dumps(d)

'{"age": 20, "score": 88, "name": "Bob"}'

反序列化方法：

>>> json\_str = '{"age": 20, "score": 88, "name": "Bob"}'

>>> json.loads(json\_str)

{'age': 20, 'score': 88, 'name': 'Bob'}

对于JSON，当只传入一个实例对象时，无法序列化；但是可以将可选参数default进行设置，把任意一个对象变成可序列化的JSON对象。

方法如：

def student2dict(std):

return {

'name': std.name,

'age': std.age,

'score': std.score

}

>>> print(json.dumps(s, default=student2dict))

{"age": 20, "name": "Bob", "score": 88}

或者简便方法为：

print(json.dumps(s, default=lambda obj: obj.\_\_dict\_\_))