Python 程序设计

Xia Tian
Email: xiat(at)ruc.edu.cn

Renmin University of China

May 24, 2016

整体内容

- 简介
- 数据类型
- 控制流
- 函数
- 模块
- 标准库
- 面向对象编程
- 异常、调试与测试
- 输入输出
- 应用 (Web，DB，etc.)

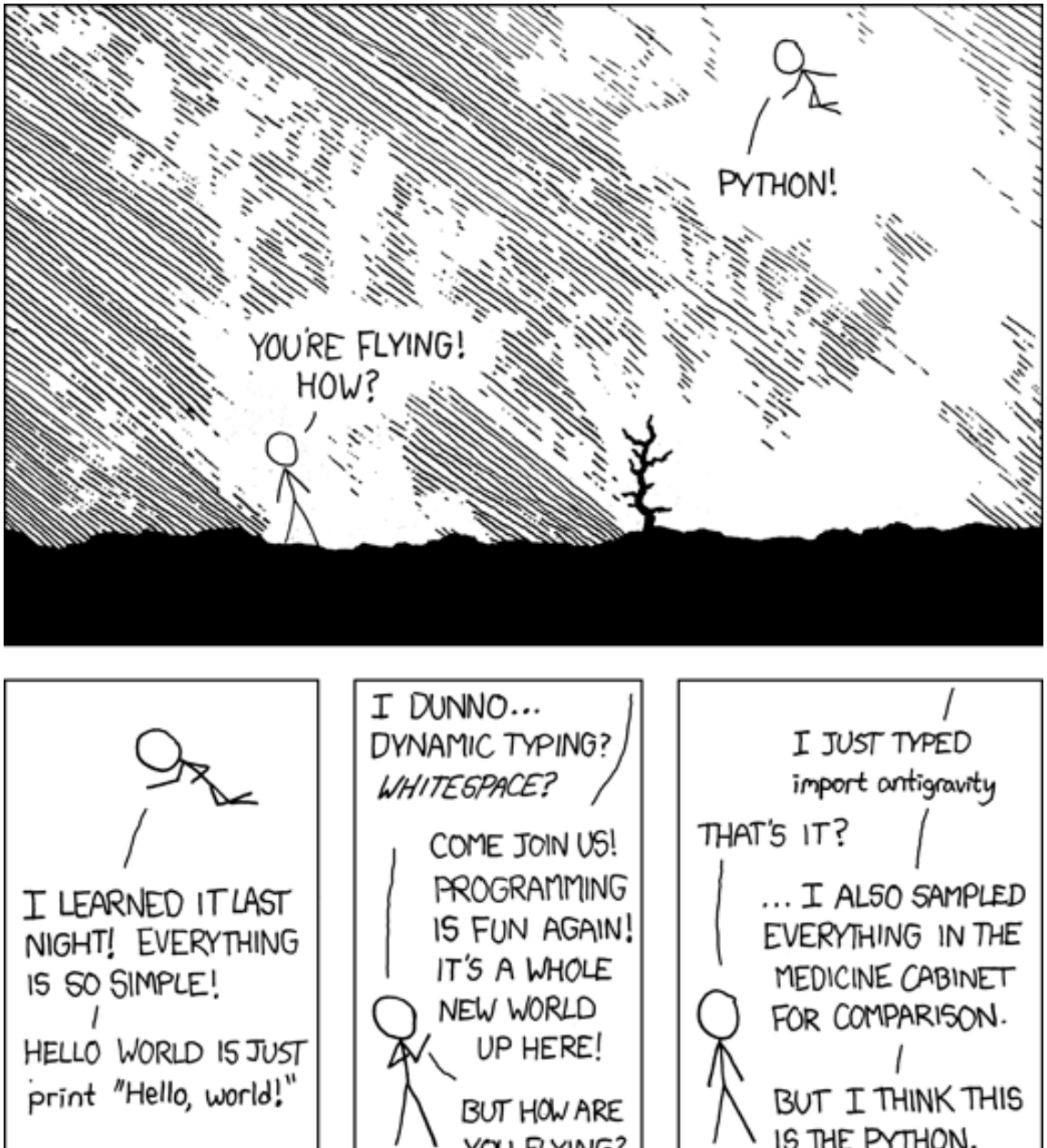
致谢

- 本课件的制作参考了部分图书和第三方网络资源，在此对原作者表达致谢和敬意，如有侵权需要从本课件中移除，请留言或联系 xiat(at)ruc.edu.cn。参考资料包括但不限于以下：
  - 廖雪峰, Python 教程
  - Introducing Python by Bill Lubanovic
  - Dive Into Python 3 by Mark Pilgrim
  - Magnus Lie Hetland, 司维等翻译，Python 基础教程 (2ed), 人民邮电出版社
  - NumPy Cookbook
  - Python for Data Analysis

CH1 简介

- 1.1 什么是 Python
- 1.2 Python 可以做什么
- 1.3 如何安装 Python
- 1.4 Python 开发环境
- 1.5 如何运行 Python 程序
- 1.6 若干例子

You're flying!



用 Python，飞一般的感觉！

- Friend：“你在飞！怎么做到的?”
- Cueball：“Python！我昨晚刚刚学会了 Python。一切都变得如此简单！写一个 Hello World 程序只要一行代码 print ”Hello World!” 就搞定了！”
- Friend：“什么情况？呃……动态类型？泛空格符?”
- Cueball：“来加入我们吧，有了 Python，编程再次变得有趣。这是一个全新的世界！”
- Friend：“但是你到底是怎么飞在天上的?”
- Cueball：“我只是输入了“import antigravity”命令而已。”
- Friend：“就这样?”
- Cueball：“我还把药柜中的药嗑了个遍……但我觉得还是 Python 的原因。”

1.1 什么是 Python

- Python 是一种既简单又强大的编程语言
- 注重如何解决问题，而不是编程语言的语法和结构
- 拥有高效的高级数据结构，简单有效地实现面向对象编程
- 语法简洁、动态解释、适用于快速应用开发和脚本编程
- 在数据科学中大有用武之地

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.

Python 之禅 by Tim Peters

优美胜于丑陋

Python 以编写优美的代码为目标

明了胜于晦涩

优美的代码应当是明了的，命名规范，风格相似

简洁胜于复杂

优美的代码应当是简洁的，不要有复杂的内部实现

复杂胜于凌乱

如果复杂不可避免，那代码间也不能有难懂的关系，要保持接口简洁

扁平胜于嵌套

优美的代码应当是扁平的，不能有太多的嵌套

间隔胜于紧凑

优美的代码有适当的间隔，不要奢望一行代码解决问题

可读性很重要

优美的代码是可读的

即便假借特例的实用性之名，也不可违背这些规则

编程语言排名 @Feb, 2016

- Java
- C
- C++
- C#
- Python ⋆
- PHP
- Visual Basic
- Perl
- JavaScript
- Delphi/Object Pascal

From: `http://www.tiobe.com/tiobe_index?page=index`

1.2 Python 可以做什么

- Almost anything
  - From system management, security, web, to data mining, machine learning ...
- 数据科学界华山论剑：R 与 Python 巅峰对决
  `http://chuansong.me/n/1458679`
- 为什么很多人喜欢 Python?
  `https://www.zhihu.com/question/28676107`

一段简单的 Python 代码

```python
#Python 3: Fibonacci series up to n
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=', ')
        a, b = b, a+b
    print()
fib(100)
```

Result: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,

1.3 如何安装 Python

- 在线直接编写运行
  - Host, run, and code Python in the cloud!
  - https://www.python.org/
- python shell 及增强版本
- 编辑器编辑代码文件，利用 python 运行

Python shell

```
1 $python3
2 Python 3.4.3+ (default, Oct 14 2015, 16:03:50)
3 [GCC 5.2.1 20151010] on linux
4 Type "help", "copyright", "credits" or "license" for more
      information.
5 >>>
```

课堂练习

- Windows 环境下 Python 程序的安装和运行测试
  - https://www.python.org/downloads/
- 注意：本课程后续讲解均以 Linux(Ubuntu) 作为操作系统环境

安装 EasyInstall

- wget␣http://peak.telecommunity.com/dist/ez_setup.py
- python␣ez_setup.py␣setuptools
- ez_install␣pip
- ez_install␣pip3

Python2 vs Python3

- Python3 是 Python2 的重大变更版本
- 有许多正在运行的程序使用了 Python2
- 系统中可以同时安装 python2 和 python3
- 与 Python2 相比，Python3 相关的工具多以 3 结尾，如 ipython3
- 本课程讲解以 Python3 为主，穿插 Python2 的语法

1.4 Python 开发环境

- Python shell
  - 自带的命令解释器
  - ipython
  - jupyter notebook
  - bpython
- Editor
  - pycharm
  - Sublime text
  - Emacs
  - Vim

ipython

- sudo␣apt-get␣install␣ipython3
- 展示

bpython

- sudo␣apt-get␣instal␣bpython3
- 展示

善用 help

利用 Python 自带的 help 函数获取帮助

```
>>> help(max)
```

```
>>> help()
```

增强命令行

- tmux
- zsh

1.5 如何运行 Python 程序

- 直接在 Python Shell 中输入，解释执行
- 保存到文件中，以.py 结尾，通过 python 运行
  - 如把前面的斐波那契数列的代码保存到文本文件中，并把文件名命名为 fib.py
  - python3 fib.py

1.5 如何运行 Python 程序

- 直接在 Python Shell 中输入，解释执行
- 保存到文件中，以.py 结尾，通过 python 运行
  - 如把前面的斐波那契数列的代码保存到文本文件中，并把文件名命名为 fib.py
  - python3␣fib.py
- 可执行的 Python 脚本
  - 在 Python 脚本文件的开头加上一行声明
    - #!/usr/bin/python3
    - chmod u+x filename.py

代码清单：fib.py

```
1  #!/usr/bin/python3
2
3
4  def fib(n):
5      '''
6      Fibonacci series up to n
7      '''
8      a, b = 0, 1
9      while a < n:
10         print(a, end=', ')
11         a, b = b, a+b
12     print()
13
14 if __name__=='__main__':
15     fib(100)
```

1.6 一个完整的 Python 程序 I

```
1  '''Convert file sizes to human-readable form.
2
3  Available functions:
4  approximate_size(size, kb_is_1024_bytes)
5      takes a file size and returns a human-readable string
6
7  Examples:
8  >>> approximate_size(1024)
9  '1.0 KiB'
10 >>> approximate_size(1000, False)
11 '1.0 KB'
12 '''
13
```

1.6 一个完整的 Python 程序 II

```
14 SUFFIXES = {1000: ['KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB',
       'YB'],
15             1024: ['KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB',
       'YiB']}
16
17
18 def approximate_size(size, kb_is_1024_bytes=True):
19     '''Convert a file size to human-readable form.
20
21     Keyword arguments:
22     size -- file size in bytes
23     kb_is_1024_bytes -- if True (default), use multiples of 1024
24                         if False, use multiples of 1000
25
```

1.6 一个完整的 Python 程序 III

```
26      Returns: string
27
28      '''
29      if size < 0:
30          raise ValueError('number must be non-negative')
31
32      multiple = 1024 if kb_is_1024_bytes else 1000
33      for suffix in SUFFIXES[multiple]:
34          size /= multiple
35          if size < multiple:
36              return '{0:.1f} {1}'.format(size, suffix)
37
38      raise ValueError('number too large')
39
```

1.6 一个完整的 Python 程序 IV

```
40 if __name__ == '__main__':
41     print(approximate_size(1000000000000, False))
42     print(approximate_size(1000000000000))
```

Code From: “Dive into Python 3”

例子解释

- 如何声明函数：def
- 可选的和命名的参数
- 文档字符串
- 代码缩进
- 异常
- 大小写

Python 基本用法预览

- 把 Python 当做计算器
- 字符串的表示
- 字符串切片

把 Python 当做计算器

- Python 解释器可以当做简单的计算器，输入表达式，即可对表达式求值
- 练习
  - 打开 Python 解释器，输入以下表达式求值
  - 2+2
  - 8/5 # 结果是 1.6？还是 1？
  - (3+5)/2
  - 8//5

字符串的表示

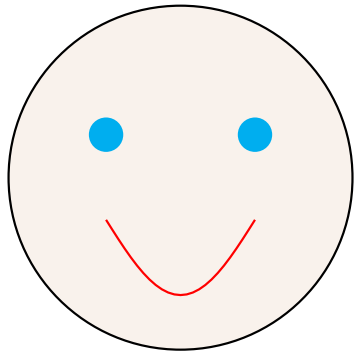
- 单引号
- 双引号
- 三引号

```
1 msg = 'hello "Python"'
2 print(msg)
3 msg = "hello 'Python'"
4 print(msg)
5 msg = '''
6     hello
7     world!
8     '''
9 print(msg)
```

字符串切片

```
>>> msg = ' 中国人民大学信息资源管理学院'

>>> msg[0]
' 中'

>>> msg[-8:-1]
' 信息资源管理学'

>>> ms g[-8:]
??

>>> msg[:6]

>>> msg[:]
```

—END—



CH8 异常、调试与测试

- 异常
  - Python 的异常处理使用方法
  - Python 的异常继承关系
  - 利用 raise 抛出异常
- 调试
  - print 调试法
  - logging 调试法
  - assert
  - pdb
- 单元测试

异常处理

- 程序在编写过程中，有大量情况需要考虑
  - 除数是否为 0
  - 打开文件时，需要判断文件是否存在，有无权限
  - ...
- 异常可以简化这一处理过程
- Python 的错误处理机制
  - try...except...finally...

try

```
1  try:
2      print('try...')
3      r = 10 / 0
4      print('result:', r)
5  except ZeroDivisionError as e:
6      print('except:', e)
7  finally:
8      print('finally...')
9  print('END')
10
```

try...
except: division by zero
finally...
END

try

```
1 try:
2     print('try...')
3     r = 10 / 2
4     print('result:', r)
5 except ZeroDivisionError as e:
6     print('except:', e)
7 finally:
8     print('finally...')
9 print('END')
10
```

try...
result: 5.0
finally...
END

Python 异常处理的规则

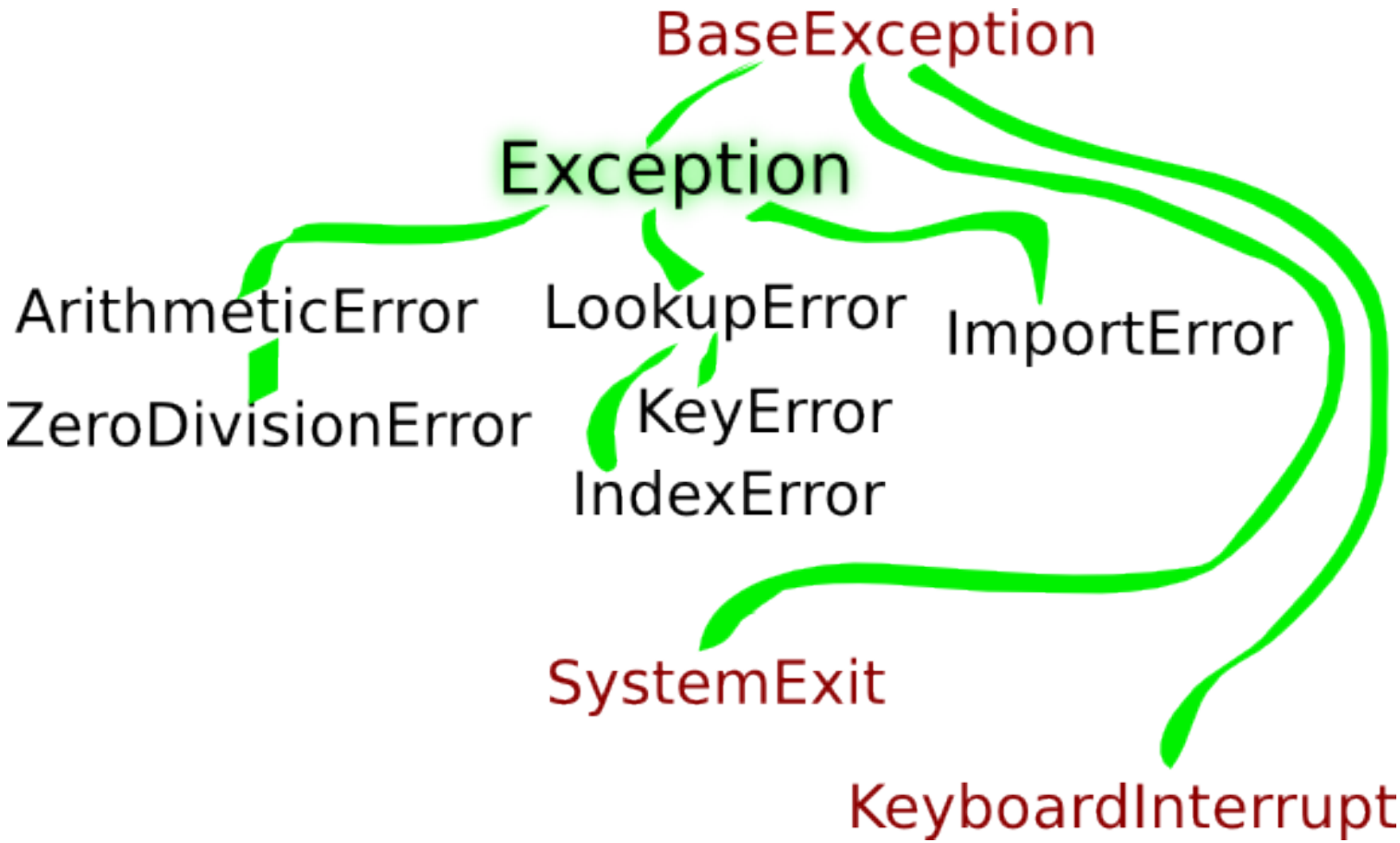
- 遇到第一个满足条件的异常，执行该异常下的语句，忽略后续的其它异常
- finally 永远会被执行

Python 异常的继承关系

- Python 的错误也是 class，都继承自 BaseException
  - 在使用 except 时需要注意的是，它不但捕获该类型的错误，还把其子类也“一网打尽”。

```
1 lst = [x for x in range(10)]
2 try:
3     n = lst[15]
4 except LookupError as e:
5     print('LookupError ', e)
6 except IndexError as e:
7     print('IndexError ', e)
8
```

Python 异常继承关系示例



https://docs.python.org/3/library/exceptions.html#exception-hierarchy

抛出异常

- 可以用 raise 语句来引发一个异常。异常/错误对象必须有一个名字，且它们应是 Error 或 Exception 类的子类。

raise 示例 I

```
1  class Student(object):
2      @property
3      def score(self):
4          return self._score
5
6      @score.setter
7      def score(self, value):
8          if not isinstance(value, int):
9              raise ValueError('score必须是整数!')
10         if value < 0 or value > 100:
11             raise ValueException('score必须在0到100之间')
12         self._score = value
```

raise 示例 II

```
1 s = Student()
2 s.score = 60
3 print(s.score)
4 s.score = 999 # Error!
```

异常可以自定义 I

- 例如：把以上的 score 判断条件不满足时，抛出的异常更改为自定义异常

```
1  class ScoreException(Exception):
2      def ___init___(self, msg):
3          self.msg = msg
4
5      def ___str___(self):
6          return 'ScoreException: ' + repr(self.msg)
7
8  class Student(object):
9      @property
10     def score(self):
11         return self._score
12
```

异常可以自定义 II

```
13      @score.setter
14      def score(self, value):
15          if not isinstance(value, int):
16              raise ScoreException('score必须是整数!')
17          if value < 0 or value > 100:
18              raise ScoreException('score必须在0到100之间')
19          self._score = value
20
21  try:
22      s = Student()
23      s.score = 60
24      print(s.score)
25      s.score = 999
26  except ScoreException as e:
```

异常可以自定义 III

```
27      print(e)
28
```

异常总结

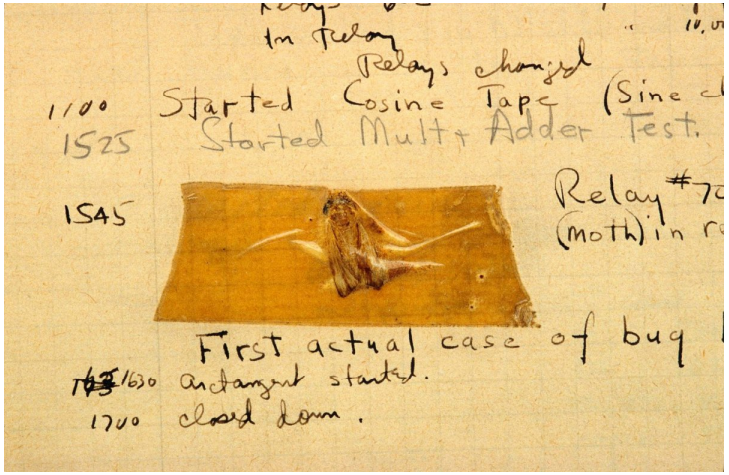
- Python 内置的 try...except...finally 用来处理错误十分方便
  - 出错时，会分析错误信息并定位错误发生的代码位置更为关键
- 程序也可以主动抛出错误，让调用者来处理相应的错误
  - 应在文档中写清楚可能会抛出哪些错误，以及错误产生的原因

调试

Bug and Debug

格蕾丝 · 赫柏 (Grace Murray Hopper)

赫柏是一位为美国海军工作的电脑专家。1945 年的一天，赫柏对 Harvard Mark II 设置好 17000 个继电器进行编程后，技术人员在进行整机运行时，它突然停止了工作。于是他们爬上去找原因，发现这台巨大的计算机内部一组继电器的触点之间有一只飞蛾，这显然是由于飞蛾受光和热的吸引，飞到了触点上，然后被高电压击死。所以在报告中，赫柏用胶条贴上飞蛾，并把“bug”来表示”一个在电脑程序里的错误”。



- 程序一次编写就能成功运行的概率很小，通常会有各种各样的错误需要调试，因此，掌握错误的调试方法非常重要。

常用的调试方法

- 观察出错的提示信息
- 利用 print 函数输出信息，观察输出结果和预期结果是否一致
- 通过日志 logging 替代 print
- 利用 assert 断言
- pdb

利用 print 进行调试

```
1 books = ['Python', 'XML', 'Information Retrieval']
2 for book in books:
3     print(book)
4
5 print('Run here!')
6 if book.price > 50:
7     print('High price.')
8
```

以上代码会抛出异常信息：

```
Traceback (most recent call last):
  File "bug.py", line 5, in <module>
    if book.price > 50:
AttributeError: 'str' object has no attribute 'price'
```

如果我们觉得前三行代码不太可能出问题，而问题很可能在后面，那

利用 logging 进行调试

print() 的结果默认输出到控制台上，不够灵活和方便，可以使用 logging 进行控制

```
1  import logging
2  logging.basicConfig(level=logging.INFO)
3
4  books = ['Python', 'XML', 'Information Retrieval']
5  for book in books:
6      logging.info(book)
7
8  logging.debug('Run here!')
9  if book.price > 50:
10     logging.warn('High price.')
11
```

assert 调试

assert 断言用于判断给定的逻辑表达式是否成立，如果不成立，就会抛出 AssertionError 异常

```
1 books = ['Python', 'XML', 'Information Retrieval']
2 assert len(books) == 3
```

启动 Python 解释器时可以用-O 参数来关闭 assert，此时的 assert 语句可看作是 pass

pdb

Python 的调试器，可以单步运行 Python 脚本，查看当前运行的代码，查看变量值

bug.py:

```
1 books = ['Python', 'XML', 'Information Retrieval']
2 for book in books:
3     print(book)
4
5 if book.price > 50:
6     print('High price.')
7
```

运行：pdb bug.py
或者：python -m pdb bug.py

pdb

- n: 执行下一条语句
- p xxx: 查看变量 xxx 的当前值
- l: 列出

pdb.set_trace()

在可能出错的地方放置 pdb.set_trace()，程序运行到该条语句时，会暂停并进入 pdb 调试环境，此时，可以用 p 指令查看变量，或者用 c 继续运行

bug2.py:

```
1 import pdb
2 books = ['Python', 'XML', 'Information Retrieval']
3 for book in books:
4     print(book)
5
6 pdb.set_trace()
7 if book.price > 50:
8     print('High price.')
9
```

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。

—END—