《像计算机科学家一样思考Python》

图书基本信息

书名：《像计算机科学家一样思考Python》

13位ISBN编号：9787115320926

10位ISBN编号：7115320926

出版时间：2013-8

出版社：人民邮电出版社

作者：(美)Allen B.Downey

页数：328

译者：赵普明

《像计算机科学家一样思考Python》

内容概要

《像计算机科学家一样思考python》按照培养读者像计算机科学家一样的思维方式的思路来教授python语言编程。全书贯穿的主体是如何思考、设计、开发的方法，而具体的编程语言，只是提供一个具体场景方便介绍的媒介。《像计算机科学家一样思考python》并不是一本介绍语言的书，而是一本介绍编程思想的书。和其他编程设计语言书籍不同，它不拘泥于语言细节，而是尝试从初学者的角度出发，用生动的示例和丰富的练习来引导读者渐入佳境。

作者从最基本的编程概念开始讲起，包括语言的语法和语义，而且每个编程概念都有清晰的定义，引领读者循序渐进地学习变量、表达式、语句、函数和数据结构。此外，书中还探讨了如何处理文件和数据库，如何理解对象、方法和面向对象编程，如何使用调试技巧来修正语法、运行时和语义错误。每一章都配有术语表和练习题，方便读者巩固所学的知识和技巧。此外，每一章都抽出一节来讲解如何调试程序。作者针对每章中所专注的语言特性，或者相关的开发问题，总结了调试的方方面面。可以说这是一种非常有益的创新，让初学编程的读者少走很多弯路。

全书共19章和3个附录，详细介绍了python语言编程的方方面面。《像计算机科学家一样思考python》是一本实用的学习指南，适合没有python编程经验的程序员阅读，也适合高中或大学的学生、python爱好者及需要了解编程基础的人阅读。对于第一次接触程序设计的人来说，是一本不可多得的佳作。

《像计算机科学家一样思考Python》

精彩短评

1、内容讲解清楚明白，非常适合python入门用，但对于学习过其他编程语言的读者来说可能会觉得进度比较慢，但作者的思路和想法确实给人很多启发，对于菜鸟如我来说收益匪浅，书中很多例子还是有一定难度的，完全吃透也不容易。
2、入门计算科学领域的人是一本不错的书籍，需要了解python语法
3、书很棒，并不像其他书籍那样以文档形式给出，确实是“ 授人以鱼不如授人以渔 ”。
4、蛮好的入门书，我看的英文的，就是有点啰嗦，适合零基础的人看
5、很薄的一本书 一天入门Python
6、一堆概念和名词解释
7、书名起得好，实际跟科学家没什么关系，我看跟Python基础教程之类的也没什么区别。
8、太基础、内容太少，看起来觉着很啰嗦，相当无聊
9、这本书还OK，遗憾的是在图形或算法都不算我的专长，不然可能能从本书中汲取更多Python的姿势。
10、作为一本入门python的书还是不错的；我是用了2年python后再看这本书的，也补充了一些以前没注意到的，但是很有用的知识点。
11、还行
12、很精简的一本书,但是读完之后很久一段时间我都拿来当参考书翻阅.五星推荐给Python初学者!
13、正在看第17章，太简单了，适合入门，我怎么觉得我在哪看到的是适合进阶。
14、这本书写的真实经典，非常简单易懂。建议看完此书后 再看 Python cook python核心编程等。。。
15、能明白一点的是，作者没有“ 像计算机科学家一样 ”思考python。
16、Python 启蒙书籍
17、翻过，太简单
18、很适合入门，基本上扫一遍就能够大致了解整个语言的特性。面向对象部分简单涉及。而对于较为复杂的装饰器、高阶函数，没有涉及。如果仅作为提高兴趣来讲，这本书再适合不过了。
19、这本书简直适合没接触过编程的人学习第一门语言时来读。语言非常通俗易懂，是一本挺好的入门书，最后附录还涉及到一点算法。第一次接触python看这本书会挺爽的。
20、课后习题很棒！
21、十一前把它终结！
22、简单,易懂。用通俗的话把程序设计这件事讲明白,包括从开始的设计,优化,除虫等。让初学者有个全面的认识,从而快速上手。
23、如果说跟孩子一起学编程是学前班课本的话，这个应该算是小学三年级水平了吧。。Well I moved faster than I thought. 希望能快快开始实战Y(^_^)Y
24、翻译流畅易懂，适合零基础入门。
25、我决定把它当作一个软件工程导论的书去读。天呐，我研究生就快要毕业了，现在还要去读一个导论类型的书籍。没错，我就是准备这么做。但我先要把Linux基础给干掉。
ps：这本书我只进行到第六章，最爽的是学习安装了setuptools和pip，并成功用这两个工具安装了swampy第三方包。用swampy坐了一些小练习也很爽。但要接着进行下去似乎是不太好。我想先要系统学习Linux和网络才是比较好的，这两个学好了，在学习Python的好多包就轻而易举了。
希望以后想起来还能继续做完这里面的习题。http://www.greenteapress.com/thinkpython/code/书里面习题的代码，作者很棒，很认真
26、还行的Python入门书
27、比国内大一入门编程的红皮书好多了，一边介绍语法，一边介绍一些软件工程的思想，从而引导菜鸟如何去思考，如何去写好程序，如何debug。菜鸟一上来就应该读这样的书，而不是读纯语法书。
28、蛮不错的书，但是真觉得不太适合初学者，重点在于题部分需要一部分前置知识，不然很容易做不出来
29、Think Python
30、做为了解python的基础书籍是非常不错的。
31、作为入门练习，应该是最佳的书了。但是在这以前要简单了解下python2跟3的区别，以及简单的

环境配置。
32、我还没开始读，书就结束了... 尴尬....
33、十分适合初学者看！会让人在脑海中搭建一个总的框架，既全面又不失简洁，里面一些题对于我还是挺有挑战性的。作为第一本编程入门学习书最好不过了。当然想要快速上手的话，应该还是直接从项目中学习来得快。
34、事实上大多数人在学习python的时候已经被不pythonic的编程习惯和思维污染了，看这本书的例子其实还算简单不是特别深入，但是很多地方需要细致的想想为什么？学会pythonic的编写程序比硬上要好得多。
35、补充了一下之前不了解的概念
36、非常不错，比起语法来更关注深层次的思考框架。
37、看书名以为是深入理解语言的书，结果是入门书
38、很爽快，也学了不少术语，推荐
39、感觉不如核心编程啊。。。
40、适合编程入门
41、这个系列的书很赞，比一些死板的教程好多了。
42、入门读物，侧重思维，辅助以大量的小程序案例分析
43、一个字: 水...
44、后期习题就做不进去了。。呵呵
45、建议先跳过章后练习题 步子大了容易扯着蛋
46、入门不错，浅显易懂
47、英文名《Think Python》翻译为中文《像计算机科学家一样思考Python》
48、看了前四章，就发现的确是本好书，作者对程序的理解足够的深入，基本的设计规则讲得很露骨。程序写多了，复杂的语言学过之后，往往容易忽视最基本的一些规则，导致有时候遇到更困难的问题时候不知所措，无从下手。重复是程序设计一个很基本的规则，但是不好掌握，延伸的问题很多，遇到问题需要将问题分解、循序渐进，最重要的还是要有信心与耐心解决一个复杂问题。
49、这本书比较适合作为编程入门，而不是python入门，它对于python语法并没有做特别多的描述，而是讲了很多如何去思考编程，学习编程的方法和道理，你要是看过任一一本Python入门，那么快速浏览即可。
8月4号，开始《Python在Unix和Linux系统管理中的应用》。
50、没基础看概念，有基础看题。相较其他同类书对初学者更为有益，不仅对部分概念的本质有涉及，还在教学中培养先进的编程思想。“ 对真实世界问题的更高阶理解可以使编程更简单 ”。

《像计算机科学家一样思考Python》

精彩书评

1、#《像计算机科学家一样思考python》并不是一本介绍语言的书，而是一本介绍编程思想的书。好吧。。内容简介里面有说，这本书我确实理解错了。但我还是不认为这本书适合0基础入门。0基础入门我推荐《Python计算与编程实践：多媒体方法》前言写的：“ - 尽量简短。学生读10页数，比不读50页书要好。(这句话什么鬼！)- 注意词汇。我尝试尽量少用术语，并在第一次使用它们时做好定义。 ”浏览完整本书之后，我就忍不住呵呵了。(我尽量不嘲讽)第一眼看这书，感觉很扎实。无论是标题的科学家字样，还是从目录章节来看。但是仔细看起来，感觉通篇概念框架，泛泛而谈。我有点无法理解这本书的定位。作为入门书籍的话，书里提到的概念术语实在是太多。你已经没办法从其他python书籍上找到属于比它还多的了。联系到前言里的声明，如果不是作者的冷幽默太黑，就是有点在故意卖弄自己的专业知识了。如果作为一本提高书籍，细节又实在过少。全是些基本操作，背后的原理细节一概不涉及。读人家50页书的废话纯粹是浪费时间，来读我这10页书就够了！(如果我没理解错前言那句话的话)这是在修仙么？我这边的知识精气更纯净哦！我真是醉了。这里的每10页背后都要花几十上百页，甚至一整本书的知识去做深入。然而作者的叙述让你提不起一点兴趣，或者你甚至意识不到要去扩展背后的知识。所以如果消化不了，吃进去多少最终还得全吐出来。感觉这本书的读者群体是那些对软件工程十分熟悉，对编程语言十分精通(设计过一门更佳)。作者给你介绍一下python的一些特点，你了解一下不同之处就可以马上开干的感觉。这样的话，这本书真是1天不到就读完了。无论是0基础学编程，还是从什么地方转过来的都好，其实都有其他很好的书籍，可以去选择。没必要在这本书上浪费时间。或许是我能力不够，上升不到科学家的层次。但要我上升到那层次了， 我估计也不会回来看这书。

《像计算机科学家一样思考Python》

章节试读

1、《像计算机科学家一样思考Python》的笔记-第1页

记录几个区别其它语法，自己较为常忘的用法于此。

循环/选择等用：

def定义函数

and or not

elif 不是 else if

带返回值函数返回用return，这一点类似C而不是matlab

多行注释'''

输入raw_input()

表示次方，a2

print语句结尾的逗号禁止产生新行

for char in fruit:
#注意in用法

>>> print s[0:5]
Monty
#注意切片用法

函数的语法upper(word),方法的语法word.upper()

for i in range(len(numbers)):
#组合使用range 和 in

t.append('d')
#append在列表结尾增加一个新元素

t1.extend(t2)
#extend将一个列表作为实参并追加所有的元素

列表方法都是无返回值的

三种列表元素删除方法：
t.pop(1)有返回值
区别del t[1] #可使用切片
t.remove('b')

将字符串分成单词 , 可以使用split方法

join和split相反。它接受一个字符串的列表并将元素串联起来。

a is b #相同的引用

字 典 (dictionary)) 和列表类似,但是更一般。 在列表中,索引必须是整数,在字典中,它们可以是(几乎)任意类型。

>>> eng2sp = dict()
花括号{}表示一个空字典。为了向字典内增加项, 你可以使用方括号:
>>> eng2sp['one'] = 'uno'

raise ValueError
#raise语句引起一个异常

```
#备忘录
known = {0:0, 1:1}
def fibonacci(n):
  if n in known:
    return known[n]
  res = fibonacci(n-1) + fibonacci(n-2)
  known[n] = res
  return res
```

函数中声明全局变量global a， 但声明时不能赋值

虽然并非必须,元组(tuples)通常用括号括起来:
>>> t = ('a', 'b', 'c', 'd', 'e')

为了用一个单独的元素生成一个元组, 你必须包括最后的逗号:
>>> t1 = 'a',
>>> type(t1)
<type 'tuple'>

>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')

这是一个返回元组的函数的例子:
def min_max(t):
return min(t), max(t)

>>> t = [1, 2, 3]
>>> random.choice(t)

如果这个文件已经存在,使用写模式打开,会将其之前的内容全部抹去从零开始。 所以一定要小心! 如果这个文件不存在,它会被创建。

>>> 'In %d years I have spotted %g %s.' % (3, 0.1, 'camels')

'In 3 years I have spotted 0.1 camels.'

读写时，‘a'模式为追加写

返回当前目录名:
>>> import os
>>> cwd = os.getcwd()

```
try:
  fin = open('bad_file')
  for line in fin:
    print line
  fin.close()
except:
  print 'Something went wrong.'
```

内置函数repr可以起到帮助。它接受任何对象作为参数,返回其转化成字符串的显示。 对于字符串来说,它将空白符转化成反斜杠序列。

可以使用assert语句,检查一个不变条件,当失败是产生一个异常:

```
def add_time(t1, t2):
  assert valid_time(t1) and valid_time(t2)
  seconds = time_to_int(t1) + time_to_int(t2)
  return int_to_time(seconds)
#这里，valid_time返回值为bool
```

定义在类内成为方法，类外为函数

```
class User(object):
    "This is user class."
    name = "tom"
    age = 0
    def __init__(self, age=22):
        self.age = age
    def showAge(self):
        print self.age
    def showClassName(self):
        print self.__class__.__name__
    def showClassDoc(self):
        print self.__class__.__doc__
```

实例化这个类：

```
user = User()
```

调用类里的属性name

```
print user.name
```

调用类里的方法showAge()

```
user.showAge()
```

约定方法的第一个参数写作self,所以可以把print_time重新写成:

```
class Time(object):
```

```
def print_time(self):
print '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)
```

init方法(“ initialization ”的简称)是一个特殊的方法,当一个对象初始化的时候调 用。它的全名是__init__(两个下划线后加init再加两个下划线)。一个 Time类的init方法看起来像是:

```
# inside class Time:
def __init__(self, hour=0, minute=0, second=0):
self.hour = hour
self.minute = minute
self.second = second
```

__str__是一个和__init__方法类似的特殊方法,用来返回一个对象的字符 串表达。

```
#好像java的ToString()
```

通过定义其它的一些特殊方法,你可以在用户定义的类上指定运算符的行为。例如,如果你为Time类定义了一个叫__add__的方法,你可以在Time对象上使用+ 运算。

```
if isinstance(other, Time):
```

内建的isinstance函数使用一个值和一个类对象,如果值是这个类的实例则返回 True。

为了调试目的,你可能发现下面这段代码非常有用:

```
def print_attributes(obj):
  for attr in obj.__dict__:
    print attr, getattr(obj, attr)
```

对于用户定义的类型,我们可以通过提供一个叫 __cmp__的方法来覆盖内建的运算符的行为。

__cmp__ 使用2个参数, self 和 other,如果第一个对象更大,返回 一个正数,第二个元素更大返回一个负数,如果相等返回0.

可以使用元组比较来使得代码更加简洁:

```
def __cmp__(self, other):
  t1 = self.suit, self.rank
  t2 = other.suit, other.rank
  return cmp(t1, t2)
```

定义一个子类的方式和定义一个普通类比较相似,但是需要在括号写是父类的名称:

```
class Hand(Deck):
```

《像计算机科学家一样思考Python》

版权说明