## Ch2-3

40 源代码.py 字节码.pyc Python虚拟机

42 可以在运行时构建并执行另一个Python程序

51 交互模式自动打印，Windows下Ctrl-Z退出

54 %python spam.py > output.txt 命令行运行源代码与流重定向，重定向是python的功能直接打开时不能用

55 运行时加.py，import时则否

60 %python spam.py < input.txt 输入流重定向

61 .py就是模块；import模块名 会运行，只有第一次有效；如果要重新运行用reload(模块名无引号)

63 import整体引入，内部属性用.访问，类似命名空间；from import拷贝属性，直接访问；方法也是属性。

64 dir(模块名无引号) 获取该模块可用变量名列表

## Ch4-6

83 Python是动态类型的，强类型（只能对某类型的对象进行有效操作）的语言。

84 精度需要时自动升级；print的形式用户友好，自动打印形式为代码形式不友好。

85 序列通用操作：len()，[]，[-1]，[1:2]，[1:]，[:]，+，\*。序列包括字符串、列表、元组。

88 数字、字符串、元组不可变，列表和字典可变。不要将重新赋值误会为可变。元组=不可变列表。

89 可作用于多种类型的通用操作都是以内置函数或表达式形式出现的如len()，特定类型的操作是以方法调用出现的如line.upper()。不可变实例的方法返回的都是新东西不是经修改的原件，可变则相反。

90 常用：先用dir看方法，再用help(实例.方法)看具体介绍。单引号、双引号均可字符串，三引号可多行，\0不会结束字符串。

94 列表解析：… = [… for … in … (if …)]。

96 列表禁止越界赋值，字典可以赋值新键，但不允许查找新键。

99 列表解析、for、sorted可以对任何遵守迭代协议的类型进行，for和列表解析可以互相转化但是列表解析快。

103 判断类型：type()

104 类：\_\_init\_\_，所有方法都有第一参数self，成员不用列出而是包含在方法中，方法和成员不能重名。

110 数字：整数自动升级长整型（无限精度尾巴L），所有英文字符大小写都可，符合直觉。类型转换 类型名()。

112 直觉运算符：not; and; or; in; is等实例; ^位异或; ~位求补; a if judge else b三元选择

117 repr()得到内部表达即自动回显用的，str()得到用户友好表达即print用的，都可把别的转换为字符串。

119 /浮点数正常而整数下取整，//无论类型都下取整

121 i写作1j。oct()，hex()函数将十进制数字转换为相应进制字符串。int(‘0x10’, 16)得16。

123 abs，pow，round(数字, 小数位数)做四舍五入直接可以用，sqrt等需要导入math

125 小数类用到再说。set(可迭代对象)得到集合或者直接{,,,}，但{}是空字典不是空集合。布尔的True和False就是整数1和0不过显示不同。

130 变量引用对象，类型属于对象而非变量。一般每个表达式生成一个值时都创建一个对象，但也有缓存常用对象作为优化。每个对象有一个类型标识符和一个引用计数器。

133 变量名2 = 变量名1：变量名2引用了变量名1在本句运行时的对象。根据这一点和Python对可变与不可变对象的不同处理来理解会发生什么。变量名1转而引用其他东西则变量名2不受影响；变量名1修改了所指向的可变对象则变量名2受影响。

135 生成新拷贝可以防止互相影响。通用方法：import copy copy.copy() copy.deepcopy()

137 ==值相等；is同一对象

## Ch7-9

142 自动在任意表达式中合并相邻字符串常量。

144 字符串中的二进制字符：处理二进制文件时候比较重要，到时候看。

146 r抑制转义。u生成Unicode类型字符串，用\u或\U编码大于1字节的2进制数字进字符串；类型不同但并不重要，共同运算时自动转换，方法通用；str(u’’)与unicode(‘’)互相转换。

150 in可对字符串进行子串搜索。

156 ord和chr在数字和字符之间转化。

158 字符串格式化：替换位置用%，串结束用%，替换一个直接写，否则写元组。替换位置完整格式%[(name)][flags][width][.precision]code。name用于按照字典格式化。vars()当前所有变量构成的字典。

164 list(字符串)转换为列表，’分隔符’.join(列表)连接为一个字符串，split。

173 列表：range()生成整数列表；extend(可迭代对象)将其内容加入；append(任何对象)在末尾增加一个，注意append!=extend；del关键字可以直接用于删除如del list[0]；+\*返回新列表而非原地修改，慢；sort原地排序；append和sort等无返回值；分片赋值：先删除分片部分，再替换为右边部分，因此未必要等长，但右边必须可迭代；index(对象)返回对象第一次出现的下标

180 字典：in可判断键存在性；dict(zip(keyslist, valueslist))从列表构建字典；用get获取可以避免无效键出错；update合并字典，同名更新；del同样可用于删除；for key in D等价于for key in D.keys()；任何不可变对象都可以作为键，比如整数；dict([(),(),(),…])从键值对列表构建字典，zip就是这个原理；dict(键列表)生成默认值的字典

191 元组：(40,)是元组而(40)是表达式；不引起语法冲突时可以忽略圆括号，但最简单的处理就是在用元组时候一直加圆括号；tuple()转换为元组；列表解析永远返回列表但可以作用于一切序列如元组

195 文件：’r’为默认；open第三参数控制缓存；flush缓存写入硬盘但不关闭；seek修改文件位置；自动回收也会关闭文件但手动关闭更值得推荐；返回空串则到底；写入不会自动换行；int等转换字符串到数字会忽略数字旁边的空白；eval可以把字符串当代码运行，通用但慢而危险；pickle用于对象存储和读取import pickle pickle.dump(内容, 文件) pickle.load(文件)；打包二进制数据：用到再说

202 自己写的类可以通过写函数来控制其是否可变、是否序列等

203 有些内置函数如list能生成拷贝；一般都是浅copy，只有copy.deepcopy()是深层copy

206 Python中一切可比较，不同类型的按类型给出结果，同一类型的比较符合直觉，字典比较按键顺序；一切有真假，if未必要用bool，0为假，空为假，None为假，其他为真

210 一般建议使用isinstance，因为类型有子类

212 重复\*、合并+、分片[]都只是复制操作对象的顶层；循环对象：包含指向自身的引用，一般可显示为[…]而不会无限循环，但有隐患所以少用

215 列表索引越界出错，分片越界按正确边界执行，L[a:a]=b则在a处插入b其后顺延，L[a:<a]则<a按a执行即插入；X,Y=Y,X不是三个式子而是对元组赋值，会导致交换

## Ch10-11

221 程序-模块-语句-表达式 表达式处理对象

224 所有Python复合语句首行以:结尾，嵌套代码缩进，如何缩进或者缩进多少无所谓但块中必须一致，()和;可以有但一般不要，一行多句简单语句时候用;分割（复合语句不能这么干），三种括号（常见()）中的语句可以跨越多行且缩进不限，\结束时候可以延续到下一行但该方法不要用，不含复合语句的复合语句主体可以放在首行:后，elif，pass无用但令空块满足语法，try:可能出错的代码块except:异常时运行的代码块else:正常时运行的代码块

238 序列赋值语句：序列=序列 则其中每个元素对应赋值，左右未必要是同一种序列，注意x,xx,xxx这种是元组也是一种序列；元组赋值可以交换两变量的值却不创建临时变量

242 多目标赋值（连等）都引用最右对象

243 增强赋值语句：+=这种，快，对扩展列表这种操作会优化为原处修改如append因此快（bug点：行为和产生新对象的L=L+[]不同），无++这种

245 变量命名：下划线或字母开头，区分大小写，包括模块名都不能用保留字，以单一\_开头的变量名不会被from .. import \*导入，交互模式下\_保留了最后运行的表达式的结果

248 比较可以串起来a<b<c

249 bug点：L = L.append(4)丢失列表，因为append这类原处修改无返回值

250 print：末尾加,可以去掉自动换行，开头加>>fileObj对该次print输出流重定向，默认输出流import sys sys.stdout，这个值也可以手动给一个有write函数的对象以进行标准输出的重定向，恢复时用sys.\_\_stdout\_\_即可，sys.stderr类似

## Ch12-14

257 Python中多路分支是写成一系列if/elif或索引可变对象形式的

260 顶层代码必须从第一栏开始

263 and和or会返回得到结果的那个操作对象而非简单的True或False，比较和not返回True或False，有短路特性，记住每个对象都有真假因此都可以运算

269 while或for的else：控制权正常离开循环（没有break）时执行，常结合break省去搜索标志，注意当循环没有执行时也会运行（bug点）

272 pass可能意味着“以后会填上”；raw\_input会在文件结尾时引发异常（如用户Ctrl-Z）

275 Python不允许赋值这类语句出现在应该是表达式出现的场合如while的条件

277 for语句中可以隐含序列赋值语句如for [a,b] in [(1,2),(3,4),(5,6)]:

280 好东西：file.xreadlines()按需求加载文字列，防止内存溢出；所有迭代工具可工作在可迭代对象上

281 迭代工具的运行：iter(可迭代对象)获得迭代器然后用它一次次调用next；next函数即迭代协议，每次被调用时返回一些东西，结束时触发StopIteration异常；读取文本文件的最佳方式：for line in open(‘’) 优点是快（迭代器以C效率运行），简单，内存使用好；有迭代器的东西：列表，字典，文件；一般隐藏了iter()调用，本质是for line in iter(open(‘’))

284 迭代工具：for，列表解析，in成员关系测试，map，sorted，sum，any，all，list和tuple内置函数，字符串join方法，序列赋值语句

285 range有1-3参数；效率起见尽可能用简单的for循环，不要用while不要用range；想跳过优先尝试分片；用for range的常见场合是在循环中进行修改，因for x in L: x += 1会丢失修改，真正的修改需要索引，但这种情况往往可以通过列表解析生成新列表解决

289 zip以1到多个可迭代对象做参数将其折叠为元组列表，长度按最短参数，和for结合可并行遍历（往往用到序列赋值）；map(函数名无括号, 1到多个可迭代对象)将各个可迭代对象折叠为元组列表，长度按最长参数其他不足补None，然后以列表中的元组分别调用函数并收集结果，当函数为None时就相当于zip的最长版本

291 当我们在迭代过程中又需要偏移量又需要该偏移处的内容时，用enumerate(可迭代对象)可得迭代器，其每次next获得(偏移量, 内容)元组 for (I,c) in enumerate(L)，一般看不到iter的调用是被省略了（281）

293 能用列表解析的地方用列表解析更快，不是原地修改而是生成新列表代替旧的（bug点）；当我们想对可迭代对象中的每个元素执行一种运算时，就处在列表解析的范畴；可以含有任意多for且每个for可带一个if得到嵌套循环，左边的for在外侧；['ad', 'ae', 'bd', 'be'] = [x + y for x in 'abc' if x < 'c' for y in 'def' if y < 'f' ]

301 文档字符串：各个东西的\_\_doc\_\_，编写方法是把字符串写在这个东西的开头，任何类型的字符串都行；dir(list)==dir([])是因为调用名称会启用构建函数从而产生实例

304 PyDoc：生成包括文档字符串和系统收集的信息在内的报表以详细介绍某样东西；启用PyDoc的基本方式：help()；更详细的网页方式：找到module docs文件打开它；标准手册：windows上安装了，上网也能找到

311 空白行在交互模式下表示结束复合语句；bug点：不要混用制表符和空格

312 for k in D.keys().sort()错误，因sort无返回值；字典迭代器不会进行排序，此处可以用sorted函数；因函数也是属性所以带不带括号都是合法的，但切记作为函数调用要用括号

## Ch15-17

321 函数也是对象，在def语句被执行时创建并连接到一个名称，相关的如多名引用、将函数作为参数传递什么的自然合法；参数通过对象引用传递给函数，自动隐含地赋值给本地变量；global修饰的变量名被认为在模块命名空间中，本地命名空间不可再有，可以在模块命名空间新创建；return可选，无return返回None；def是一个语句因此相关的如嵌套等自然合法；def包括其参数列表和默认参数等等在def语句运行时评估，其中的代码块在调用时评估；符合接口则自动多态，否则抛出异常，不要检测类型破坏多态

327 文件迭代器的特殊性：其他序列获取迭代器则迭代器重新开始，文件则在第一次迭代器运行完毕后耗尽，无法重新开始，表现出来就是只能进行一次in测试

330 每个模块是一个全局作用域，但对于其import方则全局变量成为模块对象的属性；函数内部的赋值语句（除global）如=, import, def, 参数传递都默认操作本地变量而与外层不冲突不影响；bug点：改变可变对象如L.append()不算赋值语句，会影响外层；LEGB：本地，外层函数，模块，内置

334 获取内置要用内置作用域import \_\_builtin\_\_，它的属性就是全部内置对象；可以让Python通过LEGB自动搜索并使用内置对象或者手动\_\_builtin\_\_.zip()，当内置变量名被覆盖时候用手动方法很有必要（bug点，Python对这种情况不报错）；bug点：赋值语句（除global）将在def运行时被发现并将相关变量认为本地，此时不可能在该代码块中引用EGB中非本地的同名变量，因为被赋值的变量名在命名空间内被当做本地变量而非仅在赋值以后的语句中被当做本地变量（394）

337 少用全局变量因为危险，但多线程依赖全局变量；最小化文件间变量修改，如果要这么做那么调用函数来做；bug点：文件间变量修改如在c2.py中import c1 c1.a = 33是危险的，因为通过import c2获得c1的人所看到的c1.a和c1代码中写出的a不一样，造成困惑

339 首次from sss import xxx会运行在sss中所有代码并得到xxx，如在此基础上再import其他变量则不必运行直接获得

340全局变量=引入模块后它的属性，因此可以多种方法在函数内引用全局变量且不与本地同名变量的存在发生冲突：import thismodule thismodule.var += 1或import sys glob = sys.modules[‘thismodule’]寻找已打开模块（必须已打开）glob.var += 1（交互模式下thismodule是\_\_main\_\_ 331）；注意在运行一个模块文件时里面要执行函数而且用了这招的话会很微妙，看代码，简单地说就是只用340的方法不要纠结

341 嵌套函数所依赖的E作用域变量会被保存，该特性可以用于书写工厂函数以产生一系列相似的函数并共存；类、全局变量、嵌套作用域、默认参数是函数保留状态信息的方法；默认参数就是在函数定义时参数用arg=value这样的式子，早期版本中不支持嵌套作用域则常见默认参数

344 由于def和代码块评估时间的区别(321)，

def a:

块A

def b

块B

b()

块C

这种结构中b的默认参数只能看到块A中的变量而b的代码块可以看到块AB中的变量但二者都看不到块C中的变量；由于同一原因在一个函数的定义中调用一个之后才定义的函数是可行的，只要运行该函数时另一个已经定义好就行（因为那时该函数的定义才被验证），变量也一样（458）

344 lambda是一个表达式，生成函数对象并返回，需要命名自己操作，语法lambda 参数列表:表达式，和def相同有嵌套作用域有默认参数，主体是表达式不是代码块（366）

345 bug点：嵌套作用域中的变量在嵌套定义的函数被调用时才进行查找，所以循环定义多个函数不用默认参数会造成所有函数相同（他们都看同一个变量名），解决方法是用默认参数因为它在def运行时评估能记住；注意for不构成一个单独命名空间，其循环变量如果和后面用的同名则继续变化且都影响这些函数的结果，见代码

346 bug点：默认值是以单一对象实现的，因此可变默认值如列表会在函数运行间被修改而非一直看到代码中写的默认值，可以是一种可用特性但更像一种陷阱，解决方法395

347 bug点：可变参数可以做输入或输出，避免改变应该使用copy；替代通过可变参数输出可以让return返回含多个东西的元组

352 定义参数中带\*表示收集不匹配的位置对象，\*\*表示收集不匹配的关键字对象，这两种东西调用时无法直接为他赋值；调用参数中\*表示打散序列成位置对象，\*\*表示打散字典成关键字对象；关键字赋值方法的存在使跳过有默认值的参数成为可行

358 通常最好假设参数在代码中有效，无效时Python抛出错误

360 调用或定义时参数顺序是位置参数、关键字参数或默认参数、\*name、\*\*name；匹配时先将打散内容打散放到所应在位置，然后匹配顺序是位置参数、关键字参数、\*收集其他位置参数、\*\*收集其他关键字参数、默认参数；以每个非\*/\*\*参数都有且只有一个值为匹配成功标准；不可能成功用关键字参数匹配到非\*/\*\*参数且多出位置参数

367 可以用lambda编写跳转表即行为的列表或字典；在lambda中用a if b else c写分支，用map或列表解析写循环；可以隐含序列赋值语句如map((lambda (name, age, job):age), infolist)（382）

370 apply(函数名, 参数元组[, 可选参数字典])动态调用，优势是不需要知道调用需要多少参数，现在用可变参数可做类似的事情

373 map(函数名, 1或多个列表)将列表排齐，不够补None然后依次调用函数并收集结果，结果列表和最长的列表等长；如果函数为None就是把各列表对应位置元素构成元组，类似zip但长度取长； filter和reduce参数都是一个函数一个序列；import operator模块提供了内置表达式的函数

381 列表解析是表达式，开头也是表达式，所以可以嵌套

383 生成器函数中用yield（表达式，可以在等号左右）返回一个值并挂起直到下次继续运行，return或自然终止时结束；每次执行函数返回一个新生成器，一个函数可以做多个同时生存的生成器对象；生成器对象调用next获取下一个值，结束时exception；通常用迭代工具使用生成器；生成器的优势是逐步产生结果内存使用好，并能把处理时间分开；为类定义生成器只要书写其\_\_iter\_\_函数；生成器表达式类似列表解析只是外面()，返回一个生成器对象，当作为函数的唯一参数时括号可省略

388 import time time.time()计时；\_\_name\_\_属性函数名；一般效率：内置函数map>列表解析>生成器>>for>自定函数map

391 设计思路：少用全局变量，少用可变参数返回值，少用文件间变量修改

## Ch18-21

408 import中的模块名用于识别加载对象，同时也作为所得模块对象的变量名；导入可以任意深可以循环（因重复导入失效而不会死循环？后面有实验）；生效的导入会搜索、如有必要编译字节码、执行代码创建模块对象

410 搜索路径：程序主目录（被打开console文件所在的目录或者被运行模块文件所在的目录），PYTHONPATH环境变量从左至右，标准库，在搜索目录中的.pth文件中写下的目录；合起来是sys.path可调整，优先级递减即从左到右，同一目录找到同名则行为不定，24可自定，一般还会加上工作目录不过不应该依赖它因为不确定；问题：自己写的应可以借此覆盖系统库，sys和math等不行，string等可以，不管怎么写sys.path都不行，不知道为什么？

414 发现.py&&(没有.pyc||.pyc较旧)则编译；只有.pyc则直接用，可用于保密；只有被导入的文件才有.pyc；重复import不操作，非要重来reload

415 思考基点：在程序的一次运行中，同一个模块无论被多少个其他模块导入或reload或from多少次在内存中都只有一份；头一次或者reload时构造这个对象，其他则都是获取这个对象的引用并连接到自己需要的变量名；import和from都是普通语句可出现在任何合法地方，获得的都是普通变量名遵守作用域规则

420 思考基点：from = import + 赋值 + 删除模块变量名；from xx import \*：如果有\_\_all\_\_则用赋值语句得到其中指定的任何属性，否则所有开头不是\_的（446），但这并不是私有因为import还是能看到；因为是赋值语句得到的所以name和module.name是共同引用关系，如果引用了不可变对象则对name的赋值不影响module.name，否则影响（也即影响内存中这唯一的一份模块对象，别的引用者也会受影响）；相对的，import mod后对mod.name赋值则不管可变不可变都会影响

425 \_\_dict\_\_或dir()获取模块命名空间；\_\_file\_\_获得模块文件完整路径；\_\_name\_\_获得模块文件名称，但当作为顶层文件执行时候是“\_\_main\_\_”（448）

428 运行的方式和变量名的意义完全由模块文件所决定而非运行方式，因此若A import B后运行B.func，则B.func中的global X是指B中的不是A中的

430 思考基点：reload是重新运行，需要模块对象已存在并修改这唯一的一份模块对象；允许修改程序的一些部分而不用停止整个程序；只能用在Python编写的模块；修改指对模块对象中的变量名重新赋值，因此之前from所造成的共同引用关系（不管是可变对象还是不可变对象）都被破坏，这些变量均保持原值，而通过mod.name访问的都变为新值；返回了模块对象但常被忽略；其参数是引用模块对象的变量名而不是模块名，虽然这两者常常一样但并不必须如此

435 包导入：把文件夹做成模块对象，其中的文件、文件夹、\_\_init\_\_中补充的元素等均成为其属性，适用模块的三种操作import from reload；import dir1.dir2.mod中.号表示取属性即目录路径，这就是.py要刻意忽略掉的原因；dir1文件夹必须在Python搜索路径上，dir1和dir2中必须有\_\_init\_\_；dir1所在目录dir0有无\_\_init\_\_均可，即使有在本次import中也不起作用因为未被操作

437 \_\_init\_\_的作用：标记合法目录以防止错判；首次操作时会运行以初始化并产生等价模块对象；用\_\_all\_\_可以控制from\*获得自己的哪些属性（包括文件夹、文件），若没有\_\_all\_\_则忽略文件、文件夹属性

440 为什么用包：路径更明白，简化搜索路径设置，允许使用多个同名模块

447 启用新特性：from \_\_future\_\_ import，要求出现在最前面防止新旧混用；通过检测\_\_name\_\_可确定是执行还是导入，常用于书写自我测试代码；as就是换名，可以理解为多一句赋值多一句del（也不尽然，引入时的中间变量不会影响文件中的原有变量，如原有A，import A as B后原来的A还有效），多个写作A as A1, B as B1

451 绝对导入与相对导入：没搞懂，跳过，再说

454 所有代码都在模块中，交互模式的代码在内置\_\_main\_\_模块中；少用全局变量；少修改其他人的全局变量

455 模块的属性字典\_\_dict\_\_；sys.modules字典含所有已加载模块（340），以模块名（不能用模块变量）索引；getattr(模块变量, 属性名字符串)获取属性

458 动态import：exec “import” + “moduleName”或moduleRef = \_\_import\_\_(“moduleName”)，后者大概快一点

462 reload没有传递性

462 不要递归import！不要递归import！不要递归import！没搞懂，跳过，再说

## Ch22-26

473 思考基点：类和实例都是对象，每个对象本质都是命名空间（技术上是以字典形式实现的）；连接别的命名空间并可搜索就是继承（字典连字典）（529）；方法本质上就是函数但第一个参数（self名字不重要）自动传入调用实例（所以至少要一个参数）（497，504）；属性搜索范围为继承树，由实例对象、所属类对象、所有连带父类对象构成；搜索由左向右（多根继承情况），深度优先（除新类钻石继承情况外）；属性引用时搜继承树，属性赋值时在当前对象修改或新增（若本地没有，不会去操作树上被引用那个）（该操作可被劫持，则行为不再确定）（486，503）；无论引用还是赋值总可以跳过搜索直接指定（504）；class是语句，产生类对象并赋值给变量名，执行Class时会执行其主体内每个语句（502）；每次类名调用时运行\_\_init\_\_产生实例对象

478 思考基点：程序可以取出、修改或创建其所引用的任何对象的属性（488），类、实例、模块甚至函数都行，其dir都有乱七八糟的\_\_xxx\_\_；类对象的属性往往在class块中由顶层赋值语句添加；实例对象的属性往往由\_\_init\_\_进行初始化，并在各个方法中通过self.进行控制（虽然self名字并不本质但惯例如此）

491 思考基点：变量均遵守同一套规则（作用域、共同引用等），不管他引用的是什么对象；变量del的结果是变量失效，被它引用的对象引用次数减1，若0则空间回收；继承的技术本质是字典的隐含引用

493 运算符重载\_\_add\_\_、\_\_mul\_\_在运算符左有该类实例时由左调用，\_\_radd\_\_、\_\_rmul\_\_在运算符右有左无时由右调用，不同类混合出现时左侧优先（522）；都是含义特殊的变量名，可Python自动调用也可代码调用

496 实例的\_\_class\_\_引用其类对象；类对象的\_\_bases\_\_元组引用其直接父类；基本什么对象都有\_\_dict\_\_为其属性字典；对命名空间本身的（不含继承树）属性的引用和赋值等同于对其\_\_dict\_\_的索引和赋值；dir(something)类似something.\_\_dict\_\_.keys()（529），但dir现在自动包含了继承的属性名称（552）

497 Python中的OOP其实就是在已连接的命名空间内寻找属性而已

504 instance.method(args) = class.method(instance, args)都有效；后一种可用于明确调用超类方法如superClass.\_\_init\_\_

510 查找到父类的方法并进行调用时其内部语句仍从调用实例出发进行解释，如self.xxx会从调用实例开始重新进行查找（543）；抽象超类只是个称呼，就是部分行为默认由子类提供，没提供则异常或者用assert明确要求提供；assert 表达式返回对象判断真假 打印字符串

513 \_\_getitem\_\_拦截索引运算；各迭代工具先尝试\_\_iter\_\_（即内置函数iter）获得迭代器依次next到异常，不行则尝试\_\_getitem\_\_从0开始索引到异常，二者中有任何一个都能被用于迭代，写迭代器的方法见代码；索引可以循环多次，迭代器只能用一轮再用要重新生成（之前所说文件只能in一次）

518 \_\_getattr\_\_拦截对继承树中找不到的属性的.运算；getattr内置函数会进行继承搜索（549）；\_\_getattribute\_\_新式类可用拦截一切属性引用（573）；问题：549与518冲突，\_\_getattr\_\_到底拦截树上找不到的还是本命名空间找不到的？；\_\_setattr\_\_拦截所有赋值语句，可用于模拟属性私有，但为避免死循环要用self.\_\_dict\_\_[‘name’]=xxx来赋值因为self.name又要调用\_\_setattr\_\_；\_\_repr\_\_和\_\_str\_\_返回字符串表达形式，repr()和交互模式回显只搜\_\_repr\_\_否则就默认， str()和print优先\_\_str\_\_没有则\_\_repr\_\_；\_\_call\_\_拦截对实例的调用如instance()，注意不是拦截instance.func()，可以此用实例对象模拟可保留状态的函数（相对于全局变量、嵌套作用域等是更好的方式）；\_\_del\_\_拦截析构器但很少用因为空间自动管理而且无法预测何时会析构

526 无点号赋值：本地变量创建或修改（除非global）；无点号引用：LEGB；有点号赋值：对象命名空间内创建或修改；有点号引用：搜索属性或搜索继承树；变量名赋值场所决定作用域，不受导入关系影响；bug点：类构成命名空间但不是作用域，不算LEGB，，方法无法用无点号变量引用包含它的类的属性（更别提赋值，那是在本地），不过有点号当然没问题（582）

540 方法是多态的，无法凭参数重载；实例可以做记录比字典还方便

548 pickle和shelve保存各种对象：

import pickle

pickle.dump(something, open('pickleTest.txt', 'wb')) #注意binary因为pickle用字节流

something2 = pickle.load(open('pickleTest.txt', 'rb'))

print something2 == something #True

import shelve

db = shelve.open('shelveTest.txt')

db['obj'] = obj #shelve是把对象保存成字典形式的数据库

db.close()

db2 = shelve.open(' shelveTest.txt')

print db2['obj'] == obj #True

550 id()获得对象地址；用apply或更推荐的可变参数动态调用

554 无绑定类方法对象：class.func获得，未指定调用它的实例，需要实例传第一参数来调用；绑定实例方法对象：instance.func获得，已和实例对象打包不用再传第一参数，直接调用即可；instance.func(attr)等于boundFunc = instance.func boundFunc()本质上是分了两步但一般被忽略；绑定实例方法对象可用于回调函数（557,525）

556 \_\_doc\_\_位置：模块头，类头，方法头，函数头

561 类型转换调用其实是构造实例

564 变量名压缩：类语句块中\_\_attrName名称在进入子类前会扩张为\_className\_\_attrName；这是用来满足同时使用多个父类中的同名内容的需要

566 新式类：继承树中有内置类型或者object为新式类，3.0后所有都会变成新式类；新式类在钻石继承时会宽度优先让下面的都比共同超类有优先权（非钻石继承结构时仍然深度优先简直不科学），具体怎么办的？；会不会重复搜索？；多根继承需要同时用无法用顺序解决（580）或者不想搞搜索顺序想明确控制时可以在混合点（多根继承处）明确指出

570 新式类可以在\_\_slots\_\_写名称列表，问题：具体机制很复杂，没搞清楚怎么运行的，不能用；class语句顶层可以有内容属性name=property(refHandler, setHandler, delHandler, docString)，name对类对象来讲是一个引用property对象的名称，对类的实例来讲是一个被拦截的变量名，对它的赋值语句会交由setHandler处理，另三参数类似（docString还不知道起了什么作用，跳过，不重要）；property目的是替代\_\_getattr\_\_和\_\_setattr\_\_，只拦截特定的因此较快

573 无绑定方法对象和简单函数本质一样但不等，没有实例对象就无法调用（例：class.func和func引用同一个函数对象，但class.func调用时就要求参数传入实例对象而后者没有）；静态方法和类方法对传统和新类都可用：写好后改成静态或类，method = staticmethod(method)，method = classmethod(method)；前者就是由class.method(arg)调用的简单函数，后者运行时形式一样但Python自动将类对象传入第一参数（所以定义时至少一个参数）；函数装饰器：@xxx def func等同于def func func = xxx(func)将函数包裹一层（或多层@A\n@B A(B(func))）；可用于简化上述静态\类方法的书写（Python内置的函数装饰器），也可自定义，无法改变函数原有功能但能干点别的，见代码

580 for X.i in range(5)隐含对X命名空间中i的赋值；要被大量继承的类应当尽可能独立完备，

不应对其他类的变量名有任何假设，常常用变量名压缩

## Ch27-29

599 用户自定义异常要以类形式写（字符串已不行），可选但最好继承自Exception，触发实例对象，except类对象；手动触发raise或assert；如果没异常的话else在代码块后执行；finally一定会执行，发生异常并被捕捉的话处理后执行，发生异常没被捕捉的话上传前执行，没异常的话else块后执行，即使处理过程中又发生异常也执行，基本用于清理如关闭文件、断开连接；没捕捉的异常会上传，如果传到默认异常处理会中断运行并打印信息；当异常发生Python执行第一个（从上至下，括号内从左至右，但这没什么意义，右面重复出现的是废代码但合法）符合引发的异常的except块，因此except:被语言要求必须放最后否则覆盖；一旦捕捉异常，不可能跳回异常发生的地方

610 2.5版本之前finally和其他不能共存，可以通过嵌套合并；单raise重新引发当前异常，常用于把刚捕捉的异常传递给另一个处理器或要捕捉但不希望它在此死掉时；任何时刻最多只有一个异常处于激活状态，一旦被捕捉就死了；raise时可以附加数据，except后的异常名称后可以接收

614 assert test, data

等价于

if \_\_debug\_\_: #一般自动设置为1，可以用命令行控制，移除这些常在调试时用的assert

if not test:

raise AssertionError, data

617 with expression[ as variable]:block语句与环境管理协议：运行时1.计算expression，得到环境管理器，有\_\_enter\_\_和\_\_exit\_\_方法 2.调用\_\_enter\_\_方法，如果as存在则其返回值赋给variable，否则返回值丢弃 3.块执行 4.如果引发异常则调用\_\_exit\_\_方法，含异常细节，如果方法返回False（无return则默认return None也是False）则异常重新引发，否则终止 5.如果没异常则依然调用\_\_exit\_\_方法，传入None，返回值无所谓

622 基于类的异常的优势：有附加的状态信息，支持继承；只要except分句列举了异常的类或任何超类名则可匹配，所以新增子类扩展异常层次不会破坏现有的代码

629 内置异常都在\_\_builtin\_\_中，也可以是exception模块的属性，是组织好的类：Exception异常顶层根超类，StandartError内置错误异常超类，ArithmeticError数值错误超类这三者就是一个有代表性的继承链；捕捉Exception几乎等于except:，但无法捕捉用户定义的不继承Exception的异常

631 except 类对象名, variable:时传入variable的是异常实例对象；经试验推断Exception有个\_\_init\_\_方法获取任意多参数，有个\_\_str\_\_方法用这些参数构造字符串作为对象的表达形式，默认异常处理会打印异常实例对象，以此为基点理解打印过程出现的各种现象，不过当然可以自己接手一切反正都会按照类的规则行动；

633 完整的raise：

raise instance #raise instance.\_\_class\_\_, instance

raise class, instance

raise class #raise class()

raise class, arg #raise class(arg)

raise class, (arg, …) #raise class(arg,…)

raise #重新引发异常

2和4的区分：书上说如果raise语句中没有实例对象，Python就会创建实例

637 try可以嵌套，Python会在运行时将try放入堆栈；简单地说：你对try嵌套的直觉没错，记住被抓就挂，逃走就上传就好；

641 错误都是异常但异常不都是错误，也可当信号用

644 sys.exc\_info()函数返回三元组，如没发生过异常就返回三个None，如果有就是最近一次异常的(exceptionType, exceptionInstance, traceback)

645 经常失败的运算应当放在try里；慎用except:，因为可能拦截高层try期待的异常或应当暴露出来的程序设计错误，甚至破坏基于异常的功能；用超类对异常分类以避免未来增加新异常时候的维护问题，子类异常会被父类捕捉因此不用改代码填异常

646 sys.exit(statuscode)通过引发SystemExit异常来终止程序，服从try的运行逻辑如finally；os.\_exit(statuscode)是立即强制终止，什么都不干了

651 常用工具：PyChecker找错；PyUnit和doctest测试；profile配置工具打印报告测试性能；pdb调试器；Psyco实时编译优化