第一章

一、解答题

1、Python 是什么类型的语言？

答：Python 是脚本语言

脚本语言（Scripting language）是电脑编程语言，因此也能让开发者藉以编写出让电脑听命行事的程序。以简单的方式快速完成某些复杂的事情通常是创造脚本语言的重要原则，基于这项原则，使得脚本语言通常比 C 语言、C++语言 或 java 之类的系统编程语言要简单容易。

也让脚本语言另有一些属于脚本语言的特性：

- 语法和结构通常比较简单
- 学习和使用通常比较简单
- 通常以容易修改程序的“解释”作为运行方式，而不需要“编译”
- 程序的开发产能由于运行性能

一个脚本可以使得本来要用键盘进行的相互操作自动化。一个 shell 脚本主要由原本需要在命令行输入的命令组成，或在一个文本编辑器中，用户可以使用脚本来把一些常用的操作组合成一组串行。主要用来书写这种脚本的语言叫做脚本语言。很多脚本语言实际上已经超过简单的用户命令串行的指令，还可以编写更复杂的程序。

2、IDLE 是什么？

答：IDLE 是一个 Python Shell，shell 的意思就是“外壳”，基本上来说，就是一个通过键入文本与程序交互的途径！像我们 Windows 那个 cmd 窗口，像 Linux 那个黑乎乎的命令窗口，他们都是 shell，利用他们，我们就可以给操作系统下达命令。同样的，我们可以利用 IDLE 这个 shell 与 Python 进行互动。

3、print()的作用？

答：print() 会在输出窗口中显示一些文本（在这一讲中，输出窗口就是 IDLE shell 窗口）。

4、Python 中表示乘法的符号是那个？

答：Python 中的乘号是*（星号）。

5、为什么 >>>print('I love tarena.com ' * 5) 可以正常执行，但 >>>print('I love tarena.com ' + 5) 却报错？

答：在 Python 中不能把两个完全不同的东西加在一起，比如说数字和文本，正是这个原因，>>>print('I love bj.tedu.cn ' + 5) 才会报错。这就像是在说“五只小鱼加上苍井空会是多少？”一样没有多大意义，结果可能是五，可能是六，也可能是八！不过乘以一个整数来翻倍就具有一定的意义了，前边的例子就是将 "I love bj.tedu.cn" 这个字符串打印五次。

6、现我需要在一个字符串中嵌入一个双引号，正确的做法是？

答：你有两个选择：可以利用反斜杠（\）对双引号转义：\"，或者用单引号引起这个字符串。例如：' I l"o"ve bj.tedu.cn '。

7、你知道什么是 BIF 吗？

答：BIF 就是 Built-in Functions，内置函数。为了方便程序员快速编写脚本程序（脚本就是要编程速度快快快!!!），Python 提供了非常丰富的内置函数，我们只需要直接

调用即可，例如 print() 的功能是“打印到屏幕”，input() 的作用是接收用户输入（注：Python3 用 input() 取代了 Python2 的 raw_input()）。

8、用什么方法可以知道 Python3 提供了多少个 BIF？

答：在 Python 或 IDLE 中，输入 dir(__builtins__) 可以看到 Python 提供的内置方法列表（注意，builtins 前后是两个下划线哦）其中小写的就是 BIF。如果想具体查看某个 BIF 的功能，比如 input()，可以在 shell 中输入 help(input)，就会得到这个 BIF 的功能描述。哦，答案应该是 68 个，不信你自己数数看。

9、在 Python 看来：'Tarena' 和 'tarena' 一样吗？

答：不一样，因为 Python 是一个“敏感的小女孩”，所以不要试图欺骗她，对 Python 来说，Tarena 和 tarena 是完全不同的两个名字，所以编程的时候一定要当心。不过 Python 会帮助解决可能因此出现的问题，例如只有当标识符已经赋值后（还记得吗，在课堂中说过 Python 的变量是不用先声明的）才能在代码中使用，未赋值的标识符直接使用会导致运行时错误，所以你很快就可以根据经验发现此问题。

10、在你们看来，Python 中什么是最重要的？

答：缩进！缩进是 Python 的灵魂，缩进的严格要求使得 Python 的代码显得非常精简并且有层次。所以在 Python 里对待缩进代码要十分小心，如果没有正确地缩进，代码所做的事情可能和你的期望相去甚远（就像 C 语言里边括号打错了位置）。如果在正确的位置输入冒号“:”，IDLE 会自动将下一行缩进！

11、课堂中的例子中出现了“=”和“==”，他们表示不同的含义，你在编程的过程中会不小心把“==”误写成“=”吗？有没有好的办法可以解决这个问题呢？

答：C 语言的话，如果 if(c == 1) 写成 if(c = 1)，程序就完全不按程序员原本的目的去执行，但在 Python 这里，不好意思，行不通，语法错误！Python 不允许 if 条件中赋值，所以 if c = 1: 会报错！

12、你听说过“拼接”这个词吗？

答：在一些编程语言，我们可以将两个字符串“相加”在一起，如：'I' +'Love' +'Tarena' 会得到 'ILoveTarena'，在 Python 里，这种做法叫做拼接字符串。

第二章

一、解答题

1、在 Python 中，int 表示整型，那你还记得 bool、float 和 str 分别表示什么吗？

答：bool　表示布尔类型
　　float　表示浮点数
　　str　　表示字符串

2、你知道为什么布尔类型(bool)的 True 和 False 分别用 1 和 0 来代替吗？

答：你可能听说过计算机是很“笨”的，究其根本是因为它只认识二进制数，所以所有的编程语言最终都会转换成简单的二进制序列给 CPU 按照一定的规则解析。

由于二进制只有两个数：0 和 1，因此用 0 和 1 来表示 False 和 True 再适合不过了，因为不用浪费资源在转换的过程上！

3、使用 int() 将小数转换为整数，结果是向上取整还是向下取整呢？

答：小数取整会采用比较暴力的截断方式，即向下取整。（注：5.5 向上取整为 6，向下取整为 5）

4、阅读代码，说出程序的执行结果。

```
while "C":
    print("I Love Tarena")
```

答：死循环，会一直打印“I Love Tarena!”（嗯，这也算是永远支持达内的方法之一），直到崩溃或者用户按下快捷键 CTRL + C（强制结束）

造成死循环的原因是 while 后边的条件永远为真(True)，在 Python 看来，只有以下内容会被看作假（注意冒号括号里边啥都没有，连空格都不要有!）：False None 0 "" '' () [] {}，其他一切都被解释为真！

5、请问以下代码会打印多少次“I Love Tarena!”

```
i = 10
while i:
    print("I Love Tarena")
    i =i - 1
```

答：会打印 10 次。

6、请写出与 10 < cost < 50 等价的表达式

答：(10 < cost) and (cost < 50)

7、Python3 中，一行可以书写多个语句吗？

答：可以，语句之间用分号隔开即可，不妨试试：

>>>print(“I Love Tarena”);print(“very much!”)

8、Python3 中，一个语句可以分成多行书写吗？

答：可以，一行过长的语句可以使用反斜杠或者括号分解成几行，不妨试试：

```
>>> 3 > 4 and \
    1 < 2
```

或者

```
(3 > 4 and
    1 < 2)
```

9、了解什么是“短路逻辑（short-circuit logic）”吗？

答：逻辑操作符有个有趣的特性：在不需要求值的时候不进行操作。这么说可能比较“高深”，举个例子，表达式 x and y，需要 x 和 y 两个变量同时为真(True)的时候，结果才为真。因此，如果当 x 变量得知是假(False)的时候，表达式就会立刻返回 False，而不用去管 y 变量的值。这种行为被称为短路逻辑（short-circuit logic）或者惰性求值（lazy evaluation），这种行为同样也应用与 or 操作符。实际上，Python 的做法是如果 x 为假，表达式会返回 x 的值(0)，否则它就会返回 y 的值

第三章

一、解答题

1、我们人类思维是习惯于“四舍五入”法，你有什么办法使得 int() 按照“四舍五入”的方式取整吗？

答：int() 固然没那么“聪明”，但机器是死的，人是活的！

5.4 “四舍五入”结果为：5，int(5.4+0.5) == 5

5.6 “四舍五入”结果为：6，int(5.6+0.5) == 6

4、查看一个变量的类型，可以使用 type() 和 isinstance()，你更倾向于使用哪个？

答：建议使用 isinstance()，因为它的返回结果比较直接，另外 type() 其实并没有你想象的那么简单。

5、Python3 可以给变量命名中文名，知道为什么吗？

答：Pyhton3 源码文件默认使用 utf-8 编码（支持中文）

6、Python 的 floor 除法现在使用 “//” 实现，那 3.0 // 2.0 的结果会是什么呢？

答：如果回答是 1.5 那么很遗憾，您受到 C 语言的影响比较大，Python 这里会义无反顾地执行 floor 除法原则，答案是：1.0

7、a < b < c 事实上是等于？

答：(a < b) and (b < c)

8、不使用 IDLE，你可以轻松说出 5 ** -2 的值吗？

答：0.04，也就是 1/25

幂运算操作符比其左侧的一元操作符优先级高，比其右侧的一元操作符优先级低。

9、如何简单判断一个数是奇数还是偶数？

答：使用求余可以简单得到答案：能被 2 整除为偶数，所以 x % 2 == 0，否则为奇数。

10、请用最快速度说出答案：not 1 or 0 and 1 or 3 and 4 or 5 and 6 or 7 and 8 and 9

答：如果你的回答是 0，那么很开心你中招了！

答案是：4

not or and 的优先级是不同的：not > and > or

我们按照优先级给它们加上括号：(not 1) or (0 and 1) or (3 and 4) or (5 and 6) or (7 and 8 and 9)

== 0 or 0 or 4 or 6 or 9

== 4

为啥是 4？

大家还记得第四讲作业提到的“短路逻辑”吗？3 and 4 == 4，而 3 or 4 == 3。

所以答案是：4

温馨提示：为了更好的表达你的程序，Tarena 再次呼吁有些括号还是不能省下的，毕竟不是所有程序员都跟你一样都将优先级烂透于心的。

11、还记得求闰年的作业吗？如果还没有学到“求余”操作，还记得用什么方法可以代替“%”的功能呢？

答： “if year/400 == int(year/400)”这样的方式来代替。

第四章

一、解答题

1、if not (money < 100): 上边这行代码相当于？

答：if money >= 100:

2、assert 的作用是什么？

答：assert 这个关键字我们称之为“断言”，当这个关键字后边的条件为假的时候，程序自动崩溃并抛出 AssertionError 的异常。

什么情况下我们会需要这样的代码呢？当我们在测试程序的时候就很好用，因为与其让错误的条件导致程序今后莫名其妙地崩溃，不如在错误条件出现的那一瞬间我们实现“自爆”。

一般来说我们可以用 Ta 再程序中置入检查点，当需要确保程序中的某个条件一定为真才能让程序正常工作的话，assert 关键字就非常有用了。

3、假设有 x = 1，y = 2，z = 3，请问如何快速将三个变量的值互相交换？

答：x, y, z = z, y, x

4、猜猜 (x < y and [x] or [y])[0] 实现什么样的功能？

答：这其实是 Python 的作者还没有为 Python 加入三元操作符之前，Python 社区的小伙伴们灵活的使用 and 和 or 搭配来实现三元操作符的功能，这里边有涉及到列表和切片的知识，这部分知识迫不及待的朋友可以先稍微预习下。

5、了解成员资格运算符吗？

答：Python 有一个成员资格运算符：in，用于检查一个值是否在序列中，如果在序列中返回 True，否则返回 False。

例如：

```
>>> name = 'Python'
>>> 'P' in name
True
>>> 'a' in name
False
```

当我们讲解 for 语句的时候，你看到 in 会不会就有了更深入的理解？！

6、下面的循环会打印多少次"I Love Python"？

```
for i in range(0, 10, 2):
    print('I Love Python)
```

答：5 次，因为从 0 开始，到 10 结束，步进为 2。

7、下面的循环会打印多少次"I Love Python"？

```
for i in 5:
    print('I Love Python)
```

答：会报错，上节课的课后习题我们提到了 in 是“成员资格运算符”，而不是像 C 语言那样去使用 for 语法。Python 的 for 更像脚本语言的 foreach。

8、请说明 break 和 continue 在循环中起到的作用？

答：break 语句的作用是终止当前循环，跳出循环体。

continue 语句的作用是终止本轮循环并开始下一轮循环

（这里要注意的是：在开始下一轮循环之前，会先测试 循环条件）。

9、请问 range(10) 生成哪些数？

答：会生成 range(0, 10)，list(range(0, 10)) 转换成列表是：[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]，注意

不包含 10 哦。

10、阅读以下程序会打印什么？

```
while True:
    while True:
        break
        print(1)
    print(2)
    break
print(3)
```

答：会打印：

2

3

因为 break 只能跳出一层循环，记住咯！

11、什么情况下我们要使循环永远为真？

答：

```
while Ture:
 循环体
```

同样用于游戏实现，因为游戏只要运行着，就需要时刻接收用户输入，因此使用永远为真确保游戏“在线”。操作系统也是同样的道理，时刻待命，操作系统永远为真的这个循环叫做消息循环。另外，许多通讯服务器的客户端/服务器系统也是通过这样的原理来工作的。

所以永远为“真”虽然是“死循环”，但不一定是坏事，再说了，我们可以随时用 break 来跳出循环！

12、 【学会提高代码的效率】你的觉得以下代码效率方面怎样？有没有办法可以大幅度改进(仍然使用 while)？

```
i = 0
string = ILoveTarena.com'
while i < len(string)):
    print(i)
    i += 1
```

这段代码之所以“效率比较低”是因为每次循环都需要调用一次 len() 函数（我们还没有学到函数的概念，tarena 这里为零基础的朋友形象的解释下：就像你打游戏打得正 HIGH 的时候，老妈让你去买盐……你有两种选择，一次买一包，一天去买五次，或者一次性买五包回来，老妈要就直接给她。）

```
i = 0
string = ILoveTarena.com'
length = len(string)
while i < length:
    print(i)
    i += 1
```

tarena 希望学习咱的课程的朋友不是只为了可以把程序写出来，而是追求把代码写好，写漂亮，做一个有理想、有追求的程序猿！

第五章

一、解答题

1、列表都可以存放一些什么东西？

答：我们说 Python 的列表是一个打了激素的数组，如果把数组比喻成集装箱，那么 Python 的列表就是一个大仓库，Ta 可以存放我们已经学习过的任何数据类型。

```
>>> mix = [1, 'tarena', 3.14, [1, 2, 3]]
```

2、向列表增加元素有哪些方法？

答：三种方法向列表增加元素，分别是：append()、extend() 和 insert()。

3、append() 方法和 extend() 方法都是向列表的末尾增加元素，请问他们有什么区别？

答：

append() 方法是将参数作为一个元素增加到列表的末尾。

extend() 方法则是将参数作为一个列表去扩展列表的末尾。

请看以下示例：

```
>>> name = ['P', 'y', 't', 'h', 'o', 'n']
>>> name.append('T')
>>> name
['P', 'y', 't', 'h', 'o', 'n', 'T']
>>> name.extend(['a', 'r'])
>>> name
['P', 'y', 't', 'h', 'o', 'n', 'T', 'a', 'r']
>>> name.append(['e', 'n', 'a'])
>>> name
['P', 'y', 't', 'h', 'o', 'n', 'T', 'a', 'r'], ['e', 'n', 'a']]
```

4、member.append(['竹林小溪', 'Crazy 迷恋']) 和 member.extend(['竹林小溪', 'Crazy 迷恋']) 实现的效果一样吗？

答：不一样，因为怕大家没有仔细看上一题的示例，所以不懂的请看上一题解释。

5、有列表 name = ['P', 'y', 'h', 'o', 'n']，如果我想要在元素 'y' 和 'h' 之间插入元素 't'，应该使用什么方法来插入？

答：name.insert(2, 't')

6、请问如何将下边这个列表的'达内'修改为'达内教育'？

```
list1 = [1, [1, 2, ['达内']], 3, 5, 8, 13, 18]
```

答：

```
list1 = [1, [1, 2, ['达内']], 3, 5, 8, 13, 18]
list1[1][2][0] = '达内教育'
```

7、要对一个列表进行顺序排序，请问使用什么方法？

答：列表名.sort()

8、要对一个列表进行逆序排序，请问使用什么方法？

答：

```
列表名.sort()
列表名.reverse()
```

或者

```
列表名.sort(reverse=True)
```

9、列表有两个内置方法 copy() 和 clear()，说说他们的作用。

答： copy() 方法跟使用切片拷贝是一样的：

```
>>> list2 = list1.copy()
>>> list2
[1, [1, 2, ['达内']], 3, 5, 8, 13, 18]
```

clear() 方法用于清空列表的元素，但要注意，清空完后列表仍然还在哦，只是变成一个空列表。

```
>>> list2.clear()
>>> list2
[]
```

10、你有听说过列表推导式或列表解析吧！我们看表达式：

```
>>> [ i*i for i in range(10) ]
```

你觉得会打印什么内容？

```
>>> [i*i for i in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

居然分别打印了 0 到 9 各个数的平方，然后还放在列表里边了有木有？！

列表推导式（List comprehensions）也叫列表解析，灵感取自函数式编程语言 Haskell。Ta 是一个非常有用和灵活的工具，可以用来动态的创建列表，语法如：

[有关 A 的表达式 for A in B]

例如

```
>>> list1 = [x**2 for x in range(10)]
>>> list1
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

相当于

```
list1 = []
for x in range(10):
        list1.append(x**2)
```

问题：请先在 IDLE 中获得下边列表的结果，并按照上方例子把列表推导式还原出来。

答：

```
>>> list1 = [(x, y) for x in range(10) for y in range(10) if x%2==0 if y%2!=0]
list1 = []
for x in ragne(10):
    for y in range(10):
        if x%2 == 0:
            if y%2 != 0:
                list1.append((x, y))
```

第六章

一 、解答题

1、请用一句话描述什么是列表？再用一句话描述什么是元组？

答：列表：一个大仓库，你可以随时往里边添加和删除任何东西。

元组：封闭的列表，一旦定义，就不可改变（不能添加、删除或修改）。

2、什么情况下你需要使用元组而不是列表？

答：当我们希望内容不被轻易改写的时候，我们使用元组。

当我们需要频繁修改数据，我们使用列表。

3、当元组和列表掉下水，你会救谁？

答：如果是我，我会救列表，因为列表提供了比元组更丰富的内置方法，这相当大的提高了编程的灵活性。

回头来看下元组，元组固然安全，但元组一但创建就无法修改（除非通过新建一个元组来间接修改，但这就带来了消耗），而我们人是经常摇摆不定的，所以元组只有在特殊的情况才用到，平时还是列表用的多。

4、创建一个元组，什么情况下逗号和小括号必须同时存在，缺一不可？

答：在拼接只有一个元素的元组的时候，例如我们课上举的例题：

```
>>> temp = (‘鲜花’, ‘黑夜’, ‘迷途’, ‘小布丁’)
# 如果我想在“黑夜”和“迷途”之间插入“怡静”，我们应该：
>>> temp = temp[:2] + (‘怡静’,) + temp[2:]
```

5、 x, y, z = 1, 2, 3 请问 x, y, z 是元组吗？

答：不是，所有的多对象的、逗号分隔的、没有明确用符号定义的这些集合默认的类型都是元组，

自己在交互模式下键入以下代码，并体会一下：

```
>>> x, y, z = 1, 2, 3
>>> type(x)
>>> h = x, y, z
>>> type(h)
```

6、请写出以下情景中应该使用列表还是元组来保存数据：

1）游戏中角色的属性：

2）你的身份证信息：

3）论坛的会员：

4）团队合作开发程序，传递给一个你并不了解具体实现的函数的参数：

5）航天火箭各个组件的具体配置参数：

6）NASA 系统中记录已经发现的行星数据：

答：

1) 游戏中角色的属性：列表
2) 你的身份证信息：元组
3) 论坛的会员：列表
4) 团队合作开发程序，传递给一个你并不了解具体实现的函数的参数：元组
5) 航天火箭各个组件的具体配置参数：元组
6) NASA 系统中记录已经发现的行星数据：列表

7、大家知道“列表推导式”，那请问如果我把中括号改为小括号，会不会得到“元组推导式”呢？

答：Python3 木有“元组推导式”，为嘛？没必要丫，有了“列表推导式”已经足够了。

那为什么“>>> tuple1 = (x**2 for x in range(10))”不会报错？

因为你误打误撞得到了一个生成器：

```
>>> type(tuple1)
<class 'generator'>
```

8、我们根据列表、元祖和字符串的共同特点，把它们三统称为什么？

答：序列，因为他们有以下共同点：

1）都可以通过索引得到每一个元素

2）默认索引值总是从 0 开始（当然灵活的 Python 还支持负数索引）

3）可以通过分片的方法得到一个范围内的元素的集合

4）有很多共同的操作符（重复操作符、拼接操作符、成员关系操作符）

9、请问分别使用什么 BIF，可以把一个可迭代对象转换为列表、元祖和字符串？

答：

list([iterable]) 把可迭代对象转换为列表

tuple([iterable]) 把可迭代对象转换为元祖

str(obj) 把对象转换为字符串

例如：

```
1. >>> temp = 'I love tedu.com!'
2. >>> list(temp)
3. ['I', ' ', 'l', 'o', 'v', 'e', ' ', 't', 'e', 'd', 'u', '.', 'c', 'o', 'm', '!']
```

10、你还能复述出“迭代”的概念吗？

答：所谓迭代，是重复反馈过程的活动，其目的通常是为了接近并到达所需的目标或结果。每一次对过程的重复被称为一次“迭代”，而每一次迭代得到的结果会被用来作为下一次迭代的初始值

11、你认为调用 max('I love tedu.com') 会返回什么值？为什么？

答：会返回：'v'，因为字符串在计算机中是以 ASCII 码的形式存储（ASCII 对照表：参数中 ASCII 码值最大的是'v'对应的 118。

第七章

一、解答题

1、还记得如何定义一个跨越多行的字符串吗（请至少写出两种实现的方法）？

答：

方法一：

```
>>> str1 = '''待我长发及腰，将军归来可好？
此身君子意逍遥，怎料山河萧萧。
天光乍破遇，暮雪白头老。
寒剑默听奔雷，长枪独守空壕。
醉卧沙场君莫笑，一夜吹彻画角。
江南晚来客，红绳结发梢。'''
```

方法二：

```
>>> str2 = '待卿长发及腰，我必凯旋回朝。\
昔日纵马任逍遥，俱是少年英豪。\
东都霞色好，西湖烟波渺。\
执枪血战八方，誓守山河多娇。\
应有得胜归来日，与卿共度良宵。\
盼携手终老，愿与子同袍。'
```

方法三：

```
>>> str3 = ('待卿长发及腰，我必凯旋回朝。'
'昔日纵马任逍遥，俱是少年英豪。'
'东都霞色好，西湖烟波渺。'
'执枪血战八方，誓守山河多娇。'
'应有得胜归来日，与卿共度良宵。'
'盼携手终老，愿与子同袍。')
```

2、三引号字符串通常我们用于做什么使用？

答：三引号字符串不赋值的情况下，通常当作跨行注释使用，例如：

```
'''这是一个三引号字符串用于注释的例子，
例子虽然只是简简单单的一句话，
却毫无遮掩地体现了作者用情至深，
所谓爱至深处情至简！'''
```

3、file1 = open('C:\windows\temp\readme.txt', 'r') 表示以只读方式打开“C:\windows\temp\readme.txt”这个文本文件，但事实上这个语句会报错，知道为什么吗？你会如何修改？

答： 会报错是因为在字符串中，我们约定“\t”和“\r”分别表示“横向制表符（TAB）”和“回车符”（详见：http://bbs.fishc.com/thread-39140-1-1.html），因此并不会按照我们计划的路径去打开文件。

Python 为我们铺好了解决的道路，只需要使用原始字符串操作符（R 或 r）即可：

```
>>> file1 = open(r'C:\windows\temp\readme.txt', 'r')
```

4、有字符串：str1 = '达内资源打包'，请问如何提取出子字符串：'www.tedu.com'

答：>>> str1[16:28]

5、如果使用负数作为索引值进行分片操作，按照第 4 题的要求你能够正确目测出结果吗？

答：>>> str1[-44:-32]

6、还是第 4 题那个字符串，请问下边语句会显示什么内容？

答：>>> str1[20:-36]

tedu

7、据说只有智商高于 150 的人才能解开这个字符串（还原为有意义的字符串）：str1 = 'i2sl54ovvvb4e3btere32d56u'

答： >>> str1[::3]

8、根据说明填写相应的字符串格式化符号

符　号	说　明
%	格式化字符及其 ASCII 码
%	格式化字符串
%	格式化整数
%	格式化无符号八进制数
%	格式化无符号十六进制数

%	格式化无符号十六进制数（大写）
%	格式化定点数，可指定小数点后的精度
%	用科学计数法格式化定点数
%	根据值的大小决定使用%f 或者%e
%	根据值的大小决定使用%F 或者%E

答：

符　号	说　明
%c	格式化字符及其 ASCII 码
%s	格式化字符串
%d	格式化整数
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数（大写）
%f	格式化定点数，可指定小数点后的精度
%e	用科学计数法格式化定点数
%E	作用同%e，用科学计数法格式化定点数
%g	根据值的大小决定使用%f 或者%e
%G	作用同%g，根据值的大小决定使用%F 或者%E

9、以下程序输出结果？

```
>>> "{{1}}".format("不打印", "打印")
```

答：'{1}'

10、以下代码中，a，b，c 是什么参数？

```
>>> "{a} love {b}.{c}".format(a="I", b="tedu", c="com")
'I love tedu.com'
```

答：关键字参数

11、以下代码中，{0}，{1}，{2}是什么参数？

```
>>> "{0} love {1}.{2}".format("I", "tedu", "com")
'I love tedu.com'
```

答：位置参数

12、如果想要显示 Pi = 3.14，format 前边的字符串应该怎么填写呢？

```
''.format('Pi = ', 3.1415)
```

答：

```
>>> '{0}{1:.2f}'.format('Pi = ', 3.1415)
'Pi = 3.14'
```

第八章

一、解答题

1.、知道什么是 DRY 吗？

答：DRY 是程序员们公认的指导原则：Don't Repeat Yourself.

快快武装你的思维吧，拿起函数，不要再去重复拷贝一段代码了！

2.、都是重复一段代码，为什么我要使用函数（而不使用简单的拷贝黏贴）呢？

答：使用函数:

0) 可以降低代码量（调用函数只需要一行，而拷贝黏贴需要 N 倍代码）

1) 可以降低维护成本（函数只需修改 def 部分内容，而拷贝黏贴则需要每一处出现的地方都作修改）

2) 使序更容易阅读（没有人会希望看到一个程序重复一万行"I love Tarena"）

3、 函数可以有多个参数吗？

答：可以的，理论上你想要有多少个就可以有多少个，只不过如果函数的参数过多，在调用的时候出错的机率就会大大提高，因而写这个函数的程序员也会被相应的问候祖宗，所以，尽量精简吧，在 Python 的世界里，精简才是王道！

4、 创建函数使用什么关键字，要注意什么？

答：使用"def"关键字，要注意函数名后边要加上小括号"()"，然后小括号后边是冒号":"，然后缩进部分均属于函数体的内容，例如：

```
1.  def MyFun():
2.      # 我是函数体
3.      # 我也是函数体
4.      # 我们都属于函数 MyFun()
5.
6.  # 噢，我不属于 MyFun()函数的了
```

5.、请问这个函数有多少个参数？

```
1.  def MyFun((x, y), (a, b)):
2.      return x * y - a * b
```

答：如果你回答两个，那么恭喜你错啦，答案是 0，因为类似于这样的写法是错误的！

我们分析下，函数的参数需要的是变量，而这里你试图用"元祖"的形式来传递是不可行的。

我想你如果这么写，你应该是要表达这么个意思：

```
1.  >>> def MyFun(x, y):
2.          return x[0] * x[1] - y[0] * y[1]
3.
4.  >>> MyFun((3, 4), (1, 2))
```

6、 请问调用以下这个函数会打印什么内容？

```
1.  >>> def hello():
2.          print('Hello World!')
3.          return
```

```
4.          print('Welcome To Tarena!')
```

答：会打印：

```
1.  >>> hello()

2.  Hello World!
```

因为当 Python 执行到 return 语句的时候，Python 认为函数到此结束，需要返回了（尽管没有任何返回值）。

7、请问以下哪个是形参哪个是实参？

```
01.  def MyFun(x):
02.      return x ** 3
03.
04.  y = 3
05.  print(MyFun(y))
06.
```

X 是实际参数（实参），y 是形式参数（形参），跟绝大部分编程语言一样，形参指的是函数创建过程中小括号里的参数，而实参指的是函数在调用过程中传递进去的参数。

8、函数文档和直接用”#”为函数写注释有什么不同？

给函数写文档是为了让让别人可以更好的理解你的函数，所以这是一个好习惯。

```
01.  >>> def MyFirstFunction(name):
02.          '函数文档在函数定义的最开头部分，用不记名字符串表示'
03.          print('I love FishC.com!')
```

```
def    myfirstfunction(name):
    ‘函数文档在函数定义的最开头部分，用不记名字符串表示’
    print(‘I love tarena’)
```

我们看到在函数开头写下的字符串 TA 是不会打印出来的

函数的文档字符串可以按如下方式访问：

```
01.  >>> MyFirstFunction.__doc__
02.  '函数文档在函数定义的最开头部分，用不记名字符串表示'
```

另外，我们用 help（）来访问这个函数也可以看到这个文档字符串：

```
01.  >>> help(MyFirstFunction)
02.  Help on function MyFirstFunction in module __main__:
03.
04.  MyFirstFunction(name)
05.  函数文档在函数定义的最开头部分，用不记名字符串表示
06.
```

9、使用关键字参数，可以有效避免什么问题的出现？

关键字参数，是指函数在调用的时候，带上参数的名字去指定具体调用的是哪个参数，从而可以不用按照参数的顺序调用函数，例如：

```
def    SaySome(name,words):
    print(name + ‘->’ + words)
SaySome(words=’让编程改变世界！’，name=’tarena’)
```

使用关键字参数，可以有效避免因不小心搞乱参数的顺序导致的 BUG 出现

10、使用 help(print)查看 print()这个 BIF 有哪些默认参数？分别起到什么作用？

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    # 文件类型对象，默认是sys.stdout（标准输出流）
    sep:   string inserted between values, default a space.
    # 第一个参数如果有多个值（第一个参数是收集参数），各个值之间默认用空格（space）隔开
    end:   string appended after the last value, default a newline.
    # 打印最后一个值之后默认参数一个新行标识符('\n')
    flush: whether to forcibly flush the stream.
    # 是否强制刷新流
```

4. 默认参数和关键字参数表面最大的区别是什么？

呃，其实这道题考大家有没有认真听课啦~

关键字参数是在函数调用的时候，通过参数名制定需要赋值的参数，这样做就不怕因为搞不清参数的顺序而导致函数调用出错。而默认参数是在参数定义的过程中，为形参赋初值，当函数调用的时候，不传递实参，则默认使用形参的初始值代替。

11、默认参数和关键字参数表面最大的区别是什么？

呃，其实这道题考大家有没有认真听课啦~

关键字参数是在函数调用的时候，通过参数名制定需要赋值的参数，这样做就不怕因为搞不清参数的顺序而导致函数调用出错。而默认参数是在参数定义的过程中，为形参赋初值，当函数调用的时候，不传递实参，则默认使用形参的初始值代替。

第九章

一、解答题

1、如果希望在函数中修改全局变量的值，应该使用什么关键字？

global 关键字

举个例子：

```
>>> count = 5
>>> def MyFun():
                global count
                count = 10
                print(count)

>>> MyFun()
10
>>> count
10
```

2、在嵌套函数中，如果希望在内部函数修改外部函数的局部变量，应该使用什么关键字？

nonlocal 关键字

举个例子：

```
>>> def Fun1():
                x = 5
                def Fun2():
                        nonlocal x
                        x *= x
                        return x
                return Fun2()

>>> Fun1()
25
```

3、Python 的函数可以嵌套，但要注意访问的作用域问题，请问以下代码存在什么问题

```
def outside():
    print('I am outside!')
    def inside():
        print('I am inside!')

inside()
```

复制代码

使用嵌套函数要注意一点就是作用域问题，inside()函数是内嵌在outside()函数中的，所以inside()是人妻，除了身为老公的outside()可以碰（调用），在外边或者别的函数体里是无法对其进行调用的。

正确的调用应该是：

```
def outside():
    print('I am outside!')
    def inside():
        print('I am inside!')

    inside()
outside()
```

4、请问为什么代码 A 没有报错，但代码 B 却报错了？应该如何修改？

代码A：

```
def outside():
    var = 5
    def inside():
        var = 3
        print(var)

    inside()

outside()
```

复制代码

代码B：

```
def outside():
    var = 5
    def inside():
        print(var)
        var = 3

    inside()
outside()
```

仔细一看报错的内容是：UnboundLocalError: local variable 'var' referenced before assignment，说的是变量var没有被定义就拿来使用，肯定错啦！

这里outside()函数里有一个var变量，但要注意的是，内嵌函数inside()也有一个同名的变量，Python为了保护变量的作用域，故将outside()的var变量屏蔽起来，因此此时是无法访问到外层的var变量的。

应该修改为：

```
def outside():
    var = 5
    def inside():
        nonlocal var
        print(var)
        var = 8

    inside()
outside()
```

5、请问如何访问 funIn()呢？

```
def funOut():
    def funIn():
        print('宾果！你成功访问到我啦！')
    return funIn()

```

复制代码

只需要直接调用funOut()即可：

```
funOut()
宾果！你成功访问到我啦！

```

6、以下是“闭包”的一个例子，请你目测下会打印什么内容？

```
def funX():
    x = 5
    def funY():
        nonlocal x
        x += 1
        return x
    return funY

a = funX()
print(a())
print(a())
print(a())
```

会打印：

```
6
7
8
```

7、请使用 ambda 表达式将下面的函数转变为匿名函数

```
def fun_A(x, y=3):
        return x * y
```

答：

```
lambda x, y=3 : x * y
```

8、将限免的匿名函数变成普通的函数

```
lambda x : x if x % 2 else None
```

答：

```
def is_odd(x):
        if x % 2:
                return x
        else:
                return None
```

9、感受一下使用匿名函数后给你的编程生活带来的变化

答：a、python 写一写执行脚本时，使用匿名函数就可以省下定义函数的过程，比如说我们只是需要写个简单的脚本来管理服务器时间，我们就不需要专门定义一个函数然后再写调用，使用匿名函数就可以使得代码更加精简。

b、对于一些比较抽象并且整个程序执行下来只需要调用一两次的函数，有时候给函数

起名字也是比较头疼的问题，使用匿名函数就不需要考虑命名的问题了。

c、简化代码的可读性，由于普通的函数阅读经常要跳到开头 def 定义部分，使用匿名函数可以省去这样的步骤。

10、你可以利用 filter()和 lambda 表达式快速求出 100 以内所有 3 的倍数吗?

答：

```
01.  list(filter(lambda n : not(n%3), range(1, 100)))
02.
```

11、还记得列表推导式吗？完全可以使用列表推导式代替 fiter()和 lambda 组合，你可以做到吗？

答：

```
01.  [ i for i in range(1, 100) if not(i%3)]
02.
```

12、还记得 zip 吗？使用 zip 会将两数以元组的方式绑定在一起，例如:

```
01.  >>> list(zip([1, 3, 5, 7, 9], [2, 4, 6, 8, 10]))
02.  [(1, 2), (3, 4), (5, 6), (7, 8), (9, 10)]
03.
```

但如果我希望打包的形式是灵活多变的列表而不是元组(希望是[1,2].[3,4]...)

答：

```
01.  >>> list(map(lambda x, y : [x, y], [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]))
02.  [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
03.
```

map 是可以接受多个可迭代对象的

第十章

一、解答题

1、递归在编程上的形式是如何表现的呢？

答：在编程上，递归表现为函数调用本身这么一个行为。

举个例子（递归求阶乘）：

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
number = int(input("请输入一个整数:"))
result = factorial(number)
print("%d的阶乘是:%d" % (number,result))
```

2、递归必须满足那两个基本条件？

答：1）函数调用自身

2） 设置了正确的返回条件

3、思考一下，按照递归的特性，在编程中有没有不得不使用递归的情况？

答： 例如汉诺塔，目录索引（因为你永远不知道这个目录里边是否还有目录），快速排序（二十世纪十大算法之一），树结构的定义等如果使用递归，会事半功倍，否则会导致程序无法实现或相当难以理解。

4、用递归去计算阶乘问题或斐波那契数列是很糟糕的算法，你知道为什么吗？

答： 递归的实现可以是函数自个儿调用自个儿，每次函数的调用都需要进行压栈、弹栈、保存和恢复寄存器的栈操作，所以在这上边是非常消耗时间和空间的。

另外，如果递归一旦忘记返回，或者错误的设置了返回条件，那么执行这样的递归代码就会变成一个无底洞：只进不出！所以在写递归代码的时候，千万记住口诀：递归递归，归去来兮！出来混，总有一天是要还的！

5、递归的优缺点？

答：优点：

1) 递归的基本思想是把大规模的问题转变为规模小的问题组合，从而简化问题的解决难度。
2) 有些问题使用递归使得代码简单易懂。

缺点：

1) 由于递归的原理是函数调用自个儿，所以一旦大量的调用函数本身空间和时间消耗是“奢侈的”
2) 初学者很容易错误的设置了返回条件，导致递归代码无休止调用，最终栈溢出，程序崩溃。

6、用你自己的话解释一下“迭代”这个概念

答案：迭代是重复反馈过程的活动，其目的通常是为了接近并达到所需目标或结果，每次对过程的重复称为一次“迭代”，而每一次迭代得到的结果会作为下一次迭代的初始值

7、迭代器是一个容器吗？

答案：不是，容器是可以存放数据的，例如 str，list，tuple，而迭代器就是实现了__next__()方法的对象，用来遍历容器数据的对象

8、迭代器可以回退（获取上一个值）吗？

答案：迭代器的性质决定了迭代不能回退，只能向前进行迭代，我们在实际使用迭代器的过程中，几乎不会使用回退，所以也无所谓了

9、如何判断一个容器是否具有迭代功能？

答案：判断该容器是否有__iter__()和__next__()方法

10、for 语句是如何判断迭代器里边已经取空了？

答案：迭代器通过__next__()方法每次返回容器的一个元素，并指向下一个元素，如果当前已无元素，通过抛出 StopIteration 异常表示

11、在 Python 原生的数据对象中，哪一个只能只能用迭代器进行访问

答案：set。对于原生支持顺序访问的容器类型，可以使用下标索引或迭代器进行访问，但是对于无法顺序访问的数据类型 set，只能使用迭代器进行访问

第十一章

一、解答题

1、当你听到小伙伴们再谈论“映射”,”哈希”,“散列”或者”关系数组”的时候，事实上他们就是在讨论什么呢？

答：事实上他们都是在讨论字典，都是一个概念！

2、尝试一下将数据(“F”:70,”C”:67,”h”:104,”i”:105,”s”:115)创建为一个字典并访问键“C”对应的值？

```
>>> MyDict = dict((('F', 70), ('i',105), ('s',115), ('h',104), ('C',67)))
>>> MyDict_2 = {'F':70, 'i':105, 's':115, 'h':104, 'C':67}
>>> type(MyDict)
<class 'dict'>
>>> type(MyDict_2)
<class 'dict'>
>>> MyDict['C']
67
```

3、用方括号(“[]”)括起来的数据我们叫做列表，那么使用大括号(“{}”)括起来的数据我们就叫做字典，对吗？

答：不对

```
>>> NotADict = {1, 2, 3, 4, 5}
>>> type(NotADict)
<class 'set'>

复制代码
```

4、你如何理解有些东西字典做得到，但“万能的”的列表难以实现？

举个例子：

```
In [5]: brand = ["李宁","耐克","阿迪达斯","tarena"]

In [6]: slogan = ["一切皆有可能","Just do it","Impossible is nothing","让编程改变世界"]

In [7]: print("tarena 的口号是: ",slogan[brand.index("tarena")])
tarena 的口号是:  让编程改变世界
```

答：列表 brand,slogan 的索引和相对的值是没有任何关系的，我们可以看出唯一有联系的就是两个列表间，索引号相同的元素是有关系的，所以，我们需要有字典这种映射类型的出现：

```
In [5]: brand = ["李宁","耐克","阿迪达斯","tarena"]

In [6]: slogan = ["一切皆有可能","Just do it","Impossible is nothing","让编程改变世界"]

In [7]: print("tarena 的口号是: ",slogan[brand.index("tarena")])
tarena 的口号是:  让编程改变世界
```

5、下边这些代码，他们都在执行一样的操作吗？你看得出差别吗？

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([('two', 2), ('one', 1), ('three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})

复制代码
```

答：是的，他们都在创建字典：a = dict(one = 1,two = 2,three =3),看不出差别

6、如图，你可以推测出打了马赛克部分的代码吗？

```
data = "1000,tarena,男"
MyDict = {}
(MyDict["id"],MyDict["name"],MyDict["sex"]) = [redacted]

print("ID:"+ MyDict["id"])
print("Name:"+ MyDict["name"])
print("Sex:"+ MyDict["sex"])
```

```
tarena@tedu: ~/桌面
tarena@tedu:~/桌面$ python3 note.py
ID:1000
Name:tarena
Sex:男
```

7、Python 的字典是否支持一键(Key)多值(Value)？

答：不支持，对相同的键再次赋值会将上一次的值直接覆盖。

```

>>> dict1 = {1:'one', 1:'yi'}
>>> dict1[1]
'yi'
```

8、在字典中，如果试图为一个不存在的键(Key)赋值会怎样？

答：会自动创建对应的键(key)并添加相应的值(Value)进去。

9、成员资格操作符(in 和 not in)可以检查一个元素是否存在序列当中，当然也可以用来检查一个键(Key)是否存在字典中，那么请问哪种的检查效率更高些？为什么？

答：在字典中检查键(Key)是否存在比在序列中检查指定元素是否存在更高效。因为字典的原理是使用哈希算法存储，一步到位，不需要使用查找算法进行匹配，因此时间复杂度是O(1),效率非常高。

10、Python 对键(Key)和值(Value)有没有类型限制？

答:Python 对键的要求相对要严格一些,要求他们必须是可哈希的对象,不能是可变类型(包括变量，列表，字典本身等)。

但是 Python 对值是没有任何限制的，它们可以是任意的 Python 对象。

11、请目测下边代码执行后，字典 dict1 的内容是什么？

```
>>> dict1.fromkeys((1, 2, 3), ('one', 'two', 'three'))
>>> dict1.fromkeys((1, 3), '数字')

复制代码
```

答：执行完成后，字典 dict1 的内容是：{1:”数字”，3:”数字”}

这里要注意的是,fromkeys 方法是直接创建一个新的字典，不要试图使用它来修改一个原有的字典，因为它会直接无情的把整个字典给覆盖掉。

12、如果你需要将字典 dict1 = {1:”one”,2:”two”,3:”three”}拷贝到 dict2，你应该怎么做？

答：可以利用字典的 copy()的方法：dict2 = dict1.copy(),在其他语言转移到 Python 小伙伴们刚开始可能会习惯性的直接用赋值的方法（dict2 = dict1），这样子做在 Python 中只是将对象的引用拷贝过去而已。

看以下区别：

```
>>> a = {1:'one', 2:'two', 3:'three'}
>>> b = a.copy()
>>> c = a
>>> c[4] = 'four'
>>> c
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
>>> a
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
>>> b
{1: 'one', 2: 'two', 3: 'three'}

复制代码
```

第十二章

一、解答题

1、如果你希望创建的集合是不变的，应该怎么做？

答：frozenset()

2、请问如何确定一个集合里边有多少个元素？

答：可以使用 len()函数

```
>>> num_set = set([1, 2, 3, 4, 5])
>>> len(num_set)
5
复制代码
```

3、请目测以下代码会打印什么内容？

```
>>> num_set = set([1, 2, 3, 4, 5])
>>> num_set[0]
复制代码
```

答：会报错，因为集合是无序的

4、请问 set{[1,2]}和 set([1,2])的执行结果是一样的吗？

答：不一样，set1 = set([1,2]) 会生成一个集合{1，2}，但 set1 = {[1,2]}却会报错

```
>>> set1 = {[1, 2]}
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    set1 = {[1, 2]}
TypeError: unhashable type: 'list'

复制代码
```

从报错信息上我们看到“列表不是可哈希类型”，没错，列表是可变的，它怎么可以哈希呢。

其实你再想想就会觉得很有道理，利用哈希函数计算，相同的元素得到的哈希值(存放地址)是相同的，所以在集合中所有相同的元素都会被覆盖掉，因此有了集合的唯一性。
然后你继续接着想就觉得更有道理了，通过哈希函数计算的地址不可能是按顺序排放的，所以集合才强调是无序的。

5、打开你的 IDLE,输入 set1 = {1,1.0}，你发现了什么？
答：没错，集合内容是{1,0},其实你弄懂了上一题，这一题一样容易：因为在 Python 的哈希函数会将相同的值的元素计算得到相同的地址，所以 1 和 1.0 是等值的。
6、请问如何给集合添加和删除元素？
答:使用 add()方法可以为集合添加元素,使用 remove()方法可以删除集合中已知的元素。

```
>>> num1.add(6)
>>> num1
{0, 1, 2, 3, 4, 5, 6}
>>> num1.remove(6)
>>> num1
{0, 1, 2, 3, 4, 5}
```

7、请问集合的唯一作用是什么？
答：集合几乎所有的作用就是确保里边所包含元素的唯一性，集合内不可能存在两个相同的元素。

第十三章

一、解答题

1、下边只有一种方式不能打开文件，请问是哪一种，为什么？

```
>>> f = open('E:/test.txt', 'w')   # A
>>> f = open('E:\test.txt', 'w')   # B
>>> f = open('E://test.txt', 'w')  # C
>>> f = open('E:\\test.txt', 'w')  # D

复制代码
```

答：B 不能打开文件。
Windows 在路径名中既可以接受斜线（/）也可以接受反斜线（\），不过如果使用反斜线作为路径名的分割符的话，要注意使用双反斜线(\\)进行转义，否则 python 会将反斜线进行转义，否则 python 会将反斜线进行转义，例如(\n)看成一个换行符，(\t)看作一个制表符
2、打开一个文件我们使用 open()函数，通过设置文件的打开模式，决定打开的文件具有那些性质，请问默认的打开模式是什么呢？
答：open()函数默认的打开模式是“rt”，即可读，文本的模式打开。

3、请问 >>>open(“E:\\Test.bin”,”xb”)是以什么样的模式打开文件的？
答：以“可写入以及二进制模式”打开文件“E:\\Test.bin”.
这里要注意的是“x”和“w”均是以“可写入”的模式打开文件，但以“x”模式打开的时

候，如果路径下已经存在相同的文件名，会抛出异常，而”w”模式的话会直接覆盖同名文件。
因此，”w”模式打开文件会比较危险，容易导致此前的内容遗失，因此使用”w”模式打开文件先检查该文件名是否已经存在显得非常重要！
4、尽管 python 有所谓的“垃圾回收机制”，但对于打开了的文件，在不需要用到的时候我们仍然需要使用 f.close()将文件对象“关闭”，这是为什么呢？
答：python 拥有垃圾收集机制，会在文件对象的引用计数降至零的时候自动关闭文件，所以在 python 编程里，如果忘记关闭文件并不会造成内存泄漏那么危险。
但并不是说就可以不要关闭文件，如果你对文件进行了写入操作，那么你应该在完成写入之后关闭文件。因为 python 可能会缓存你写入的数据，如果中间断电，那些缓存的数据根本不会写入到文件中。所以，为了安全起见，要养成使用完文件后立刻关闭的优雅习惯。
5、如何将一个文件对象(f)中的数据存放进列表中？
答：list(f) ,是不是非常的方便！
6、如何迭代打印出文件对象(f)的每一行数据？
答：直接使用 for 语句把文件对象迭代出来即可：

```
for each_line in f:
        print(each_line)

复制代码
```

7、文件对象的内置方法 f.read([size=-1])作用是读取文件对象内容，size 参数是可选的，那如果设置了 size = 10，例如 f.read(10),将返回什么内容呢？
答：将返回从文件指针开始（注意这里不是文件头哦）的连续 10 个字符
8、如何获得文件对象(f)当前文件指针的位置？
答：f.tell() 就能告诉你
9、还是视频中的那个演示文件(record.txt),请问为何 f.seek(45.0)不会出错，但 f.seek(46)就出错了呢？

```
>>> f.seek(46)
46
>>> f.readline()
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    f.readline()
UnicodeDecodeError: 'gbk' codec can't decode byte 0xe3 in position 4: illegal multibyte sequence

复制代码
```

答：因为使用 f.seek()定位的文件指针是按字节为单位来进行计算的，演示文件（record.txt）是以 GBK 进行编码的，按照规则，一个汉字需要占用两个字节，f.seek(45)的位置位于字符”小”的开始位置，因此可以正常打印，而 f.seek(46)的位置刚好位于字符“小”的中间位置，因此按照 GBK 编码的形式无法将其解码！
10、使用 pickle 的什么方法存储数据？
答：pickle.dumple(data,file)#第一个参数是待存储的数据对象，第二个参数是目标存储的文件对象，注意要先使用’wb’的模式 open 文件
11、使用 pickle 的什么方法读取数据？
答：pickle.load(file)#参数是目标存储文件对象，注意要先使用’rb’的模式 open 文件
12、使用 pickle 能不能保存为”*.txt”类型文件？

答：可以，不过打开后是乱码，因为是以二进制的模式写入的。

13、pickle 的实质是什么？

答：pickle 的实质就是利用一些算法将你的数据对象“腌制”成二进制文件，存储在磁盘上，当然也可以放在数据库或者通过网络传输到另一台计算机上。

第十四章

一、解答题

1. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

```
>>> my_list = [1, 2, 3, 4,,]
```

答：语法错误
SyntaxError: invalid syntax

2. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

```
>>> my_list = [1, 2, 3, 4, 5]
>>> print(my_list[len(my_list)])

```

答：len(my_list) 是获得列表的长度，这里长度为5，该列表各个元素的访问索引号分别是：[0, 1, 2, 3, 4]，因此试图访问 my_list(5) 会引发 IndexError: list index out of range 异常。

3. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

```
>>> my_list = [3, 5, 1, 4, 2]
>>> my_list.sorted()

```

答：初学者容易疏忽的错误，列表的排序方法叫 list.sort()，sorted() 是BIF。因此会引发 AttributeError: 'list' object has no attribute 'sorted' 异常。

4. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

```
>>> my_dict = {'host': 'http://www.tedu.cn ', 'port': '80'}
>>> print(my_dict['server'])

```

答：尝试访问字典中一个不存在的“键”引发 KeyError: 'server' 异常，为了避免这个异常发生，可以使用 dict.get() 方法：

```
if not my_dict.get('server'):
        print('您所访问的键【server】不存在！')

```

复制代码

5. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

```
def my_fun(x, y):
        print(x, y)

f(x=1, 2)

```

复制代码

答：如果使用关键字参数的话，需要两个参数均使用关键字参数 f(x=1, y=2)

6. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

```
f = open('C:\\test.txt', wb)
f.write('I love FishC.com!\n')
f.close()

```

答：注意 open() 第二个参数是字符串，应该 f = open('C:\\test.txt', 'wb') 。wb不加双引号 Python 还以为是变量名呢，往上一找，艾玛没找着……引发 NameError 异常。由于打开文件失败，接着下边一连串与 f 相关的均会报同样异常。

7. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

```
def my_fun1():
        x = 5
        def my_fun2():
                x *= x
                return x
        return my_fun2()

my_fun1()

```

复制代码

答：闭包的知识大家还记得不？ Python 认为在内部函数的 x 是局部变量的时候，外部函数的 x 就被屏蔽了起来，所以执行 x *= x 的时候，在右边根本就找不到局部变量 x 的值，因此报错。

在 Python3 之前没有直接的解决方案，只能间接地通过容器类型来存放，因为容器类型不是放在栈里，所以不会被“屏蔽”掉。容器类型这个词儿大家是不是似曾相识？我们之前介绍的字符串、列表、元祖，这些啥都可以往里的扔的就是容器类型啦。

于是乎我们可以把代码改造为：

```
def my_fun1():
        x = [5]
        def my_fun2():
                x[0] *= x[0]
                return x[0]
        return my_fun2()

my_fun1()

```
复制代码

但是到了 Python3 的世界里，又有了不少的改进，如果我们希望在内部函数里可以修改外部函数里的局部变量的值，那么也有一个关键字可以使用，就是 nonlocal：

```
def my_fun1():
        x = 5
        def my_fun2():
                nonlocal x
                x *= x
                return x
        return my_fun2()

my_fun1()

```
复制代码

> **8. 结合你自身的编程经验，总结下异常处理机制的重要性？**
>
> 答：由于环境的不确定性和用户操作的不可以预知性都可能导致程序出现各种问题，因此异常机制最重要的无非就是：增强程序的健壮性和用户体验，尽可能的捕获所有预知的异常并写好处理的代码，当异常出现的时候，程序自动消化并恢复正常（不至于崩溃）。

0. 我们使用什么方法来处理程序中出现的异常？

答：使用 try……except 搭配来捕获处理程序中出现的异常。

try:
检测范围
except Exception[as reason]:
出现异常（Exception）后的处理代码

1. 一个 try 语句可以和多个 except 语句搭配吗？为什么？

答：可以。因为 try 语句块中可能出现多类异常，利用多个 except 语句可以分别捕获并处理我们感兴趣的异常。

```
try:
    sum = 1 + '1'
    f = open('我是一个不存在的文档.txt')
    print(f.read())
    f.close()
except OSError as reason:
    print('文件出错啦T_T\n错误原因是：' + str(reason))
except TypeError as reason:
    print('类型出错啦T_T\n错误原因是：' + str(reason))

```

复制代码

2. 你知道如何统一处理多类异常吗？

答：在 except 后边使用小括号“()”把多个需要统一处理的异常括起来：

```
try:
    int('abc')
    sum = 1 + '1'
    f = open('我是一个不存在的文档.txt')
    print(f.read())
    f.close()
except (OSError, TypeError):
    print('出错啦T_T\n错误原因是：' + str(reason))

```

复制代码

3. except 后边如果不带任何异常类，Python 会捕获所有（try 语句块内）的异常并统一处理，但小甲鱼却不建议这么做，你知道为什么吗？

答：因为它会隐藏所有程序员未想到并且未做好准备处理的错误，例如用户输入ctrl+c试图终止程序会被解释为KeyboardInterrupt异常。

4. 如果异常发生在成功打开文件后，Python 跳到 except 语句执行，并没有执行关闭文件的命令（用户写入文件的数据就可能没有保存起来），因此我们需要确保无论如何（就算出了异常退出）文件也要被关闭，我们应该怎么做呢？

答：我们可以使用 finally 语句来实现，如果 try 语句块中没有出现任何运行时错误，会跳过 except 语句块执行 finally 语句块的内容。

如果出现异常，则会先执行 except 语句块的内容再接着执行 finally 语句块的内容。总之，finally 语句块里的内容就是确保无论如何都将被执行的内容！

5. 请恢复以下代码中马赛克挡住的内容，使得程序执行后可以按要求输出。

代码：

```
try:
    for i in range(3):
        for j in range(3):
            [redacted]
            print(i, j)
except KeyboardInterrupt:
    print('退出啦！')
```

输出：

```
>>> ================================ RESTART ================================
>>>
0 0
0 1
0 2
1 0
1 1
1 2
退出啦！
>>>
```

答：这道题比较考脑瓜，你想到了吗？

```
try:
    for i in range(3):
        for j in range(3):
            if i == 2:
                raise KeyboardInterrupt
            print(i, j)
except KeyboardInterrupt:
    print('退出啦！')

```

复制代码

第十五章

一、解答题

1、使用 with 语句固然方便，但如果出现异常的话，文件还会自动关闭吗？

答：while 语句会自动处理文件的打开和关闭，如果中途出现异常，会清理执行代码，

然后确保文件自动关闭。

2、你可以换一种形式写出线面的伪代码吗？

```
with A() as a:
    with B() as b:
        suite
```

答：with 处理多个项目的时候，可以用逗号隔开写成一条语句。

```
with A() as a, B() as b:
    suite
```

3、使用 with 语句改写代码，让 python 去关心文件的打开与关闭吧

```
def file_compare(file1, file2):
    f1 = open(file1)
    f2 = open(file2)
    count = 0 # 统计行数
    differ = [] # 统计不一样的数量

    for line1 in f1:
        line2 = f2.readline()
        count += 1
        if line1 != line2:
            differ.append(count)

    f1.close()
    f2.close()
    return differ

file1 = input('请输入需要比较的头一个文件名：')
file2 = input('请输入需要比较的另一个文件名：')

differ = file_compare(file1, file2)

if len(differ) == 0:
    print('两个文件完全一样！')
else:
    print('两个文件共有【%d】处不同：' % len(differ))
    for each in differ:
        print('第 %d 行不一样' % each)
```

4、你可以利用异常的原理，修改下面的代码使得更高效率的实现吗？

答：使用条件语句的代码非常直观明了，但是效率不高。因为程序会两次访问字典的键，一次判断是否存在（例如 if name in contacts），一次获得值（例如 print(name + ':'+contacts[name]) ）。

如果利用异常解决方案，我们可以简单避开每次需要使用 in 判断是否键存在字典中的操作。因为只要当键不存在字典中，会触发 KeyError 异常，利用此特性我们可以修改代码如下

```
print('|--- 欢迎进入通讯录程序 ---|')
print('|--- 1：查询联系人资料  ---|')
print('|--- 2：插入新的联系人  ---|')
print('|--- 3：删除已有联系人  ---|')
print('|--- 4：退出通讯录程序  ---|')

contacts = dict()

while 1:
    instr = int(input('\n请输入相关的指令代码：'))

    if instr == 1:
        name = input('请输入联系人姓名：')
        if name in contacts:
            print(name + ' : ' + contacts[name])
        else:
            print('您输入的姓名不再通讯录中！')

    if instr == 2:
        name = input('请输入联系人姓名：')
        if name in contacts:
            print('您输入的姓名在通讯录中已存在 -->> ', end='')
            print(name + ' : ' + contacts[name])
            if input('是否修改用户资料（YES/NO）：') == 'YES':
                contacts[name] = input('请输入用户联系电话：')
        else:
            contacts[name] = input('请输入用户联系电话：')

    if instr == 3:
        name = input('请输入联系人姓名：')
        if name in contacts:
            del(contacts[name])          # 也可以使用dict.pop()
        else:
            print('您输入的联系人不存在。')

    if instr == 4:
        break

print('|--- 感谢使用通讯录程序 ---|')

```

5、在 python 中，else 语句能跟哪些语句进行搭配？

答：在 python 中，else 语句不仅能跟 if 语句搭配，构成“要么怎么样，要么不怎么样的语境”，它还能循环语句，构成“干完了能怎样，干不完就别想怎样”，的语境；其实 else 语句还能够跟我们刚刚讲的异常处理进行搭配，构成“没有问题 那就干吧”的语境。

6、请问以下例子中，循环中的 break 语句会跳过 else 语句吗？

```
def showMaxFactor(num):
    count = num // 2
    while count > 1:
        if num % count == 0:
            print('%d最大的约数是%d' % (num, count))
            break
        count -= 1
    else:
        print('%d是素数！' % num)

num = int(input('请输入一个数：'))
showMaxFactor(num)
```

答：会，因为如果将 else 语句与循环语句（whlie 和 for 语句）进行搭配，那么只有在循环正常执行完成后才会执行 else 语句块的内容。

7、以下代码会打印什么内容？

```
try:
        print('ABC')
except:
        print('DEF')
else:
        print('GHI')
finally:
        print('JKL')

```

答：只有 except 语句中的内容不打印，因为 try 语句块中并没有异常，则 else 语句块也会被执行。

ABC

GHI

JKL

第十六章

一、解答题

1、对象中的属性和方法，在编程中实际是什么？

答案：变量（属性）和函数（方法）。

2、类和对象是什么关系呢？

答案：类和对象的关系就如同模具和用这个模具制作出的物品之间的关系。一个类为它的全部对象给出了一个统一的定义，而他的每个对象则是符合这种定义的一个实体，因此类和对象的关系就是抽象和具体的关系。

3、如果我们定义了一个猫类，那你能想象出由“猫”类实例化的对象有哪些？

答案：叮当猫，咖啡猫，Tom（Tom&Jerry），Kitty（Hello Kitty）……

4、类的定义有些时候或许不那么“拟物”，有时候会抽象一些，例如我们定义一个矩形类，那你会为此添加哪些属性和方法呢?

答案：属性可以是长和宽，方法可以是计算周长、面积等。

5、类的属性定义应该尽可能抽象还是尽可能具体？

答案：正确的做法是应该尽可能的抽象，因为这样更符合面向对象的思维。

6、请用一句话概括面向对象的几个特征？

封装	答案：对外部隐藏对象的工作细节
继承	答案：子类自动共享父类之间数据和方法的机制
多态	答案：可以对不同类的对象调用相同的方法，产生不同的结果

7、函数和方法有什么区别？

答案：细心的童鞋会发现，方法跟函数其实几乎完全一样，但有一点区别就是方法默认有一个 self 参数，这个方法是什么意思？请听一下详细分解。

8、以下代码体现了面向对象编程的什么特征？

```
>>>“达内.com”.count('o')
1
>>> [1, 1, 2, 3, 5, 8].count(1)
2
>>> (0, 2, 4, 8, 12, 18).count(1)
0
```

答案：体现了面向对象编程的多态特征。

9、当程序员不想把同一段代码写几次，他们发明了函数解决了这种情况。当程序员已经有了一个类，而又想建立一个非常相近的新类，他们会怎么做呢？

答案：他们会定义一个新类继承已有的这个类，这样子就只需要简单添加如重写需要的方法即可。例如已有鸟类，那么如果要重新定义一个麻雀类，我们只需要让麻雀类继承已有的鸟类，然后重写翅膀的属性为“土灰色”即可（据说麻雀的翅膀是土灰色的）

10、self 参数的作用是什么？

答案：绑定方法，据说有了这个参数，Python 再也不会傻傻分不清是哪个对象在调用方法了，你可以认为方法中的 self 其实就是实例对象的唯一标志。

11、如果我们不希望对象的属性或方法被外部直接引用，我们可以怎么做？

答案：我们可以在属性或方法名字前边加上双下划线，这样子从外部是无法直接访问到，会显示 AttributeError 错误。

```
>>> class Person:
__name = '达内'
        def getName(self):
                return self.__name

>>> p = Person()
>>> p.__name
Traceback (most recent call last):
  File "<pyshell#56>", line 1, in <module>
    p.__name
AttributeError: 'Person' object has no attribute '__name'
>>> p.getName()
'达内'
```

我们把 getNone 方法称之为“访问器”。Python 事实上时采用一种叫“name mangling”技术，将以双下划线开头的变量名巧妙的改了个名字而已，我们仍然可以在外部通过“_类名__变量名”的方式访问：

```
>>> p._Person__name'达内'
```

当然我们并不提倡这种抬杠较真粗暴不文明的访问方式……

12、类在实例化后哪个方法会被自动调用？

答案：__init__方法会在类实例化时被自动调用，我们称之为魔法方法。你可以重写这个方法，为对象定制初始化方案。

13、请解释下边代码错误的原因：

```
class MyClass:
        name = '达内'
        def myFun(self):
                print("Hello达内!")

>>> MyClass.name
'达内'
>>> MyClass.myFun()
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    MyClass.myFun()
TypeError: myFun() missing 1 required positional argument: 'self'
>>>
```

答案：首先你要明白类、类对象、实例对象是三个不同的名词。

我们常说的类指的是类定义，由于“Python 无处不对象”，所以当类定义完之后，自然就是类对象。在这个时候，你可以对类的属性（变量）进行直接访问（MyClass.name）。

一个类可以实例化出无数的对象（实例对象），Python 为了区分是哪个实例对象调用了方法，于是要求方法必须绑定（通过 self 参数）才能调用。而未实例化的类对象直接调用方法，因为缺少 self 参数，所以就会报错。

第十七章

一、解答题

1、继承机制给程序员带来最明显的好处是？

答案：可以偷懒，据说这是每一个优秀程序员的梦想！

如果一个类 A 继承自另一个类 B，就把这个 A 称为 B 的子类，把 B 称为 A 的父类、基类或超类。继承可以使得子类具有父类的各种属性和方法，而不需要再次编写相同的代码（偷懒）。在子类继承父类的同时，可以重写定义某些属性，并重写某些方法，即覆盖父类的原有属性和方法，使其获得与父类不同的功能。另外，为子类追加新的属性和方法也是常见的做法。

2、如果按以下方式重写魔法方法__init__,结果会怎样？

```
class MyClass:
    def __init__(self):
        return "I love 达内.com!"

```

答案：会报错，因为__init__特殊方法不应当返回除了 None 以外的任何对象。

```
>>> myClass = MyClass()
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    myClass = MyClass()
TypeError: __init__() should return None, not 'str'

```

3、当子类定义了与相同名字的属性或方法时，Python 是否会自动删除父类的相关属性或方法？

答案：不会删除！Python 的做法跟其他大部分面向对象编程语言一样，都是将父类属性或方法覆盖，子类对象调用的时候会调用到覆盖后的新属性或方法，但父类的任然存在，知识子类对象“看不到”。

4、假设已经有鸟类的定义，现在我要定义企鹅类继承于鸟类，但我们都知道企鹅是不会飞的，我们应该如何屏蔽父类（鸟类）中飞的方法？

答案：覆盖父类对象，例如将函数体内容写 pass，这样调用 fly 方法就没有任何反应了。

```
class Bird:
        def fly(self):
                print("Fly away!")

class Penguin(Bird):
        def fly(self):
                pass

>>> bird = Bird()
>>> penguin = Penguin()
>>> bird.fly()
Fly away!
>>> penguin.fly()

```

5、super 函数有什么“超级”的地方？

答案：super 函数超级之处在于你不需要明确给出任何基类的名字，它会自动帮您找出所有基类以及对应的方法。由于你不用给出基类的名字，这就意味着你如果需要改变了类继承关系，你只要改变 class 语句里的父类即可，而不必在大量代码中去修改所有被继承的方法。

6、多重继承使用不当会导致重复调用（也叫钻石继承、菱形继承）的问题，请分析以下代码在实际编程中有可能导致什么问题？

```
class A():
    def __init__(self):
        print("进入A…")
        print("离开A…")

class B(A):
    def __init__(self):
        print("进入B…")
        A.__init__(self)
        print("离开B…")

class C(A):
    def __init__(self):
        print("进入C…")
        A.__init__(self)
        print("离开C…")

class D(B, C):
    def __init__(self):
        print("进入D…")
        B.__init__(self)
        C.__init__(self)
        print("离开D…")

```

答案：多重继承容易导致重复调用问题，下边实例化 D 类后我们发现 A 被前后进入了两次（有童鞋说两次就两次呗，我女朋友还不止呢……）

这有什么危害？我举个例子，假设 A 的初始化方法里有一个计数器，那这样 D 一实例化，A 的计数器就跑两次（如果遭遇多个钻石结构重叠还要更多），但明显是不符合程序设计的初衷的（程序应该可控，而不能受到继承关系影响）

```
>>> d = D()
进入D…
进入B…
进入A…
离开A…
离开B…
进入C…
进入A…
离开A…
离开C…
离开D…
```

为了让大家明白，这里只是举例最简单的钻石继承问题，在实际编程中，如果不注意多重继承的使用，会导致比这个复杂 N 倍的现象，调试起来不是一般的痛苦……所以一定要尽量避

免使用多重继承。

7、如何解决上一题中出现的问题？

答案：super 函数再次大显神威。

```
class A():
    def __init__(self):
        print("进入A…")
        print("离开A…")

class B(A):
    def __init__(self):
        print("进入B…")
        super().__init__()
        print("离开B…")

class C(A):
    def __init__(self):
        print("进入C…")
        super().__init__()
        print("离开C…")

class D(B, C):
    def __init__(self):
        print("进入D…")
        super().__init__()
        print("离开D…")

>>> d = D()
进入D…
进入B…
进入C…
进入A…
离开A…
离开C…
离开B…
离开D…
```

8、继承机制给程序员带来最明显的好处是？

答案：可以偷懒，据说这是每一个优秀程序员的梦想！

如果一个类 A 继承自另一个类 B，就把这个 A 称为 B 的子类，把 B 称为 A 的父类、基类或超类。继承可以使得子类具有父类的各种属性和方法，而不需要再次编写相同的代码（偷懒）。在子类继承父类的同时，可以重写定义某些属性，并重写某些方法，即覆盖父类的原有属性和方法，使其获得与父类不同的功能。另外，为子类追加新的属性和方法也是常见的做法。

9、如果按以下方式重写魔法方法__init__,结果会怎样？

```
class MyClass:
    def __init__(self):
        return "I love 达内.com!"
```

答案：会报错，因为__init__特殊方法不应当返回除了 None 以外的任何对象。

```
>>> myClass = MyClass()
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    myClass = MyClass()
TypeError: __init__() should return None, not 'str'

```

10、当子类定义了与相同名字的属性或方法时，Python 是否会自动删除父类的相关属性或方法？

答案：不会删除！Python 的做法跟其他大部分面向对象编程语言一样，都是将父类属性或方法覆盖，子类对象调用的时候会调用到覆盖后的新属性或方法，但父类的任然存在，知识子类对象“看不到”。

11、假设已经有鸟类的定义，现在我要定义企鹅类继承于鸟类，但我们都知道企鹅是不会飞的，我们应该如何屏蔽父类（鸟类）中飞的方法？

答案：覆盖父类对象，例如将函数体内容写 pass，这样调用 fly 方法就没有任何反应了。

```
class Bird:
        def fly(self):
                print("Fly away!")

class Penguin(Bird):
        def fly(self):
                pass

>>> bird = Bird()
>>> penguin = Penguin()
>>> bird.fly()
Fly away!
>>> penguin.fly()

```

12、super 函数有什么“超级”的地方？

答案：super 函数超级之处在于你不需要明确给出任何基类的名字，它会自动帮您找出所有基类以及对应的方法。由于你不用给出基类的名字，这就意味着你如果需要改变了类继承关系，你只要改变 class 语句里的父类即可，而不必在大量代码中去修改所有被继承的方法。

13、多重继承使用不当会导致重复调用（也叫钻石继承、菱形继承）的问题，请分析以下代码在实际编程中有可能导致什么问题？

```
class A():
    def __init__(self):
        print("进入A…")
        print("离开A…")

class B(A):
    def __init__(self):
        print("进入B…")
        A.__init__(self)
        print("离开B…")

class C(A):
    def __init__(self):
        print("进入C…")
        A.__init__(self)
        print("离开C…")

class D(B, C):
    def __init__(self):
        print("进入D…")
        B.__init__(self)
        C.__init__(self)
        print("离开D…")

```

答案：多重继承容易导致重复调用问题，下边实例化 D 类后我们发现 A 被前后进入了两次（有童鞋说两次就两次呗，我女朋友还不止呢……）

这有什么危害？我举个例子，假设 A 的初始化方法里有一个计数器，那这样 D 一实例化，A 的计数器就跑两次（如果遭遇多个钻石结构重叠还要更多），但明显是不符合程序设计的初衷的（程序应该可控，而不能受到继承关系影响）

```
>>> d = D()
进入D…
进入B…
进入A…
离开A…
离开B…
进入C…
进入A…
离开A…
离开C…
离开D…
```

为了让大家明白，这里只是举例最简单的钻石继承问题，在实际编程中，如果不注意多重继承的使用，会导致比这个复杂 N 倍的现象，调试起来不是一般的痛苦……所以一定要尽量避免使用多重继承。

14、如何解决上一题中出现的问题？

答案：super 函数再次大显神威。

```
class A():
    def __init__(self):
        print("进入A…")
        print("离开A…")

class B(A):
    def __init__(self):
        print("进入B…")
        super().__init__()
        print("离开B…")

class C(A):
    def __init__(self):
        print("进入C…")
        super().__init__()
        print("离开C…")

class D(B, C):
    def __init__(self):
        print("进入D…")
        super().__init__()
        print("离开D…")

>>> d = D()
进入D…
进入B…
进入C…
进入A…
离开A…
离开C…
离开B…
离开D…
```

第十八章

一、解答题

1、什么是组合（组成）？

答案：Python 继承机制很有用，但容易把代码复杂化以及依赖隐含继承。因此，经常的时候，我们可以使用组合来代替。在 Python 里组合其实很简单，直接在类定义中把需要的类放进去实例化就可以了。

例子：

```
//麻雀类
class Sparrow:
    def __init__(self,x):
        self.num = x
//鹦鹉类
class Parrot:
    def __init__(self,x):
        self.num = x
//天空类
class Sky:
    def __init__(self,x,y):
        self.sparrow = Sparrow(x)
        self.parrot = Parrot(y)

    def print_num(self):
        print("天空里总共有乌龟  %d 只，小鱼  %d 条！" % (self.sparrow.num,self.parrot.num))
>>> sky = Sky(1,10)
>>> sky.print_num()
```

2、什么时候用组合，什么时候用继承？

答案：根据实际应用场景确定。简单的说，组合用于“有一个”的场景中，继承用于“是一个”的场景中。例如，天空中有一个鸟，河里有一条鱼，地上有一个蜘蛛，这些都适合使用组合。青瓜是瓜，男人是人，鲨鱼是鱼，这些都应该使用继承啦。

3、类对象是在什么时候产生？

答案：当你这个类定义完的时候，类定义就变成类对象，可以直接通过“类名.属性”或者“类名.方法名()”引用或使用相关的属性或方法。

4、如果对象的属性跟方法名字相同，会怎样？

答案：如果对象的属性跟方法名相同，属性会覆盖方法。

```
01. class C:
02.         def x(self):
03.                 print('Xman')
04.
05. >>> c = C()
06. >>> c.x()
07. Xman
08. >>> c.x = 1
09. >>> c.x
10. 1
11. >>> c.x()
12. Traceback (most recent call last):
13.   File "<pyshell#20>", line 1, in <module>
14.     c.x()
15. TypeError: 'int' object is not callable
16.
```

5、请问以下类定义中哪些是类属性，哪些是实例属性？

```
class C:
        num = 0
        def __init__(self):
                self.x = 4
                self.y = 5
                C.count = 6

```

答案：num 和 count 是雷属性（静态变量），x 和 y 是实例属性。大多数情况下，你应该考虑使用实例属性，而不是类属性（类属性通常仅用来跟踪与类相关的值）。

6、请问以下代码中，bb 对象为什么调用 printBB()方法失败？

```
class BB:
        def printBB():
                print("no zuo no die")

>>> bb = BB()
>>> bb.printBB()
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    bb.printBB()
TypeError: printBB() takes 0 positional arguments but 1 was given

```

答案：因为 Python 严格要求方法需要有实例才能被调用，这种限制其实就是 Python 所谓的绑定概念。所以 Python 会自动把 bb 对象作为第一个参数传入，所以才会出现 TypeError："需要 0 个参数，但实际传入了 1 个参数"。

正确的做法应该是：

```
class BB:
        def printBB(self):
                print("no zuo no die")

>>> bb = BB()
>>> bb.printBB()
no zuo no die

```

7、哪个特征让我们一样就能认出这货是魔法方法？

答：以双下划线开头 双下划线结尾（魔法方法总是被双下划綫包围）

1、类实例化对象所调用的第一个方法是什么？

答：是 __new__ 方法,__new__是在一个对象实例化的时候所调用的第一个方法。跟其它魔法方法不同，他的第一个参数不是 self 而是个类(cls)，而其它的参数会直接传递给__init__方法的。

8、什么时候我们需要在类中明确写出_init_方法?

答：当我们的实例对象需要有明确的初始化步骤的时候，你可以在__init__方法中部署初始化的代码

比如:

```
# 我们定义一个矩形类，需要长和宽两个参数，拥有计算周长和面积两个方法。
# 我们需要对象在初始化的时候拥有“长”和“宽”两个参数，因此我们需要重写__init__方法
# 因为我们说过，__init__方法是类在实例化成对象的时候首先会调用的一个方法，大家可以理解吗？

class Rectangle:
        def __init__(self, x, y):
                self.x = x
                self.y = y
        def getPeri(self):
                return (self.x + self.y) * 2
        def getArea(self):
                return self.x * self.y

>>> rect = Rectangle(3, 4)
>>> rect.getPeri()
14
>>> rect.getArea()
12
```

9、请问下面代码存在什么问题？

```
class Test:
        def __init__(self, x, y):
                return x + y

```

答：__init__方法不能用其他返回值 他的返回值只能是 None。

10、请问__new__方法是负责什么任务？

答：实例化对象的时候的一些特殊处理，__new__方法主要任务是返回一个实例化对象，通常是参数 cls 这个类的实例化对象，当然你也可以返回其它对象

11、__del__魔法方法什么时候会被自动调用?

答：如果说__init__和__new__方法是对象的构造器的话，那么 Python 也提供了一个析构器，叫做__del__方法。当对象将要被销毁的时候，这个方法就被调用。

但是一定要注意的是，并非 del x 就相当于自动调用 x.__del__(),__del__方法是当垃圾回收机制回收这个对象的时候调用的。

第十九章

一、解答题

1、自 Python2.2 之后，对类和类型进行了一个统一，做法是将 int(),float(),str(),list(),tuple()这些 BIF 转换为工厂函数。所谓的工厂函数，其实是什么原理？

答：工厂函数，其实就是一个类对象。当你调用他们的时候，事实上

就是创建一个相应的实例对象。

```
# a 和 b 是工厂函数（类对象） int 的实例对象
>>> a = int('123')
>>> b = int('345')
>>> a + b
468
```

2、当实例对象进行加法操作的时候，会自动调用什么魔法方法？
答：对象 a 和 b 相加时，Python 会自动根据对象 a 的__add__魔法方法进行加法操作。
3、下面的代码有问题吗？

```
class Foo:
        def foo(self):
                self.foo = "I love tarenaC    !"
                return self.foo

>>> foo = Foo()
>>> foo.foo()
'I love  tarebaC   '
```

答:这是一个温柔的陷阱，这种 BUG 非常难排查，所以一定要注意：类的属性名和方法名绝对不能相同！如果代码这么些，就会有一个难以排查的 BUG 出现了：

```
class Foo:
        def __init__(self):
                self.foo = "I love  tarenaC    "
        def foo(self):
                return self.foo

>>> foo = Foo()
>>> foo.foo()
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    foo.foo()
TypeError: 'str' object is not callable
```

4、写出下列算数运算符对应的魔法方法(附答案)

运算符	对应的魔法方法
+	__add__(self, other)
-	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)
divmod(a, b)	__divmod__(a, b)
**	__pow__(self, other[, modulo])
<<	__lshift__(self, other)
>>	__rshift__(self, other)
&	__and__(self, other)
^	__xor__(self, other)
|	__or__(self, other)

5、一下代码能说明 Python 支持什么风格？

```
01.  def calc(a, b, c):
02.          return (a + b) * c
03.
04.  >>> a = calc(1, 2, 3)
05.  >>> b = calc([1, 2, 3], [4, 5, 6], 2)
06.  >>> c = calc('love', tarenaC , 3)
07.  >>> print(a)
08.  9
09.  >>> print(b)
10.  [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
11.  >>> print(c)
12.  lovetarenaClovetarenaClovetarenaC
```

答：说明 Python 支持鸭子类型(duck typing)风格(自行百度)

6、对象相加(a+b)，如果 a 对象有__add__方法，请问 b 对象的__radd__会被调用吗？

答：不会！

实验如下：

```
>>> class Nint(int):
        def __radd__(self, other):
                print("__radd__ 被调用了！")
                return int.__add__(self, other)

>>> a = Nint(5)
>>> b = Nint(3)
>>> a + b
8
>>> 1 + b
__radd__ 被调用了！
4
```

7、Python 什么时候会调用到方运算的魔法方法？

答：例如 a+b，如果 a 对象的__add__方法没有实现或者不支持相应的操作，那么 Python 就会自动调用 b 的__radd__方法。

8、请问如何在继承的类中调用基类的方法？

答：使用 super()这个 BIF 函数。

```
class Derived(Base):
    def meth (self):
        super(Derived, self).meth()

```

9、如果我要继承的基类是动态的(有时候是 A，有时候是 B)，我应该如何部署我的代码，以便基类可以随意改变。

答：你可以先为基类定义一个别名，在类定义的时候，使用别名代替你要继承的基类。如此，当你想要改变基类的时候，只需要修改给别名赋值的语句即可。顺便说一下，当你的资源是视情况而定的时候，这个小技巧很管用。

比如：

```
BaseAlias = BaseClass  # 为基类取别名

class Derived(BaseAlias):
    def meth(self):
        BaseAlias.meth(self)  # 通过别名访问基类
```

10、尝试自己举一个例子说明如何使用类的静态属性

答：类的静态属性很简单，在类中直接定义的变量(没有 self.)就是静态属性，引用类的静态属性”类名.属性名”的形式。

类的静态属性应用(计算该类被实例化的次数)：

```
class C:
    count = 0  # 静态属性

    def __init__(self):
        C.count = C.count + 1  # 类名.属性名的形式引用

    def getCount(self):
        return C.count
```

11、尝试自己举例说明如何使用类的静态方法，并指出使用类的静态方法有何需要注意的地方？

答：静态方法是类的特殊方法，静态方法只需要在普通方法的前边加上@staticmethod 即可。

```
class C:
        @staticmethod  # 该修饰符表示 static() 是静态方法
        def static(arg1, arg2, arg3):
                print(arg1, arg2, arg3, arg1 + arg2 + arg3)

        def nostatic(self):
                print("I'm the f**king normal method!")
```

静态方法最大的优点是：不会绑定到实例对象上，换而言之就是节省开销。

```
>>> c1 = C()
>>> c2 = C()
# 静态方法只在内存中生成一个，节省开销
>>> c1.static is C.static
True
>>> c1.nostatic is C.nostatic
False
>>> c1.static
<function C.static at 0x03001420>
>>> c2.static
<function C.static at 0x03001420>
>>> C.static
<function C.static at 0x03001420>
# 普通方法每个实例对象都拥有独立的一个，开销较大
>>> c1.nostatic
<bound method C.nostatic of <__main__.C object at 0x03010590>>
>>> c2.nostatic
<bound method C.nostatic of <__main__.C object at 0x032809D0>>
>>> C.nostatic
<function C.nostatic at 0x0328D2B8>
```

使用的时候需要注意的地方：静态方法并不需要 self 参数，因此即使是适用对象去访问，self

也不会传进去。

```
>>> c1.static(1, 2, 3)
1 2 3 6
>>> C.static(1, 2, 3)
1 2 3 6
```

第二十章

一、解答题

1、请问一下代码的作用是什么？这样写正确吗？(如果还不正确，请改正)

```
def __setattr__(self, name, value):
        self.name = value + 1
```

答：这段代码试图在对象的属性发生赋值操作的时候，将实际的值+1 赋值给相应的属性。但这写法是错误的，因为每当属性被赋值的时候，__setattr__()会被调用，而里边的 self.name=value+1 语句又会再次触发__setattr__()调用，导致无限递归。
代码如下:

```
def __setattr__(self, name, value):
        self.__dict__[name] = value + 1
```

或者

```
def __setattr__(self, name, value):
        super().__setattr__ = value + 1
```

2、自定义该类的属性被访问的行为，你应该重写那个魔法方法？
答:__getattribute__(self,name)
3、在不上机验证的情况下，你能推断以下代码分别会显示什么吗？

```
>>> class C:
        def __getattr__(self, name):
                print(1)
        def __getattribute__(self, name):
                print(2)
        def __setattr__(self, name, value):
                print(3)
        def __delattr__(self, name):
                print(4)


>>> c = C()
>>> c.x = 1
# 位置一，请问这里会显示什么？
>>> print(c.x)
# 位置二，请问这里会显示什么？
```

答：位置一会显示 3 因为 c.x = 1 是赋值操作，所以会访问__setattr__()魔法方法；位置二会显示 2 和 None，因为 x 是属于实例对象 c 的属性，所以 c.x 是访问一个存在的属性，因此会访问__getattribute__()魔法方法，但我们重写了这个方法，使得他不能按照正常的逻辑返回属性值，而是打印一个 2 代替，由于我们没有写返回值，所以紧接着返回 None 并被 print() 打印出来。

4、请指出以下代码的问题所在

```
class Counter:
        def __init__(self):
                self.counter = 0
        def __setattr__(self, name, value):
                self.counter += 1
                super().__setattr__(name, value)
        def __delattr__(self, name):
                self.counter -= 1
                super().__delattr__(name)
```

答：

```
class Counter:
        def __init__(self):
                self.counter = 0 # 这里会触发 __setattr__ 调用
        def __setattr__(self, name, value):
                self.counter += 1
"""既然需要 __setattr__ 调用后才能真正设置 self.counter 的值，所以这时候 self.counter 还没有定义，所以没法 += 1，错误的根源。"""
                super().__setattr__(name, value)
        def __delattr__(self, name):
                self.counter -= 1
                super().__delattr__(name)
```

5、请问一下代码的作用是什么？这样写正确吗？(如果还不正确，请改正)

```
def __setattr__(self, name, value):
        self.name = value + 1
```

答：这段代码试图在对象的属性发生赋值操作的时候，将实际的值+1 赋值给相应的属性。但这写法是错误的，因为每当属性被赋值的时候，__setattr__()会被调用，而里边的 self.name=value+1 语句又会再次触发__setattr__()调用，导致无限递归。

代码如下:

```
def __setattr__(self, name, value):
        self.__dict__[name] = value + 1
```

或者

```
def __setattr__(self, name, value):
        super().__setattr__ = value + 1
```

6、自定义该类的属性被访问的行为，你应该重写那个魔法方法？

答:__getattribute__(self,name)

7、在不上机验证的情况下，你能推断以下代码分别会显示什么吗？

```
>>> class C:
        def __getattr__(self, name):
                print(1)
        def __getattribute__(self, name):
                print(2)
        def __setattr__(self, name, value):
                print(3)
        def __delattr__(self, name):
                print(4)


>>> c = C()
>>> c.x = 1
# 位置一，请问这里会显示什么？
>>> print(c.x)
# 位置二，请问这里会显示什么？
```

答：位置一会显示 3 因为 c.x = 1 是赋值操作，所以会访问__setattr__()魔法方法；位置二会显示 2 和 None，因为 x 是属于实例对象 c 的属性，所以 c.x 是访问一个存在的属性，因此会访问__getattribute__()魔法方法，但我们重写了这个方法，使得他不能按照正常的逻辑返回属性值，而是打印一个 2 代替，由于我们没有写返回值，所以紧接着返回 None 并被 print() 打印出来。

8、请指出以下代码的问题所在

```
class Counter:
        def __init__(self):
                self.counter = 0
        def __setattr__(self, name, value):
                self.counter += 1
                super().__setattr__(name, value)
        def __delattr__(self, name):
                self.counter -= 1
                super().__delattr__(name)
```

答：

```
class Counter:
        def __init__(self):
                self.counter = 0 # 这里会触发 __setattr__ 调用
        def __setattr__(self, name, value):
                self.counter += 1
"""既然需要 __setattr__ 调用后才能真正设置 self.counter 的值，所以这时候 self.counter 还没有定义，所以没法 += 1，错误的根源。"""
                super().__setattr__(name, value)
        def __delattr__(self, name):
                self.counter -= 1
                super().__delattr__(name)
```

第二十一章

一、简答题

1、请尽量用自己的语言描述什么是描述符

答案：有时候，某些应用程序可能会有一个微妙的需求，需要你设计更为复杂的操作来响应（例如当变量被访问时，创建一个日志记录）。最好的解决方案就是编写一个用于执行这些“更为复杂的操作”的特殊函数，然后指定它在属性被访问时运行，那么一个具有这种函数的对象被称为描述符。

简单的说，描述符就是一个类，一个至少实现__get__、__set__、__delete__三个特殊方法中一个的类。

2、描述符类中，分别通过哪些魔法方法实现对属性的 get、set 和 delete 操作的？

答案：__get__、__set__、__delete__

__get__(self, instance, owner)-用于访问属性，返回属性的值

__set__(self, instance, value)-在属性分配操作中调用，不返回任何内容

__delete__(self, instance, owner)-控制删除操作，不返回任何内容

3、请问以下代码，分别调用 test.a 和 test.x，哪个会打印”getting...”

```
class MyDes:
    def __get__(self, instance, owner):
        print('getting....')
class Test:
    a = MyDes()
    x = a
test = Test()
```

答案：都会打印

4、请问以下代码打印什么内容

```
class MyDes:
    def __init__(self, value=None):
        self.val = value
    def __get__(self, instance, owner):
        return self.val - 20
    def __set__(self, instance, value):
```

```
        self.val = value +10
        print(self.val)
class C:
    x = MyDes()
if __name__ == '__main__':
    c = C()
    c.x = 10
    print(c.x)
```

答案： 20 0

*需要注意的是，print(c.x)访问了 c 的 x 属性，因此值减少了 20

5、请问以下代码会打印什么内容

```
class MyDes:
    def __init__(self, value=None):
        self.val = value
    def __get__(self, instance, owner):
        return self.val ** 2
class Test:
    def __init__(self):
        self.x = MyDes(3)
test = Test()
print(test.x)
```

答案：<__main__.MyDes object at 0x024C35B0>

访问实例层次上的描述符 x，只会访问描述符本身。为了让描述符正常工作，它们必须定义在类的层次上。如果你不这么做，那么 Python 无法为你自动调用__get__和__set__方法

6、Python 基于序列的三大容器类是什么？

答案：list 列表、tuple 元组、str 字符串

7、Python 中允许我们自定义容器类，如果定义一个不可变类型的容器类，就不能定义什么方法？

答案：如果我们想要定制一个不可变的容器类，类似 str 的，就不能定义像__setitem__和__delitem__这类能够修改容器内数据的方法

8、Python 如果希望定制容器类支持 reverse()内置函数，那么应该定义什么方法？

答案：应该定义__reverse__()函数，提供对内置函数 reverse()的支持

9、题目：容器类都应该提供“查询容量”的方法，那么需要定义什么方法呢？

答案：在 Python 中我们使用 len()函数来查询容器的“容量”，所以容器需要定义__len__()方法

10、通过哪些方法我们可以实现对容器的读、写和删除操作

答案：__getitem__()读，__setitem__()写，__delitem__()删除

11、为什么说 Python 中的协议不是正式的？

答案：在 Python 中，协议更像是一种指南，Python 并不会严格要求一定要怎样做，而是要我们靠着自觉和经验把事情做好，就像吉多说的，“大家都是成年人”。

第二十二章

一、解答题

1、通常，一般的函数执行从第一行开始，并在什么情况下结束

答案：对于调用一个普通的 Python 函数，一般从函数的第一行代码开始执行，结束于 return 语句、异常或所有语句执行完毕，一旦函数将控制权交给调用者，这意味着全部结束。函数中所做的工作以及保存在局部变量中的所有数据都将丢失。如果再次调用该函数，一切重新开始。

2、什么叫做协同程序

答案：所谓的协同程序就是可以运行的独立函数调用，函数可以暂停或者挂起，并在需要的时候从程序离开的地方继续执行或重新开始。

Python 通过生成器来实现类似于协同程序的概念：生成器可以暂时挂起程序，并保留函数中的局部变量等数据，然后在再次调用它的时候，从上次暂停的位置继续执行下去。

3、生成器所能实现的任何操作都可以由迭代器来代替吗，为什么？

答案：都可以，生成器的本身就是基于迭代器实现的，生成器只需要一个 yield 就可以，但它内部会自动创建__iter__()函数和__next__()函数

4、一个函数改造成生成器，说白了就是把什么语句改为 yield 语句？

答案：将 return 语句改为 yield 语句

5、生成器最大的作用是什么？

答案：可以“保留现场”，当下一次执行该函数的时候直接从上一次结束的地方开始，不必从头开始，节省内存空间和响应时间

6、如下，get_prime()是一个判断数字是否为素数的函数，请问第二行代码有何作用？

```
def get_primes(number):
    while True:
        if is_prime(number):
            yield number
        number += 1
```

答案：这个 endless loop 是为了确保该生成器函数永远不会执行到函数结尾，只要调用 next()就会生成一个值，这是一个处理无穷序列的常用方法

7、说到底，python 的模块是什么？

答：模块就是程序。没错，所谓模块就是平时我们写的任何代码，然后保存的每一个”.py”结尾的文件，都是一个独立的模块。

8、我们现在有一个 hello.py 的文件，里边有一个 hi()函数：

```
def hi():
    print(“Hi everyone, I love bj.tedu.cn!”)
```

请问我如何在另外一个源文件 tset.py 里边使用 hello.py 的 hi()函数呢？

答：只需要在 test.py 中导入 hello 模块（文件名=模块名）即可使用 hello.py 中的 hi()函数。

9、你知道的总共有几种导入模块的方法？

答：我们总共学习了 3 种导入模块的方法。

第一种：import 模块名

第二种：from 模块名 import 函数名

第三种：import 模块名 as 新名字

10、曾经我们讲过有办法阻止 from…import * 导入你的“隐私”属性，你还记得是怎么做的吗？

答：如果你不想模块中的某个属性被 from…import * 导入，那么你可以给你不想导入的属性名称前边加上一个下划线（_）。不过需要注意的是，如果使用 import …导入整个模块，或者显示地使用 import xx._oo 导入某个属性，那么这个隐藏的方法就不起作用了。

11、倘若有 a.py 和 b.py 两个文件，内容如下：

```
# a.py
def sayHi():
    print("嗨，我是A模块～")
# b.py
def sayHi():
    print("嗨，我是B模块～")
```

那么我在 test.py 文件中执行以下操作，会打印什么结果？

```
# test.py
from a import sayHi
from b import sayHi
sayHi()
```

答：会打印 “嗨，我是 B 模块~”，因为第二次导入的 b 模块把 a 模块的同名函数 sayHi（）给覆盖了，这就是所谓命名空间的冲突。所以，在项目中，特别是大型项目中我们应该避免使用 from…import…除非你非常明确不会造成命名冲突。

12、执行下边 a.py 或 b.py 任何一个文件，都会报错，请尝试解释一下此现象。

```
# a.py
from b import y
def x():
    print("x")
```

```
# b.py
from a import x
def y():
    print("y")
```

答：这个是循环嵌套导入问题。无论运行 a.py 或 b.py 哪一个文件都会抛出 ImportError 异常。这是因为在执行其中一个文件（a.py）的加载过程中，会创建模块对象并执行对应的字节码。但当执行第一个语句的时候需要导入另一个文件（from b import y），因

此 CPU 会转而去加载另一个文件（b.py）。同理，执行另一个文件的第一个语句（from a import x）恰好也是需要导入之前的文件（a.py）。此时，之前的文件处于仅导入第一条语句的阶段，因此其对应的字典中并不存在 x，故抛出异常。
解决方案是直接使用 import 语句导入：

```
# a.py
import b
def x():
    print("x")

# b.py
import a
def y():
    print("y")
a.x()
```