- 列表
  - 什么是列表
  - 从列表提取单个元素
  - 从列表提取多个元素
  - 给列表增加/删除元素
- 数据类型：字典
  - 什么是字典
  - 给字典增加/删除元素
- 列表和字典的异同
  - 列表和字典的不同点
- 作业
- 元组（tuple）

第5关 循环for..in ,while

- for...in...循环语句
  - for循环：空房间
  - for循环：一群排队办业务的人
  - range()函数
  - for循环：办事流程
- while循环
  - while循环：放行条件
  - while循环：办事流程
- 两种循环对比
- 作业
- pop()函数

第6关 布尔值，break，continue，pass，else语句

- 用数据做判断：布尔值
  - 两个数值做比较
  - 直接用数值做运算
  - 布尔值之间的运算
- 四种新的语句
  - break语句
  - continue语句
  - pass语句
  - else语句
- 总结
- 小练习
- 作业

第7关 项目实操 小游戏大学问

- 项目实操
  - 明确项目目标
  - 分析过程，拆解项目
  - 逐步执行，代码实现

- 版本1.0：自定属性，人工PK
- 版本2.0：随机属性，自动PK
- 版本3.0：打印战果，三局两胜

学习format()函数

第8关 编程学习的瓶颈

瓶颈1：知识学完就忘

用法查询笔记

深度理解笔记

知识管理

瓶颈2：缺乏解题能力

如何解题

作业

第9关 函数（定义和调用函数，函数的重要概念）

函数是什么

定义和调用函数

- 定义函数
- 调用函数

函数重要概念

return语句

变量作用域

作业

extend

第10关 项目实操 田忌赛马

明确项目目标

分析过程，拆解项目

逐步执行，代码实现

- 版本1.0：封装函数，自定属性
- 版本2.0：随机角色，随机属性
- 版本3.0：询问玩家出场顺序
- 版本4.0：3V3战斗，输出战果

作业

index()函数

第11关 bug

bug 1：粗心

bug 2：知识不熟练

bug 3：思路不清

bug 4：被动掉坑

作业

第12关 两种编程思维，类

两种编程思维

类是一个函数包

类中可以放置函数和变量

第0关 PRINT()函数，转义字符，变量赋值

```
import time

print ('在'+time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())+'，我写了人生中第一行Python代码\n它的内容虽然简单，不过是平凡的一句print(520)\n但我知道：我的编程之路，将从最简单的520开始\n在我点击运行的同时，一切在在这一刻开始变得不同\n以下，是这行代码的运算结果：' )
print(520)
```

```
在2019-09-25 09:47:17，我写了人生中第一行Python代码
它的内容虽然简单，不过是平凡的一句print(520)
但我知道：我的编程之路，将从最简单的520开始
在我点击运行的同时，一切在在这一刻开始变得不同
以下，是这行代码的运算结果：
520
```

PRINT()函数

print()函数的主要功能：

打印内容.让计算机把你给它的指令结果，显示在屏幕的终端上。

```
print('千寻')
```

```
千寻
```

print()函数由两部分构成：

1. 指令：print；

2. 指令的执行对象：在print后面的括号里的内容。

虽然你只是输入了一个简单的print，但在背后，这段Python代码却帮你做了这样的一些事情：

- （0）我们向计算机发出指令：“打印‘千寻’这两个字”；
- （1）Python把这行代码编译成计算机能听懂的机器语言；
- （2）计算机做出相应的执行；
- （3）最后把打印结果呈现在我们面前。

引号的用法

> print(520) print('千寻')

这个单引号是干啥用的？为什么打印文字的时候需要加上引号呢？

这是因为，计算机的脑回路比较特别：只能理解数字，却读**不懂文字**。因为数字和数学运算是标准化、有固定格式的，而文字却能够千变万化。

这，便是print()函数中引号的用法：当括号内有引号的时候，就相当于告诉计算机——你不需要执行多余的操作，我输入什么，你就打印什么。

此时，计算机会听话地执行你的命令，直接把引号内的内容打印出来。

在print()函数内不仅能使用单引号，还能使用双引号，两者的效果没什么区别，都能让你打印出一行文本。

有些时候，单引号和双引号可能会在括号内同时出现，比如print("Let's play")。

这种情况下，需要注意区分清楚哪个引号是属于print()函数结构，哪个引号是属于你要让计算机打印的内容，可别“混搭”了。

说这么多，运行一下代码就懂了。直接在下面代码框中点击运行，观察运行结果。（点击运行前先思考一下，在第三个print()函数中，哪个引号才属于print()函数结构）

```
print('一起玩吧')
#括号内是单引号的情况。
print("一起玩吧")
#括号内是双引号的情况。
print("Let's play")
#括号内单双引号同时存在的情况。
```

```
一起玩吧
一起玩吧
Let's play
```

很明显，在print("Let's play")中，没有被打印出来的引号就属于print()函数结构啦。

不过，在print()函数中，引号里的内容其实也不一定非得是文字，还可以是英文和数字。

但无论你往引号里放啥内容，计算机看到带引号的内容时都会理解为：哦，那我就原样把引号里的内容“复印”一份，显示在终端上吧。 相应地，在括号内没有引号的情况下，我们应该往括号内放入计算机能够“理解”的内容，如：数字或数学运算。

此时，print函数会让计算机尝试去“读懂”括号里的内容，并打印最终的结果。

怎么样才叫读懂呢？难道计算机还能给你做个阅读理解题不成？

要不然，我们来猜猜看，在代码框输入下列代码的话，计算机会在终端输出什么结果？ print(1+1)

```
print(1+1)
```

```
2
```

在这里，计算机并没有再原样打印“1+1”，因为print(1+1)是计算机能直接读懂的数学运算，所以，它会直接打印出最终的运算结果：“2”。这就是计算机“读懂内容”的表现。

注意

注意，前方高能预警，接下来，我要告诉你一个99%的初学者都踩过的代码bug。

在Python中，默认所有正确的语法，包括标点符号都是【英文】。不小心用了中文标点的话，计算机会无法识别，然后报错。

在终端里，你能看到的最常见的符号报错提示就是**【syntaxError:invalid syntax】**（语法错误：无效语法）。

看到这个报错提示时，你的第一反应就应该是：我的标点符号用对了吗？是英文输入法吗？然后再去检查自己语法有没有写对。 别小看了这个小小的标点符号，它可是许多人、甚至是专业程序员的绊脚石——花了一下午的时间找bug，最后发现只是一个符号出了问题。

所以，我们在debug（解决程序报错）的时候，需要下意识地找找自己是否犯了这样细小却致命的错误。

好啦，重要提示说完，我们来继续学习下一个知识点。

还记得吗？刚才，我们已经用print()函数打印出了千寻的卖身契。

```
print('我愿意留在汤婆婆的澡堂里工作两年，第一年在锅炉房和锅炉爷爷一起烧锅炉水，第二年在澡堂给客人搓背，如果我违背工作内容的话，将在这个世界变成一头猪。')
```

```
我愿意留在汤婆婆的澡堂里工作两年，第一年在锅炉房和锅炉爷爷一起烧锅炉水，第二年在澡堂给客人搓背，如果我违背工作内容的话，将在这个世界变成一头猪。
```

希望你能让卖身契文字的每一个逗号后面都换行显示，打印出这样“自动换行”的效果。

```
print('我愿意留在汤婆婆的澡堂里工作两年，
第一年在锅炉房和锅炉爷爷一起烧锅炉水，
第二在澡堂给客人搓背，
如果我违背工作内容的话，
将在这个世界变成一头猪。')
```

```
  File "<ipython-input-6-1c6e1da4ff0e>", line 1
    print('我愿意留在汤婆婆的澡堂里工作两年，
                                  ^
SyntaxError: EOL while scanning string literal
```

发现了吧，直接在括号内换行根本行不通，反而还让计算机报错了，在print()函数内部不能这样手动换行。

因为计算机是一行行地往下执行命令的，因此，想让它重复执行“换行”这个动作，你得在每一个需要换行的地方都给它一个新的指令才行。

这就要介绍第二种换行的方法了：在print内部使用三引号'''（连续输入三个单引号）来实现自动换行。

嘿嘿，没想到吧，除了单引号和双引号之外，print()里面居然还能放三引号。直接运行下面这段代码，看这次能不能实现汤婆婆想要的效果。

```
print('''我愿意留在汤婆婆的澡堂里工作两年，
第一年在锅炉房和锅炉爷爷一起烧锅炉水，
第二年在澡堂给客人搓背，
如果我违背工作内容的话，
将在这个世界变成一头猪。
''')
```

其实，还有第三种实现换行的办法：使用转义字符。

转义字符

转义字符是一种特殊的字符常量，在编程语言中，我们用转义字符表示不能直接显示的字符，比如换行键、后退键，回车键等。

其中，被用来“换行”的转义字符是\n。

我们可以在需要换行的地方后面都加上转义字符中的换行符号\n。因此，汤婆婆的卖身契还可以这样改：

```
print('我愿意留在汤婆婆的澡堂里工作两年，\n第一年在锅炉房和锅炉爷爷一起烧锅炉水，\n第二年在澡堂给客人搓背，\n如果我违背工作内容的话，\n将在这个世界变成一头猪。')
```

除了\n之外，转义字符还有很多，它们的特征就是反斜杠【+想要实现的转义功能首字母】。

比如换行\n代表（+newline）；退格\b代表（+backspace）；回车\r代表（+return）。大家可以按照这种方法记住转义字符的含义。

我把常用的一些转义字符做了个总结：

别担心，这张图上的内容你不需要背下来，先收藏图片，以后要使用转义字符时再看图查找就好。

除了排版文字之外，转义字符还能让计算机对符号进行区分。

有时候，我们在打印的过程中，可能会遇到符号重复的问题，尤其是在需要打印英文的时候。比如，下面这个例子：

```
print('let's go')
```

这段代码会让计算机报错，因为Python是从左往右匹配单（双）引号的——当它读到一个引号时，会在心里嘀咕：呀，这是不需理解，原样打印的内容。当它再读到下一个引号时，想：好！要打印的内容结束了！

所以，在计算机眼里，上面的【'Let'】是要打印的内容，而后面的东西呢，它并不能读懂，于是产生一个报错。

我们来拆解一下这段错误代码：第一个和第三个单引号属于print()函数的结构，第二个单引号则是纯粹的符号。

要让计算机学会区分第二个单引号，我们需要用到转义字符\。

```
print('let\'s go')
```

现在，你已经学会了使用基本的转义字符来更好地打印内容，在后面的学习里，你会在实践中掌握更多转义字符的用法。

为了让她能拿回自己的名字，不被汤婆婆永远留在鬼神的世界。你需要帮助千寻将她的名字"收纳"好，等她将名字取出的那一天。

为了完成"收纳"的行动，你需要借助【变量和赋值】的力量。

变量与赋值

我们先来看看下面这行代码

```
name='千寻'
```

这就是一个常见的"给变量赋值"的动作。在这里，name就是一个变量，这行代码的意思，就是把"千寻"这两个字【赋值】给了"name"这个【变量】。

这就好比，为了让千寻在日后能在计算机中更方便地找到她的名字，我们帮她把名字收纳进了一个小盒子里，并在这个盒子上贴了个叫"name"的标签。

可是，我们为什么要对信息进行"收纳"呢?

因为，每台计算机都要存储和操作成千上万的数据，这就等同于我们家里囤了成千上万的东西一般，不加以整理的话，根本找不到想要的东西在哪。

为了让家看上去更整洁，我们可以把不同的东西放进不同的盒子里，并且做好标记。这样家里不仅会整整齐齐的，也会更方便我们自己取用东西。

同理，计算机也是这么做的，它会把成千上万的数据放在不同的"盒子"里，这样才方便它存储和操作数据。

这个"盒子"在Python中被称为变量，你可以在这个"盒子"里放任何你想放的内容。

而这个往盒子里装东西的过程，在代码的魔法世界被称为【赋值】。

在千寻签完合约后，汤婆婆就把装她名字的"盒子"夺走了。她告诉千寻：你现在有了新名字，叫做小千。

于是，"千寻"这两个字对于她来说成为了一个陌生的字眼，她再也不记得原本的名字了。

这时，装千寻名字的"盒子"就被重新赋值变成了：

```
name='小千'
```

现在，请你运行下面这段代码，看一下打印函数的最终输出结果是什么。

```
name='千寻'

name='小千'

print(name)
```

```
小千
```

你的结果是什么，是不是【小千】？可这又是为什么呢？我们对“name”这个变量第一次的赋值不是【'千寻'】吗？为什么打印的结果是第二次的赋值呢？

这就涉及到了变量的特点：变量之所以称为变量，是因为它保存的数据是可以随意变化的。

刚才我们讲，我们可以将变量当作一个盒子，你可以在这个盒子里放任何内容，但盒子都有它的最大容量，不能无限制地塞东西。

偏偏在代码世界里，盒子的容量又特别小，仅为1。所以当你放进新的东西时，旧的东西就会被替代掉。

在我们上面的案例中，第一行代码：【name='千寻'】表达的意思是：把千寻这个名字放到name这个变量“盒子”中。

由于，计算机是一行行自上而下执行代码的，所以当运行到第二行代码【name='小千'】时，变量“name”里存储的【'千寻'】就被替换成了【'小千'】。

所以，当运行到第三行print(name)时，我们打印出的结果，就自然而然的是小千了。

就这样，汤婆婆拿走了千寻的名字，剥夺了她的自我。其实，不仅是千寻，名字对于我们每个人来讲都具有非同寻常的意义，它从出生起就跟随我们，帮我们形成独立的个体。

而给变量取名字也是如此，我们要注意变量的命名规范，才能让我们在纷繁复杂的代码世界，更为方便地使用它。

变量的命名规范

所以，在变量与赋值中，你要注意的第一个规则就是：你需要学会给变量取一个合适的、标准的名字。

对于变量的命名，我们可以遵循以下规范：

比如说，如果信息是姓名，那么变量可以命名为name；如果信息是数字，那么变量名则应该叫做number。

虽然变量可以起任何名字，但是起合适的名字，才能让我们一眼找到对应的数据，所以number是最合适数字存放的变量。

第二点要注意的规则是：在代码世界中，赋值符号=不是左边等于右边的意思。仅仅表示赋值动作：把右边的内容放到了左边的盒子中。

代表左右两边相等的符号，是比较运算符==，虽然他俩长得像，但代表的却是完全不同的意思，可千万不要搞混了。 下面，让我们跟着变量和赋值的用法，回到千寻的故事中，帮千寻逃脱汤婆婆的魔掌。

时间不断流逝，千寻意识到，自己不能甘于机械地打工，必须要回到人类世界了！于是，她去找汤婆婆对峙，希望汤婆婆把名字还给她。

汤婆婆心想：反正她肯定记不得自己的名字了，我就随便写几个让她选吧，选对了就放她走。于是，汤婆婆在纸上写下了这几个名字：

确实，时间早已冲淡了小千对于名字的记忆，她早就不记得自己叫什么了。但是善良、勇敢、坚强的小千赢得了汤婆婆的助手白龙的帮助。那么，白龙要怎么做才能让小千获得真正的名字呢?

需要你来助一臂之力的时候到了！请你运用学过的赋值知识点，将储存在我们盒子中的名字交给白龙。

请将上面三个名字用name依次赋值，并保证交给白龙的变量内，存放的是小千真正的名字。（提示：使用print()函数，变量的值总是等于最后一次赋给它的内容。）

总结

```
import time,random

print('''============================

           埋める年齢

======================''')

time.sleep(2)

print('你好呀ヽ✿゜▽゜)ノ我叫小埋~想和我玩吗？先猜对我的年龄吧~~')

time.sleep(2)

user_guess=int(input('我的年龄在1-10岁之间，请输入你猜的答案吧(♦◡♦)（必须是整数1-10哦）'))

print('哦~~你猜我'+str(user_guess)+'岁呀，来看看你猜的对不对……')

time.sleep(2)
```

```
age=random.randint(1,10)

if user_guess==age:
    print('嘻嘻！猜对啦！我们做好朋友吧（￣▽￣）↗再见啦！')
elif user_guess<age:
    print('猜错啦！我才没有那么小呢(＞﹏＜)不和你玩了！')
else:
    print('猜错啊！我才没有那么老呢（；´д｀）ゞ不和你玩了！')
```

```
==============================

            埋める年齢

======================
你好呀ヽ✿゜▽゜)ノ我叫小埋~想和我玩吗？先猜对我的年龄吧~~
```

作业

1. 打印皮卡丘
2. 打印无脸男对千寻说的话

```
print( '''
      へ          /|
    /\7        ∠_/
   /  |      /   /
  |  Z _,<   /      /`ヽ
  |          ヽ     /    〉
   Y           `   /     /
  ｲ●  、 ●     ⊂⊃〈    /
  ()   へ         |   \〈
    >ｰ 、_   ィ    |  / /
   / へ     /  ﾉ<|  \ \
   ヽ_ﾉ    (_/    |/ /
    7              |/
    >―r ￣￣`ｰ―__''')
```

```
print('千寻你好，人们叫我\'无脸男\'\n这个世界的人都选择无视我\n只有你看到了我并和我打招呼\n我感到很孤单，很孤单\n你愿意和我成为朋友吗？')
```

第1关 数据类型，数据拼接，数据转换

数据类型

数据类型有三种

- 字符串(str)
- 整数(int)
- 浮点数(float)

他们三者分别具有不同的属性：

字符串(str)

首先，我们要讲的就是代码届的"坦克"——字符串：

字符串英文string，简写str。作为Python届的坦克，此数据类型组团必备随处可见，皮糙肉厚战斗力爆表。作为最常用的数据类型，字符串的识别方式非常简单——有层名为【引号】的皮。

只要是被【单/双/三引号】这层皮括起来的内容，不论那个内容是中文、英文、数字甚至火星文。只要是被括起来的，就表示是字符串类型。

```
slogan = '命运！不配做我的对手！'
attack = "308"
gold = "48g"
blood = '''+101'''
achieve = "First Blood!"

print(slogan)
print(attack)
print(gold)
print(blood)
print(achieve)
```

```
命运！不配做我的对手！
308
48g
+101
First Blood!
```

关于字符串，大家一定要记得：字符串类型必须有引号的辅助。不然你这坦克，就只能被报错按在地上摩擦摩擦，躺遍召唤峡谷的每个角落。

下面，我们来小结一下字符串的知识点。

可是有的时候，某些数据它脱了引号的马甲，也照样能打印出来。

而这，就是我们要介绍的，常见数据类型中的第二种：代码届的刺客——整数。

整数

整数英文为integer，简写做int。代码世界的整数，和我们数学课本中定义的一样：是正整数、负整数和零的统称，是没有小数点的数字。

可我为什么说整数是代码届的刺客呢？其实，就本质来讲，整数和刺客一样，是一个灵活多变、可攻可受的脆皮英雄。

首先，整数可以不穿名为引号的马甲，直接跟print()函数组团搞输出，比如下列代码：

```
print(566)
```

```
566
```

但是由于没有引号这层皮的保护，整数只能单独“行动”。一旦碰上其他文字类数据，譬如中文、英文。整数这个脆皮刺客，立马被报错秒杀。

```
print(6小灵童)
print(6skr)
```

```
  File "<ipython-input-3-7965f0c05cac>", line 1
    print(6小灵童)
            ^
SyntaxError: invalid syntax
```

其次，整数的“灵活多变”又体现在它的用处上，它不仅可以脱离引号被直接打印，还可以和运算符号结合，进行数学计算。

下面，就请你来点击运行按钮，跑一下这些代码。

```
print(499*561+10620-365)
```

```
290194
```

不过说到计算，肯定要使用运算符了。Python世界的算数运算符，和我们平时在纸上写的运算符号有相同点，也有不同点。

首先，我们来看看不同的部分——样式：Python的运算符在写法上，与我们平时使用的运算符稍微有点区别。 运算优先级：Python世界的运算优先级，和我们平时的计算优先级是一样的。

掌握了方法，下一步就是实操“刺客”——整数的计算功能了。请你根据以下代码框中的题目，写出运算公式并计算出结果。【要点提示：运算符，运算优先级，print()函数】

```
print((362-330)/10)
```

```
3.2
```

下面，我们来总结一下整数这部分的知识点： 到现在为止，我们已经知道了字符串和整数，这两种数据类型长啥样。可是，我们刚刚计算铭文攻击时产生的结果【3.2】，跟上面那两种类型长得都不一样，那【3.2】又叫啥呢?

它就是我们常见三种数据类型中的最后一种：代码届自带眩晕技能的法师——浮点数。

浮点数

整数是不带小数点的数字，那么相对的，带小数点的数字，就是浮点数。非常好识别，比如下列代码中的数字都是浮点数。

浮点数的英文名是float，与整数(int)和字符串(str)不同，浮点数没有简写。

那我为什么说浮点数能让你眩晕呢？虽然浮点数识别起来非常简单，但是其运算却晦涩难懂、让人头秃。

正如世界级C++大佬Herb Sutter说的：“世上的人可以分为3类：一种是知道自己不懂浮点运算的；一种是以为自己懂浮点运算的；最后一种是极少的专家级人物，他们想知道自己是否有可能，最终完全理解浮点运算。”

所以，浮点运算没有你想的那么简单。那么现在，请大家跟我通过一道题，来感受一下浮点数的“眩晕技能”：

请你先心算一道题【0.55+0.3】，结果是不是【0.85】？下面我们让Python算一下，请直接点击运行按钮，并阅读答案：

```
print(0.55+0.3)
```

```
0.8500000000000001
```

这是因为，Python计算浮点数的方式与我们不一样。Python计算浮点数时，会先把0.55和0.3转化成二进制数【注：二进制数由0和1表示，逢二进一】，如下列代码：

```
#进制转换
0.55(十进制) =0.10001100110011001100110011001100110011001100110011001101(二进制)
0.3(十进制) =0.01001100110011001100110011001100110011001100110011001101(二进制)
```

然后，这俩二进制数通过二进制法则运算后，再通过复杂的计算公式，将二进制结果转成十进制小数。

经过这两次转换，小数点后面就产生了多余的“尾巴”。这样，就造成了我们与程序计算结果的差异。

不过对于浮点数，你也不用太担心，因为在前期我们很少会涉及浮点数运算。而在后期，随着学习的深入，你会接触到更多的相关知识，辅助你对浮点数的进一步理解。

然后，我们来看一下浮点数这部分的主要知识：

以上，就是我们最常接触的三种数据类型：坦克字符串、刺客整数、法师浮点数的全部内容了。

数据拼接

数据拼接的方法可简单了，就是利用数据拼接符号【+】，将需要拼接的变量连在一起就行了。

```
hero = '亚瑟'
enemy = '敌方'
action = '团灭'
gain = '获得'
achieve = 'ACE称号'

print(hero+action+enemy+gain+achieve)
```

```
亚瑟团灭敌方获得ACE称号
```

有了拼接符号，我不仅可以输出亚瑟团灭敌方获得ACE称号，还可以调整变量的拼接顺序，输出不同的内容。

下面我们就来试一下，请你补全下列代码，同时打印出两个结果：【结果1】亚瑟秒掉李白获得First Blood；【结果2】李白秒掉亚瑟获得First Blood。【要点提示：print()函数，拼接符号+，无需标点符号，输出结果需与要求一模一样】。

```
hero1 = '亚瑟'
hero2 = '李白'
action = '秒掉'
gain = '获得'
achieve = 'First Blood'

print(hero1+action+hero2+gain+achieve)
print(hero2+action+hero1+gain+achieve)
```

```
亚瑟秒掉李白获得First Blood
李白秒掉亚瑟获得First Blood
```

学会了数据拼接，就能让我们的数据整合更为灵活，组合出任意我们想要的内容。

而这，就是我们数据拼接部分的要点：

那么，为了输出结果【亚瑟秒杀5敌方获得Penta Kill】，我们是不是可以把代码写成这样？

```
hero = '亚瑟'
enemy = '敌方'
action = '秒杀'
gain = '获得'
number = 5
achieve = 'Penta Kill'

print(hero+action+number+enemy+gain+achieve)
```

```
---------------------------------------------------------------------------
----

TypeError                                 Traceback (most recent call
last)

<ipython-input-9-2245ce2ea417> in <module>
      6 achieve = 'Penta Kill'
      7
----> 8 print(hero+action+number+enemy+gain+achieve)
```

```
TypeError: can only concatenate str (not "int") to str
```

梦想是美好的，但现实是残酷的，我们立马被报错【TypeError：can only concatenate str (not "int") to str】（类型错误：只能将字符串与字符串拼接）按在地上摩擦摩擦。

虽然通过报错提示，我知道了错误来源于print()函数内，数据类型的拼接错误。但是到底是哪个数据类型不对呢？我们要怎么查看数据类型呢？

拿刚刚那段报错代码来说，由于变量是由我们定义的，所以我们可以用人眼查找错误源。

但是在未来，当我们学会了更为复杂的命令，可以用代码对其他人提问，收集其他人的答案时。我们无法确定他们回复你的内容，是用什么数据类型写的。

所以，我们要学会善用Python的魔法，使用Python中一个已有的，既简单又实用的函数，来帮助我们查询不确定数据的类型。而这个函数，就是type()函数。

TYPE()函数

那么，这个type()函数要怎么使用呢？答案就是：只需把查询的内容放在括号里就行。

只要你把内容写在括号里，type()函数就会立即将它的类型查询出来。下面，请你用刚才的代码来试一下。请直接点击运行按钮，只要运行通过就请继续课程。

```
hero = '亚瑟'
enemy = '敌方'
action = '秒杀'
gain = '获得'
number = 5
achieve = 'Penta Kill'

type(hero)
type(enemy)
type(action)
type(gain)
type(number)
type(achieve)
```

```
str
```

哎？怎么什么都没出现呢？刚刚不是说过，只要按照type(需要查询的数据)这个格式写对了，就能出现数据的类型吗？

其实，什么都没有就对了。对于Python来说，你只是下了一个命令——给我查询类型哟～然后Python就老老实实地查询了类型，再然后……就没有了。

在你的命令里，并没有告诉Python："你查到以后，记得告诉我这个类型的结果啊～"所以，异常老实的Python就把查询到的数据类型……存起来了，没告诉你。

那么，我们如何输出结果呢？

答对了，我们需要运用在第0关学习过的打印函数print()函数。

下面，请你再实操一下，补全代码，运用print()函数，将查询函数的结果打印出来。例如：print(type(hero))

```
print(type(hero))
```

```
<class 'str'>
```

```
print(type(hero))
print(type(enemy))
print(type(action))
print(type(gain))
print(type(number))
print(type(achieve))
```

```
<class 'str'>
<class 'str'>
<class 'str'>
<class 'str'>
<class 'int'>
<class 'str'>
```

终端里出现的结果，是不是除了5个<class 'str'>，还有1个<class 'int'>?

str代表字符串，全称为string，整数的英文简称为int，全称为integer。

在type()函数的帮助下，我们知道了：原来字符串里面出了一个“奸细”整数。难怪当时在终端区会给我报错说：数据类型不同呢。

可是为什么不同类型的数据不能拼接在一起呢？一句话：圈子不同不相融。

我打个比方，这就类似我饭阿森纳，你饭热刺。虽说咱俩都是足球粉丝，但喜欢的球队却是死对头，所以根本没办法交朋友，没法把咱俩放在一起。

但是，在某些时候，咱俩总得有低头不见抬头见，在一个球场看球的时候。那么这时候要怎么办呢？要如何把两个不同类型的数据拼在一起呢?

这就涉及到了我们这关最后一个知识点：数据转换。

数据转换

负责转换数据类型的函数一共有3种：str()、 int()和float()。

下面，我们先来学习一下，能帮我们解决刚才的代码Bug，拿到五杀的str()函数。

str()函数

str()函数能将数据转换成其字符串类型，不管这个数据是中文、数字、标点还是火星文，只要放到括号里。这个数据就能摇身一变，成为字符串类型。

下面，请你看看老师是如何运用str()函数解决报错，完成数据拼接的。请先观察下方代码的书写，然后点击运行按钮。

```
hero = '亚瑟'
enemy = '敌方'
action = '秒杀'
gain = '获得'
number = 5
achieve = 'Penta Kill'

print(hero+action+str(number)+enemy+gain+achieve)
```

```
亚瑟秒杀5敌方获得Penta Kill
```

是不是挺简单的？我们只需通过str(number)一个步骤，便可以将整数类型的5转化为字符串类型的5 。成功完成数据拼接，拿到五杀。

```
slogan = '我一个电话立马有'
character = '山兔'
unit = '只'
place = '在你家楼下'
action = 'hola hola'
number = 10000

print(slogan+str(number)+unit+character+place+action)
```

```
我一个电话立马有10000只山兔在你家楼下hola hola
```

整数转换字符串类型的方法是不是很方便？那如果我告诉你，还有一种转换成字符串的方法，而且这种方法你已经学会了呢？

那就是借用引号的帮助，比方说我一个电话立马有10000只山兔在你家楼下hola hola这个结果，我也可以写成下面这样：

```
print(slogan+'10000'+unit+character+place+action)
```

你看，我们是不是用两种不同的写法：str()函数和引号，输出了同一种结果？

可是，为什么引号内我们使用的是数字，而不是变量名number呢？这是因为，当我们使用引号时，引号里的东西，都会被强制转换为字符串格式。

所以，如果我们把变量名number放进引号里后，被强制转换为字符串的，将是变量名number，而不是变量名代表的数字10000。

可能有人会在心里吐槽：既然引号用着这么麻烦，那我直接用str()呗～你看这多省事。

其实，我并不是单单地在教你编程方法，而是想传达给你一种编程思维——条条大路通罗马。

代码的世界千变万化，同一个结果可能有千百万种答案，如果我们只心安理得地，局限在一种思维模式下。那这个思维定势一定会阻挡你的进步，阻止你成为真·大佬。

所以，我希望你能尽情创新，尝试不同的选择，就像老祖宗说的：胆大心细，绝对没错。

好了，题外话到此为止。现在我们来实操一下，锻炼一下编程思维。请补齐下方代码，并用两种方法打印出结果：我一个电话立马有500个吴亦凡在你家楼下skr skr。

```
slogan = '我一个电话立马有'
character = '吴亦凡'
unit = '个'
place = '在你家楼下'
action = 'skr skr'
number = 500

print(slogan+str(number)+unit+character+place+action)
print(slogan+'500'+unit+character+place+action)
```

```
我一个电话立马有500个吴亦凡在你家楼下skr skr
我一个电话立马有500个吴亦凡在你家楼下skr skr
```

下面，我们来小结一下str()函数部分的知识点。 既然整数类型能转换为字符串，那字符串类型能转换为整数吗？ 那必须的。

int()函数

数据转换为整数类型的方法也很简单，就是int()函数。其使用方法同str()一样，将你需要转换的内容放在括号里就行，像这样：int(转换的内容)。

下面我们来试验一下，请补全代码，计算出bug和hair这两个变量的和。

```
bug = '666'
hair = '0'

print(int(bug)+int(hair))
```

```
666
```

不过对于int()函数的使用，大家要注意一点：只有符合整数规范的字符串类数据，才能被int()强制转换。

别看它虽然只有一句话，但它其实带有三层含义：首先，整数形式的字符串比如'666'和'0'，可以被int()函数强制转换。

其次，文字形式，比如中文、火星文或者标点符号，不可以被int()函数强制转换。

最后，小数形式的字符串，由于Python的语法规则，也不能使用int()函数强制转换。

比方说下列代码，如果点击运行，程序会报错【ValueError：invalid literal for int() with base 10: '3.8'

```
print(int('3.8'))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
last)

<ipython-input-18-b69d84cf2e14> in <module>
----> 1 print(int('3.8'))
```

```
ValueError: invalid literal for int() with base 10: '3.8'
```

这是不是意味着，浮点数不能转换成整数类型了？

不，虽然浮点形式的字符串，不能使用int()函数。但浮点数是可以被int()函数强制转换的。

下面，请你先观察下方代码框中的代码，然后点击运行按钮。

```
print(int(3.8))
```

```
3
```

上方的代码串，就是一条将浮点数3.8强制转换的语句。但是，为什么输出的结果是3呢?

你看，int()函数的本质是将数据转换为整数。所以对于浮点数，int()函数会做取整处理。但是，同我们平时对小数四舍五入的处理方法不同，int()函数会直接抹零，直接输出整数部分。

最后，我们来小结一下int()函数的知识点。

可是，如果遇到这种情况：字符串就是小数形式，比如'3.8'这种。我不想让它变为整数3，想让它保持小数形式的3.8，那我要怎么做呢?

这就涉及到了我们这关，最后一个知识点——float()函数。

float()函数

首先float()函数的使用，也是将需要转换的数据放在括号里，像这样：float(数据)。

其次，float()函数也可以将整数和字符串转换为浮点类型。但同时，如果括号里面的数据是字符串类型，那这个数据一定得是数字形式。

那么，经过之前str()和int()操练，float()函数是不是好懂了一些?

下面，我们来打段代码练练手。请补齐代码，将下列变量，转换为浮点类型，并打印出结果。

```
height = 183.5
weight = 79
age = '30'

print(float(height))
print(float(weight))
print(float(age))
```

```
183.5
79.0
30.0
```

总结

下面，我们来解决刚进入float()函数部分时，我们的问题：字符串就是小数形式，比如'3.8'，我如何将这个字符串，直接转换为浮点类型呢?

比方说我想使用以下变量，输出这样一个结果人工智障说：3.8+1等于4。

```
word = '3.8'
number = 1
sentence = '人工智障说：3.8+1等于'

print(sentence+str(int(float(word)+number)))
```

```
人工智障说：3.8+1等于4
```

```
word = '3.8'
number = 1
sentence = '人工智障说：3.8+1等于'
print(float(word))
```

```
3.8
```

```
print(float(word)+number)
```

```
4.8
```

```
print(int(float(word)+number))
```

```
4
```

总结

练习

1. str()打印
2. 使用数据转换str()、int()、float()及数据拼接符号+，打印一句话

```
#第一句话：1人我编程累碎掉的节操满地堆
#第二句话：2眼是bug相随我只求今日能早归

number1 = 1
number2 = 2
unit1 = '人'
unit2 = '眼'
line1 = '我编程累'
line2 = '是bug相随'
sentence1 = '碎掉的节操满地堆'
sentence2 = '我只求今日能早归'

print(str(number1)+unit1+line1+sentence1)
print(str(number2)+unit2+line2+sentence2)
```

```
1人我编程累碎掉的节操满地堆
2眼是bug相随我只求今日能早归
```

```
#请运用所给变量，使用数据转换str()、int()、float()及数据拼接符号+，打印一句话：
脸黑怪我咯7张蓝票一个SSR都没有

slogan = '脸黑怪我咯'
number = '7.8'
unit = '张'
sentence = '蓝票一个SSR都没有'

print(slogan+str(int(float(number)))+unit+sentence)
```

```
脸黑怪我咯7张蓝票一个SSR都没有
```

第2关 条件判断 嵌套 如何写嵌套代码

天猫精灵，打开灯。那么，由于缺少开门的前提条件，即使你进了门，灯也打不开。

所以，我们下的命令应该是这样：天猫精灵，如果我开门，你就打开客厅的灯。这时，我们就从本拉登，变成了本开灯。

而这个下命令的如果…就…逻辑，就是与计算机沟通的逻辑——条件判断，其作用就是明确地让计算机知道：在什么条件下，该去做什么。 对于Python也是同样。Python之所以能做很多自动化任务，比如自动抓取网页关键词，自动下载小电影等，就是因为它可以执行条件判断。

Python的条件判断，就像霍格沃茨的分院帽，它通过识别你的条件，产生对应的结果，把你分在适合的学院中。

下面，我们来get一下逻辑判断的趣味性。请同学们先点击下面代码框左侧的运行选项，然后在右侧的终端处按照提示语，填写选择的数字。

```
import time
print('如果你英勇无畏，那你就是小狮子')
time.sleep(2)
print('如果你高傲纯血，蛇院欢迎你')
time.sleep(2)
print('如果你睿智公正，你会在拉文克劳找到一席之地')
time.sleep(2)
print('如果你勤劳诚实，赫奇帕奇会成为你最强的后盾')
time.sleep(2)
print('那么，如果让你来选择的话，你会加入哪个学院呢？')
time.sleep(2)
print('请在以下四个选项【1 格兰芬多；2 斯莱特林；3 拉文克劳；4 赫奇帕奇】中，选择你想去的学院吧！')
```

```
如果你英勇无畏，那你就是小狮子
如果你高傲纯血，蛇院欢迎你
如果你睿智公正，你会在拉文克劳找到一席之地
如果你勤劳诚实，赫奇帕奇会成为你最强的后盾
那么，如果让你来选择的话，你会加入哪个学院呢？
请在以下四个选项【1 格兰芬多；2 斯莱特林；3 拉文克劳；4 赫奇帕奇】中，选择你想去的学院吧！
```

条件判断

在Python宇宙，条件判断语句总共有三种表现形式，我们先从最简单的单向判断：if开始说起：

单向判断：if

A long time ago in a galaxy far, far away……有一位出生于泰坦星，无情、残忍、实力强劲的强者——紫薯精！啊不对，这位强者的名字叫灭霸……

相传，宇宙中存在着六颗宝石——力量、时间、空间、灵魂、现实和心灵。这六颗宝石是宇宙中最强大的东西。如果一个人同时拥有六颗宝石，ta就可以实现包括毁灭宇宙在内的所有事情。

你发现，上面的故事中，有一个如果……就。那么，如果我们要用代码来"翻译"这段话，就可以用条件判断语句中的单项判断：if来表述。直接运行下面这段代码，看看结果。

```
stonenumber=6
#为宝石数量赋值

if stonenumber>=6:
    #条件：如果你拥有的宝石数量大于等于6个
    print('你拥有了毁灭宇宙的力量')
    #结果：显示'你拥有了毁灭宇宙的力量'的毁灭宇宙的力量'的结果
```

```
你拥有了毁灭宇宙的力量
```

那么，这段代码是如何实现的呢?

首先，第一行代码，用赋值运算符=对当前情况进行赋值：将你拥有的宝石个数6赋值给变量——宝石数stonenumber。

第二步，确定if条件：如果变量stonenumber的值>=个数6时，就执行冒号后，下一行的命令。

第三步，用print()命令打印出结果：你拥有了毁灭宇宙的力量。

所以，if语句的单向判断逻辑，我们可以这样归纳：

在这里，你可能注意到了一个细节：在条件判断代码中的冒号:后、下一行内容的前面，会空几个格，但这是为什么呢？ 首先，在计算机的沟通语言中，空格的学名叫缩进，比如我们写文章都要空两个格，这就叫首行缩进。

对于Python而言，冒号和缩进是一种语法。它会帮助Python区分代码之间的层次，理解条件执行的逻辑及先后顺序。

并且，在if条件语言中，缩进不需要我们手动按空格键。当你用英文输入法打:后按回车，我们的开发工具（用来编写Python代码的程序）为了方便大家编程，会自动实现下一行代码，向右缩进的功能。

此时，被缩进的内容（print()函数）和if条件语句组成了一个代码块（一个整体），成为了if条件下的内部命令。

这意味着：如果赋值满足if条件，计算机就会准确执行if条件内部的命令。

通俗点讲，我们可以把if当作一个黑社会大哥，冒号后的内容都是他的小弟，当老大if提出一个条件时，他组织下的小弟就会自动执行任务。

```
stonenumber=6
#为宝石数量赋值

if stonenumber>=6:
    #条件：如果你拥有的宝石数量大于等于6个
print('你拥有了毁灭宇宙的力量')
    #结果：显示‘你拥有了毁灭宇宙的力量’的毁灭宇宙的力量‘的结果
```

```
  File "<ipython-input-10-b78376334eb9>", line 6
    print('你拥有了毁灭宇宙的力量')
        ^
IndentationError: expected an indented block
```

是不是会出现IndentationError: expected an indented block（缩进错误：期望一个缩进块）的报错？ 这是因为，当我们去掉缩进时，if条件和print命令成为了两个不同的代码组，属于平行关系。你看，print小弟都自立为王了，他自然就不会执行if老大的命令了。

所以，我们总结一下单向判断：if的代码逻辑及语法格式：

下面，我们回到灭霸的故事：当灭霸得知无限宝石的能力之后，他决定集齐六颗宝石，完成灭掉一半宇宙生灵，实现他所谓的“人口控制”计划。

很快，他就通过完虐对手的几场战斗，从其他人手中抢到了两颗宝石，展现漫威宇宙“最强反派”的实力。

虽然还没完成收集6颗宝石的目标，但现在拥有了2颗宝石的灭霸，会拥有怎样的能力呢?

请修改下方代码框中无限宝石的代码：在if条件不变的情况下，修改第一行的赋值语句，把【stonenumber=6】改为【stonenumber=2】，并点击运行，看看会出现什么结果。

```
stonenumber=2
#为宝石数量赋值

if stonenumber>=6:
    #条件：如果你拥有的宝石数量大于等于6个
    print('你拥有了毁灭宇宙的力量')
    #结果：显示‘你拥有了毁灭宇宙的力量’的毁灭宇宙的力量‘的结果
```

这时，你可能会在心里犯嘀咕：纳尼？别提结果，为什么连个报错都没有呢?

别急，在这个操作下，终端里不显示结果就是正常的。我来解释一下这是怎么回事儿。

首先，你要知道，第一行的赋值语句【stonenumber=6】是后面if条件判断语句的前提情况，如果没有这个前提，后面的if条件就没有被判断的对象了。

老师讲过，计算机是一行行往下执行代码的。所以，当第一行赋值情况的前提不满足if的条件时，Python的逻辑就会判断：条件不满足，无法执行if条件下的命令，从而自动跳过，执行下一行命令。但是，在上面的代码中，除了if之外，我们并没有设置其他的命令去让Python执行。因此，Python自然就无法输出结果。

也就是说，由于灭霸的宝石数量还太少，没有满足6颗宝石的if条件，if条件下的打印命令就无法生效，自然没办法获得摧毁宇宙的力量。

作为最强反派，灭霸当然不会就此善罢甘休，他要继续寻找宝石，直到集齐6颗，达到他的目的为止。

那么，这个如果6颗宝石的条件没满足，就做其他事情——寻找宝石的判断，在Python中要如何实现呢?

双向判断：if…else…

在获得了收藏家手里的现实宝石后，拥有三颗宝石的灭霸选择用狡猾、残忍的手段，逼迫自己的养女卡魔拉，说出了第四颗灵魂宝石的下落——沃弥尔星。

在Python宇宙，我们可以利用if的双向判断形式：if…else…去实现这个行动：当宝石数量还没达到6颗的时候，需要带着卡魔拉去沃弥尔星寻找灵魂宝石。

下面，请你直接运行下面的代码，看程序是如何识别的：

```
stonenumber=3
#赋值语句：为宝石数量赋值

if stonenumber>=6:
    #条件：如果你拥有的宝石数量大于等于6个
    print('你拥有了毁灭宇宙的力量')
    #结果：显示‘你拥有了毁灭宇宙的力量’的结果
else:
    #条件：当赋值不满足if条件时，执行else下的命令
    print('带着卡魔拉去沃弥尔星寻找灵魂宝石')
    #结果：显示‘去找灵魂宝石’的结果
```

```
带着卡魔拉去沃弥尔星寻找灵魂宝石
```

但很多时候，我们不能把鸡蛋放在一个篮子里，要做好两手准备：如果不满足条件时，我们要怎么办。

Python则很贴心地，让我们借用if…else…语句，让码农们有了另一种选择——【如果…不满足，就…】 还记得我们在讲单向判断if条件时，老师说过if条件下的内容会自动缩进，所以按照这个思路，else前也应该有缩进才对。

可在这个多向判断语句里，为什么我们还要手动删去else前的空格呢？那就让我们先来看看，如果有空格会发生什么吧。

请抄写下方代码框中的代码，将else改为缩进格式（即else前空两格）并点击运行（当终端出现红色报错提醒后，点击跳过本题）。

```
stonenumber=3

if stonenumber>=6:
    print('你拥有了毁灭宇宙的力量')
    else:
    #试试看让else缩进个2空格，看会发生什么。
    print('带着卡魔拉去沃弥尔星寻找灵魂宝石')
```

```
  File "<ipython-input-14-11de95df6976>", line 5
    else:
       ^
SyntaxError: invalid syntax
```

Python是不是又爆出了让我们头痛的报错【SyntaxError: invalid syntax】（语法错误：无效语法），把我们按在地上摩擦摩擦？

那么，这是为什么呢？为了找到错误，我们得先了解一下正确语法下if和else运行的原理：

首先，当else条件前没缩进时，if和else各自抱团，形成两个不同的代码块。这时，if条件和else条件是平级的。

其次，在平级关系的前提下，他们表示的意思是条件和其他条件的互斥关系——如果不满足if条件，就执行else其他条件。 如果用我们上面宝石的例子讲解的话，if定义的就是宝石数>=6，而else定义的则是宝石数<6。

而由于赋值【stonenumber=3】，并不满足【if stonenumber>=6:】这个条件，所以不能执行【print('你拥有了毁灭宇宙的力量')】的命令。

只能走第二条else:的路——执行print('带着卡魔拉去沃弥尔星寻找宝石')的结果。

现在，我们归纳一下if...else正确运行的语句逻辑：

1. if和else两个条件判断是平级关系；

2. 当if条件不被满足时，才会执行else下的命令。

看到没，想让else正确运行有两个关键点——平级和if前提条件：当else有一个和它平级的if前提条件，且这个前提条件无法执行时，才会轮到else发挥作用。

这下，你明白了刚才那段代码报错的原因了吧：

1. 在语法方面，else缺少和它平级的if前提条件（见下图）；
2. 第一行的赋值也无法满足if条件。 所以，Python才会犯懵：你到底想让我干啥？算了我报个错吧……

下面，我们来做个练习巩固一下if...else...的知识点：前天晚上，酱酱无比纠结自己要不要吃夜宵，所以她打算上秤称一下，如果体重超过100斤，就不吃了，没有的话，就放心吃。结果……刚好101斤（泪目）……

请在下面代码框里用if...else...把酱酱的夜宵判断语句写出来。【要点提示：1. 为酱酱的体重weight赋值；2. 开始判断，如果if体重超过100斤，打印结果不吃了，否则else，打印结果放心吃吧；3. if和else语句后需跟英文冒号:】

```
weight=int(input('酱酱的体重？'))
if weight>=100:
    print('求求你别吃了。')
else:
    print('放心吃吧，我请')
```

```
酱酱的体重？110
求求你别吃了。
```

刚才你自己用if...else...写出了一个基本的条件判断语句。现在，问题又来了：else一定要和if组团，才能运行吗？

并不是，虽然想要让else生效，一定需要一个平级的前提条件，但这个前提条件却不一定是条件判断语句，还可以是后面你会学到的其他命令语句。

可这个和else平级的命令有个限制条件：它不能是一个“一步到胃”的命令——像print()一样直接得出结果的命令，它得是一个需要再处理的命令。

什么叫再处理呢？白话点讲就是：我得先把食物嚼碎了，咽下去才能进到胃里。回归到计算机语言，就是要让计算机先做个判断（如先运行if条件），看是否通过，如果不通过，再运行else。

举一些具体的例子：像我们未来会学到的，让你痛并快乐着的while循环、for循环，这些都是需要再处理的命令，所以他们也可以作为else的平级命令。

下面，请你先看一下这段代码。这是老师在第五关会教给大家的知识点for循环（现在也许你还不能完全看懂这段代码，不要紧，只关注代码中if和else的结构就好。）

```
for i in range(10):
    if i == 11:
        print('找到结果')
        break
else:
    print('未找到结果')
```

```
未找到结果
```

发现了吗？在这块代码中，else和if不仅不是平级，而且if语句还缩进了，并且else在if外，比if还高一级。那么，这段代码可以打印出结果吗？

请大家将以上代码，按照格式一字不落地抄写在代码区域中（注意缩进及符号的格式），并点击运行。

你运行出“未找到结果”了吗？在上面这段代码中，由于if缩进，成为了for循环的“小弟”。所以else和for的内部条件if无关，只和它的平级条件for循环有关。 这就意味着：根据计算机一行行往下执行命令的语法特点，计算机会先执行for循环下的代码块，完成后再执行else下的命令。

最后，我们来总结一下if…else…语句的知识点。 摆在红女巫面前的问题条件是：灭霸已经拥有5颗宝石，如果宝石数大于等于6，世界会被灭霸毁灭；

但是，如果想让宝石数停留在5个及以下，她必须要亲手毁掉幻视头上的宝石，同时，还要从灭霸手中至少夺回一颗宝石，才会有胜利的希望；

或者，还有第三种胜利的可能——如果想让灭霸手中一个宝石都没有，让一切都未曾发生，那么我们需要穿越时空，回到过去。

那么，这三个如果在Python中要如何实现呢？

多向判断：if…elif…else…

国际惯例，在思考如何用代码实现某个目标时，首先，我们需要先来梳理一下逻辑。

通过上面的故事情节我们知道，在代码中，我们需要添加三个如果：如果宝石数>=6，如果3<宝石数<=5，如果宝石数=0。并产生3个对应的执行结果。

那么，在判断3个或3个以上的条件时，我们就需要借助Python中的多向判断命令：if…elif…else…。

这三者共同构成了多向判断的命令逻辑关系：如果if的条件不满足，就按顺序看是否满足elif的条件，如果不满足elif的条件，就执行else的命令。

并且，当判断的条件超过3个时，中间的多个条件都可以使用elif，如下图。 那么，这个代码要怎么写呢？我们还是在代码中来感受一下elif的逻辑，直接点击运行就好：

```
stonenumber=5
#为宝石数量赋值

if stonenumber>=6:
    #条件：如果你拥有的宝石数量大于等于6个
    print('你拥有了毁灭宇宙的力量')
    #结果：显示‘就拥有了毁灭宇宙的力量’的结果
elif 3<stonenumber<=5:
    #条件：如果想让宝石数量停留在4至5个
    print('红女巫需要亲手毁掉幻视额头上的心灵宝石')
else:
    #条件：当赋值不满足if和elif条件时，执行else下的命令，宝石数量在3个以下
    print('需要惊奇队长逆转未来')
    #结果：显示‘需要惊奇队长逆转未来’的结果
```

```
红女巫需要亲手毁掉幻视额头上的心灵宝石
```

通过上面if和else的积累，多向判断elif的逻辑是不是很好理解?

首先，第一行的赋值，也就是整个条件判断的大前提会按照从上到下的顺序挨个试一遍，看满足哪个条件，满足了就不往下走，不满足就继续试，试到出结果为止。

其次，elif的运行本质上和else一样，都需要一个和elif平级的前提，但是这个前提只能是if。

最后，elif已经带有其他条件的意思，所以elif后也可以不接else。

在上面的代码里，大前提stonenumber=5会对下面的if `elif` else条件逐个扫描，看看自己满足哪一个，就执行哪个条件底下的命令。

很显然，第二个条件elif 3<stonenumber<=5刚好能与stonenumber=5的前提相匹配，于是，执行elif底下的命令：print('红女巫需要亲手毁掉幻视额头上的心灵宝石')

下面，请你练习一下，试着不要看刚才的代码，参考下面的提示，自己来重写一遍刚才我们学过的那段代码，通过修改宝石数的赋值语句，让输出结果变为需要惊奇队长逆转未来。

提示：（1）为宝石数量赋值（数量可为0-3任意数字）（2）条件1：如果宝石数量>=6，显示你拥有了毁灭宇宙的力量（3）条件2：如果3<宝石数量<=5，显示红女巫需要亲手毁掉幻视额头上的心灵宝石（4）条件3：如果是其它情况，显示需要惊奇队长逆转未来

下面，我们来总结一下elif的知识点： 但是，不管复仇者联盟怎么竭尽全力，灭霸还是得到了最后一颗宝石。stonenumber的赋值也变为了6，自此，灭霸拥有了毁灭宇宙的力量。

未有任何喘息，在所有人还在悲伤、震惊、痛苦中徘徊时，灭霸举着镶满了6颗宝石的无限手套打出了响指。

然后，我们迎来了最不想看到的结果：

（友情提示：前方高能预警，非战斗人员请迅速撤离现场……）

```
import time
print('一切都化为虚无')
time.sleep(1.5)
print('一切都化为尘埃')
time.sleep(1.5)
print('他们就这样随风消散')
time.sleep(1.5)
print('冬兵')
time.sleep(1.5)
print('格鲁特')
time.sleep(1.5)
print('红女巫')
```

```
一切都化为虚无
一切都化为尘埃
他们就这样随风消散
冬兵
格鲁特
红女巫
```

虽然套着超级英雄的衣服，但蜘蛛侠彼得·帕克也仅仅是个18岁的高中生而已，在死亡面前他会感到害怕。当然，他也像所有18岁的青少年一样，有着自己成长的烦恼。

他曾只顾忙着在街头巷尾当超级英雄，结果在期末历史考试里只考了26分，荣获“学渣”提名。

但这个“学渣”评价并不是随便来的，而是通过规则一步步过滤出来的：

考试成绩评价规则：

1. 如果成绩大于等于60分，就是及格，在此前提下：
    - 如果成绩大于等于80分，属于优秀范围；
    - 否则，属于一般范围；
2. 如果成绩小于60分，就是不及格，在此前提下：
    - 如果成绩小于30分，平时太不认真，属于学渣了；
    - 如果成绩大于等于30分，那么，至少还能抢救一下；

像这种如果底下还有如果、条件里还套着条件的情况，我们要如何用Python把上面的规则写出来，并得出专属于彼得·帕克的成绩评价呢？

答案就是，我们需要用到嵌套

IF嵌套

if嵌套的应用场景，简单来讲就是：在基础条件满足的情况下，再在基础条件底下增加额外的条件判断。

就像上面的基础条件是60分及格，想要判断优秀和一般还要增加额外条件。

因此，彼得·帕克的成绩评价规则，用if嵌套可以这样写：

```
historyscore=26


if historyscore>=60:
    print('你已经及格')
    if historyscore>=80:
        print('你很优秀')
    else:
        print('你只是一般般')
else:
    print('不及格')
    if historyscore<30:
        print('学渣')
    else:
        print('还能抢救一下')

print('程序结束')
```

```
不及格
学渣
程序结束
```

然出现了三个扎心的结果……那么，在Python中，我们如何同时得出不及格、学渣、程序结束这三个结果的呢?

首先，我们先从整体总览一下，这段彼得·帕克成绩评价规则的代码总共分为四部分（分别是赋值、if、else、print()），两个大代码组。 根据缩进看，这四个部分都是平级关系。并且，在if和else的代码组里面，又包含了条件判断命令if...else...。这正是嵌套在实际代码中的使用方式。

这其中，if hsitoryscore>=60是第一条件；if historynumber>=80是在满足了60分后的第一个额外条件；而与if historynumber>=80平级的else条件，则是满足60分后的第二个额外条件。

那么，这种嵌套命令我们要如何理解呢？这就需要你回想一下，老师曾教给你的4个知识点：1.缩进；2.冒号:后的从属关系；3.命令按顺序执行；4.if和else条件的互斥。

首先，我们曾学过，缩进相同的平级命令都是老大，都是有头有脸的人物，谁都不能怠慢。所以，第一步，计算机就要按顺序一条一条地跟老大们“打招呼”（按顺序执行命令）。

那么，计算机执行的顺序就是：（1）接受带头大哥1号的赋值；（2）因为if大哥和else大哥是只能留一个的互斥关系，所以计算机要根据赋值，判断赋值的内容是满足2号还是3号的条件。

如果满足if大哥的条件，就执行if大哥的命令，如果不满足就执行else大哥的命令；第三步，执行带头大哥4号的固定命令。

经过判断，我们知道了，蜘蛛侠26分的历史成绩要找带头大哥3号。然而，问题又来了，else大哥后面不仅跟着三个print()，还有if，所以我到底要输出哪个？

你看一个体制健全生生不息的黑社会，哪有扁平式管理的？不都是老大下还有中层，中层下还有小头目，小头目下还有组长，组长下才是真正搬砖的。

所以，计算机语法的执行逻辑永远都不会变：else下的代码也要一行一行往下执行，根据缩进判断层级、在对应条件下运行对应的层级，并输出对应层级下的结果。

所以，我们的运行顺序是，先输出组长1号的结果不及格，然后再判断历史成绩26分的赋值符合组长2号：历史成绩<60分且<30分还是组长3号：历史成绩<60分且>=30分的条件。

下面，我们再通过一张导图理一下if嵌套的逻辑，老师建议你把这张图收藏起来，接下来写嵌套代码的时候可以翻看：

要注意的是，elif也是可以放进嵌套里的，就是在上述结构的基础上，多加一个带头大哥elif条件，以及底下跟着的elif条件的小弟就好。因为用法及结构是一样的，在这里就不重复讲解了。

说了这么多，对于if嵌套你掌握得怎么样？可以自己脱离课本，自己码出一段if嵌套了吗？

下面，我将教给你一个写出嵌套条件代码的小技巧。

如何写嵌套代码

if嵌套由于涉及多个条件判断，并且是条件套条件的判断，所以对于新手来说，可能会将逻辑搞混。为了理清逻辑，我们可采用扒洋葱大法写if嵌套。

如果我们想要自己写出上面“彼得·帕克的历史成绩”那样的嵌套代码，第一步就是：我们要搞定最外层那张皮，将最基础的命令和条件写出来。

那么，我们的思考逻辑应该是这样（1）把彼得·帕克26分的历史成绩赋值到变量historyscore；（2）用if…else写最基础判断条件如果historyscore>=60时，打印你已经及格，否则，打印不及格；（3）用print()输出程序结束。

来来来，跟老师动起来，请根据上面提示自己写一遍代码，并点击运行。看在成绩为26分的赋值情况下，只套了第一层条件判断皮的代码，会出现怎样的结果：

```
historyscore=26
#赋值语句：为历史成绩赋值

if historyscore>=60:
    #条件：当历史成绩大于等于60时：
    print('你已经及格')

else:
    #条件：当历史成绩小于60时
    print('不及格')
    #结果：输出不及格的结果

print('程序结束')
#打印最终结果
```

```
不及格
程序结束
```

运行通过了吗？正确的话，你应该会看到终端显示不及格、程序结束。

第二步，在“第一层皮”里装东西。也就是在条件【historyscore>=60】下，增加额外条件。

其中，额外条件1：当历史成绩大于80分，显示结果你很优秀；额外条件2：当历史成绩在60到80分之间，显示结果：你只是一般般。

下面，请你先看一眼正确的代码，并注意我是怎样安排额外条件的代码的。

```
historyscore=26
#赋值语句：为历史成绩赋值

if historyscore>=60:
    #条件：当历史成绩大于等于60时
    print('你已经及格')
    #结果：输入及格的结果
    if historyscore>=80:
        #条件：当历史成绩大于等于60，且历史成绩大于等于80
        print('你很优秀')
        #结果：输出你很优秀的结果
    else:
        #条件：当历史成绩大于等于60，且小于80时
```

```
        print('你只是一般般')
        #结果：输出你只是一般般的结果

else:
    #条件：当历史成绩小于60时
    print('不及格')
    #结果：输出不及格的结果

print('程序结束')
#打印最终结果
```

```
不及格
程序结束
```

最后一步，你要为另一个大条件else增加额外条件了！

请你以下方代码框中出现的代码为基础，在else底下增加上面两个额外条件。

其中，额外条件1：当历史成绩小于60，同时还小于30时，输出结果学渣；额外条件2：当历史成绩小于60，但大于等于30时，输出结果还能抢救一下。

总结

作业

1. if elif else
2. 嵌套

```
stonenumber=1

if stonenumber>=4:
    print('获得了打败灭霸的力量，反杀稳了')
elif 1<=stonenumber<=3:
    print('可以全员出动，殊死一搏')
else:
    print('没办法了，只能尝试呼叫惊奇队长')
```

```
可以全员出动，殊死一搏
```

其中判断工资水平的代码需要满足如下条件：

1.如果月工资小于等于500美元，显示"欢迎进入史塔克穷人帮前三名"

```
1.1如果月工资在100-500美元之间，显示“请找弗瑞队长加薪”
1.2如果月工资小于等于100美元，显示“恭喜您荣获“美元队长”称号！”
```

2.如果月工资在500-1000美元之间（含1000美元），打印"祝贺您至少可以温饱了。"

3.其他情况下，如果工资大于1000美元，打印"经济危机都难不倒您！"

```
3.1如果工资在1000-20000美元（含20000美元）之间，打印“您快比钢铁侠有钱了！”
3.2如果月工资大于20000美元，打印“您是不是来自于瓦坎达国？”
```

4.不管赋值改变后输出结果如何，都需固定打印"程序结束"

```
gongzi=60

if gongzi<=500:
    print('欢迎进入史塔克穷人帮前三名')
    if gongzi>=100:
        print('请找弗瑞队长加薪')
    else:
        print('恭喜您荣获“美元队长”称号！')
elif 500<gongzi<=1000:
    print('祝贺您至少可以温饱了。')
else:
    print('经济危机都难不倒您！')
    if gongzi<=20000:
        print('您快比钢铁侠有钱了！')
    else:
        print('您是不是来自于瓦坎达国？')

print('程序结束')
```

```
欢迎进入史塔克穷人帮前三名
恭喜您荣获“美元队长”称号！
程序结束
```

第3关 INPUT函数 综合

们不仅要掌握Python的码法，还要掌握Python的底层逻辑，利用正确的语言和合理的逻辑构造命令，最后再对计算机输入自己的命令。

这样我们才能以最真实的自己，得到计算机的回应；才能突破认知局限，从代码这面镜子中看到真我。

而这个传递信息的输入动作，就是我们今天的重点——input()函数，它将会完成你与Python的第一次互动沟通，产生不可思议的Python魔法。

INPUT()函数

```
import time
print('亲爱的同学：')
time.sleep(1)
print('我们愉快地通知您，您已获准在霍格沃茨魔法学校就读。')
time.sleep(2)
print('随信附上所需书籍及装备一览。')
time.sleep(1)
print('学期定于九月一日开始。')
time.sleep(1)
print('鉴于您对魔法世界的不熟悉，')
time.sleep(1)
print('我们将指派魔咒学老师—吴枫教授带您购买学习用品。')
```

```
亲爱的同学：
我们愉快地通知您，您已获准在霍格沃茨魔法学校就读。
随信附上所需书籍及装备一览。
学期定于九月一日开始。
鉴于您对魔法世界的不熟悉，
我们将指派魔咒学老师—吴枫教授带您购买学习用品。
```

print()函数是人给程序下达一个打印命令，程序想都不想，一比一地打印出结果，这种程序向屏幕输出信息的过程，就是人与计算机的单向沟通。

但是，如果想实现真正的人机互动沟通，就要有来有往。比如，校长问你要不要来霍格沃茨学魔法，那她就要给程序输入一个提问命令：你要来霍格沃茨吗？

程序得令后，立马把校长的提问一字不改地显示在计算机屏幕上，问：你要来霍格沃茨吗？

那么，这个人类用键盘向电脑提供指令，然后通过电脑把问题显示在显示器上，等人回答的过程，就是通过input()函数实现的。 那在代码的魔法世界，input()函数是如何实现自身价值的呢？这关，老师会从四个角度逐个击破input()函数。

input()函数的使用

先，让我们通过一段代码，来看一看input()函数是如何使用的：

```
input('请铲屎官输入宠物的名字：')
```

```
请铲屎官输入宠物的名字：张昊
```

```
'张昊'
```

input()函数是输入函数。就上面例子来讲，它需要你输入针对括号内'请铲屎官输入你宠物的名字：'的答案。

所以，当你在函数的括号内写出问题时，input()函数会将此问题原样显示在屏幕上，并在终端区域等待你针对此问题的回答。

可是，我们为什么要在终端处输入回答呢？不输入行不行？

事实上，我们可以把input()函数当作一扇链接现实世界与代码世界的门。

当问题从代码世界传递给我们，可我们却没有回答时，这扇等待输入的input()大门，就会一直处于敞开状态，一直等着你往里送回答。

而当我们输入信息后，这扇input()大门便会关上，继续执行下面的命令。

input()函数是输入函数，当你在函数的括号内写出问题时，input()函数会将此问题原样显示在屏幕上，并在终端区域等待你针对此问题的回答。

下面，我们来总结一下input()函数的使用方法： 我们反复强调代码是一行行运行的，所以即使input()函数通过了，门关上了，由于input()函数下没有其他命令，自然就不会有结果产生。

但是，我们输入的回答并没有丢失，它被被计算机储存在了程序中。

那么，如果我想要从程序海洋中找到刚刚输入的回答，利用它产生一个结果，比如表达我对主子的爱要怎么办？

```
请铲屎官输入宠物的名字：摩卡
#提出的问题

I Love 摩卡!
#显示的结果
```

按老规矩我们先梳理一下思维逻辑。首先，为了表达我对宠物的爱意，我得先知道宠物的名字，所以我得先搜集宠物名字的信息。

那么，我们思维的第一步，就是使用input()函数搜集信息：

```
input('请铲屎官输入宠物的名字：')
#运用input函数搜集信息
```

然而，当我想将搜集到的数据和【'I Love'】拼接起来，并打印出结果时，问题来了。虽然我用input()函数搜集到了一个信息，可我如何从程序海洋中调出这个信息，进行数据拼接呢?

这就涉及到了input()函数的第二个知识点——函数结果的赋值。

input()函数结果的赋值

首先，我们可以对变量进行赋值，这样当我们想提取数据时，只要直接打印变量名，就能唤醒程序对数据的记忆。

这个赋值逻辑，在input()函数中也是行得通的。我们可以通过赋值，达到随时提取输入结果的目的。但是在逻辑层面，我们需要拐一个弯。

比如，以我们下面这串代码为例：

```
name = input('内容')
#函数赋值
```

虽然看上去像是给input()函数赋值，但实际上，我们是将input()函数的执行结果（收集的信息）赋值给变量name。

通俗来讲，我们放进name这个盒子里的东西，不是input()函数中提的问题，而是我们通过input()函数收集到的别人的答案。

这样，不管你在终端输入的内容是什么，不管你的回答改变多少次。只要是对input()函数所提问题的回答，都会被存储在变量中。等待你打印变量时，将回答提取出来，展示在显示屏上。

而这些展示在终端的信息/回答/数据，在代码世界，我们可以称其为输入值——我们输入给函数的内容。

拿我们上个铲屎官代码举例：

```
input('请铲屎官输入宠物的名字：')
#运用input()函数搜集信息
```

当点击运行后，我们在终端处输入的信息——宠物的名字摩卡，就是输入值。

所以，为了能随时且方便地提取输入值（输入的结果），我们需要把input()函数的结果赋给变量。

下面，我们来测试一下：

```
name=input("请铲屎官输入宠物的名字：")
print(name)
```

```
请铲屎官输入宠物的名字：Ryan
Ryan
```

```
name = input('请铲屎官输入宠物的名字：')
print('I Love '+name +'!')
```

```
请铲屎官输入宠物的名字：RYan
I Love RYan!
```

这里有一个重点，在我们理解代码时，脑中的思维顺序是先搜集信息，再把input()函数的结果赋给变量。

但是为了书写规范及防止漏掉信息，我们需要按照代码书写格式，优先对input()函数的结果进行赋值。

首先，我们需要对input()函数的结果进行赋值，然后使用input()函数搜集信息，最后再用print()函数输出结果。

我们再用一张图，来直观地总结一下。

掌握了input()函数的综合使用场景，我们再来回看一下本关卡最开始，霍格沃茨来信的代码。

```
print('那么，您的选择是什么？`1`接受，还是`2`放弃呢？')

choice = input('请输入您的选择：')
#变量赋值

if choice == '1':
#条件判断:条件1
```

```
    print('霍格沃茨欢迎您的到来。')
    #条件1的结果

else:
#条件判断：其他条件
    print('您可是被梅林选中的孩子，我们不接受这个选项。')
    #其他条件的结果
```

```
那么，您的选择是什么？`1`接受，还是`2`放弃呢？
请输入您的选择：2
您可是被梅林选中的孩子，我们不接受这个选项。
```

从思维角度出发，简单来讲就是：我给你两个选择，你从中挑一个，然后我给你显示对应的结果。

所以，代码的第一步就是赋值，并通过input()函数提问。

```
choice = input('请输入您的选择：')
#变量赋值
```

第二步，我们要根据回答进行条件判断：如果回答是1，就显示条件1的结果；如果选了其他选项，则显示其他条件的结果。

```
choice = input('请输入您的选择：')
#变量赋值

if choice == '1':
#条件判断:条件1
    print('霍格沃茨欢迎您的到来。')
    #条件1的结果

else:
#条件判断：其他条件
    print('您可是被梅林选中的孩子，我们不接受这个选项。')
    #其他条件的结果
```

代码是写完了，但问题也来了：为什么if条件下的变量choice是字符串'1'呢？如果不是字符串格式，是整数1会出现什么结果呢？

下面，请你抄写上段代码，并将if的条件从字符串格式的'1'，改为整数格式的1。点击运行后，在终端处先后输入1和2并观察运行结果【提示：点击运行 → 输入1 → 点击重做 → 写代码 → 点击运行 → 输入2）】

```
choice = input('请输入您的选择：')
#变量赋值

if choice == 1:
#条件判断:条件1
    print('霍格沃茨欢迎您的到来。')
    #条件1的结果
else:
#条件判断：其他条件
    print('您可是被梅林选中的孩子，我们不接受这个选项。')
    #其他条件的结果
```

怎么样？是不是不管你输入的是1还是2，显示的结果都是else条件下的结果：'您可是被梅林选中的孩子，我们不接受这个选项。'？

是为什么呢？我们输入的就是条件选项中的数字1和2啊？怎么就无法出现对应的结果呢？

在这里，我们就涉及到了input()函数的第三个知识点：

input()函数的数据类型

现在，让我们先从计算机语言的逻辑，按行拆解一下这段代码组：


```
choice = input('请输入您的选择：')
#变量赋值

if choice == 1:
#条件判断:条件1
    print('霍格沃茨欢迎您的到来。')
    #条件1的结果
else:
#条件判断：其他条件
    print('您可是被梅林选中的孩子，我们不接受这个选项。')
    #其他条件的结果
```


那么，我们先来看看第一行代码中的赋值语句。

```
choice = input('请输入您的选择：')
#变量赋值
```

首先，我们从整体来看，这段代码表示的是——利用input()函数接收数据，并将数据赋值给变量choice。

其次，我们再细分着看，这行代码的每个部分代表着什么：

于input()函数来说，不管我们输入的回答是什么，不管你输入的是整数1234，还是字符串我爱摩卡，input()函数的输入值（搜集到的回答），永远会被强制性地转换为字符串类型。（Python3固定规则）

跟"你大爷还是你大爷，你大妈已经不是你大妈"一个道理：我爱摩卡即使被强制转换，它还是字符串类型的我爱摩卡；但是我们输入的整数1234就会被强制地转换为字符串的'1234'。

所以，不管我们在终端区域输入什么，input()函数的输入值一定是字符串。

就这样，由于我们用赋值运算符=把input()函数的字符串结果，赋给了变量choice，所以这个变量也一定是字符串类型。

现在，我们确定了变量choice的数据类型是字符串。下面，我们再来看后面的代码。

```
if choice == 1:
#条件判断:条件1
    print('霍格沃茨欢迎您的到来。')
    #条件1的结果

else:
#条件判断：其他条件
    print('您可是被梅林选中的孩子，我们不接受这个选项。')
    #其他条件的结果
```

我们先来看第一块代码组的if条件语句【if choice==1】并进行拆解：

根据第2关所学的条件判断知识，我们可以将第一行的if条件解释为：如果变量choice等于整数1时，则执行此if条件下的结果。

这回你是不是发现了什么问题？还记得我们在拆解input()函数时，提到的变量choice的数据类型吗？在第一次变量赋值时，由于input()函数的输入值是字符串，所以变量choice也是字符串；但是到了if条件判断时，if的条件又是【choice == 1】—— 变量choice等于整数1。 这就相当于王X聪准备接手家族事业，但是王X聪他爸王X林开了个条件：当王X聪是猫时，才能把产业给他。

但因为王X聪是个人类，无法满足动物猫的条件，所以自然无法执行猫条件下，继承家业的结果。

就这样，因为我们输入的信息永远是字符串，永远不能满足if的整数条件。所以，不管你输入的是什么，程序只有一个选择：执行else下的结果。

那么，不管你的选择是什么，永远只能打印出else下'您可是被梅林选中的孩子，我们不接受这个选项。'这个结果。

下面，请你展示自己的魔法。请下段代码中if条件的错误语句，改为正确的等于字符串类型，写在下方代码框中，并点击运行。

下面，我们来通过一张图再来加深一下，这个在未来非常有用，也非常重要的知识点： 那么，我们就需要一个更为省力的简便方法，将所有输入值的数据类型，一次性全部变为整数。

input()函数结果的强制转换

input()函数的输入值在某些情况下，是可以变为整数的。那么，你还记得字符串转变为整数要怎么做吗？ int()是第1关的知识点强制类型转换，可以将字符串强制性地转换为整数。

有了int()函数，我们就可以从input()函数的源头，将输入的内容转换为整数。

```
choice = int(input('请输入您的选择：'))
#将输入值强制转换为整数，并赋值给变量choice
```

但是，这串代码看起来像是把input()函数整个强制转换了。

可实际上，我们是将input()函数取得的结果，进行强制转换，并将强制转换的结果赋值存在名为choice的变量盒里。

这样，就算if条件是整数，程序也可以准确运行。

下面，请你接下我的接力棒。补齐下面的条件判断代码：如果选择1，打印"霍格沃茨欢迎您的到来"；其他时，打印"您可是被梅林选中的孩子，我们不接受这个选项"。【要点提示：if……else条件判断，变量赋值，等于整数】

```
choice = int(input('请输入您的选择：'))
if choice==1:
    print('霍格沃茨欢迎您的到来')
else:
    print('您可是被梅林选中的孩子，我们不接受这个选项')
```

最后，我们再来归纳一下input()函数数据类型强制转换的知识点：

总结

前四关总结

在第0关我们初识了Python世界最简单，也最常用的一个函数——print()函数。

你别看这个函数看起来简单，但是深挖下去，这个最简单的打印函数也内有乾坤，它包含着程序世界最基础的运算逻辑。

但是，如果想要达成和计算机互动沟通的目的，还需要掌握计算机的沟通语言。

只有当计算机明白我们说的是什么时，它才能给予我们正确的反馈。

与我们人类一样，计算机的运行也需要正规的逻辑，所以在第2关，我们掌握了如何与计算机沟通的逻辑。

拥有了语言和逻辑，我们就可以和计算机进行交互式沟通了。

然后，我们就来到了今天的关卡，学习了input()函数。

可是，学会了这些知识点，我们要如何才能自己写一段代码，比如霍格沃茨来信这样的代码呢？代码组的大致格式是什么呢？ 回头看看，经历了这四关，你是不是发现，在不知不觉中学习了这么多东西？更接触到了一个全新的世界。

作业

```
name=input("哈利·波特的猫头鹰叫做")
print(name)
```

```
哈利·波特的猫头鹰叫做火火
火火
```

```
number=int(input('罗恩吃的巧克力数量'))

if number >10:
    print("罗恩要给哈利100块")
else:
    print('哈利就给罗恩100块')
```

```
罗恩吃的巧克力数量3
哈利就给罗恩100块
```

```
Q1=input('您好，欢迎古灵阁，请问您需要帮助吗？')

if Q1=='需要':
    Q2=input("请问您需要什么帮助呢？1 存取款；2 货币兑换；3 咨询")
    if Q2=='1':
        print('推荐你去存取款窗')
    elif Q2=='2':
        print('金加隆和人民币的兑换率为1:51.3，即一金加隆=51.3人民币')
        N=float(input('请问您需要兑换多少金加隆呢？'))
        print('那么，您需要付给我'+str(N*51.3)+'人民币。')
    else:
        print('推荐你去咨询窗口')
else:
    print('好的，再见。')
```

```
您好，欢迎古灵阁，请问您需要帮助吗？需要
请问您需要什么帮助呢？1 存取款；2 货币兑换；3 咨询2
金加隆和人民币的兑换率为1:51.3，即一金加隆=51.3人民币
请问您需要兑换多少金加隆呢？21
那么，您需要付给我1077.3人民币。
```

第4关 列表，字典，元组

这一关我们就要学习两种新的数据类型：列表、字典。

不过在这之前，我想先和你聊一聊“计算机”与“数据”之间水乳交融的关系。

计算机名字里就有【计算】两字，如果计算机离开了数据，就如巧妇难为无米之炊。所以说，数据对于计算机很重要。

总的来说，计算机有3种方式利用数据：

- 第一种：直接使用数据，比如print()语句，可以直接把我们提供的数据打印出来，通常所见即所得。
- 第二种：计算和加工数据
- 第三种：用数据做判断是怎么一回事

```
# 直接运行即可
# 计算机会使用数据来做判断
a = int(input('请输入你的年龄：'))
#如果输入不了数字，请切换成英文输入法
if a<0:
    print('你还在娘胎里呢。')
elif a == 0:
    print('欢迎来到这个世界。')
elif a < 18:
    print('小小的年纪还不懂什么是爱')
else:
    print('你已经是个成熟的大人了，要学会照顾自己。')
```

```
请输入你的年龄：21
你已经是个成熟的大人了，要学会照顾自己。
```

可以看出，计算机在这里是【利用数据用做逻辑判断】

那现在我们对【计算机】和【数据】的关系有了一定的了解吧，也正因为数据的重要性，所以对于编程新手来说，掌握主要的数据类型是重中之重。

这一关，我们就会接触两种新的数据类型——列表和字典，你会发现，它们比我们学过的"整数、浮点数、字符串"更加高级，更有"包容性"。

为什么这么说呢？前面学的几种类型，每次赋值只能保存一条数据。如果我们需要使用很多数据的时候，就会很不方便。

而列表和字典的作用，就是可以帮我们存储大量数据，让计算机去读取和操作。

列表

首先我们来看看列表。为了感受列表的作用，我们来玩玩角色扮演的游戏：从现在起，你就是一个新班级的班主任了！

第一天，班上来了50个新鲜的面孔。你让学生把名字写在花名册上，方便上课时一个个点名。

如果只能用已学的知识来解决这个问题，我们需要将每个学生的名字都赋值到一个变量名，然后再分别打印。代码是这样的：

```
student1 = '党志文'
student2 = '浦欣然'
student3 = '罗鸿朗'
student4 = '姜信然'
student5 = '居俊德'
student6 = '宿鸿福'
```

```
student7 = '张成和'
student8 = '林景辉'
student9 = '戴英华'
student10 = '马鸿宝'
student11 = '郑翰音'
student12 = '厉和煦'
student13 = '钟英纵'
student14 = '卢信然'
student15 = '任正真'
student16 = '翟彭勃'
```

但我们知道，在编程世界里，最忌讳的就是"重复性劳动"。这一百行代码打下来，即使是复制黏贴修改的，分分钟也要抓狂。

实际上呢，只要学会了列表和循环（剧透：循环下一关会讲，可先忽略），3行代码就能搞定。

```
students = ['党志文', '浦欣然', '罗鸿朗', '姜信然', '居俊德', '宿鸿福', '张成和', '林景辉', '戴英华', '马鸿宝', '郑翰音', '厉和煦', '钟英纵', '卢信然', '任正真', '翟彭勃', '蒋华清', '双英朗', '金文柏', '饶永思', '堵宏盛', '濮嘉澍', '戈睿慈', '邰子默', '于斯年', '扈元驹', '厍良工', '甘锐泽', '姚兴怀', '殳英杰', '吴鸿福', '王永年', '宫锐泽', '黎兴发', '朱乐贤', '关乐童', '养永寿', '养承嗣', '贾康成', '韩修齐', '彭凯凯', '白天干', '瞿学义', '那同济', '衡星文', '公兴怀', '宫嘉熙', '牧乐邦', '温彭祖', '桂永怡']
for i in students:
    print(i+'在不在？')
```

```
党志文在不在？
浦欣然在不在？
罗鸿朗在不在？
姜信然在不在？
居俊德在不在？
宿鸿福在不在？
张成和在不在？
林景辉在不在？
戴英华在不在？
马鸿宝在不在？
郑翰音在不在？
厉和煦在不在？
钟英纵在不在？
卢信然在不在？
任正真在不在？
翟彭勃在不在？
蒋华清在不在？
双英朗在不在？
金文柏在不在？
饶永思在不在？
堵宏盛在不在？
濮嘉澍在不在？
戈睿慈在不在？
邰子默在不在？
```

```
于斯年在不在?
扈元驹在不在?
厍良工在不在?
甘锐泽在不在?
姚兴怀在不在?
殳英杰在不在?
吴鸿福在不在?
王永年在不在?
宫锐泽在不在?
黎兴发在不在?
朱乐贤在不在?
关乐童在不在?
养永寿在不在?
养承嗣在不在?
贾康成在不在?
韩修齐在不在?
彭凯凯在不在?
白天干在不在?
瞿学义在不在?
那同济在不在?
衡星文在不在?
公兴怀在不在?
宫嘉熙在不在?
牧乐邦在不在?
温彭祖在不在?
桂永怡在不在?
```

发现了吗？在第一行代码里，赋值号右边不再像字符串那样只能放一个名字，而是放了50个。

这就是我们要新认识的第一个数据类型——列表，下面我会从4个方面来介绍列表的用法。

什么是列表

首先，我们来看看列表（list）的代码格式： 图中的('小明','小红','小刚')就是一个列表。

一个列表需要用中括号[ ]把里面的各种数据框起来，里面的每一个数据叫作“元素”。每个元素之间都要用英文逗号隔开。

这就是列表的标准格式，现在请你创建一个列表名为list1的列表，列表里有三个元素：'小明'、18、1.70，并将其打印出来：

```
list1=['小明',18,1.70]
print(list1)
```

```
['小明', 18, 1.7]
```

恭喜你，成功掌握了列表的规范写法以及打印列表的方法。而且，你也用代码验证了一个知识点：列表很包容，各种类型的数据（整数/浮点数/字符串）无所不能包。

不过，很多时候，我们只需要用到列表中的某一个元素，好比老师上课点名时，不会说“所有的同学都站起来回答一下这个问题”。

所以，问题来了：列表中具体的某个元素，要如何取出来?

从列表提取单个元素

这就涉及到一个新的知识点：偏移量。列表中的各个元素，好比教室里的某排学生那样，是有序地排列的，也就是说，每个元素都有自己的位置编号（即偏移量）。 从上图可得：

1. 偏移量是从0开始的，而非我们习惯的从1开始；
2. 列表名后加带偏移量的中括号，就能取到相应位置的元素。

所以，我们可以通过偏移量来对列表进行索引（可理解为搜索定位），读取我们所需的元素。

假如你现在要喊小明来回答问题，用代码怎么写呢？请补充下列代码，利用列表的偏移量来打印出'小明'这个元素。

```
students = ['小明','小红','小刚']
print(students[0])
```

```
小明
```

现在我们已经知道了如何从列表中取出一个元素，那如果要同时取好几个呢？所以我们接着学习如何从列表中取出多个元素。

从列表提取多个元素

这次，老师不会直接告诉你需要用到的知识，而是想让你自己总结出规律。

请运行以下代码，对比代码和终端最后的结果，尝试发现当中的规律。老师过会儿会考考你。

一个提醒：下列:左右两边的数字指的是列表中元素的偏移量，记住偏移量（索引）始终是从0开始的。

```
list2 = [5,6,7,8,9]
print(list2[:])
# 打印出[5,6,7,8,9]
print(list2[2:])
# 打印出[7,8.9]
print(list2[:2])
# 打印出[5,6]
print(list2[1:3])
#打印出[6,7]
print(list2[2:4])
#打印出[7,8]
```

```
[5, 6, 7, 8, 9]
[7, 8, 9]
[5, 6]
[6, 7]
[7, 8]
```

上面这种用冒号来截取列表元素的操作叫作切片，顾名思义，就是将列表的某个片段拿出来处理。这种切片的方式可以让我们从列表中取出多个元素。

- 前半句：
    - 冒号左边空，就要从偏移量为0的元素开始取；
    - 右边空，就要取到列表的最后一个元素。
- 后半句：
    - 冒号左边数字对应的元素要拿，
    - 右边的不动（可再回顾下代码）。

现在就请你来试验一下吧，请修改代码，用切片将列表中小明和小红两个元素一起取出来。

另外，我们要注意一个细节：偏移量取到的是列表中的元素，而切片则是截取了列表的某部分，所以还是列表，清你运行下列代码看一看。

```
students = ['小明','小红','小刚']
print(students[2])
print(students[2:])
```

```
小刚
['小刚']
```

给列表增加/删除元素

过了一周，你正上着课呢，教导主任突然领了一个新学生“小美”，说是转校生，要插到你们班。这时，我们就需要用到append()函数给列表增加元素，append的意思是附加，增补。

我们还是通过代码来试验一番，请你运行代码，并思考规律：（注：报错后，可读一下报错信息，然后在第6行开头加个#号，将其注释掉再运行）

```
list3 = [1,2]
list3.append(3)
print(list3)

list3.append([4,5])
print(list3)
```

```
[1, 2, 3]
[1, 2, 3, [4, 5]]
```

```
list3 = [1,2]
list3.append(3)
print(list3)

list3.append(4,5)
#list3.append([4,5])
print(list3)
```

```
[1, 2, 3]
```

```
---------------------------------------------------------------------------
----

TypeError                                 Traceback (most recent call
last)

<ipython-input-11-9f76c1f0eea8> in <module>
      3 print(list3)
      4
----> 5 list3.append(4,5)
      6 #list3.append([4,5])
      7 print(list3)
```

```
TypeError: append() takes exactly one argument (2 given)
```

这句话的意思是：append后的括号里只能接受一个参数，但却给了两个，也就是4和5。所以，用append()给列表增加元素，每次只能增加一个元素。

append函数并不生成一个新列表，而是让列表末尾新增一个元素。而且，列表长度可变，理论容量无限，所以支持任意的嵌套。

```
students = ['小明','小红','小刚']
students.append('小美')
print(students)
```

```
['小明', '小红', '小刚', '小美']
```

棒哟～现在你已经知道如何增添列表中的元素了。

又是新的一天，你接到一个电话，小红生病请假了，今天不来上课。所以，你要将小红从列表中删除。

提示：需要用到del语句。请你先读一下Python官方文档对"del语句"的解释：（懂得阅读官方文档也是编程学习中一个重要能力）

```
students = ['小明','小红','小刚','小美']
del students[1]
print(students)
```

```
['小明', '小刚', '小美']
```

事实上del语句非常方便，既能删除一个元素，也能一次删除多个元素（原理和切片类似，左取右不取）。

数据类型：字典

众所周知，一个老师的日常就是出卷、改卷。这次期中考呢，小明、小红、小刚分别考了95、90和90分。

假如我们还用列表来装数据的话，我们需要新创建一个列表来专门放分数，而且要保证和姓名的顺序是一致的，很麻烦。

所以类似这种名字和数值（如分数、身高、体重等）两种数据存在一一对应的情况，用第二种数据类型——"字典"（dictionary）来存储会更方便。

什么是字典

同样的，我们先来看一下字典是长怎么样的： 仔细看下，字典和列表有3个地方是一样的：

1. 有名称；
2. 要用=赋值；
3. 用逗号作为元素间的分隔符。

而不一样的有两处：

1. 列表外层用的是中括号[]，字典的外层是大括号{}；
2. 列表中的元素是自成一体的，而字典的元素是由一个个键值对构成的，用英文冒号连接。如'小

明':95，其中我们把'小明'叫键（key），95叫值(value)。

这样唯一的键和对应的值形成的组合，我们就叫做【键值对】，上述字典就有3个【键值对】：'小明':95、'小红':90、'小刚':90

如果不想口算，我们可以用len()函数来得出一个列表或者字典的长度（元素个数），括号里放列表或字典名称。

```
students = ['小明','小红','小刚']
scores = {'小明':95,'小红':90,'小刚':90}
print(len(students))
print(len(scores))
```

```
3
3
```

这里需要强调的是，字典中的键具备唯一性，而值可重复。也就是说字典里不能同时包含两个'小明'的键，但却可以有两个同为90的值。

现在，我们尝试将小明的成绩从字典里打印出来。这就涉及到字典的索引，和列表通过偏移量来索引不同，字典靠的是键。

```
scores = {'小明': 95, '小红': 90, '小刚': 90}
print(scores['小明'])
```

```
95
```

这便是从字典中提取对应的值的用法。和列表相似的是要用[]，不过因为字典没有偏移量，所以在中括号中应该写键的名称，即字典名(字典的键)。

现在你也知道如何取出字典里的值了。

小刚拿到试卷后，下课后来找你，说把他总分算错了，应该是92分。你看了一下，发现还真的是。于是，你在成绩册上将90划掉，改成了92。

这个操作在代码里对应的是字典的删除和增加。

给字典增加/删除元素

```
album = {'周杰伦':'七里香','王力宏':'心中的日月'}
del album['周杰伦']
print(album)

album['周杰伦'] = '十一月的萧邦'
print(album)
print(album['周杰伦'])
```

```
{'王力宏': '心中的日月'}
{'王力宏': '心中的日月', '周杰伦': '十一月的萧邦'}
十一月的萧邦
```

我们可以发现：删除字典里键值对的代码是del语句del 字典名(键)，而新增键值对要用到赋值语句字典名(键) = 值。

那么，请你把小刚的成绩改成92分吧。对了，新来的小美也考了，得了85。请你对字典里进行修改和新增，然后将整个字典都打印出来。

```
scores = {'小明': 95, '小红': 90, '小刚': 90}
del scores['小刚']
scores['小刚']=92
scores['小美']=85
print(scores)
```

```
{'小明': 95, '小红': 90, '小刚': 92, '小美': 85}
```

好了。至此，我们可以总结一下字典的基础知识：

列表和字典的异同

列表和字典同作为Python里能存储多条数据的数据类型，有许多共同点，也有值得我们注意的不同点，那么接下来我们先来看看不同点。

列表和字典的不同点

一个很重要的不同点是列表中的元素是有自己明确的"位置"的，所以即使看似相同的元素，只要在列表所处的位置不同，它们就是两个不同的列表。我们来看看代码：

```
# 如果==左右两边相等，值为True，不相等则为False。
print(1 == 1)
```

```
# 1等于1，所以值为True
print(1 == 2)
# 1不等于2，所以为False

students1 = ['小明','小红','小刚']

students2 = ['小刚','小明','小红']
print(students1 == students2)

scores1 = {'小明':95,'小红':90,'小刚':100}
scores2 = {'小刚':100,'小明':95,'小红':90}
print(scores1 == scores2)
```

```
True
False
False
True
```

而字典相比起来就显得随和很多，调动顺序也不影响。因为列表中的数据是有序排列的，而字典中的数据是随机排列的。

这也是为什么两者数据读取方法会不同的原因：

- 列表有序，要用偏移量定位；
- 字典无序，便通过唯一的键来取值。

我们先来看第一个共同点：在列表和字典中，如果要修改元素，都可用赋值语句来完成。

```
list1 = ['小明','小红','小刚','小美']
list1[1] = '小蓝'
print(list1)

dict1 = {'小明':'男'}
dict1['小明'] = '女'
print(dict1)
```

```
['小明', '小蓝', '小刚', '小美']
{'小明': '女'}
```

所以，上面修改小刚成绩的时候，其实直接用赋值语句即可，del语句通常是用来删除确定不需要的键值对。

```
scores = {'小明':95,'小红':90,'小刚':90}
#del scores['小刚']
#如果只需要修改键里面的值，可不需要del语句
scores['小刚'] = 92
```

第二个共同点其实之前已经略有提及，即支持任意嵌套。除之前学过的数据类型外，列表可嵌套其他列表和字典，字典也可嵌套其他字典和列表。

先来看看第一种情况：列表嵌套列表。你在班级里成立了以四人为单位的学习小组。这时，列表的形式可以写成：

```
students = [['小明','小红','小刚','小美'],['小强','小兰','小伟','小芳']]
```

students这个列表是由两个子列表组成的，现在有个问题是：我们要怎么把小芳取出来呢?

可能你数着小芳是列表的第7个元素(从0开始)，所以想students(7)不就能取到小芳吗?

事情当然没有那么简单，当我们在提取这种多级嵌套的列表/字典时，要一层一层地取出来，就像剥洋葱一样： 现在，我们确定了小芳是在students(1)的列表里，继续往下看。

小芳是students(1)列表里的第三个元素，所以要取出小芳，代码可以这么写：

```
students = [['小明','小红','小刚','小美'],['小强','小兰','小伟','小芳']]
print(students[1][3])
print(students[1][1])
```

```
小芳
小兰
```

下来，我们再来看看第二种情况：字典嵌套字典。

和列表嵌套列表也是类似的，需要一层一层取出来，比如说要取出小芳的成绩，代码是这样写：

```
scores = {
    '第一组':{'小明':95,'小红':90,'小刚':100,'小美':85},
    '第二组':{'小强':99,'小兰':89,'小伟':93,'小芳':88}
    }
print(scores['第二组']['小芳'])
print(scores['第一组']['小刚'])
```

```
88
100
```

我们再来提高下难度，看看列表和字典相互嵌套的情况，可以将代码和注释结合起来看。

```
# 最外层是大括号，所以是字典嵌套列表，先找到字典的键对应的列表，再判断列表中要取出元素的偏移量
students = {
    '第一组':['小明','小红','小刚','小美'],
    '第二组':['小强','小兰','小伟','小芳']
    }
print(students['第一组'][3])
#取出'第一组'对应列表偏移量为3的元素，即'小美'

# 最外层是中括号，所以是列表嵌套字典，先判断字典是列表的第几个元素，再找出要取出的值相对应的键
scores = [
    {'小明':95,'小红':90,'小刚':100,'小美':85},
    {'小强':99,'小兰':89,'小伟':93,'小芳':88}
    ]
print(scores[1]['小强'])
#先定位到列表偏移量为1的元素，即第二个字典，再取出字典里键为'小强'对应的值，即99。
```

```
小美
99
```

```
students = {
    '第一组':['小明','小红','小刚','小美'],

    '第二组':['小强','小兰','小伟','小芳']
    }
scores = [
    {'小明':95,'小红':90,'小刚':100,'小美':85},
    {'小强':99,'小兰':89,'小伟':93,'小芳':88}
    ]
print(students['第一组'][2])
print(scores[0]['小刚'])
```

```
小刚
100
```

作业

```
list1 = [{'嫉妒':'envy'},{'恨':'hatred'},{'爱':'love'}]
print(list1[2]['爱'])
# 第一步：取出列表中的第三个元素（list1[2]），字典{'爱':'love'};
# 第二步：取出list1[2]中键'爱'所对应的值，即'love'（list1[2]['爱']）。
```

```
dict1 = {1:['cake','scone','puff'],2:['London','Bristol','Bath'],3:
['love','hatred','envy']}

print(dict1[3][0])

# 第一步：取出字典中键为3对应的值（dict1[3]），即['love','hatred','envy']。
# 第二步：再取出列表['love','hatred','envy']中的第一个元素（dict1[3][0]）。

tuple1 = ('A','B')

list2 = [('A','B'),('C','D'),('E','F')]
print(tuple1[0])
print(list2[1][1])
# 从代码里，也可看出：1.元组内数据的提取也是用偏移量；2.元组也支持互相嵌套。
```

```
love
love
A
D
```

元组（TUPLE）

下面，介绍一种新的数据类型：元组（tuple）。 可以看到：元组和表格很相似，不过，它是用小括号来包的。 元组和列表都是序列,提取的方式也是偏移量，如 tuple1(1)、tuple1(1:)。另外，元组也支持任意的嵌套。 请你根据以上提供的信息，将tuple1中的A和list2中的D打印出来。看到了，理解了，运用了，就能够掌握了。

```
townee = [
    {'海底王国':['小美人鱼''海之王''小美人鱼的祖母''五位姐姐'],'上层世界':['王
子','邻国公主']},
    '丑小鸭','坚定的锡兵','睡美人','青蛙王子',
    [{'主角':'小红帽','配角1':'外婆','配角2':'猎人'},{'反面角色':'狼'}]
    ]
print(townee[5][1]['反面角色'])
```

```
狼
```

第5关 循环FOR..IN ,WHILE

要实现"重复、自动地执行代码"，有两种循环语句可供我们选择使用：一种是for...in...循环语句，另一种是while循环语句。

"循环"在计算机中是非常重要，是最基础的编程知识，为了讲解得更加清楚，我们将循环这一章节分成了上下两关。这一关，我们将对两种循环语句形成初步的了解，并学会简单的应用。

话不多说，我们先一起来看看第一种循环的方式：for...in...循环，它也被简称为for循环。

FOR...IN...循环语句

还记得前一关里，班主任点名的例子吗？本来要重复50次才能完成的任务，最后变成了3行代码。

```
student = ['党志文', '浦欣然', '罗鸿朗', '姜信然', '居俊德', '宿鸿福', '张成和', '林景辉', '戴英华', '马鸿宝', '郑翰音', '厉和煦', '钟英纵', '卢信然', '任正真', '翟彭勃', '蒋华清', '双英朗', '金文柏', '饶永思', '堵宏盛', '濮嘉澍', '戈睿慈', '邰子默', '于斯年', '扈元驹', '厍良工', '甘锐泽', '姚兴怀', '殳英杰', '吴鸿福', '王永年', '宫锐泽', '黎兴发', '朱乐贤', '关乐童', '养永寿', '养承嗣', '贾康成', '韩修齐', '彭凯凯', '白天干', '瞿学义', '那同济', '衡星文', '公兴怀', '宫嘉熙', '牧乐邦', '温彭祖', '桂永怡']
for i in student:
    print(i+'在不在？')
```

```
党志文在不在？
浦欣然在不在？
罗鸿朗在不在？
姜信然在不在？
居俊德在不在？
宿鸿福在不在？
张成和在不在？
林景辉在不在？
戴英华在不在？
马鸿宝在不在？
郑翰音在不在？
厉和煦在不在？
钟英纵在不在？
卢信然在不在？
任正真在不在？
翟彭勃在不在？
蒋华清在不在？
```

```
双英朗在不在?
金文柏在不在?
饶永思在不在?
堵宏盛在不在?
濮嘉澍在不在?
戈睿慈在不在?
邰子默在不在?
于斯年在不在?
扈元驹在不在?
厍良工在不在?
甘锐泽在不在?
姚兴怀在不在?
殳英杰在不在?
吴鸿福在不在?
王永年在不在?
宫锐泽在不在?
黎兴发在不在?
朱乐贤在不在?
关乐童在不在?
养永寿在不在?
养承嗣在不在?
贾康成在不在?
韩修齐在不在?
彭凯凯在不在?
白天干在不在?
瞿学义在不在?
那同济在不在?
衡星文在不在?
公兴怀在不在?
宫嘉熙在不在?
牧乐邦在不在?
温彭祖在不在?
桂永怡在不在?
```

这里的第2-3行就是for循环。

先看一段最简单的for循环代码，了解它的格式：

```
for i in [1,2,3,4,5]:
        print(i)
```

```
1
2
3
4
5
```

终端上依次出现了列表里的所有数字，对吧？我们用大白话来打个比方，以便更好地理解这段代码的意义：

```
for i in [1,2,3,4,5]:
   print(i)

#有一群数字在排队办业务，也就是列表[1,2,3,4,5]
#它们中的每一个被叫到号的时候(for i in)，就轮流进去一个空房间办业务
#每一个数字进去房间之后，都对计算机说：“喂，我要办这个业务：帮忙把我自己打印出来”，也就是print(i)
#然后计算机忠实的为每一个数字提供了打印服务，将1,2,3,4,5都打印在了屏幕上
```

for循环的3个要点即是:

1. 空房间;
2. 一群等着办业务的人;
3. 业务流程

我们一个一个来看:

for循环：空房间

房间的学名叫【元素】（item），你可以把它当成是一个变量。那么首先，我们需要给房间取一个名字，也就是“变量名”。

为什么我总是用i？因为英文是item，所以i是常用名嘛。但其实你给这个房间取什么名字都行。

来，直接运行一下代码试试。

```
for i in [1,2]:
    print(i)

for number in [1,2]:
    print(number)

for LOVE in [1,2]:
       print(LOVE)
```

```
1
2
1
2
1
2
```

是不是三种结果都一样？这就对啦，不必拘束姓名。

在for循环结束之后，我们还能使用这个房间，不过这时候房间里的人是谁呢？请运行代码体验一下:

```
for i in [1,2,3,4,5]:
    print(i)

print('事情全部办完了！现在留在空房间里的人是谁？')
print(i)
```

```
1
2
3
4
5
事情全部办完了！现在留在空房间里的人是谁？
5
```

原来，业务结束之后，最后一个走进去的5留在了房间里，被打印了出来。

搞清楚了什么是"空房间"，我们再看看下一个要点：

for循环：一群排队办业务的人

我们刚刚看到的"一群排队办业务的人"，都是以列表的形式出现：(1,2,3,4,5)。还有哪些数据类型也属于"一群排队办业务的人"呢？

我觉得你已经猜到了，就是字典。来直接体验一下代码运行效果：

```
dict = {'日本':'东京','英国':'伦敦','法国':'巴黎'}

for i in dict:
    print(i)
```

```
日本
英国
法国
```

我们用print(i)把"空房间"i打印了出来，发现i会逐个接待字典中的每一个【键】。

字典、列表和字符串'吴承恩'都是一群排队办业务的人，但a = 5并不是。

为什么'吴承恩'也可以？

不要怀疑，字符串也属于"一群排队办业务的人"。打个比方，'吴承恩'三个字就像一家三口，但走进空房间办业务时，这家人是可以一个一个进去的。来试试吧。

```
for i in '吴承恩':
    print(i)
```

```
吴
承
恩
```

而整数、浮点数是不属于“一群排队办业务的人”的，如果把它们放在for循环里，代码会报错。

现在我们理解了：列表，字典，字符串都可以是“一群排队办业务的人”。

```
for i in[1,2,3,4,5]:
    print(i)
```

```
1
2
3
4
5
```

还是这段代码，代码的运行结果你应该已经了然于心，是1,2,3,4,5依次出现。也就是说，当这一群排队的人依次序走进空房间，每个人都会把业务办完。

这个过程，在Python中的学名就叫做【遍历】。

Python是遍历数据结构（列表、字典等），一一访问其中的数据。

除了列表，字典，字符串三种数据类型，我们还可以遍历其他的数据集合。比如和for循环常常一起搭配使用的：range() 函数。

range()函数

```
for i in range(3):
    print(i)
```

```
0
1
2
```

运行后，你看到了整数0，1，2，是不是？使用range(x)函数，就可以生成一个从0到x-1的整数序列。

它还有更多用法，再来看这段代码，并运行：

```
for i in range(13,17):
    print(i)
```

```
13
14
15
16
```

使用range(a,b) 函数，你可以生成了一个【取头不取尾】的整数序列。

你可能会想问，我没事儿取这些整数出来干嘛？嗯，这是个好问题。再来看一段代码，并运行：

```
for i in range(3):
    print('我很棒')
```

```
我很棒
我很棒
我很棒
```

重要的事情说三遍，哈哈。像这样，有了range()函数之后，当你想把一段代码固定重复n次时，就可以直接使用for i in range(n)解决问题。

来练习一下：如果你要重复打印"书桓走的第n天，想他"，n为0到10，你会怎么写？

```
for i in range(12):
    print('书恒走的第'+str(i)+'天')
```

```
书恒走的第0天
书恒走的第1天
书恒走的第2天
书恒走的第3天
书恒走的第4天
书恒走的第5天
书恒走的第6天
书恒走的第7天
书恒走的第8天
书恒走的第9天
书恒走的第10天
书恒走的第11天
```

range()函数还有一种用法，我们来直接运行体验一下：

```
for i in range(0,10,3):
    print(i)
```

```
0
3
6
9
```

你观察出规律了么？这里range(0,10,3)的意思是：从0数到9（取头不取尾），数数的间隔为3。 好啦，我们来实战一下：请你用for循环完成1到10的整数分别乘以5的计算，并打印出来，效果就像这样：

```
for i in range(1,11):
    #要用11表示1-10
    print(i*5)
```

```
5
10
15
20
25
30
35
40
45
50
```

我们最后来理解一下for循环的第三个要点：

for循环：办事流程

我们以下面的代码为例：

```
for i in [1,2,3,4,5]:
    print(i*5)
```

这两行代码中，in表示从“一群排队办业务的人”（字符串、列表、字典等）中依次取值，这个刚才我们已经学过了。

在循环的过程中，“一群排队办业务的人”会被依次取出，然后走进房间i去办理业务。

但办事流程呢？在这里流程很简单，都是print(i*5)。然后在i=1的情况下，执行一遍流程；在i=2的情况下，再执行一遍流程……一直执行到i=5，也就把这群人的事情全部办完了：

“办事流程”的学名是【for子句】。格式是【冒号】后另起一行，【缩进】写命令。

格式对于编程来说是一件再怎么强调也不为过的“小事”，所以在这里我们再强调一下for循环的格式：

到这里，你已经掌握for循环的基本语法了，我们再做一些练习。

假设你要做大采购，让小明买醋，小红买油，小白买盐，小张买米；我们先把这堆事情写成一个字典：
d = {'小明':'醋','小红':'油','小白':'盐'}

如果我们把这四个人要买的东西依次print出来，要写4句print语句：

```
d={'小明':'醋','小红':'油','小白':'盐','小张':'米'}
print(d['小明'])
print(d['小红'])
print(d['小白'])
print(d['小张'])
```

```
醋
油
盐
米
```

```
d = {'小明':'醋','小红':'油','小白':'盐','小张':'米'}

for i in d:
    print(d[i])
```

```
醋
油
盐
米
```

到这里，for循环的3个要点都讲解完毕： 好了，我们学完了for循环，我们来看看另一种循环方式。

WHILE循环

先来看看while循环长啥样：

```
a = 0

while a < 5:
    a = a + 1
    print(a)
```

```
1
2
3
4
5
```

还是1,2,3,4,5依次出现，对吧？我们也用大白话去解释一下这段代码。

和for循环语句不同，while语句没有“空房间”，也不是“把一群排队办业务的人做完”。它是“在一定的条件下”，“按照流程办事”。

```
a = 0                   #先定义变量a，并赋值
while a < 5:            #设定一个放行条件：a要小于5，才能办事
    a = a + 1  # 满足条件时，就办事：将a+1
    print(a)   # 继续办事：将a+1的结果打印出来
```

```
1
2
3
4
5
```

很明显，while循环有2个要点：1.放行条件；2.办事流程。咱们先看第一个。

while循环：放行条件

while在英文中表示“当”，while后面跟的是一个条件。当条件被满足时，就会循环执行while内部的代码（while子句）。

所以while循环本质上像是一个哨卡：只要事情符合条件，那就一遍又一遍的“按流程办事”。

就像在上面的例题中，只要a<5这个条件成立，就不停地办事（把a+1的结果打印出来），直到条件不成立，办事流程就停止。

同样，while语句也要注意代码规范： 和for循环一样，冒号和内部代码的缩进都是必不可少的。┌(ᴗ ^ ᴗ)┐ 写不规范的话计算机又给你报错，然后你又卡在莫名其妙的地方很久噢.....

下面，我们用一个好玩儿的案例来说明while的具体用法，不晓得同学们有没有看过金庸的武侠小说《神雕侠侣》？

小说里的主角小龙女（古墓派掌门人）自小在终南山上的古墓里生活。这个门派有个规定，除非有男人愿意为掌门人死，否则掌门人永远不能出墓门半步。

在Python中，这个故事的逻辑就被翻译成：当（while）没有男人愿意为小龙女去死的时候，小龙女要一直一直一直生活在古墓里，这就是一种循环。只有当条件（没有男人愿意为小龙女去死）为假的时候，就可以打破循环，小龙女就能出古墓下山了。

这段代码我先写出来，你看看~

```
man = ''  # 注：''代表空字符串
while man != '有':  #注：!=代表不等于
    man = input('有没有愿意为小龙女死的男人？没有的话就不能出古墓。')
print('小龙女可以出古墓门下山啦~')
```

- 第1行代码：定义了变量man为空的字符串。使用变量前要先定义变量并为变量赋值，下面我们还会继续用到变量man。
- 第2行代码：while后面有一个条件，当这个条件被满足时，即man != '有'时，放行、办事。开始执行循环内部代码，即第3行的代码，开始询问。
- 第3行代码：询问现在有男人愿意为小龙女死吗？输入完信息后，回到第2行代码，重新判断条件真假。直到条件被判断为假，即man == '有'，while循环才结束。
- 第4行代码：while 循环结束后的代码，也是循环外部的代码。因为当有男人愿意为小龙女死的

时候，while 后面的条件就为假，此时，程序会结束循环，去运行第五行代码。

我们来运行一下好了，同样需要你输入内容。你可以先随便输入几个数据，只要你输入的不是有，代码就会一直循环；直到你输入有，循环会结束。

好了，你感受过了while循环的逻辑之后，现在我想请你自己动手练一练。放心，没那么难，有疑问的话看看小龙女的例题。

这个画风突变的故事是这样的：你家的大门是密码门，密码是你的生日816。当输错密码错误的时候，会提示"请尝试输入密码："。直到密码输入正确，就会提示"欢迎回家！"。

```
password=''
while password!='816':
    password=input('请尝试输入密码：')

print('欢迎回家')
```

```
请尝试输入密码：816
欢迎回家
```

while循环：办事流程

while循环，在满足条件的时候，会一轮又一轮地循环执行代码。

我们来做道选择题，请看代码回答问题：

```
a = 0

while a < 5:
    a = a + 1
print(a)
```

a从0开始，每次循环都加1，当a被加到5的时候，就不会满足循环的条件哨卡就不会通过，于是循环就会结束。最会打印a。

这道题跟一开始的例题非常相似，唯一的区别在于print(a)有没有缩进。

```
# 之前的例题
a = 0

while a < 5:
    a = a + 1
    print(a)
```

```
# 本题
a = 0

while a < 5:
    a = a + 1
print(a)
```

有缩进的时候，print(a)也是循环中的"办事流程"，会将数字逐一打印。没有缩进的时候，循环中的"办事流程"就只有做加法，print(a)也就只会打印循环结束时的最后一个数字。

稍微有点费事是不？我们来做一下分解，看看每一行代码的含义：

```
a = 0  # 定义了一个变量a
while a < 5:  # 当a小于5的时候，就自动执行后续缩进部分的语句
    print('现在a的值是：' + str(a)) #加一个print看看现在的a是多少
    a = a + 1  # 每执行一次循环，变量a的值都加1
    print('加1后a的值是：' + str(a)) #加一个print看看加1后的a是多少
print(a)
```

可以看到，最后一轮循环的时候a=4，然后最后a的值被加1后等于5：

所以，缩进后的【while子句】才是会被循环执行的"办事流程"，这一点，你应该差不多明白了吧~

那，我们再来做一个小练习。

之前，我们用for循环解过"1到10分别乘以5"的题目：

```
a=0
while a<11:
    print(a*5)
    a=a+1
```

```
0
5
10
15
20
25
30
35
40
45
50
```

到这里，for循环和whlie循环的知识就都学完了~

for循环和while循环都可以帮我们完成重复性的劳动，那到底两个循环有什么区别，什么时候用for什么时候用while呢？让我们来对比一下。

两种循环对比

for循环和whlie循环最大的区别在于【循环的工作量是否确定】，for循环就像空房间依次办理业务，直到把【所有工作做完】才下班。但while循环就像哨卡放行，【满足条件就一直工作】，直到不满足条件就关闭哨卡。

所以说，当我们【工作量确定】的时候，我们就可以让for循环来完成重复性工作。反之，要【工作量不确定时】可以让while循环来工作：

```
# 适合用for...in...循环
for i in '神雕侠侣':
•     print(i)

# 适合用while循环
password = ''
while password != '816':
•     password = input('请尝试输入密码：')
```

要把字符串'神雕侠侣'拆成一个个字符打印出来，这件事【工作量确定】，适合用for循环。

而对于"输入密码，判断输入的密码是否正确"这件事，我们并不知道要判断几遍才能得到正确密码，所以【工作量不确定】，适合用while循环。

不过有一种情况for循环和while循环都可以解决问题，那就是【把一件事情做N遍】：

```
#用for循环把诗句打印3遍
for i in range(1,4) :
    print('明日复明日，明日何其多。')


j = 1
while j<4 :
    print ('明日何其多，明日何其多。')
    j =j+1
```

```
明日复明日，明日何其多。
明日复明日，明日何其多。
明日复明日，明日何其多。
明日何其多，明日何其多。
明日何其多，明日何其多。
明日何其多，明日何其多。
```

你会看到，两者都能做。不过for循环的代码相对更简洁一些。

让我们总结一下什么时候用for什么时候用while：

作业

```
for i in range(1,8):
    if i != 4:
        print(i)
```

```
1
2
3
5
6
7
```

```
n=0
while n < 8:
    if n!=4:
        print(n)
    n=n+1
```

```
0
1
2
3
5
6
7
```

```
students = ['小明','小红','小刚']

for i in range(3):
    student1 = students.pop(0)  # 运用pop()函数，同时完成提取和删除。
    students.append(student1)  # 将移除的student1安排到最后一个座位。
    print(students)
```

```
['小红', '小刚', '小明']
['小刚', '小明', '小红']
['小明', '小红', '小刚']
```

POP()函数

我们先介绍一下列表中的pop()函数，用于移除列表中的一个元素（默认最后一个元素），并且返回该元素的值。

可以将其理解为提取和删除的融合：

1. 提取：取到元素，对列表没有影响；
2. 删除：删除列表的元素。 而移除，则是同时做到取到元素，并且删除列表中的元素。

第6关 布尔值，BREAK，CONTINUE，PASS，ELSE语句

我们还是接着上一关的while循环讲起，还记得while循环怎么写吧？

```
a = 0
while a < 5:
    a = a + 1
    print(a)
```

之前说到，while循环是需要有一个“条件”的，当条件被满足，才能开启循环。

在上面这个小例题中，这一点不难理解：我先定义了变量a，然后将0赋值给a；接下来，面对a<5这个条件，计算机就会判断条件满足，用术语来说就是条件为【真】（True），然后开启循环。

直到a+1不断执行，到了 a=5时，条件a<5不再满足，即条件为【假】（False），这段循环就结束了。

真真假假，计算机是怎么做出这样的判断的呢？简单的数学题倒是一看便知，但当逻辑更复杂的时候，计算机怎么决定自己要不要进行下一步行动呢？比如下面这道小题：

```
while False:
    print('while False')
```

我猜你的表情大概是ヾ(。0‸0)ﾉ゛这样的。没事，很快你就能秒懂了，先不理它。

要看懂这种奇怪的条件判断，就要熟悉一个编程基础知识：布尔值。

用数据做判断：布尔值

之前我们提到，计算机利用数据有三种方式：

1. 直接使用数据，
2. 计算和加工数据，
3. 用数据做判断。

除了while循环，我之前学过的if...elif...else语句，也涉及到【利用数据用做逻辑判断】。当逻辑判断通过才会继续执行： 当然，if和while有个显著的区别。那就是if语句只会执行一次，而while是循环语句，只要条件判断为真，就一直循环执行。

这个"判断"的过程，在计算机的世界里是如何发生的呢？

计算机的逻辑判断，只有两种结果，就是True（英文意思是"真"）和False（英文意思是"假"），没有灰色地带。这个计算真假的过程，叫做【布尔运算】。

而True和False，也就叫做【布尔值】。

我们举个例子，请你运行以下代码，看看结果是True还是False：

```
print(3<5)
print(3>5)
print('长安'=='长安')
print('长安'!='金陵')
```

```
True
False
True
True
```

print()括号内的计算其实就是【布尔运算】。终端上出现的True和False我们称为【布尔值】。

第1行和第2行代码很简单啦。我们看后面两行：两个字符串'长安'相等，结果正确，所以打印为True。而字符串'长安'和字符串'金陵'做比较，结果不相等，!=代表不等号，结果正确，打印为True。

只有当条件判断为True时，if和while后面的子句才会执行下去。

好，现在我们来体验一下上面出现过的这段奇怪的代码，你会发现它执行后，终端上不显示任何结果：

```
while False:
    print('while False')
```

这是因为while循环执行，必须要while后的条件为"真"，而【布尔值】False本身就是"假"，所以这个循环一次都没运行就结束了。

同理，3>5这个条件恒为"假"，如果把while False换成while 3>5，效果会是一样的——什么都不会出现。

```
while 3>5:
    print('while False')
```

这时候可能就有人会问了："那如果我用while True或者while 3<5来做条件，会怎么样？"

我们就来体验一下。怕你生气，先剧透：请注意，这段代码将会无限运行，陷入【死循环】，你需要在键盘上按几次【ctrl+c】来强制结束运行。

```
while True:
    print('while True')
```

这是因为【布尔值】True直接把条件判断的结果设置为真，也就是条件永远正确。所以代码会无限循环，必须手动强制退出。

【布尔值】对条件判断语句有一样的效果：

```
if False:
    print('if False')
if True:
    print('if True')
```

```
if True
```

我们会发现，只有if True成功运行，将结果显示在了屏幕上。当然，if语句不会陷入死循环。

我们做一个总结：

两个数值做比较

其实刚才我们print(3>5)，计算机会先做一次布尔运算，判断3是否大于5，然后再把判断的结果以【布尔值】的方式打印出来。

用两个数值做比较的【布尔运算】，主要包含以下情况：

主要记住前两种==和!=，这两种在条件判断中用得非常多。还有大于>和小于<了，这都属于数学常识了。

另外，特别提醒大家一点：在代码中，A == B表示A和B相等，==表示相等关系；而=表示给变量赋值。=和==虽然长得相似，但没有任何关系。

对于=和==，初学者很容易混淆写错，导致报错。请你来修改一下下面的代码，让它能正确运行：

```
password = input('请输入密码：')

if password == 'abc':
    print('密码正确！')
else:
    print('密码错误！')
```

我们接着看布尔运算的第二种方式：直接用数值做运算。

直接用数值做运算

请看这段代码：

```
if 1:
    print('熊猫')
```

是不是觉得怪怪的，没关系，我们来运行一下，看终端会给我们一个啥样的结果。

```
if 1:
    print('熊猫')
```

终端打印出了一个字符串熊猫。我们看看这两行代码，这是一段使用了条件判断的代码。因为终端打印出了'熊猫'，说明 if 后面的条件为真。

可是怪就怪在，if后面接的不像是一个“条件”，而是一个数字。

其实，整数1在这里就是作为一个条件，被判断为真（True）。这就是数值本身作为一个条件，被判断真假的情况。

那为什么可以这样操作呢?

因为在Python中已经设定好什么数据为真，什么为假。假的是有限的，那么除了假的，就都是真的。请看下图： 这个表的左侧一列，它们在Python中被判定为假，比如False、0、''(空字符串)等等。假的东西是有限的，那么除了假的，其他就都是真的。比如上一个例子中出现的整数1，就是真的。

至于None，它代表的是【空值】，自成一派，数据类型是NoneType。要注意它和0的区别，0是整数0，可并非什么都没有。

好，现在请看这段代码：

```
if '开心':
    print('熊猫')
if '':
    print('熊猫')
```

答对啦，'开心'这个字符串作为一个条件时，被判定为真，所以if后面的条件满足，计算机会执行if的下一行代码；而''（空字符串）本身作为一个条件时，被判定为假，if后面的条件为假，计算机不会执行if的下一行代码。

我们可以使用bool()函数来查看一个数据会被判断为真还是假。这个函数的用法与type()函数相似（还有印象吧~），在bool()函数括号中放入我们想要判断真假的数据，然后print出来即可。

我们来做个试验,用bool()函数把前面我们提到的各种数据都放进去判断一下，打印出来看看。请阅读代码后再直接运行：

```
print('以下数据判断结果都是【假】：')
print(bool(False))
print(bool(0))
print(bool(''))
print(bool(None))

print('以下数据判断结果都是【真】：')
print(bool(True))
print(bool(1))
print(bool('abc'))
```

好啦，现在你明白了"直接用数值做布尔运算"是怎么一回事，再来看看第三种情况。

布尔值之间的运算

你会接触到and、or、not、in、not in五种运算，别怕，只是看起来多，我保证不难。

还是用例子来说明吧。我们先看看【and】和【or】。请先阅读代码，然后直接运行：

```
a = 1
b = -1

print('以下是and运算')
if a==1 and b==1:    # 【b实际上是-1】
    print('True')
else:
    print('False')

print('以下是or运算')
if a==1 or b==1:  # 【b实际上是-1】
    print('True')
else:
    print('False')
```

注意到屏幕上的结果了么？ a==1 and b==1的意思是【a=1并且b=1】，要两个条件都满足，才能判断为True，而a==1 or b==1的意思是【a=1或者b=1】，只要两个条件满足一个，就能判断为True。

这里把and和or的计算逻辑和大家做一个总结：

接下来我们看看【not】运算。这个运算很简单，表示翻转的意思。not True就等于False，not False就等于True。

最后，我们讲一下【in】和【not in】两种运算。

【in】的意思是"判断一个元素是否在一堆数据之中"，【not in】反之。这个超简单的。

```
list = [1,2,3,4,5]
a = 1

print(bool(a in list))
print(bool(a not in list))
```

如果涉及到的数据集合是字典的话，【in】和【not in】就可以用来判断字典中是否存在某个【键】

```
dict = {'法国':'巴黎','日本':'东京','中国':'北京'}
a = '法国'

print(bool(a in dict))
```

到这里，我们学完了5种布尔值的运算方式，不难，就是有点多，我还有点啰嗦，对吧~

让我们对所有的布尔计算方式做个总结。连续点击回车后你会拥有3张总结图，不用细看，保存下来整理成你自己的笔记吧~

掌握了布尔值，以后我们可以写出更加简洁的while循环。

比如说，昨天用while循环重复执行100遍任务，代码会这样写：

```
i = 1
while i<101 :
   print('把这句话打印100遍')
   i = i+1
```

今天理解布尔运算原理后，可以把这段代码改造成更"程序员"的方式：

```
i = 100
while i:
   print('把这句话打印100遍')
   i = i-1
```

第二种方式对计算机来说是更简单的，因为计算次数减少了（减少了计算i<100,i<99……这个过程），这也是程序员在写代码时的思考方式。

我们以后还会经常看到布尔运算和布尔值在代码中的运用，今天对布尔值的讲解就到这里。 接下来，是本关的第二个主题：循环里的四种新语句

四种新的语句

这4种新语句配合for循环和while循环，可以让循环发挥更多的功能。我们逐个来看。

break语句

我们先来看看break语句。break的意思是"打破"，是用来结束循环的，一般写作if...break。它的写法长这样：

```
# break语句搭配for循环
for...in...:
    ...
    if ...:
        break

# break语句搭配while循环
while...(条件):
    ...
    if ...:
        break
```

在这里，if...break的意思是如果满足了某一个条件，就提前结束循环。记住，这个只能在循环内部使用。

我们运行一下代码来理解这一点。下面是一个for循环代码，本来会循环5次，但循环到第4次的时候就被break语句打断，然后结束循环了。

```
for i in range(5):
    print('明日复明日')
    if i==3:  # 当i等于3的时候触发
        break # 结束循环
```

下面是一个while循环代码，本来会循环5次，但循环到第3次的时候就被break语句打断，然后结束循环了。

```
i = 0
while i<5:
    print('明日复明日')
    i = i+1
    if i==3:  # 当i等于3的时候触发
        break # 结束循环
```

小小地提醒你，break前面一共缩进了【8个空格】。

这是因为if之下的语句要缩进4个空格（按一次Tab键），这里的if...break又嵌套在while或for循环中，而循环语句本身又要缩进4个空格。这样一来，break就缩进了【4+4=8】个空格（按两次Tab键）。【注】：Tab键和空格键不能同时混用。

```
while True:
    print('上供一对童男童女')
    t = input('孙悟空来了吗')
    if t == '来了':
        break
print('孙悟空制服了鲤鱼精，陈家庄再也不用上供童男童女了')
```

第1行代码：while True我们在上面见过了，这个条件恒为真，就会开启无限循环。而while True常和break语句搭配使用，你也可以学着使用这种写法。

第2行代码：打印上供一对童男童女的字符串。

第3行代码：请用户输入一个信息。

第4行代码：如果用户输入的信息是“来了”，那么if后面的条件被满足，执行下面的代码break；如果没有，回到while True 继续循环。

第5行代码：break表示结束循环，然后去执行循环外部的代码，即第6行代码，打印孙悟空制服了鲤鱼精的字符串。

【练习时间来咯】接下来是你表现的机会！我们来练习一下break语句的编程。

我想请你写下这样一个程序，功能是请用户输入密码，如果输入了错误的密码，就一直循环请用户继续输入；如果输入了正确的密码，就结束循环。设定这个密码为'小龙女'。（其实之前做过这题，有印象吗？）

提示：① 用while True开启无限循环。② 在循环内部用input() 函数获取用户的数据。 ③ 使用if...break，如果变量p == '小龙女'，那就break结束循环。否则又开始循环。④ 结束循环后，打印--通过啦~

```
while True:
    p=input('请用户输入密码')
    if p=='小龙女':
        break
print('通过了')
```

第1行：用while True 开启了一个无限循环。你也可以用while 1 或其他形式开启一个无限循环。

第2行：用input() 函数获取到了一个数据。第3行：如果这个数据等于之前设定的密码，那么就第4行：结束循环。如果这个数据不等于之前设定的密码，那就回到第1行while True继续循环。

第5行：结束循环后，就打印--通过啦。这已经是循环外部的代码了，所以没有缩进噢。

break子句，咱们就讲到这里。接下来“继续”，continue语句。

continue语句

continue的意思是“继续”。这个子句也是在循环内部使用的。当某个条件被满足的时候，触发continue语句，将跳过之后的代码，直接回到循环的开始。

它的写法是这样的：

```
# continue语句搭配for循环
for...in...:
    ...
    if ...:
        continue
    ...

# continue语句搭配while循环
while...(条件):
    ...
    if ...:
        continue
    ...
```

请观察代码，然后运行代码看看是什么结果：

```
# continue语句搭配for循环
for i in range(5):
    print('明日复明日')
    if i==3:  # 当i等于3的时候触发
        continue # 回到循环开头
    print('这句话在i等于3的时候打印不出来')
```

```
# continue语句搭配while循环
i = 0
while i<5:
    print('明日复明日')
    i = i+1
    if i==3:  # 当i等于3的时候触发
        continue # 回到循环开头
    print('这句话在i等于3的时候打印不出来')
```

上面的代码，当i==3的时候会触发continue语句，于是回到了循环开头，跳过了一句print语句。

```
while True:
    q1 = input('第一问：你一生之中，在什么地方最是快乐逍遥？')
    if q1 != '黑暗的冰窖':
        continue
    print('答对了，下面是第二问：')
    q2 = input('你生平最爱之人，叫什么名字？')
    if q2 != '梦姑':
        continue
    print('答对了，下面是第三问：')
    q3 = input('你最爱的这个人相貌如何？')
    if q3 == '不知道':
        break
print('都答对了，你是虚竹。')
```

别怕哈，表面上看这个代码老长老长，其实不难，我们先看1-5行代码。

第1、2行我就不说了，从第3行开始，当第一问的答案不是“黑暗的冰窖”时，就说明答错了，必须从头开始，所以使用continue提前开始循环，回到第1行代码。

如果在第3行，有人回答对了，那么if q1的条件为假，就会继续执行第5行的代码。之后的代码都是同样的道理。

再看10-13行代码，到了最后一行，如果答对，就可以用break结束循环。答错的话，重新开始，这里就不需要continue了。

为了让你更好地理解，请你运行下面的代码：

①你可以随便输入，感受一下循环的逻辑，比如第一个答案对，第二个答案错；或者第一、二个答案对，第三个答案错。

②当你玩得差不多了，请依次输入`黑暗的冰窖’，点击enter，再输入`梦姑’，点击enter，再输入`不知道’，点击enter，就能结束循环。

你可以看到continue的作用就是就是当某个条件为真时，又提前回到循环，而不会执行下面的代码。关于continue语句，我就讲到这里。接下来是pass语句。

pass语句

pass语句就非常简单了，它的英文意思是“跳过”。

```
a = int(input('请输入一个整数:'))
if a >= 100:
    pass
else:
    print('你输入了一个小于100的数字')
```

这个代码的意思是：当a>=100的时候，跳过，什么都不做。其他情况，也就是a<100的时候，执行一个print语句。

如果没有pass来占据一个位置表示"什么都不做"，以上的代码执行起来会报错：（请你先体验一下报错，然后把pass语句加上。）

else语句

最后一种else语句，我们在条件判断语句见过【else】，其实，else不但可以和if配合使用，它还能跟for循环和while循环配合使用。

举一个例子；

```
for i in range(5):
    a = int(input('请输入0来结束循环，你有5次机会:'))
    if a == 0:
        print('你触发了break语句，循环结束，导致else语句不会生效。')
        break
else:
    print('5次循环你都错过了，else语句生效了。')
```

以上这段代码，请你做两种尝试：

1. 连续五次不输入零；
2. 输入一次0跳出循环。完成一种尝试后，你可以点击【重做】按钮，再来一次尝试。

```
for i in range(5):
    a = int(input('请输入0结束循环，你有5次机会:'))
    if a == 0:
        print('你触发了break语句，导致else语句不会生效。')
        break
else:
    print('5次循环你都错过了，else语句生效了。')
```

所以，用一句话总结，当循环中没有碰到break语句，就会执行循环后面的else语句，否则就不会执行。

在while循环中，else的用法也是如此，格式一模一样：

```
while...(条件):
    ...
else:
    ...
```

我们来做一个练习：把之前那段for循环的代码改成while循环，要求运行起来效果一模一样。

```
i=0
while i<6:
    a=int(input('请输入0结束循环，你有五次机会：'))
    i=i+1
    if a==0:
        print('你触发了break语句，导致else语句不会生效。')
        break
else:
    print('5次循环你都错过了，else语句生效了。')
```

总结

小练习

我们一起来完成这个"猜大小游戏"的编程，我的思路大概是这样的：

1.一个人在心里想好一个数————所以这个数字是提前准备好的，所以可以设置一个变量来保存这个数字。我就设置我的数字为24。

2.然后让另一个人猜————所以可以使用input()函数来接收另一个人输入的数字，并用int()转化为整数。

3.直到猜对为止————天知道几次才能猜对，所以肯定需要用到循环，并且由于不知道要循环几次，所以适合while循环。

4.如果他猜的数比24小就告诉他"太小了"，如果他猜的数比24大就告诉他"太大了"——这里一看"如果……就……"的描述，就知道应该用if...else...写一个条件判断。

```
n=24
while True:
    a=int(input('你猜我的数字是：'))
    if a>n:
        print('太大了')
```

```
        continue
    elif a<n:
        print('太小了')
        continue
    else:
        print('猜对了')
        break
```

```
n=int(input('你的密码数字是多少？'))
i=0
while i<3:
    g=int(input('你猜猜秘密数字是多少？'))
    i=i+1
    if g==n:
        print('猜对了')
        break
    elif g>n:
        print('太大了')
    else:
        print('太小了')
else:
    print('失败了')
```

```
secret = 24
for i in range(3):
    guess = input('guess which number is my secret:')
    if  int(guess) ==secret:
        print('正确！你很棒哦。')  #输出结果
        break
    elif int(guess)>secret:
        print('你猜的太大了，请重新猜猜~')
    else:
        print('你猜的太小了，请重新猜猜~')
else:
    print('给你3次机会都猜不到，你失败了。')
```

作业

```
movies = {
'妖猫传':['黄轩','染谷将太'],
'无问西东':['章子怡','王力宏','祖峰'],
'超时空同居':['雷佳音','佟丽娅'],
}

actor=input('请问你想查询哪个演员? ')
for movie in movies:#遍历所有的电影
    actors=movies[movie]#读取出每个电影对应的演员列表
    if actor in actors:
        print(actor+'出演了电影'+movie)
```

```
#理解使用代码
c={
'1':['no.1'],
'2':['no.2']
}
print(c['1'])
```

```
while True:
    a=input('犯人a你认不认罪? ')
    b=input('犯人b你认不认罪? ')

    if a == '不认':
        if b =='不认':
            print('两人各判3年')
            break
        else:
            print('a*20,b*1')
    else:
        if b=='不认':
            print('b*20,a*1')
        else:
            print('a, b ten years')
```

```
## break的使用，参考答案
while True:
    a = input('A，你认罪吗? 请回答认罪或者不认')
    b = input('B，你认罪吗? 请回答认罪或者不认')
    if a == '认罪' and b == '认罪':
        print('两人都得判10年，唉')
    elif a == '不认' and b == '认罪':
        print('A判20年，B判1年，唉')
    elif a == '认罪' and b == '不认':
        print('A判1年，B判20年')
    else:
```

```
        print('都判3年，太棒了')
        break  # 当满足开头提到的条件时，跳出循环。
```

```
## 第二问 修改代码
i=1
while i:
    a = input('A，你认罪吗？请回答认罪或者不认')
    b = input('B，你认罪吗？请回答认罪或者不认')
    i=i+1
    if a == '认罪' and b == '认罪':
        print('两人都得判10年，唉')
    elif a == '不认' and b == '认罪':
        print('A判20年，B判1年，唉')
    elif a == '认罪' and b == '不认':
        print('A判1年，B判20年')
    else:
        print('都判3年，太棒了')
        print(i-1)
        break  # 当满足开头提到的条件时，跳出循环。
```

```
A，你认罪吗？请回答认罪或者不认认罪
B，你认罪吗？请回答认罪或者不认不认
A判1年，B判20年
A，你认罪吗？请回答认罪或者不认认罪
B，你认罪吗？请回答认罪或者不认不认
A判1年，B判20年
A，你认罪吗？请回答认罪或者不认不认
B，你认罪吗？请回答认罪或者不认不认
都判3年，太棒了
3
```

第7关 项目实操 小游戏大学问

项目实操

项目实操的流程总结为如下三步： 明确项目目标，是指我们希望程序达成什么目的，实现什么功能，从而帮我们将项目拆解成不同的单元；而一个妥当的拆解方案，难度适度递增，能帮我们逐步顺利执行，最终完成项目。这三个步骤可以说是环环相扣的。

那么接下来，我们就正式进入项目实操，在这个过程中，需要你多动动脑，动动手。

明确项目目标

今天且让我扮演一下产品经理的角色。我们此次要实现的需求是：人机PK小游戏。

简单来说，这个游戏中，会随机生成玩家和敌人的属性，同时互相攻击，直至一方血量小于零。

另外，这样的战斗会持续三局，采取三局两胜制，最后输出战斗结果，公布获胜方。

分析过程，拆解项目

编写代码，我们无须苛求一步到位。尤其对于刚接触编程的学习者来说，层层递进、逐渐提升难度才能达到更好的练习效果。

为了让你暖暖身，同时照顾大部分同学的学习节奏，我从“功能叠加、难度递增”这个角度考虑，将我们要实现的小游戏拆分成了三个版本。

版本1.0，主要是帮我们理清战斗逻辑。而版本2.0和3.0，会涉及到一些新的知识点，到时遇到了再和大家介绍。

当项目被清晰地拆解后，剩下的就是去逐步执行，也就是重复“执行→遇到问题→解决问题→继续执行”这个循环的过程。

下面，开始正式写代码咯，让我们一个一个版本来攻克吧！

逐步执行，代码实现

版本1.0：自定属性，人工PK

第一阶段的代码，我们的主要任务是理清战斗的逻辑，再用print()函数将战斗过程打印在终端。

我们先来思考一下，一个人机PK游戏最基础的元素是什么，我们可以拿最经典的拳皇游戏来脑补一下。
根据这一版本的设定，我们要做的主要有三步：

1. 规定并显示出玩家和敌人的属性
2. 双方同时互相攻击，血量根据对方的攻击力扣除
3. 若有一方血量小于等于0，游戏结束。

为了让我们的思路保持清晰，画成流程图就是这样子的：

别说你平时不玩游戏，不知道该怎么动手。这个版本的所有步骤，都还很不“智能”，只用到了唯一一个函数Print()。也就是说，我们只要把步骤一个一个打印上去，就算成功啦。

好，我们从第1步开始：设定【玩家】和【敌人】的属性，即【血量】和【攻击】。

好，我们从第1步开始：设定【玩家】和【敌人】的属性，即【血量】和【攻击】。

```
print('【玩家】血量：100 攻击：50')  # 自定义玩家角色的血量和攻击
print('【敌人】血量：100 攻击：30')  # 自定义敌人角色的血量和攻击
```

第2步：手动计算攻击一次，双方各自所剩的血量。

```
print('你发起了攻击，【敌人】剩余血量50')  # 人工计算敌人血量：100-50=50
print('敌人向你发起了攻击，【玩家】剩余血量70')  # 人工计算玩家雪量：100-30=70
```

第3步：继续做人工计算：算一算，玩家攻击2次敌人，敌人的血量就等于0了，这时候可以结束战斗，打印游戏结果。

```
print('你发起了攻击，【敌人】剩余血量0')  # 双方同时攻击，若血量出现小于等于0，游戏结束
print('敌人向你发起了攻击，【玩家】剩余血量40')

print('敌人死翘翘了，你赢了！') # 打印结果
```

很简单吧！现在我们要做的，就是把这三段代码拼起来，然后我会加一些修饰视觉的换行符和分割线，让运行结果看得更清楚一点。

```
print('【玩家】\n血量：100\n攻击：50')  # 自定义玩家角色的血量和攻击，用换行符'\n'来优化视觉
print('------------------------')  # 辅助功能，起到视觉分割的作用，让代码的运行结果更清晰

print('【敌人】\n血量：100\n攻击：30')
print('------------------------')

print('你发起了攻击，【敌人】剩余血量50')  # 人工计算敌人血量：100-50=50
print('敌人向你发起了攻击，【玩家】剩余血量70')  # 人工计算玩家血量：100-30=70
print('------------------------')

print('你发起了攻击，【敌人】剩余血量0')  # 双方同时攻击，若血量出现小于等于0，游戏结束
print('敌人向你发起了攻击，【玩家】剩余血量40')
print('------------------------')

print('敌人死翘翘了，你赢了！') # 打印结果
```

```
【玩家】
血量：100
攻击：50
------------------------
【敌人】
```

```
血量：100
攻击：30
------------------------
你发起了攻击，【敌人】剩余血量50
敌人向你发起了攻击，【玩家】剩余血量70
------------------------
你发起了攻击，【敌人】剩余血量0
敌人向你发起了攻击，【玩家】剩余血量40
----------------------
敌人死翘翘了，你赢了！
```

唔...虽然看起来还有点儿意思，但所有信息一下子都蹦跶出来，一点都没有体现游戏的进程感。

所以，为了让打印出的东西能有时间间隔地依次出现，我们需要设置一个类似"计时器"的东西。在Python里，我们需要用到两行代码来实现：（敲黑板，很简单的新知识）

```
import time    #调用time模块
time.sleep(secs)
#使用time模块下面的sleep()函数，括号里填的是间隔的秒数（seconds，简称secs）
#time.sleep(1.5)就表示停留1.5秒再运行后续代码
```

这里有个新名词——模块，它是Python里一个重要的概念，我会在第16关给你详细介绍一番。

你可以把模块想象成是一个装着许多神奇函数的百宝箱，不过想要使用这个百宝箱里的函数，得先用import 模块名 这样一句代码来打开它。

然后这里我们想使用time模块里的sleep()函数，也就是让代码运行结果不要一次性全部出现，而是分批分批的出现。就要写成time.sleep(secs)的形式。

如果我想设置成打印的信息间隔1.5秒出现，代码就可以这么写：

```
import time  #通常import语句会写到代码的开头

print('【玩家】\n血量：100\n攻击：50')
print('------------------------')
time.sleep(1.5)
#暂停1.5秒，再继续运行后面的代码

print('【敌人】\n血量：100\n攻击：30')
print('------------------------')
time.sleep(1.5)
#同上

print('你发起了攻击，【敌人】剩余血量50')
print('敌人向你发起了攻击，【玩家】剩余血量70')
print('------------------------')
time.sleep(1.5)
```

```
print('你发起了攻击，【敌人】剩余血量0')
print('敌人向你发起了攻击，【玩家】剩余血量40')
print('------------------------')
time.sleep(1.5)

print('敌人死翘翘了，你赢了！')
```

```
【玩家】
血量：100
攻击：50
------------------------
【敌人】
血量：100
攻击：30
------------------------
你发起了攻击，【敌人】剩余血量50
敌人向你发起了攻击，【玩家】剩余血量70
------------------------
你发起了攻击，【敌人】剩余血量0
敌人向你发起了攻击，【玩家】剩余血量40
------------------------
敌人死翘翘了，你赢了！
```

呼～总算完成了版本1.0，不过我想你一定在心里默默吐槽，一句句用print()写也太蠢太弱鸡了吧。

没错，不过代码嘛，总得一步步实现。就先当作是小小的热身。

而且，这个版本的代码还有两个明显的缺陷：一是玩家和敌人的属性（血量&攻击）是我自己说了算，那胜负早已没有悬念；二是战斗过程中血量的变化要自己手动算，那要计算机有何用?

你放心，这些都是我们会在版本2.0解决的问题。

版本2.0：随机属性，自动PK

如前所述，这个阶段，我们主要新增【随机属性】和【自动战斗】两个功能，画成流程图是这样子的：想一想：自己来定义双方角色的属性，那简直是黑箱操作，胜负早已注定。所以，为了游戏公平，我们要让属性由自己说了算变成随机生成。

现在问题来了，要随机生成属性（数字），这课堂里又没教.google

我们看这段文字可以发现，要随机生成整数，就要用到random模块里的randint()函数，括号里放的是两个整数，划定随机生成整数的范围。

请你阅读下列代码注释，然后点击运行。

```
#可以多运行几次，看看结果是不是随机生成的~

import random
#调用random模块，与
a = random.randint(1,100)
# 随机生成1-100范围内（含1和100）的一个整数，并赋值给变量a
print(a)
```

```
62
```

1. 定义两个变量，来存储玩家血量和玩家攻击力的数值
2. 血量是100-150的随机数，攻击力是30-50的随机数
3. 将两个变量打印出来

```
import random

player_life=random.randint(100,150)
player_attack=random.randint(30,50)

print(player_attack)
print(player_life)
```

```
43
109
```

应该没问题吧？不过可能稍微卡住的你不是生成随机数，反而是给变量取名字。

标准的变量名最好是用英文来表达含义，如果是多个单词组成，需要用英文下划线_来隔开。

对于取英文变量名，很多英语水平在高考即巅峰的同学会感到头疼，这里我推荐大家一个网站:CODELF（https://unbug.github.io/codelf），输入中文就可以看到别人是怎么命名的。

好，我们已经知道如何生成随机属性，下面我们就要将属性展示打印出来，请阅读下列代码，弄懂每一行的含义：

```
import time
import random
#也可合并写成一行：import time,random

# 生成随机属性
player_life = random.randint(100,150) # “player_life” 代表玩家血量
player_attack = random.randint(30,50) # “player_attack” 代表玩家攻击
enemy_life = random.randint(100,150) # “enemy_life” 代表敌人血量
enemy_attack = random.randint(30,50) # “enemy_attack” 代表敌人攻击
```

```
# 展示双方角色的属性
print('【玩家】\n'+'血量：'+str(player_life)+'\n攻
击：'+str(player_attack))
#player_life和player_attack的数据类型都是整数，所以拼接时需要先用str()转换
print('------------------------')
time.sleep(1)
#暂停一秒再执行后续代码
print('【敌人】\n'+'血量：'+str(enemy_life)+'\n攻击：'+str(enemy_attack))
print('------------------------')
```

```
【玩家】
血量：122
攻击：44
------------------------
【敌人】
血量：148
攻击：35
------------------------
```

那截至目前，我们已经完成了随机生成属性和展示属性，接下来我们就来实现"自动战斗"。 要怎么实现自动战斗呢？如果一头雾水的话，可以先尝试从版本1.0的人为战斗来寻找规律：

```
print('【玩家】 血量：130  攻击：50')
print('【敌人】 血量：150  攻击：40')

print('你发起了攻击，【敌人】剩余血量100')
print('敌人向你发起了攻击，【玩家】剩余血量90')
print('------------------------')

print('你发起了攻击，【敌人】剩余血量50')
print('敌人向你发起了攻击，【玩家】剩余血量70')
print('------------------------')

print('你发起了攻击，【敌人】剩余血量0')
print('敌人向你发起了攻击，【玩家】剩余血量40')
print('------------------------')

print('敌人死翘翘了，你赢了！')
```

我们可以发现，4-6行这3行是重复出现的结构，除了数字是灵活变动之外，其余是一毛一样的。

说到循环，我们就要思考是要使用for循环还是while循环了。

因为现在双方的血量和攻击是随机生成，不是固定的。所以我们不知道具体要战斗多少回合才能分出胜负，也就是循环次数不明确，那自然要用while循环。

我们进一步思考：while后面要接什么条件呢，也就是说什么条件下，战斗过程会一直持续呢？请你思考一下。

一个简单的问题，在什么条件下，战斗过程（循环）会一直持续?

如果有一方血量小于等于0，战斗就结束了。所以我们现在确定了让循环执行需要满足的条件就是——双方血量均大于零，也就是不死不休。

可见while后面要同时满足两个条件，即这两个条件要同时为真，所以我们要用and来连接，用代码来表示就是：

```
while (player_life >= 0) and (enemy_life >= 0):
#and两边的条件分别用括号括起，是一种习惯，方便阅读
```

现在我们确定了执行while循环的条件，接下来就是要填充循环内部的内容。

根据刚才的分析，我们希望循环的内容是双方互相攻击，掉血的过程。

```
print('你发起了攻击，【敌人】剩余血量xxx')
print('敌人向你发起了攻击，【玩家】剩余血量xxx')
print('------------------------')
```

其中【敌人】剩余血量=敌人当前血量-玩家攻击，【玩家】剩余血量=玩家当前血量-敌人攻击。

事实上我们之前已经定义好了这四个变量，每一次互相伤害后，player_life（玩家血量）和enemy_life（敌人血量）都会被重新赋值，所以转换为代码逻辑就是：

```
player_life = player_life - enemy_attack
enemy_life = enemy_life - player_attack
#赋值语句的执行顺序是先计算等号右边，再赋值给左边的变量
```

好，自动攻击的基础逻辑也已经理清楚了。我们先合并一下这之前写过的代码。

```
import time,random

# 生成随机属性
player_life = random.randint(100,150)
player_attack = random.randint(30,50)
enemy_life = random.randint(100,150)
enemy_attack = random.randint(30,50)

# 展示双方角色的属性
```

```
print('【玩家】\n'+'血量：'+str(player_life)+'\n攻
击：'+str(player_attack))
#player_life和player_attack都是整数类型，所以拼接时需要先用str()转换
print('------------------------')
time.sleep(1)
print('【敌人】\n'+'血量：'+str(enemy_life)+'\n攻击：'+str(enemy_attack))
print('------------------------')
time.sleep(1)

while (player_life >0) and (enemy_life > 0):
    player_life = player_life - enemy_attack
    enemy_life = enemy_life - player_attack

    print('你发起了攻击，【敌人】剩余血量'+str(enemy_life))
    print('敌人向你发起了攻击，【玩家】剩余血量'+str(player_life))
    print('------------------------')
    time.sleep(1.5)
```

```
【玩家】
血量：142
攻击：30
------------------------
【敌人】
血量：118
攻击：46
------------------------
你发起了攻击，【敌人】剩余血量88
敌人向你发起了攻击，【玩家】剩余血量96
------------------------
你发起了攻击，【敌人】剩余血量58
敌人向你发起了攻击，【玩家】剩余血量50
------------------------
你发起了攻击，【敌人】剩余血量28
敌人向你发起了攻击，【玩家】剩余血量4
------------------------
你发起了攻击，【敌人】剩余血量-2
敌人向你发起了攻击，【玩家】剩余血量-42
------------------------
```

你应该能感受到，版本2.0总算像模像样了，慢慢逼近我们的项目目标。

不过它还没有实现：打印出每局结果，三局两胜，并打印最终战果的功能。这就是我们在版本3.0要增加的功能。

一鼓作气，让我们顺势攻克最后一座山头。

版本3.0：打印战果，三局两胜

对比版本2.0，在版本3.0中，我们想要增加的功能是：

1. 打印战果：每局战斗后，根据胜负平的结果打印出不同的提示；
2. 三局两胜：双方战斗三局，胜率高的为最终赢家。 我反复解释新增功能，是因为这样不断地明确项目的阶段性目标，可以让自己持续专注地推进项目。

就像我给的提示一样，结果是有三种可能性的，对应的条件如下图所示：

```
import time,random

# 生成随机属性
player_life = random.randint(100,150)
player_attack = random.randint(30,50)
enemy_life = random.randint(100,150)
enemy_attack = random.randint(30,50)

# 展示双方角色的属性
print('【玩家】\n'+'血量：'+str(player_life)+'\n攻
击：'+str(player_attack))
#player_life和player_attack都是整数类型，所以拼接时需要先用str()转换
print('------------------------')
time.sleep(1)
print('【敌人】\n'+'血量：'+str(enemy_life)+'\n攻击：'+str(enemy_attack))
print('------------------------')
time.sleep(1)

while (player_life >0) and (enemy_life > 0):
    player_life = player_life - enemy_attack
    enemy_life = enemy_life - player_attack

    print('你发起了攻击，【敌人】剩余血量'+str(enemy_life))
    print('敌人向你发起了攻击，【玩家】剩余血量'+str(player_life))
    print('------------------------')
    time.sleep(1.5)
# 打印战果
if player_life > 0 and enemy_life <= 0:
    print('敌人死翘翘了，你赢了')
elif player_life <= 0 and enemy_life > 0:
    print('悲催，敌人把你干掉了！')
else:
    print('哎呀，你和敌人同归于尽了！')
```

```
【玩家】
血量：142
攻击：45
------------------------
【敌人】
血量：149
攻击：42
------------------------
```

```
你发起了攻击，【敌人】剩余血量104
敌人向你发起了攻击，【玩家】剩余血量100
----------------------
你发起了攻击，【敌人】剩余血量59
敌人向你发起了攻击，【玩家】剩余血量58
----------------------
你发起了攻击，【敌人】剩余血量14
敌人向你发起了攻击，【玩家】剩余血量16
----------------------
你发起了攻击，【敌人】剩余血量-31
敌人向你发起了攻击，【玩家】剩余血量-26
----------------------
哎呀，你和敌人同归于尽了！
```

同样的，我们可以将其拆分成两个部分：先来个三局，再判断最终胜负。 首先我们来看，三局战斗也是一个可以循环的结构，且循环次数是固定的，所以要用到for循环。

在这里我们可以使用for i in range( )的结构，我们先来回顾一下之前学过的range()函数：

现在，你有思路了吗？尝试把代码打出来吧，让战斗循环三局。（先不用统计最后的结果）

给两个提示：

1. 想清楚哪些代码要嵌套到for循环里，即一局战斗里包括什么信息。确定了for写在哪里之后，一句战斗包含的所有信息都要缩进；
2. 细节也需要留意，如局与局之间要怎么区分开来（时间间隔&打印局数信息）

```
import time,random
for i in range(1,4):
    print('这是第'+str(i)+'局战斗')
# 生成双方角色，并生成随机属性。
    player_life = random.randint(100,150)
    player_attack = random.randint(30,50)
    enemy_life = random.randint(100,150)
    enemy_attack = random.randint(30,50)
# 展示双方角色的属性
    print('【玩家】\n'+'血量：'+str(player_life)+'\n攻
击：'+str(player_attack))
    print('------------------------')
    time.sleep(1)
    print('【敌人】\n'+'血量：'+str(enemy_life)+'\n攻
击：'+str(enemy_attack))
    print('------------------------')
    time.sleep(1)

# 双方PK
```

```
这是第1局战斗
```

```
【玩家】
血量：123
攻击：49
------------------------
【敌人】
血量：113
攻击：39
------------------------
这是第2局战斗
【玩家】
血量：123
攻击：48
------------------------
【敌人】
血量：137
攻击：35
------------------------
这是第3局战斗
【玩家】
血量：129
攻击：33
------------------------
【敌人】
血量：128
攻击：40
------------------------
```

如果做起来有些障碍，检查一下是否存在上面提示的这几个问题：

1. for循环语句的位置放的对不对？这个关键在于，你想让哪些信息被循环展示。例如：如果你错将for循环语句放在了【随机属性】 和【自动战斗】之间，那每一局的战斗信息会是一样的，也就不存在什么三局两胜了。
2. 你写完for循环语句后，需要缩进的信息【整体】缩进了吗？如果没有缩进，可能存在报错，或者只有部分战斗信息循环的情况。
3. 细节注意到了吗？局与局之间要有明显间隔，那我们可以同时使用time.sleep()和print('现在是第x局')来完美解决这个问题。此外，遇到各种报错的话，记得去搜索一下，看看报的是什么错，先自己尝试解决看看。

好，现在来看我的答案，请你主要看3-5行（后面内容都要放在for循环内部），然后运行看看效果。

```
import time,random

for i in range(1,4):
    time.sleep(1.5)  # 让局与局之间有较明显的有时间间隔
    print(' \n————现在是第'+str(i)+'局，ready go!————')  # 作为局的标记

    player_life = random.randint(100,150)
    player_attack = random.randint(30,50)
    enemy_life = random.randint(100,150)
    enemy_attack = random.randint(30,50)
```

```
    # 展示双方角色的属性
    print('【玩家】\n'+'血量：'+str(player_life)+'\n攻
击：'+str(player_attack))
    print('------------------------')
    time.sleep(1)

    print('【敌人】\n'+'血量：'+str(enemy_life)+'\n攻
击：'+str(enemy_attack))
    print('------------------------')
    time.sleep(1)
```

```
————现在是第1局，ready go!————
【玩家】
血量：111
攻击：30
------------------------
【敌人】
血量：100
攻击：40
------------------------

————现在是第2局，ready go!————
【玩家】
血量：113
攻击：31
------------------------
【敌人】
血量：107
攻击：30
------------------------

————现在是第3局，ready go!————
【玩家】
血量：132
攻击：40
------------------------
【敌人】
血量：101
攻击：31
------------------------
```

OK，打三局这个需求也成功了。现在我们距离最后的终点只剩一步之遥，只有“统计三局两胜的结果”这个功能还没实现了。

我们可以想一想，平常我们是怎么统计比赛结果呢?

好比乒乓球比赛，有一方赢了一局就翻一下计分牌，让数字+1，最后看哪边的数字大就是哪边获胜。

对于计算机也是如此：它靠数据思考，比如拿数据做计算、做条件判断、做循环等。所以这里的关键就在于，要给计算机数据。

那么仿照计分牌的做法，我们的解决方法也就出来了：采取计分的方式，赢一局记一分，平局不计分。

所以，我们要给计算机一个空白的“计分板”，用于存放【玩家】和【敌人】每一局的得分。

```
player_victory = 0
#存放玩家赢的局数。
enemy_victory = 0
#存放敌人赢的局数
```

那什么情况下，这两个变量会变动（+1）呢？自然是要与具体每一局的结果挂钩，这时候可以回看我们计算输赢的条件判断语句。

```
if player_life > 0 and enemy_life <= 0:  #玩家赢
    print('敌人死翘翘了，你赢了')
elif player_life <= 0 and enemy_life > 0: #敌人赢
    print('悲催，敌人把你干掉了！')
else:                                    #平局
    print('哎呀，你和敌人同归于尽了！')
```

然后，我们将敌人和玩家各自赢的局数给算出来：

```
player_victory = 0
enemy_victory = 0

if player_life > 0 and enemy_life <= 0:
    player_victory = player_victory + 1
    print('敌人死翘翘了，你赢了！')
elif player_life <= 0 and enemy_life > 0:
    enemy_victory  = enemy_victory + 1
    print('悲催，敌人把你干掉了！')
else:
    print('哎呀，你和敌人同归于尽了！')
```

这样三局过后，player_victory和enemy_victory会被赋上新的值。给你一个小技巧：player_victory = player_victory + 1，总是这样写有点烦人，我们可以写作player_victory += 1，这两个代码是等价的，都代表"如果if后的条件满足，变量就+1"。

这也是程序员是追求“极简”的体现。好，我们把这段代码替换一下：

```
player_victory = 0
enemy_victory = 0

if player_life > 0 and enemy_life <= 0:
    player_victory += 1
    print('敌人死翘翘了，你赢了！')
elif player_life <= 0 and enemy_life > 0:
    enemy_victory  += 1
    print('悲催，敌人把你干掉了！')
else:
    print('哎呀，你和敌人同归于尽了！')
```

现在，我们只需要再用一次条件判断，比较两个变量的大小就能知道谁输谁赢了。

我会把思维逻辑展示给你，然后请你将它转换为代码就可以了。注意思考：这一次的条件判断，要不要缩进呢?

```
import time,random

player_victory = 0
enemy_victory = 0

for i in range(1,4):
    time.sleep(2)  # 让局与局之间有较明显的有时间间隔
    print(' \n————现在是第'+str(i)+'局————')  # 作为局的标记

    player_life = random.randint(100,150)
    player_attack = random.randint(30,50)
    enemy_life = random.randint(100,150)
    enemy_attack = random.randint(30,50)

    # 展示双方角色的属性
    print('【玩家】\n'+'血量：'+str(player_life)+'\n攻
击：'+str(player_attack))
    print('------------------------')
    time.sleep(1)
    print('【敌人】\n'+'血量：'+str(enemy_life)+'\n攻
击：'+str(enemy_attack))
    print('------------------------')
    time.sleep(1)

    # 双方PK
    while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        print('你发起了攻击，【玩家】剩余血量'+str(player_life))
        print('敌人向你发起了攻击，【敌人】剩余血量'+str(enemy_life))
        print('------------------------')
        time.sleep(1.5)
```

```
    #打印最终战果
    if player_life > 0 and enemy_life <= 0:
        player_victory += 1
        print('敌人死翘翘了，你赢了！')
    elif player_life <= 0 and enemy_life > 0:
        enemy_victory += 1
        print('悲催，敌人把你干掉了！')
    else:
        print('哎呀，你和敌人同归于尽了！')

if player_victory > enemy_victory :
    time.sleep(1)
    print('【最终结果：你赢了！】')
elif enemy_victory > player_victory:
    print('【最终结果：你输了！】')
else:
    print('【最终结果：平局！】')
```

```
————现在是第1局————
【玩家】
血量：121
攻击：30
------------------------
【敌人】
血量：126
攻击：31
------------------------
你发起了攻击，【玩家】剩余血量90
敌人向你发起了攻击，【敌人】剩余血量96
------------------------
你发起了攻击，【玩家】剩余血量59
敌人向你发起了攻击，【敌人】剩余血量66
------------------------
你发起了攻击，【玩家】剩余血量28
敌人向你发起了攻击，【敌人】剩余血量36
------------------------
你发起了攻击，【玩家】剩余血量-3
敌人向你发起了攻击，【敌人】剩余血量6
------------------------
悲催，敌人把你干掉了！

————现在是第2局————
【玩家】
血量：147
攻击：47
------------------------
【敌人】
血量：118
攻击：31
```

```
------------------------
你发起了攻击，【玩家】剩余血量116
敌人向你发起了攻击，【敌人】剩余血量71
------------------------
你发起了攻击，【玩家】剩余血量85
敌人向你发起了攻击，【敌人】剩余血量24
------------------------
你发起了攻击，【玩家】剩余血量54
敌人向你发起了攻击，【敌人】剩余血量-23
------------------------
敌人死翘翘了，你赢了！

————现在是第3局————
【玩家】
血量：112
攻击：37
------------------------
【敌人】
血量：103
攻击：39
------------------------
你发起了攻击，【玩家】剩余血量73
敌人向你发起了攻击，【敌人】剩余血量66
------------------------
你发起了攻击，【玩家】剩余血量34
敌人向你发起了攻击，【敌人】剩余血量29
------------------------
你发起了攻击，【玩家】剩余血量-5
敌人向你发起了攻击，【敌人】剩余血量-8
------------------------
哎呀，你和敌人同归于尽了！
【最终结果：平局！】
```

不过，这还没完呢。作为一个程序员，代码是我们的名片，我们会追求更加优雅的，方便他人阅读的代码，所以上述代码还有一些优化空间。

所以，以下是彩蛋时间，我会教大家一个新的知识点——【格式化字符串】，作为这一关的收尾。

什么意思呢，上面有这么两行代码，是用来展示双方角色的属性的：

```
print('【玩家】\n'+'血量：'+str(player_life)+'\n攻
击：'+str(player_attack))
print('【敌人】\n'+'血量：'+str(enemy_life)+'\n攻击：'+str(enemy_attack))
```

我们在用+拼接字符串和变量的时候，常常需要考虑变量是什么类型的数据，如果不是字符串类型，还先需要str()函数转换。

并且一句话常常要拼接成好几个部分，然后我们要考虑每一对引号' '的起始位置，好麻烦，相信你多少会有点体会。

所以，为了更方便地实现不同数据类型的拼接，用【格式符%】是更常用更便利的一种方式。

我们可以把%想象成：图书馆里用来占位的一本书。先占一个位置，之后再填上实际的变量。举个例子：下面这两种写法是相同的，请你着重研究下第二行的语法。

```
print('血量：'+str(player_life)+' 攻击：'+str(player_attack))
print('血量：%s 攻击：%s' % (player_life,player_attack))
```

我们看到格式符%后面有一个字母s，这是一个类型码，用来控制数据显示的类型。%s就表示先占一个字符串类型的位置。

还有其他常见的类型码如下图所示： 占完位置之后，我们要以%的形式在后面补上要填充的内容，如此一来我们就免去了转换类型的烦恼。如果是多个数据，就要把它们放进括号，按顺序填充，用逗号隔开。

举个例子，你可以运行一下，对比下列输出的结果：

```
lucky = 8

print('我的幸运数字是%d' % lucky)
print('我的幸运数字是%d' % 8)
print('我的幸运数字是%s' % '小龙女的生日816')
print('我的幸运数字是%d和%d' % (8,16))
```

```
我的幸运数字是8
我的幸运数字是8
我的幸运数字是小龙女的生日816
我的幸运数字是8和16
```

一个小小的提示：%后面的类型码用什么，取决于你希望这个%占住的这个位置的数据以什么类型展示出来，如果你希望它以字符串形式展示，那就写%s，如果你希望它以整数形式展示，那就写%d。

```
print('我的幸运数字是%d，而且我不知道%d' % (8,16))  #8以整数展示
print('我的幸运数字是%s' % 8)  #8以字符串展示
print(8) #整数8与字符串'8'打印出来的结果是一样的
print('8')
```

```
我的幸运数字是8，而且我不知道16
我的幸运数字是8
8
8
```

选用了不同的类型码，打印出的结果却是一样，原因也已经在代码注释中写清楚了：因为整数8与字符串'8'的打印结果是一样的，所以选两种类型码都OK。但这种"都OK"的情况仅限于整数，对文字是行不通的：

```
print('我的幸运数字是%s' % '小龙女的生日816')
print('我的幸运数字是%d' % '小龙女的生日816')
```

```
我的幸运数字是小龙女的生日816
```

```
---------------------------------------------------------------------------
----

TypeError                                 Traceback (most recent call
last)

<ipython-input-20-54fefa130b7b> in <module>
      1
      2 print('我的幸运数字是%s' % '小龙女的生日816')
----> 3 print('我的幸运数字是%d' % '小龙女的生日816')
```

```
TypeError: %d format: a number is required, not str
```

会报错，对吧？好啦，现在你应该更懂这个格式化字符串该怎么用了。

那就看回我们之前的代码，如果把一开始用+拼接的字符串都替换成%格式符表示，我们先替换一部分试试

```
import time
import random

player_victory = 0
enemy_victory = 0

for i in range(1,4):
    time.sleep(1.5)
```

```
    print('  \n————现在是第 %s 局————' % i)
    #对比之前：(' \n————现在是第'+str(i)+'局————')
    player_life = random.randint(100,150)
    player_attack = random.randint(30,50)
    enemy_life = random.randint(100,150)
    enemy_attack = random.randint(30,50)

    print('【玩家】\n血量：%s\n攻击：%s' % (player_life,player_attack))
    print('------------------------')
    time.sleep(1)
    print('【敌人】\n血量：%s\n攻击：%s' % (enemy_life,enemy_attack))
    print('------------------------')
    time.sleep(1)

    while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        print('你发起了攻击，【玩家】剩余血量%s' % player_life)
        print('敌人向你发起了攻击，【敌人】的血量剩余%s' % enemy_life)
        print('------------------------')
        time.sleep(1.2)

    if player_life > 0 and enemy_life <= 0:
        player_victory += 1
        print('敌人死翘翘了，你赢了！')
    elif player_life <= 0 and enemy_life > 0:
        enemy_victory += 1
        print('悲催，敌人把你干掉了！')
    else:
        print('哎呀，你和敌人同归于尽了！')

if player_victory > enemy_victory :
    time.sleep(1)
    print('\n【最终结果：你赢了！】')
elif enemy_victory > player_victory:
    print('\n【最终结果：你输了！】')
else:
    print('\n【最终结果：平局！】')
```

```
————现在是第 1 局————
【玩家】
血量：118
攻击：33
------------------------
【敌人】
血量：130
攻击：45
------------------------
你发起了攻击，【玩家】剩余血量73
敌人向你发起了攻击，【敌人】的血量剩余97
```

你发起了攻击，【玩家】剩余血量28
敌人向你发起了攻击，【敌人】的血量剩余64

你发起了攻击，【玩家】剩余血量-17
敌人向你发起了攻击，【敌人】的血量剩余31

悲催，敌人把你干掉了！

————现在是第 2 局————
【玩家】
血量：125
攻击：45

【敌人】
血量：110
攻击：40

你发起了攻击，【玩家】剩余血量85
敌人向你发起了攻击，【敌人】的血量剩余65

你发起了攻击，【玩家】剩余血量45
敌人向你发起了攻击，【敌人】的血量剩余20

你发起了攻击，【玩家】剩余血量5
敌人向你发起了攻击，【敌人】的血量剩余-25

敌人死翘翘了，你赢了！

————现在是第 3 局————
【玩家】
血量：115
攻击：30

【敌人】
血量：121
攻击：46

你发起了攻击，【玩家】剩余血量69
敌人向你发起了攻击，【敌人】的血量剩余91

你发起了攻击，【玩家】剩余血量23
敌人向你发起了攻击，【敌人】的血量剩余61

你发起了攻击，【玩家】剩余血量-23
敌人向你发起了攻击，【敌人】的血量剩余31

悲催，敌人把你干掉了！

【最终结果：你输了！】

学习format()函数

format()函数是从 Python2.6 起新增的一种格式化字符串的函数，功能比课堂上提到的方式更强大。

format()函数用来占位的是大括号{}，不用区分类型码（%+类型码）。

具体的语法是：'str.format()'，而不是课堂上提到的'str % ()'。

而且，它对后面数据的引用更灵活，不限次数，也可指定对应关系。

看完左侧的代码、结果和注释，你就懂上面几句话的意思了。

```
import time
import random

player_victory = 0
enemy_victory = 0

while True:
    for i in range(1,4):
        time.sleep(1.5)
        print('  \n————现在是第 {} 局————'.format(i))
        player_life = random.randint(100,150)
        player_attack = random.randint(30,50)
        enemy_life = random.randint(100,150)
        enemy_attack = random.randint(30,50)

        print('【玩家】\n血量：{}\n攻击：
{}'.format(player_life,player_attack))
        print('------------------------')
        time.sleep(1)
        print('【敌人】\n血量：{}\n攻击：
{}'.format(enemy_life,enemy_attack))
        print('------------------------')
        time.sleep(1)

        while player_life > 0 and enemy_life > 0:
            player_life = player_life - enemy_attack
            enemy_life = enemy_life - player_attack
            print('敌人发起了攻击，【玩家】剩余血量{}'.format(player_life))
            print('你发起了攻击，【敌人】的血量剩余{}'.format(enemy_life))
```

```
————现在是第 1 局————
【玩家】
```

血量：125
攻击：50

【敌人】
血量：108
攻击：48

敌人发起了攻击，【玩家】剩余血量77
你发起了攻击，【敌人】的血量剩余58
敌人发起了攻击，【玩家】剩余血量29
你发起了攻击，【敌人】的血量剩余8
敌人发起了攻击，【玩家】剩余血量-19
你发起了攻击，【敌人】的血量剩余-42

————现在是第 2 局————
【玩家】
血量：110
攻击：33

【敌人】
血量：100
攻击：39

敌人发起了攻击，【玩家】剩余血量71
你发起了攻击，【敌人】的血量剩余67
敌人发起了攻击，【玩家】剩余血量32
你发起了攻击，【敌人】的血量剩余34
敌人发起了攻击，【玩家】剩余血量-7
你发起了攻击，【敌人】的血量剩余1

————现在是第 3 局————
【玩家】
血量：111
攻击：36

【敌人】
血量：113
攻击：47

敌人发起了攻击，【玩家】剩余血量64
你发起了攻击，【敌人】的血量剩余77
敌人发起了攻击，【玩家】剩余血量17
你发起了攻击，【敌人】的血量剩余41
敌人发起了攻击，【玩家】剩余血量-30
你发起了攻击，【敌人】的血量剩余5

————现在是第 1 局————
【玩家】
血量：138
攻击：47

【敌人】

```
血量：115
攻击：43
------------------------
敌人发起了攻击，【玩家】剩余血量95
你发起了攻击，【敌人】的血量剩余68
敌人发起了攻击，【玩家】剩余血量52
你发起了攻击，【敌人】的血量剩余21
敌人发起了攻击，【玩家】剩余血量9
你发起了攻击，【敌人】的血量剩余-26

————现在是第 2 局————
【玩家】
血量：139
攻击：33
------------------------
【敌人】
血量：148
攻击：50
------------------------
敌人发起了攻击，【玩家】剩余血量89
你发起了攻击，【敌人】的血量剩余115
敌人发起了攻击，【玩家】剩余血量39
你发起了攻击，【敌人】的血量剩余82
敌人发起了攻击，【玩家】剩余血量-11
你发起了攻击，【敌人】的血量剩余49

————现在是第 3 局————
【玩家】
血量：140
攻击：41
------------------------
【敌人】
血量：144
攻击：40
------------------------
敌人发起了攻击，【玩家】剩余血量100
你发起了攻击，【敌人】的血量剩余103
敌人发起了攻击，【玩家】剩余血量60
你发起了攻击，【敌人】的血量剩余62
敌人发起了攻击，【玩家】剩余血量20
你发起了攻击，【敌人】的血量剩余21
敌人发起了攻击，【玩家】剩余血量-20
你发起了攻击，【敌人】的血量剩余-20

————现在是第 1 局————
【玩家】
血量：120
攻击：37
------------------------
【敌人】
血量：124
攻击：48
```

```
----------------------
敌人发起了攻击，【玩家】剩余血量72
你发起了攻击，【敌人】的血量剩余87
敌人发起了攻击，【玩家】剩余血量24
你发起了攻击，【敌人】的血量剩余50
敌人发起了攻击，【玩家】剩余血量-24
你发起了攻击，【敌人】的血量剩余13

————现在是第 2 局————
【玩家】
血量：136
攻击：47
----------------------
【敌人】
血量：118
攻击：35
----------------------
敌人发起了攻击，【玩家】剩余血量101
你发起了攻击，【敌人】的血量剩余71
敌人发起了攻击，【玩家】剩余血量66
你发起了攻击，【敌人】的血量剩余24
敌人发起了攻击，【玩家】剩余血量31
你发起了攻击，【敌人】的血量剩余-23

————现在是第 3 局————
【玩家】
血量：106
攻击：43
----------------------
【敌人】
血量：110
攻击：31
----------------------
敌人发起了攻击，【玩家】剩余血量75
你发起了攻击，【敌人】的血量剩余67
敌人发起了攻击，【玩家】剩余血量44
你发起了攻击，【敌人】的血量剩余24
敌人发起了攻击，【玩家】剩余血量13
你发起了攻击，【敌人】的血量剩余-19

————现在是第 1 局————
【玩家】
血量：108
攻击：45
----------------------
【敌人】
血量：129
攻击：43
----------------------
敌人发起了攻击，【玩家】剩余血量65
你发起了攻击，【敌人】的血量剩余84
敌人发起了攻击，【玩家】剩余血量22
```

```
你发起了攻击，【敌人】的血量剩余39
敌人发起了攻击，【玩家】剩余血量-21
你发起了攻击，【敌人】的血量剩余-6

————现在是第 2 局————
【玩家】
血量：123
攻击：38
------------------------
```

第8关 编程学习的瓶颈

瓶颈1：知识学完就忘

忘了就忘了吧，无所谓。等到要用的时候“临时抱佛脚”也是行得通的，毕竟编程是“开卷”的，你可以随时查阅。

究其原因，我们学编程是为了获得一门实用的技能。因此，所有的编程知识并不是要靠死记硬背来【记住】，而是自己真正吸收，学会【怎么用】。

在编程学习中，【掌握】知识和【记住】知识是两个概念：【掌握知识】是知道有这个知识点，并了解它该往哪用，而【记住知识】只是强行记忆知识点，但不一定知道怎么用，用在哪。

然而，Python涉及到的知识点是非常多的，即便把脑子塞爆也不可能都【记住】。所以在初期，很多知识只需要你在使用的时候有个印象，遇到不明确的，就去翻看学习记录，找到知识点的具体用法，再运行代码检验。

在这里，我要和大家介绍一个对我而言很有效的学习方法：【案例笔记法】。它包括了两种类型的笔记：【用法查询笔记】和【深度理解笔记】。

用法查询笔记

先来说【用法查询笔记】，【用法查询笔记】就是记录知识点的基础用法，它是你的学习记录，能供你快速查阅，加深对知识的印象。

要记住的是：不管你用什么格式记笔记，目的只有一个，就是“方便自己查询使用”。

当然你可以多次登陆学习系统复习或者搜索相关教程，但这样可能要花一点时间才能找到想要的信息，所以将笔记统一记录在可随时查阅的文件里，不失为一个更便捷的好方法。

我自己在初学编程的时候，非常重视在案例中理解、消化知识。因此，我的笔记会以案例居多。

比如先来看一个最简单的“算数运算符”相关笔记：（扫一眼就行）

大家可以发现，这张笔记是用小例子来说明不同算数运算符的用法，并且用注释说明了（1）代码含义（2）实际运行结果。

其中代码含义往往用【#注释……】写在代码后面，实际运行效果往往用【#》》注释……】写在代码下方。

这样注释的话会更直观，比如当要区分'%'和'//'用法的时候就能一目了然。

我们再来看看与“列表”相关的笔记，这里，我记了一些对列表的常见操作。

虽然课堂上我们也是以案例带出知识点，但老师还是建议你能在课后制作一份不同于课堂的案例，写出“代码含义”和“实际运行结果”的注释，这样才能加深对知识的第一印象，需要的时候才想得起笔记在哪。

最后，再来看看以“字典”知识为例的笔记，当然啦，要记什么内容取决于你对具体知识点的熟悉度。

像这样，当你逐步解锁后续关卡，你就能积累越来越多的【用法查询笔记】： 除此之外，当你遇见有些课堂上老师没有涉及到的零碎知识点，也可以补充到笔记里，方便日后查看。

有了这些【查询使用笔记】，在学习新知识的时候，你可以及时回顾已学知识，在练习或实操的时候，你可以快速查看某个知识的具体使用细节。

于是，你的学习过程立马轻松了很多，因为很多知识并不要求自己直接记在脑子里，可以放到【用法查询笔记】中。

当然，“轻松”的意思，不是指我们可以在学习上摸鱼，而是让我们可以把更多时间精力花在关键之处。

说完了【用法查询笔记】，接下来我们看看【深度理解笔记】。

深度理解笔记

就像开头所说，有些知识光有印象还不够，还需要你知道这些知识该如何使用。所以我们需要【深度理解笔记】来总结、理解知识的使用场景。

【深度理解笔记】重在“理解”，所以笔记内容主要是记录对知识的理解。比如你可以看下我对于使用“循环”知识的理解笔记：

可以看到，如果说【用法查询笔记】解决的是“知识点是什么”的问题，那么【深度理解笔记】更侧重解决“为什么要用”以及“怎么用这个知识点”的问题。

通常来说，我们需要回答“什么时候需要用到这个知识？这个知识有什么常见用法？这个知识和其他知识有什么不同？”之类的问题，并写下自己的思考过程。

我们再来看一个关于列表和字典的例子：

当然，深度理解笔记不是一蹴而就的，它会随着你对某一知识的理解程度的加深而不断完善。

作为萌新，你可以尝试从“什么时候需要用到这个知识”记起，在后续练习和实操的过程，当你积累越来越多的时候，进一步补充笔记，比如补充这个知识的多种使用技巧。

知识管理

如果你学Python学到“走火入魔”的话，你会发现【深度理解笔记】和【用法查询笔记】本质上就是一个“字典嵌套列表”，其中【深度理解笔记】是键，【用法查询笔记】是值。

基于这个结构，我们的知识框架就搭建起来了。因为我们有【深度理解笔记】，所以在解决一个编程问题的时候，我们可以轻易想到需要用到哪块知识，接着就可以去【用法查询笔记】里寻找相关的知识细节和具体案例，形成解题的思路。

在解决了问题之后，我们还可以把一些特别典型的案例，继续补充在【用法查询笔记】，是不是有点像以前上学时我们所做的错题集。

同样地，在解题的过程中，我们可能会专门搜索并自学一些额外知识，比如random模块（里面有许多随机函数）的使用方法，也可以一并记录在笔记里。

这样子，随着我们的知识库存越来越多，知识结构越来越完整，我们使用知识也会得心应手。

所以如果你发现自己是“学完即忘”的金鱼记忆体质，不妨试一试老师上述所教的笔记法。当然，第一步是不要怕懒哈哈。

瓶颈2：缺乏解题能力

接下来，我们来看看困扰许多学员的第二个瓶颈：缺乏解题能力，也就是看到题目，隐隐中知道要用什么知识，但就是会像便秘一样死活卡住。

我认为这其中一个很重要的原因是还没有形成解题的思路，殊不知，解题技巧也是需要练习的。虽说解题思路是因人而异，但从中提炼共性的话，我会将解题步骤分为以下几步：

如何解题

接下来，我会用一道经典的题目来详细解释上述步骤：用Python打印出九九乘法表。

分析问题是解决问题的第一步，所以拿到题目的时候我们首先要思考“这是一个怎样的问题”。

题目要求是"打印九九乘法表"，为了让问题变得清晰，我们得先知道"九九乘法表"长什么样，即我们最终想要呈现在终端结果是什么样。

假设我们的目标是在终端打印出这种格式的九九乘法表.

知道了需要实现的结果后，我们就来思考一下，解决这个问题到底要用到什么知识。

首先很明显，要打印信息就必须用到最基本的print()函数。

虽然最取巧的办法是像下面这样写，但显然这么操作，Python会很生气的，因为实在太太太大材小用了。

```
print('1 X 1 = 1')
print('1 X 2 = 2  2 X 2 = 4')
print('1 X 3 = 3  2 X 3 = 6  3 X 3 = 9')
print('1 X 4 = 4  2 X 4 = 8  3 X 4 = 12  4 X 4 = 16')
...
...
```

我们可以留意到，九九乘法表是有一定规律，重复的结构，所以我们可以想到用循环来处理。

同时，我们能看到数字是在灵活变动的，所以用"格式化字符串"来为变动的数字预留位置会方便些。

现在我们知道需要用到"循环"和"格式化字符串"相关知识，那我们就可以开始思考切入点：先从什么地方入手来解决这个问题。

我们再来观察九九乘法表，我们会发现一个规律：每一行的等式里，第一位数会递增，第二位数则会保持不变，并且在第几行就会有多少个等式。

也就是说，每一行其实都可以是一个小循环，那我们就可以以此为切入点，先把每一行的小循环写出来，再进一步寻找规律。

记住从简单的切入点入手，解决一部分问题，往往会让问题变得更简单，更容易被解决。

说起循环，我们就会想到while循环和for循环，

当循环次数是确定的时候，我们优先使用for循环。像这种知识，就可以记在关于循环的【深度理解笔记】里。

那么，现在请你小试牛刀，分别用for循环打印出九九乘法表的第二行和第三行吧，

```
for i in range(1,3):
    print(str(i) + ' X 2 = ' + str(i*2))

for i in range(1,4):
    print(str(i) + ' X 3 = ' + str(i*3))
```

```
1 X 2 = 2
2 X 2 = 4
1 X 3 = 3
2 X 3 = 6
3 X 3 = 9
```

你会发现字符串只能与字符串用'+'号拼接，要拼接数字的话还得先用str()转换，煞是麻烦。所以用我们刚才所说的格式化字符串处理会轻松得多。

这时，之前讲的【用法查询笔记】也能派上用场了。 所以，刚刚的那段代码就可以写成：（主要看格式符的用法，然后点击运行）

```
for i in range(1,3):
    print('%d X %d = %d' % (i,2,i*2))
for i in range(1,4):
    print('%d X %d = %d' % (i,3,i*3))
```

但是，运行后你会发现这样写的话每一个等式都会换行，而我们想要的结果是：解决问题的时候发现了新的问题，所以，我们回到第2步重新思考：

现在我们希望print出来的东西不换行，那怎么实现呢?

正如我反复强调的，很多零碎的知识点老师无法都直接教给你们，需要我们碰到实际问题的时候再去补充，这时就需要我们发挥“搜索大法”，主动搜索新知识。

这不，点进链接，我们一下子就能找到答案。（目前我们学习的是Python3）

原来print()函数里有个参数'end'是用来控制换行行数和结尾字符，比如说，你可以运行下列代码，感受一下，留意'end='后面的内容：

```
print('hello',end='')
print('world')

print('hello',end='  ')
print('world')

print('hello',end='!')
print('world')
```

```
helloworld
hello  world
hello!world
```

我们又要来尝试解决问题了。现在请在刚才代码的基础上优化，让它打印出不换行的效果。

```
for i in range(1,3):
    print('%d X %d = %d' % (i,2,i*2),end=' ')

for i in range(1,4):
    print('%d X %d = %d' % (i,3,i*3),end=' ')
```

```
1 X 2 = 2 2 X 2 = 4 1 X 3 = 3 2 X 3 = 6 3 X 3 = 9
```

但是这样子会发现，天啊真是磨人的小机精，现在所有的等式又都挤在一行了 = =。

别愁，问题很好解决，这里就教给大家一个小技巧，用print('')来控制换行，请你直接运行下下面的代码看一看：

```
1
for i in range(1,3):
    print('%d X %d = %d' % (i,2,i*2),end = '   ')
print('') #用来换行

for i in range(1,4):
    print('%d X %d = %d' % (i,3,i*3),end = '   ')
print('') #用来换行
```

```
1 X 2 = 2   2 X 2 = 4
1 X 3 = 3   2 X 3 = 6   3 X 3 = 9
```

可算输出了我们想要的结果了，那么依葫芦画瓢，把所有的等式放在一起就是：

```
for i in range(1,2):
    print( '%d X %d = %d' % (i,1,i*1) ,end = '   ' )
print('')

for i in range(1,3):
    print( '%d X %d = %d' % (i,2,i*2) ,end = '   ' )
print('')

for i in range(1,4):
    print( '%d X %d = %d' % (i,3,i*3) ,end = '   ' )
print('')

for i in range(1,5):
    print( '%d X %d = %d' % (i,4,i*4) ,end = '   ' )
print('')

for i in range(1,6):
    print( '%d X %d = %d' % (i,5,i*5) ,end = '   ' )
print('')

for i in range(1,7):
```

```
    print( '%d X %d = %d' % (i,6,i*6) ,end = '   ' )
print('')

for i in range(1,8):
    print( '%d X %d = %d' % (i,7,i*7) ,end = '   ' )
print('')

for i in range(1,9):
    print( '%d X %d = %d' % (i,8,i*8) ,end = '   ' )
print('')

for i in range(1,10):
    print( '%d X %d = %d' % (i,9,i*9) ,end = '   ' )
print('')
```

```
1 X 1 = 1
1 X 2 = 2   2 X 2 = 4
1 X 3 = 3   2 X 3 = 6   3 X 3 = 9
1 X 4 = 4   2 X 4 = 8   3 X 4 = 12   4 X 4 = 16
1 X 5 = 5   2 X 5 = 10   3 X 5 = 15   4 X 5 = 20   5 X 5 = 25
1 X 6 = 6   2 X 6 = 12   3 X 6 = 18   4 X 6 = 24   5 X 6 = 30   6 X 6 = 36
1 X 7 = 7   2 X 7 = 14   3 X 7 = 21   4 X 7 = 28   5 X 7 = 35   6 X 7 = 42
7 X 7 = 49
1 X 8 = 8   2 X 8 = 16   3 X 8 = 24   4 X 8 = 32   5 X 8 = 40   6 X 8 = 48
7 X 8 = 56   8 X 8 = 64
1 X 9 = 9   2 X 9 = 18   3 X 9 = 27   4 X 9 = 36   5 X 9 = 45   6 X 9 = 54
7 X 9 = 63   8 X 9 = 72   9 X 9 = 81
```

虽然看起来还很不优雅，但相信我到了这一步，离最后胜利只差一步之遥了！

我们可以看到以上代码是有规律的三行结构，同一结构重复了九次，这提醒我们可以再用一层循环嵌套，来彻底解决重复劳动。

我们再来看看九九乘法表，可以发现行数是固定的，范围是range(1,10)，而列数（等式）则是逐行加一，即第N行就有N个等式。

基于此和上述已完成的代码，你想到了如何用两层for循环消灭重复吗？请尝试一下，最外层的循环已经替你写好。（最终代码可以控制在四行）

```
for i in range(1,10):
    for j in range(1,i+1):
        print('%d × %d= %d'%(j,i,i*j),end='   ')
    print('   ')
```

```
1 × 1= 1
1 × 2= 2  2 × 2= 4
1 × 3= 3  2 × 3= 6  3 × 3= 9
1 × 4= 4  2 × 4= 8  3 × 4= 12  4 × 4= 16
1 × 5= 5  2 × 5= 10  3 × 5= 15  4 × 5= 20  5 × 5= 25
1 × 6= 6  2 × 6= 12  3 × 6= 18  4 × 6= 24  5 × 6= 30  6 × 6= 36
1 × 7= 7  2 × 7= 14  3 × 7= 21  4 × 7= 28  5 × 7= 35  6 × 7= 42  7 × 7=
49
1 × 8= 8  2 × 8= 16  3 × 8= 24  4 × 8= 32  5 × 8= 40  6 × 8= 48  7 × 8=
56  8 × 8= 64
1 × 9= 9  2 × 9= 18  3 × 9= 27  4 × 9= 36  5 × 9= 45  6 × 9= 54  7 × 9=
63  8 × 9= 72  9 × 9= 81
```

```python
for i in range (1,10):
    for j in range(1,10):
        print('%d X %d = %d' % (j,i,i*j),end = '  ')
        if i==j:
            print('')
            break
```

```
1 X 1 = 1
1 X 2 = 2  2 X 2 = 4
1 X 3 = 3  2 X 3 = 6  3 X 3 = 9
1 X 4 = 4  2 X 4 = 8  3 X 4 = 12  4 X 4 = 16
1 X 5 = 5  2 X 5 = 10  3 X 5 = 15  4 X 5 = 20  5 X 5 = 25
1 X 6 = 6  2 X 6 = 12  3 X 6 = 18  4 X 6 = 24  5 X 6 = 30  6 X 6 = 36
1 X 7 = 7  2 X 7 = 14  3 X 7 = 21  4 X 7 = 28  5 X 7 = 35  6 X 7 = 42
7 X 7 = 49
1 X 8 = 8  2 X 8 = 16  3 X 8 = 24  4 X 8 = 32  5 X 8 = 40  6 X 8 = 48
7 X 8 = 56  8 X 8 = 64
1 X 9 = 9  2 X 9 = 18  3 X 9 = 27  4 X 9 = 36  5 X 9 = 45  6 X 9 = 54
7 X 9 = 63  8 X 9 = 72  9 X 9 = 81
```

如果用while循环，也是可以的，不过对比一下，是不是还是第一种解法的代码最简洁呢～

```python
i = 1
while i <= 9:
    j = 1
    while j <= i:
        print('%d X %d = %d' % (j,i,i*j),end = '  ')
        j += 1
    i+=1
    print(' ')
```

```
1 X 1 = 1
1 X 2 = 2  2 X 2 = 4
1 X 3 = 3  2 X 3 = 6  3 X 3 = 9
1 X 4 = 4  2 X 4 = 8  3 X 4 = 12  4 X 4 = 16
1 X 5 = 5  2 X 5 = 10  3 X 5 = 15  4 X 5 = 20  5 X 5 = 25
1 X 6 = 6  2 X 6 = 12  3 X 6 = 18  4 X 6 = 24  5 X 6 = 30  6 X 6 = 36
1 X 7 = 7  2 X 7 = 14  3 X 7 = 21  4 X 7 = 28  5 X 7 = 35  6 X 7 = 42
7 X 7 = 49
1 X 8 = 8  2 X 8 = 16  3 X 8 = 24  4 X 8 = 32  5 X 8 = 40  6 X 8 = 48
7 X 8 = 56  8 X 8 = 64
1 X 9 = 9  2 X 9 = 18  3 X 9 = 27  4 X 9 = 36  5 X 9 = 45  6 X 9 = 54
7 X 9 = 63  8 X 9 = 72  9 X 9 = 81
```

到这里，我们就成功打印出九九乘法表了，撒花！

通过这个案例，你发现了吗，我们的解题步骤其实也是一个循环～所以当我们遇到复杂的题不要怂，学会拆解问题，找到突破口，就能一步步KO掉难题啦！ 最后，我们再来总结一下这一关我们所要攻克的两大瓶颈。

作业

```python
list1 =  [91, 95, 97, 99]
list2 =  [92, 93, 96, 98]
list_1=[]
for item in list1:
    list_1.append(item)
for item in list2:
    list_1.append(item)
print (list_1)
```

```
[91, 95, 97, 99, 92, 93, 96, 98]
```

```python
list1 =  [91, 95, 97, 99]
list2 =  [92, 93, 96, 98]
list_1=list1+list2
print(list_1)
```

```
[91, 95, 97, 99, 92, 93, 96, 98]
```

```
list1 =  [91, 95, 97, 99]
list2 =  [92, 93, 96, 98]

# 把 A 组成绩赋值给一个新列表，用来存合并的成绩——这个细节要注意！
list3 =list1
list3.extend(list2)
print(list3)
list3.sort()
print(list3)
```

```
[91, 95, 97, 99, 92, 93, 96, 98]
[91, 92, 93, 95, 96, 97, 98, 99]
```

```
import numpy as np  # 导入 numpy库，下面出现的 np 即 numpy库

scores1 =  [91, 95, 97, 99, 92, 93, 96, 98]
scores2 = []

average = np.mean(scores1)  # 一行解决。
print('平均成绩是：{}'.format(average))

for score in scores1:
    if score < average:
        scores2.append(score)
        continue  # 少于平均分的成绩放到新建的空列表中，然后继续判断。
print(' 低于平均成绩的有：{}'.format(scores2))  # 上个关卡选做题的知识。


# 下面展示一种NumPy数组的操作，感兴趣的同学可以自行去学习哈。
socres3 = np.array(scores1)
print(' 低于平均成绩的有：{}'.format(socres3[socres3<average]))
```

```
平均成绩是：95.125
 低于平均成绩的有：[91, 95, 92, 93]
 低于平均成绩的有：[91 95 92 93]
```

第9关 函数（定义和调用函数，函数的重要概念）

函数是什么

与数学中的函数不同，在Python中，函数不是看上去冰冷无聊的规则和公式，而是有实打实的、有自己作用的代码。而且，它已经是你的老朋友了。

之所以说函数是我们的老朋友，是我们在先前的课堂中就已经接触到一些Python自带（学名叫“内置”）的函数了。

比如说当我们需要实现“打印”这个功能，我们会用到print()；当我们需要实现“获取数据长度”这个功能，我们会要到len()。这些都是设定好了，可以直接拿过来就用的功能，这就叫做“组织好的代码”。

函数（Function）能实现的功能从简单到复杂，各式各样，但其本质是相通的：“喂”给函数一些数据，它就能内部消化，给你“吐”出你想要的东西。

这就像自动贩卖机，只不过贩卖机是喂点钱，吐出来一些吃的喝的用的东西；而Python函数则是喂各种各样的数据，吐出来各种各样的功能。 眼尖的话，你会发现图片里的函数后面都跟了个括号。而括号里放的东西，也就是我们需要输入的数据，它在函数中被称作【参数】。【参数】指向的是函数要接收、处理怎样的数据。

比如print()函数的括号里可以放不同的参数，根据你放的参数不同，print()函数会帮你在屏幕上显示不同的内容。

```
print('Hello World')
print('万物的终极答案')
print(42)
```

括号里面的字符串和整数，都是print() 函数的参数。

说了这么多函数的特征，现在我们一起看看函数的定义：

还是用贩卖机来打比方，贩卖机是设定好可以直接使用（组织好的），可以重复上架售卖不同的物品（重复使用），功能是卖东西（单一功能）。

而函数呢？以print()函数为例，它也是设定好可以直接使用（组织好的），不论你想打印什么参数都可以（重复使用），而print函数能实现的单一功能就是“打印”。

好，现在问题来了，就像贩卖机不总是有我们想要的东西。除了Python自带的内置函数，我们能不能根据自己编写程序的需要，自己定义一个独一无二的函数呢？

答案是肯定的，下面我就来教大家如何DIY一个函数。

定义和调用函数

第一步，我们需要去定义一个函数，想象这个函数的名字、功能是什么。

定义函数

要定义函数，你要这样写： 照着这个语法，我们先来举个最简单的例子，定义一个打招呼的函数。

```
def greet(name):
    print(name+'早上好')
    return
```

来一起读代码。第1行：def的意思是定义(define)，greet是【函数名】（自己取的），再搭配一个括号和冒号，括号里面的name是参数（参数名也是自己取）。

第2行：def下一行开始缩进的代码是函数要实现的功能，也叫【函数体】。这里的函数体展现出的功能就是：打印出"name+ 早上好"这句话。

第3行：一个简单的return。函数内部一旦遇到return语句，就会停止执行并返回结果。没有return语句的函数，Python也会在末尾隐性地加上return None，即返回None值（return None可以简写为return。）所以你也会看到，我们接下来的很多例子是省略了return语句的。

关于return返回值这个概念，我想你可能会有些困惑。没有关系，return的具体作用我会放在这一关的后面一点来详细讲解，我们先继续往下看。

定义函数的语法并不难，但有些注意事项一开始要特别注意，我标记在下面代码块的注释里，请你一定仔细阅读下。

```
#函数名：最好是取体现函数功能的名字，一般用小写字母和单下划线、数字等组合
def greet(name):
#参数：根据函数功能，括号里可以有多个参数，也可以不带参数，命名规则与函数名相同
#规范：括号是英文括号，后面的冒号不能丢
    print(name+'早上好')
#函数体：函数体就是体现函数功能的语句，要缩进，一般是四个空格
    return
```

看到第三行注释，你可能会有点疑问，怎么括号里可以有参数，又能没有参数？接下来我举个小例子，你可以对比一下。

```
#第一个函数
def pika1():
    print('我最喜爱的神奇宝贝是皮卡丘')

#第二个函数
def pika2(name):
    print('我最喜爱的神奇宝贝是'+name)

#第三个函数
def pika3(name,person):
    print('我最喜爱的神奇宝贝是'+name)
    print('我最喜爱的驯兽师是'+person)
```

几个函数的功能都是打印出句式相同的话，区别在于第一个函数总是输出固定的一句话，所以不需要带参数。

而第二个函数需要参数name的参与，所以括号里需要带上name，第三个函数则需要两个参数来完成功能。所以说，参数的数量需要视函数想要实现什么功能来设置。

现在你可以运行下这些代码，看看会发生什么。

```
#第一个函数
def pika1():
    print('我最喜爱的神奇宝贝是皮卡丘')

#第二个函数
def pika2(name):
    print('我最喜爱的神奇宝贝是'+name)

#第三个函数
def pika3(name,person):
    print('我最喜爱的神奇宝贝是'+name)
    print('我最喜爱的驯兽师是'+person)
```

嗯，终端是不是一片空白，这就对了。

截至目前，我们完成了【定义函数】。但定义函数只是将函数的内部功能封装起来（组织好），它们就像是神奇宝贝里的精灵球，安静地待着，只有听见你的召唤时才会出场，为你所用。

定义完函数的下一步就是调用函数，召唤神奇宝贝的时刻到了。

调用函数

那怎么调用函数呢？很简单，喊出它的名字即可。在Python里，就是输入函数名和参数对应的值。就像这样：

```
def pika1():
    print('我最喜爱的神奇宝贝是皮卡丘')
#该函数没有参数，直接调用函数名。记得英文括号一定不能少

pika1()
def pika2(name):
    print('我最喜爱的神奇宝贝是'+name)
#需要给参数name赋值，每次调用都可以给参数赋不同的值
pika2('皮卡丘')
pika2('喷火龙')

def pika3(name,person):
    print('我最喜爱的神奇宝贝是'+name)
    print('我最喜爱的驯兽师是'+person)
#需要给两个参数分别赋值，并用逗号隔开，否则会报错
pika3('可达鸭','小霞')
```

```
我最喜爱的神奇宝贝是皮卡丘
我最喜爱的神奇宝贝是皮卡丘
我最喜爱的神奇宝贝是喷火龙
我最喜爱的神奇宝贝是可达鸭
我最喜爱的驯兽师是小霞
```

通过这个例子，我们明白调用函数最关键的是：得弄清楚这个函数有多少个参数，如何给参数赋值，这个过程在函数里叫做参数的传递(pass)。

那么，定义和调用函数的基本语法就讲得差不多了。这里想强调的是，为了让大家理解基本概念，目前举的例子都只是在实现非常简单的功能。

但函数的作用远不仅如此，它能将复杂的语句和功能统一封装进一个函数名里，调用者只需明白函数能实现什么，根据需要传递参数即可。

例如，这个tree()函数能打印出一颗圣诞树，参数是圣诞树的层高。现在请你调用该函数两次，分别传递参数4和8，点击运行。

```
def tree(Height):
    print('Merry Christmas!')
    for i in range(Height):
        print((Height-i)*2*' '+'o'+ i*'~x~o')
        print(((Height-i)*2-1)*' '+(i*2+1)*'/'+'|'+(i*2+1)*'\\')
tree(4)
tree(8)
```

```
Merry Christmas!
        o
       /|\
      o~x~o
     ///|\\\
    o~x~o~x~o
   /////|\\\\\
  o~x~o~x~o~x~o
 ///////|\\\\\\\
Merry Christmas!
               o
              /|\
             o~x~o
            ///|\\\
           o~x~o~x~o
          /////|\\\\\
         o~x~o~x~o~x~o
        ///////|\\\\\\\
       o~x~o~x~o~x~o~x~o
      /////////|\\\\\\\\\
     o~x~o~x~o~x~o~x~o~x~o
    ///////////|\\\\\\\\\\\
   o~x~o~x~o~x~o~x~o~x~o~x~o
```

```
   /////////////|\\\\\\\\\\\\\
  o~x~o~x~o~x~o~x~o~x~o~x~o~x~o
 ///////////////|\\\\\\\\\\\\\\\
```

接下来呢，我会针对几个有关函数的重要概念做些展开，和大家一起打牢基础。

函数重要概念

前面我们提到，设置与传递参数是函数的重点。而善解人意的Python呢，支持非常灵活的参数形态。我们就先来了解一下这些参数类型。

主要的参数类型有：

- 位置参数
- 默认参数
- 不定长参数

我会用一个案例把这些参数串起来。

假设你最近开了一家深夜食堂，顾客可以任意点菜。但因为人手不足，所以只能为每个人提供一份开胃菜和一份主食。如果写成函数的形式，这个函数就会有两个参数。

（我先查了查英文词典，确认开胃菜和主食的英文分别是appetizer和course，哈哈）

```
def menu(appetizer,course):
    print('一份开胃菜：'+appetizer)
    print('一份主食：'+course)

menu('话梅花生','牛肉拉面')
```

```
一份开胃菜：话梅花生
一份主食：牛肉拉面
```

这里的'话梅花生'和'牛肉拉面'是对应参数appetizer和course的位置顺序传递的，所以被叫作【位置参数】，这也是最常见的参数类型。之前的神奇宝贝函数也有用到：

```
def  menu(appetizer,course):
    print('一份开胃菜：'+appetizer)
    print('一份主食：'+course+'\n')
#还记得转义字符\n吧，表示换行

menu('牛肉拉面','话梅花生')
menu('话梅花生','牛肉拉面')

menu(course='牛肉拉面',appetizer='话梅花生')
```

```
一份开胃菜：牛肉拉面
一份主食：话梅花生

一份开胃菜：话梅花生
一份主食：牛肉拉面

一份开胃菜：话梅花生
一份主食：牛肉拉面
```

好，位置参数是怎么一回事，你现在已经差不多明白了吧。

回到这个食堂的故事。经营了一阵子之后，为了吸引更多的人流，你决定给每个顾客免费送上一份甜品绿豆沙，这时候你就可以用到【默认参数】，注意：默认参数必须放在位置参数之后。

如果一个参数的值是相对固定的，那么设置默认参数就免去了每次都要传递的麻烦。但默认参数并不意味着不能改变，试试运行下列结果。

```
def  menu(appetizer,course,dessert='绿豆沙'):
    print('一份开胃菜：'+appetizer)
    print('一份主食：'+course)
    print('一份甜品：'+dessert)


menu('话梅花生','牛肉拉面')
menu('话梅花生','牛肉拉面','银耳羹')
#银耳羹对应参数dessert
```

```
一份开胃菜：话梅花生
一份主食：牛肉拉面
一份甜品：绿豆沙
一份开胃菜：话梅花生
一份主食：牛肉拉面
一份甜品：银耳羹
```

一个萝卜一个坑，因为前两个参数已经有对应的值传递，Python会自动将'银耳羹'传递给参数dessert。

了解完默认参数，我们接着往下看。

后来呢，盛夏来袭，你觉得卖烧烤是个不错的主意。但问题是每个人点的烤串数量都不同，你也不能限定死数量，这时候【不定长参数】就能派上用场，即不确定传递参数的数量。

它的格式比较特殊，是一个星号*加上参数名，来看下面的例子。

```
def menu(*barbeque):
    print(barbeque)
menu('烤鸡翅','烤茄子','烤玉米')

#这几个值都会传递给参数barbeque
```

```
('烤鸡翅', '烤茄子', '烤玉米')
```

你会发现输出的是这样的结果：('烤鸡翅', '烤茄子', '烤玉米')，这种数据类型叫做元组(tuple)，曾在第4关的必做练习中与你打过照面。我们就趁势来复习一下：

元组的写法是把数据放在小括号()中，它的用法和列表用法类似，主要区别在于列表中的元素可以随时修改，但元组中的元素不可更改。

当然我们也可以先生成一个元组，再传入参数。上述代码等价于：

```
order=('烤鸡翅','烤茄子','烤玉米')
#元组的长度没有限制
def menu(*barbeque):
    print(barbeque)

menu(*order)
```

```
('烤鸡翅', '烤茄子', '烤玉米')
```

和列表一样，元组是可迭代对象，这意味着我们可以用for循环来遍历它，这时候的代码就可以写成：

```
def menu(appetizer,course,*barbeque,dessert='绿豆沙'):
    print('一份开胃菜：'+appetizer)
    print('一份主菜：'+course)
    print('一份甜品：'+dessert)
    for i in barbeque:
        print('一份烤串：'+i)

menu('话梅花生','牛肉拉面','烤鸡翅','烤茄子','烤玉米',)
```

```
一份开胃菜：话梅花生
一份主菜：牛肉拉面
一份甜品：绿豆沙
一份烤串：烤鸡翅
一份烤串：烤茄子
一份烤串：烤玉米
```

需要注意的是，这时候默认参数也需要放在不定长参数的后面，即dessert='绿豆沙'要放在*barbeque后面，否则传递的值会对应不上。现在请你重现上面的代码（自己手打哦~），注意参数的顺序，调用函数时可以换成你爱吃的食物。

```
def chide(xiaocai,zhushi,tianpin='黑森林可乐',*shaokao):
    print('一盘小菜：'+xiaocai)
    print('主食来啦：'+zhushi)
    print('一份甜品：'+tianpin)
    for i in shaokao:
        print('烧烤之一：'+i)

chide('炒豆苗','馒头','樱桃布丁','羊肉串','肉筋','板筋','烤韭菜')
```

```
一盘小菜：炒豆苗
主食来啦：馒头
一份甜品：樱桃布丁
烧烤之一：羊肉串
烧烤之一：肉筋
烧烤之一：板筋
烧烤之一：烤韭菜
```

我仿佛听到有人的肚子在咕咕叫，加把劲，课堂已经上了一大半了！

现在，你明白了参数的不同形态，以后就可以视实际情况的需要，灵活设置不同的参数类型啦。

接下来，我们着重讲下函数的第二个拓展概念：return语句。

return语句

前面我们提到，return是返回值，当你输入参数给函数，函数就会返回一个值给你。事实上每个函数都会有返回值，像我们之前学过的len()函数。

```
a=[1,2,3]
print(len(a))
```

```
3
```

当你把参数a放进len() 函数中，它返回来的是3这个数值（列表的长度）。之后，我们就可以调用这个值，比如用print()函数打印出来或干点儿别的。

像常见的type()函数、数据类型转换函数，还有我们之前学过的bool()都是这样，会返回一个值。

而print()函数本身比较特殊，它在屏幕上显示完相关的文本内容就没了，并不会返回一个值给我们。所以，它返回的是空值（None）。

在自定义函数的时候，我们就可以用return语句规定该函数要返回什么值给我们。带return语句的函数是这样的：

```
def niduoda(age):
    if age < 12:
        return '哈，是祖国的花朵啊'
    elif age < 25:
        return '哇，是小鲜肉呢'
    else:
        return '嗯，人生才刚刚开始'

print(niduoda(30))
```

```
嗯，人生才刚刚开始
```

参数值30在条件判断中满足else的条件，所以函数返回'嗯，人生才刚刚开始'，再由print()函数打印出来。

提醒一下，函数也是可以互相嵌套的，在这个例子中，niduoda()函数就被嵌套在print()函数里。

可能你会觉得在这个例子中，直接用print不就行了吗，为啥还要用return呢？还有，我们前面讲了那么多函数，好像都是省略了return的啊，比如刚讲的深夜食堂函数和神奇宝贝函数：

```
#神奇宝贝函数
def pika2(name):
    print('我最喜爱的神奇宝贝是'+name)
pika2('皮卡丘')
pika2('喷火龙')

#深夜食堂函数
def menu(appetizer,course,*barbeque,dessert='绿豆沙'):
    print('一份开胃菜：'+appetizer)
    print('一份主菜：'+course)
    print('一份甜品：'+dessert)
    for i in barbeque:
        print('一份烤串：'+i)

menu('话梅花生','牛肉拉面','烤鸡翅','烤茄子','烤玉米')
```

```
我最喜爱的神奇宝贝是皮卡丘
我最喜爱的神奇宝贝是喷火龙
一份开胃菜：话梅花生
一份主菜：牛肉拉面
一份甜品：绿豆沙
一份烤串：烤鸡翅
一份烤串：烤茄子
一份烤串：烤玉米
```

其实是因为在这些例题中，我们的函数功能都是第一时间把参数打印出来。而在很多时候，当多个函数之间相互配合时，我们并不需要第一时间就将结果打印出来，而是需要将某个返回值先放着，等到需要的时候再做进一步的处理。

下面来看个例子：

在我们关于爱情的天真幻想中，我希望我的梦中情人拥有XXX的脸蛋和XXX的身材。

现在需求如下：

一、分别定义两个函数，参数为人名，能够返回字符串'XXX的脸蛋'和'XXX的身材'；

二、将上述两个函数的返回值拼接在一起之后，再打印出来。请你先思考一下，再点击回车。

相信对你来说并不难，如果一步步地写，代码应该是这样的：

```
def face(name):
    return name+'的脸蛋'
def body(name):
    return name+'的身材'

face('Ryan')
body('Ryan')
print('我的梦中情人：'+face('李若彤') +' + ' + body('林志玲'))
```

```
我的梦中情人：李若彤的脸蛋 + 林志玲的身材
```

前面也讲到，函数可以互相嵌套，所以第6,7行调用face()和body()函数的两行代码可以省略，因为第12行的print()函数中其实已经有调用这两个函数了。

这只是一个非常简单的例子，在类似这种多个函数相互配合的代码中，我们就会非常需要return语句，来帮我们先保留某个函数的返回值，等要用到的时候再调出来用。

但是这样的代码还有个问题，当我想多次调用函数的时候，就需要先复制print那行代码，再分别修改两个函数里的参数。这样的操作既不简洁，也不优雅。就像这样：

```
def face(name):
    return name + '的脸蛋'
def body(name):
    return name + '的身材'

print('我的梦中情人：'+face('李若彤') +' + ' + body('林志玲'))
#要再次调用就要复制一遍，然后改参数
print('我的梦中情人：'+face('新垣结衣') +' + ' + body('长泽雅美'))
```

```
我的梦中情人：李若彤的脸蛋 + 林志玲的身材
我的梦中情人：新垣结衣的脸蛋 + 长泽雅美的身材
```

所以更常见的做法是：再定义一个主函数main()，参数调用前两个函数的返回值。老师先给出代码，你可以琢磨一下，主要思考第5行和第6行代码。

```
def face(name):
    return name + '的脸蛋'
def body(name):
    return name + '的身材'
def main(dream_face,dream_body):
    return '我的梦中情人：' + face(dream_face) + ' + ' + body(dream_body)

print(main('李若彤','林志玲'))
print(main('新垣结衣','长泽雅美'))
```

```
我的梦中情人：李若彤的脸蛋 + 林志玲的身材
我的梦中情人：新垣结衣的脸蛋 + 长泽雅美的身材
```

main()函数内部分别调用了face()和body()函数，参数dream_face和dream_body传递给了face()和body()函数的参数name，得到返回值，并打印。看起来有点绕，我们将函数运行的步骤分解，就一目了然了。 好，接下来要讲的是return的一个特别的用法，我们看到前面所有的return语句，都是返回1个值，如果要返回多个值，我们该怎么办呢?

还是看这段代码：

```
def face(name):
    return name + '的脸蛋'
def body(name):
    return name + '的身材'
print('我的梦中情人：'+face('李若彤') +' + ' + body('林志玲'))
```

在这段代码中，我们定义了两个函数，每个return语句分别返回关于`脸蛋"和`身材"的值。如果要一次性返回这两个值，我会这样做，你先看代码：

```
def lover(name1,name2):
    face = name1 + '的脸蛋'
    body = name2 + '的身材'
    return face,body
a=lover('李若彤','林志玲')
print('我的梦中情人：'+a[0]+' + '+a[1])
```

```
我的梦中情人：李若彤的脸蛋 + 林志玲的身材
```

为什么要这样写，第7行代码是啥意思？怎么看上去像是列表？你大概有这样的疑惑了吧。

我们先来看看，一次性返回的两个值，是什么数据类型。

```
def lover(name1,name2):
    face = name1 + '的脸蛋'
    body = name2 + '的身材'
    return face,body
a=lover('李若彤','林志玲')
print(a)
```

```
('李若彤的脸蛋', '林志玲的身材')
```

是('李若彤的脸蛋', '林志玲的身材')这样一个元组，对不对？又是元组。

事实上，Python语言中的函数返回值可以是多个，而其他语言都不行，这是Python相比其他语言的简便和灵活之处。一次接受多个返回值的数据类型就是元组。

而元组与列表其实都是数据的"序列"，元组取某个位置的值的操作，与列表是一模一样的，即tuple()。也是因此，我们会这样去写代码：

```
def lover(name1,name2):
    face = name1 + '的脸蛋'
    body = name2 + '的身材'
    return face,body

a=lover('李若彤','林志玲')
#此时return的值为元组 a = ('李若彤的脸蛋', '林志玲的身材')
print('我的梦中情人：'+a[0]+' + '+a[1])
```

现在你大概对return的作用有了更深的了解吧。我们继续总结：

```
#第一个函数
def fun():
    a ='I am coding'
print(fun())

#第二个函数
def fun():
    a='I am coding'
    return a

print(fun())
```

```
None
I am coding
```

第一个函数因为没有return语句，返回None值，所以打印出None。 此外，return还有一个“副作用”：一旦函数内部遇到return语句，就会停止执行并返回结果。你可以运行下列的代码：

```
def fun():
  return 'I am coding.'
  return 'I am not coding.'

print(fun())
```

```
I am coding.
```

函数内部遇到第一个return就会返回，所以只会打印出'I am coding'。

好啦，关于return语句，我们的知识讲解就到这里。

现在呢，趁热打铁，请你做道题，巩固一下学过的知识。 题目是这样的： 一、定义一个带有两个参数的函数，函数的功能是返回两个参数中较大的那个值；

二、调用函数，将99的平方和8888赋值给参数，并将较大值打印出来。

我们一起来构思一下这段代码该怎么写：

1. 肯定要先定义函数，给函数命名，并且带两个参数；
2. 两个参数中较大的那个值，就是比大小呗，两个数字比大小就三种情况，大于小于等于，所以可以用到if...elif...else。
3. 要返回，那if...elif...else的三个子句，应该都是return。
4. 调用这个函数，把99**2和8888放进去，然后print，搞定！

好啦，请你将这段代码写出来，当然你也可以用自己的思路！

```
def bigger(number1,number2):
    if number1 > number2:
        return number1
    elif number1 < number2:
        return number2
    else:
        return number1+'='+number2

print(bigger(99**2,8888))
```

```
9801
```

```
def big_num(x,y):
    if x>y:
        return x
    elif x==y:
        return '一样大'
    else:
        return y

print(big_num(99**2,8888))
```

```
9801
```

好，return讲完了，现在我们来看今天的最后一个函数重要概念，变量作用域。

变量作用域

当我们定义一个函数的时候，很重要的事情就是理解函数中变量的作用域。下面请你先认真看下面两行字：

- 第一点：在一个函数内定义的变量仅能在函数内部使用（局部作用域），它们被称作【局部变量】。
- 第二点：在所有函数之外赋值的变量，可以在程序的任何位置使用（全局作用域），它们叫【全局变量】。

认真看完了吗？好，我们来看一个例子。

```
x=99   #全局变量x
def num():
    x=88 #局部变量x
    print(x)

num()
#打印局部变量x
print(x)
#打印全局变量x
```

```
88
99
```

你可以看到，两个变量都叫x，一个在函数外部，一个在函数内部。

因为x=99是在函数外赋值的，所以第一个变量x是全局变量；x=88是在函数内赋值的，所以第二个变量x是局部变量。 虽然变量的名字相同（都是x），但因为全局变量和局部变量处在不同的"作用域"中，所以它们彼此井水不犯河水，都能打印出相应的结果。

但为了让程序更易读以及避免可能会出现的麻烦，建议局部变量和全局变量【不要】取相同的名字，以免混淆。

你可以将定义的函数想象成一个私人房间，所以里面存数据的容器（变量）是私有的，只能在个人的房间里使用；而在函数外存数据的变量是公用的，没有使用限制。

有几点需要注意。就像你不希望合租的人随意使用你私人房间里的物品，全局作用域中的代码中也不能使用任何局部变量。来看看一个新手容易踩的坑：

```
def egg():
#定义了一个叫egg的函数
    quantity = 108
#定义了一个变量quantity，并赋值为108
egg()
#调用egg() 函数
print(quantity)
# 打印egg() 函数里面的变量quantity
```

```
---------------------------------------------------------------------------
----

NameError                                 Traceback (most recent call
last)

<ipython-input-53-37d7ccc47bd1> in <module>
      5 egg()
      6 #调用egg() 函数
----> 7 print(quantity)
      8 # 打印egg() 函数里面的变量quantity
```

```
NameError: name 'quantity' is not defined
```

报错信息提示：前3行的代码都没毛病，坏就坏在第5行代码。然后问题是：NameError: name 'quantity' is not defined

这句话翻译成中文就是：变量quantity并没有被定义。其实，我们定义了，只不过是在函数egg() 内定义的，所以这是个局部变量。但问题是我们不能在定义的函数egg() 外面，也就是不能在全局作用域中使用这个局部变量。

那该怎么办呢？其实也很简单，就像私人房间里的人可以使自由使用公共区域的物品，在函数内部的局部作用域，是可以访问全局变量的。

```
quantity = 108
#定义变量quantity，这不是在我们定义的函数内的，所以是全局变量。

def egg():
#定义一个函数，叫egg()
   print(quantity)
#函数内的功能是打印变量quantity
egg()
#调用这个函数
```

```
108
```

显然计算机成功地把变量quantity打印了出来。这是因为全局变量在哪里都可以被使用。

总之记住一句话，当变量处于被定义的函数内时，就是局部变量，只能在这个函数内被访问；当变量处于被定义的函数外时，就是全局变量，可以在程序中的任何位置被访问。

如果你非要将局部变量变成全局变量，就像把私人房间的东西挪到公共区域，可不可以呢？Python也是能够满足你的，只不过要用到一种新的语句global语句，就像这样子：

```
def egg():
    global quantity
#global语句将变量quantity声明为全局变量
    quantity = 108
egg()

print(quantity)
```

```
108
```

我将以上三段代码放在一起，你可以仔细对比一下，注意第一个函数的写法是错误的。

```
def egg():
    quantity = 108
egg()
print(quantity)
#会报错，不能在函数外部（全局作用域）使用函数内的局部变量
```

```
quantity = 108
def egg():
   print(quantity)
egg()
#函数内的局部作用域，可以访问全局变量

def egg():
    global quantity
    quantity = 108
egg()
print(quantity)
#global语句可以将局部变量声明为全局变量
```

最后，来做道题检验一下吧。

```
m1 = 3
def myfunction():
    print(m1)
    m2 = 8
    print(m2)

myfunction()
```

```
3
8
```

答对啦！m1是全局变量，是可以在整个程序内访问的，所以可以在定义的函数myfunction() 内访问。m2是局部变量，print(m2) 是在函数内部调用的，所以也没问题。由此，调用myfunction()也是没问题的。

呼，今天的课到这里就结束了～知识点看起来很多，是因为函数是Python里非常重要的工具！只有掌握了这些基础，我们才有可能去写出更酷炫的代码。

作业

```
# 查看注释，运行代码。
import random
import time

def choujiang(q,w,e):  # 定义一个抽奖函数，带有3个参数，也就是3位候选人
    luckylist = [q,w,e]  # 定义一个中奖名单的列表
```

```
    a = random.choice(luckylist)  # 在中奖名单里面随机选择
    print('开奖倒计时',3)
    time.sleep(1)
    print('开奖倒计时',2)
    time.sleep(1)
    print('开奖倒计时',1)
    time.sleep(1)
    image = '''
    /\_)o<
    |      \\
    | o . o|
    \_____/
    '''
    print(image)
    print('恭喜'+a+'中奖! ')

choujiang('虚竹','萧峰','段誉')  # 调用函数
```

```
开奖倒计时 3
开奖倒计时 2
开奖倒计时 1

    /\_)o<
    |      \
    | o . o|
    \_____/

恭喜虚竹中奖!
```

新知识点：

- 一种新的列表生成方式
- extend 的新用法
- 列表生成式

```
num2 = list(range(1,6))
num2.extend('ABCDE')
list2 = [m+n for m in ['天字', '地字'] for n in '一二']
list3 = [n*n for n in range(1,11) if n % 3 == 0]

print(num2)
print(list2)
print(list3)
```

```
[1, 2, 3, 4, 5, 'A', 'B', 'C', 'D', 'E']
['天字一', '天字二', '地字一', '地字二']
[9, 36, 81]
```

```
# 生成扑克牌：返回一个扑克牌列表，里面有52个元组，对应52张牌。

def cards():
  color = ['红心', '方块', '梅花','黑桃']  # 将花色放在一个列表中待用
  num = list(range(2, 11))
  num.extend('JQKA')  # 通过两行代码，生成一个 2-A 的数字列表。
  return [(x, y) for x in color for y in num ]  # 用列表生成式完成扑克牌的生成。

print(cards())
# 注：花色对应的正式单词是：suit和ran，上面为了好理解所以用了 color。
```

```
[('红心', 2), ('红心', 3), ('红心', 4), ('红心', 5), ('红心', 6), ('红心', 7), ('红心', 8), ('红心', 9), ('红心', 10), ('红心', 'J'), ('红心', 'Q'), ('红心', 'K'), ('红心', 'A'), ('方块', 2), ('方块', 3), ('方块', 4), ('方块', 5), ('方块', 6), ('方块', 7), ('方块', 8), ('方块', 9), ('方块', 10), ('方块', 'J'), ('方块', 'Q'), ('方块', 'K'), ('方块', 'A'), ('梅花', 2), ('梅花', 3), ('梅花', 4), ('梅花', 5), ('梅花', 6), ('梅花', 7), ('梅花', 8), ('梅花', 9), ('梅花', 10), ('梅花', 'J'), ('梅花', 'Q'), ('梅花', 'K'), ('梅花', 'A'), ('黑桃', 2), ('黑桃', 3), ('黑桃', 4), ('黑桃', 5), ('黑桃', 6), ('黑桃', 7), ('黑桃', 8), ('黑桃', 9), ('黑桃', 10), ('黑桃', 'J'), ('黑桃', 'Q'), ('黑桃', 'K'), ('黑桃', 'A')]
```

extend

- 描述

  extend() 函数用于在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）。

- extend()方法语法：

  list.extend(seq)

- 参数

  seq -- 元素列表。

第10关 项目实操 田忌赛马

这次的项目实操流程，和项目1是一模一样的： 接下来，我们一步一步来推进这个项目的实现吧。首先是：明确这个项目的目标。

明确项目目标

这次的小游戏项目是在项目1的基础上进行功能的叠加，之所以这么设置，是想让大家认识到一个小游戏的功能也能做得很复杂，这需要我们非常灵活地调动所学的知识。

我希望你能在这个过程中，逐渐感受到一个程序的需求会经历一个从0到1，从1到100，从雏形到丰满的阶段，所以难度还是不小的哦。

如标题所暗示的，我们会在这个项目里实现"田忌赛马"这个功能，即玩家可自行选择角色出场顺序，与电脑进行3V3的战斗。

我们来看下项目最终的运行效果： 因为是3V3，所以也会根据三局的战斗输出最终的赛果：

还记得我们项目1的结果吗？项目2可以说是对项目1的全面升级。 综上，我们可以得出两个项目的区别：

想必你多少能感受到项目2的难度了吧，别担心，老师会带着你一步步完成的。

上一关我们学习了函数的基础知识，在这个项目中，我们会主要使用函数的相关知识来完成，你会发现，函数还挺妙的！

分析过程，拆解项目

我们先回顾一下之前项目1是怎么拆分的：

那么现在请你尝试在下面的代码区将项目2也拆分成数个个版本吧，在这个过程中你或许能大概知道我们需要用到哪些知识。

提示：可以回看项目运行效果，以及两个项目的对比图。

```
#请完成项目2的拆解。
#可自行添加版本

print('''
版本1.0：手动取名字，手动安排敌我顺序，使用上次工程自动战斗进程
版本2.0：随机名字，敌方随机顺序
版本3.0：可随意安排我方出场顺序
''')
```

```
版本1.0：手动取名字，手动安排敌我顺序，使用上次工程自动战斗进程
版本2.0：随机名字，敌方随机顺序
版本3.0：可随意安排我方出场顺序
```

你的拆解方案很可能也是可行的。为了更平稳地达成所愿，达到教学效果，我们这次就一起按下面的既定方案推进。 我解释一下这个拆分逻辑：因为函数是新学的知识，所以，版本1.0主要起到的是热身作用，让我们能学以致用。

另外，因为游戏的功能会封装在不同的函数里来实现，且后续封装的函数会受到前面的影响。所以，版本2.0可先专注于随机选出角色，生成随机属性，做好战前准备。

版本3.0是这个项目的重头戏，排序功能会单拎出来；版本4.0和项目1的实现逻辑是类似的，所以老师会挑重点来讲。

明确了每一阶段的任务后，接下来就是好戏开场，让我们开始逐步用代码实现功能！

逐步执行，代码实现

你要记住，写代码时碰到问题是正常的。它是一个“执行→遇到问题→解决问题→继续执行”的循环，不过相信循环总有break的时刻，问题是可以被解决的鸭！

让我们先一起来攻克版本1.0：

版本1.0：封装函数，自定属性

我们先来看一个代码，温习一下函数的用法。

```
# 语法：用 def 语句定义一个函数，括号放参数，冒号不要忘
def fav_num(num1,num2):
    # 函数的功能是打印出一句话，包含两个参数，用到了格式化字符串
    print('我喜欢的数字是%d，还有%d。' % (num1,num2))

fav_number(5,8)
#调用fav_num()函数
```

下面我们来看看我们在项目1中所写的代码，这是一段展示双方属性的代码。

```
player_life = 100
player_attack = 35
enemy_life = 105
enemy_attack = 33

print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
print('-------------------------')
time.sleep(1)
print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
print('-----------------------')
```

现在，请你练练手。1. 根据以上代码定义一个名为show_role的函数，功能是打印出双方属性; 2. 传入2-5行的参数，调用这个函数。

```
import time

def show_role(player_life,player_attack,enemy_life,enemy_attack):
  print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
  print('-------------------------')
  time.sleep(1)
  print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
  print('-----------------------')

show_role(100,35,105,33)
```

```
【玩家】
血量：100
攻击：35
-------------------------
【敌人】
血量：105
攻击：33
-----------------------
```

show_role函数只是展示一下角色的属性，还没有开始PK。所以这个项目至少就涉及到两个函数了。

来复习一下函数的特性：一般情况下，一个函数只负责一个单一的功能，以便编写、理解和维护。 而当程序需要同时调用多个函数的话，一个习惯的用法是定义一个主函数main()，将其他函数放在主函数main()内部，起到一种封装的效果。

还是先看个例子复习一下：

```
#第一个函数：打印我喜欢的两个数字
def fav_num(num1,num2):
```

```
    print('我喜欢的数字是%d，还有%d。' % (num1,num2))

#第二个函数：打印两个这两个数字之和
def add_num(num1,num2):
    print('这两个数字相加的和为%d。'% (num1 + num2))

#定义主函数，让我们可以一次性调用两个函数
def main(num1,num2):
    fav_num(num1,num2)
    add_num(num1,num2)

main(7,9)
main(4,7)
```

```
我喜欢的数字是7，还有9。
这两个数字相加的和为16。
我喜欢的数字是4，还有7。
这两个数字相加的和为11。
```

这样，我们就通过一个主函数将具有不同功能的函数拼在一起。只需一行代码，就可调用不同的函数。

那么，接下来我们就看看怎么将项目1的部分代码封装成不同的函数，再用主函数打包，最后调用主函数。一段项目1的长代码来啦，看仔细~

```
#项目1的部分代码
import time

#展示角色
player_life = 100
player_attack = 35
enemy_life = 105
enemy_attack = 33

print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
print('------------------------')
time.sleep(1)
print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
print('------------------------')

#双方PK
while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        print('你发起了攻击，【敌人】剩余血量%s'%(enemy_life))
        print('敌人向你发起了攻击，【玩家】的血量剩余%s'%(player_life))
        print('------------------------')
        time.sleep(1.2)
```

```
#打印战果
if player_life > 0 and enemy_life <= 0:
    print('敌人死翘翘了，这局你赢了')
elif player_life <= 0 and enemy_life > 0:
    print('悲催，这局敌人把你干掉了！')
else:
    print('哎呀，这局你和敌人同归于尽了！')
print('-----------------------')
```

```
【玩家】
血量：100
攻击：35
-----------------------
【敌人】
血量：105
攻击：33
-----------------------
你发起了攻击，【敌人】剩余血量70
敌人向你发起了攻击，【玩家】的血量剩余67
-----------------------
你发起了攻击，【敌人】剩余血量35
敌人向你发起了攻击，【玩家】的血量剩余34
-----------------------
你发起了攻击，【敌人】剩余血量0
敌人向你发起了攻击，【玩家】的血量剩余1
-----------------------
敌人死翘翘了，这局你赢了
-----------------------
```

可以看到上面代码，能分成显示【展示角色】、【双方PK】和【打印战果】三部分，根据单个函数实现单个功能的原则，所以我们也可以封装成三个函数，放在主函数里。

第一个【展示角色】的函数show_role刚刚我们已经写好了。现在请你写一写第二个【双方PK】的函数式代码，函数名定为pk_role。

```
def pk_role(player_life,player_attack,enemy_life,enemy_attack):
    while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        time.sleep(1)
        print('你发起了攻击，【敌人】剩余血量%s'%(enemy_life))
        print('敌人向你发起了攻击，【玩家】剩余血量%s'%(player_life))
        print('-----------------------')
```

现在就剩下第三个【打印战果】的函数了，我在跟助教讨论这个项目时，助教告诉我之前有很多同学自然而然地写出了这么一段函数式代码：

```
#打印战果，原来的代码
if player_life > 0 and enemy_life <= 0:
    print('敌人死翘翘了，这局你赢了')
elif player_life <= 0 and enemy_life > 0:
    print('悲催，这局敌人把你干掉了！')
else:
    print('哎呀，这局你和敌人同归于尽了！')
print('-----------------------')

#打印战果，函数式代码
def show_result(player_life,enemy_life):
    if player_life > 0 and enemy_life <= 0:
        print('敌人死翘翘了，这局你赢了')
    elif player_life <= 0 and enemy_life > 0:
        print('悲催，这局敌人把你干掉了！')
    else:
        print('哎呀，这局你和敌人同归于尽了！')
    print('-----------------------')
```

```
敌人死翘翘了，这局你赢了
-----------------------
```

看上去很简单，加了一句def语句就搞定，而且乍一看好像没什么毛病。但其实中间是出了一些差错的。我们暂时把这三个函数合并，并定义了主函数，来找找bug出在哪，你可以运行看看：

```
import time

# 展示角色
def show_role(player_life,player_attack,enemy_life,enemy_attack):
    print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
    print('-----------------------')
    time.sleep(1)
    print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
    print('-----------------------')

# 双方PK
def pk_role(player_life,player_attack,enemy_life,enemy_attack):
    while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        time.sleep(1)
        print('你发起了攻击，【敌人】剩余血量%s'%(enemy_life))
        print('敌人向你发起了攻击，【玩家】剩余血量%s'%(player_life))
```

当你运行代码的时候，你会发现：明明是玩家赢了或电脑赢了，最后的输出结果都是平局。不信，你可以传入不同的参数，多调用几次看看。

如果你已经找到了其中的bug，可以来修改代码。如果你不知道为什么，就往下听老师的讲解哈。

```
import time

# 展示角色
def show_role(player_life,player_attack,enemy_life,enemy_attack):
    print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
    print('------------------------')
    time.sleep(1)
    print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
    print('------------------------')

# 双方PK
def pk_role(player_life,player_attack,enemy_life,enemy_attack):
    while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        time.sleep(1)
        print('你发起了攻击，【敌人】剩余血量%s'%(enemy_life))
        print('敌人向你发起了攻击，【玩家】剩余血量%s'%(player_life))
```

怎么样，你知道谜底了吗？老师也不卖关子了，问题出在打印战果的函数show_result()的参数里。 调用主函数时，player_life和enemy_life传入的参数100，105同样会传入show_result()函数里，也就是show_result(100,105)。

所以做条件判断时，双方的血量都大于0，属于else的情况，所以自然输出的结果是平局。

这里老师提供一个解决方案：将show_result()函数放在pk_role()函数内部，只需要2个小小的操作即可，请关注代码的第20-21行，及最后的主函数注释。

```
import time

# 展示角色
def show_role(player_life,player_attack,enemy_life,enemy_attack):
    print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
    print('------------------------')
    time.sleep(1)
    print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
    print('------------------------')

# 双方PK
def pk_role(player_life,player_attack,enemy_life,enemy_attack):
    while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        time.sleep(1)
        print('你发起了攻击，【敌人】剩余血量'+str(enemy_life))
```

```
            print('敌人向你发起了攻击，【玩家】剩余血量'+str(player_life))
            print('----------------------')
        #把打印战果函数放在PK函数内部
        show_result(player_life,enemy_life)

# 打印战果
def show_result(player_life,enemy_life):
    if player_life > 0 and enemy_life <= 0:
        print('敌人死翘翘了，这局你赢了')
    elif player_life <= 0 and enemy_life > 0:
        print('悲催，这局敌人把你干掉了！')
    else:
        print('哎呀，这局你和敌人同归于尽了！')
    print('----------------------')

#  （主函数）展开战斗全流程
def main(player_life,player_attack,enemy_life,enemy_attack):
    show_role(player_life,player_attack,enemy_life,enemy_attack)
    pk_role(player_life,player_attack,enemy_life,enemy_attack)
    #删除了调用show_role函数的1行代码

main(100,35,105,33)
```

```
【玩家】
血量：100
攻击：35
------------------------
【敌人】
血量：105
攻击：33
------------------------
你发起了攻击，【敌人】剩余血量70
敌人向你发起了攻击，【玩家】剩余血量67
------------------------
你发起了攻击，【敌人】剩余血量35
敌人向你发起了攻击，【玩家】剩余血量34
------------------------
你发起了攻击，【敌人】剩余血量0
敌人向你发起了攻击，【玩家】剩余血量1
------------------------
敌人死翘翘了，这局你赢了
------------------------
```

这个操作相当于两步：

1. 在pk_role()函数内部调用show_result()函数，调用函数的语法是直接写函数名和参数即可，所以只需要加1行代码；
2. 由于调用pk_role()函数就相当于已经调用了show_result()函数，所以主函数中调用show_result()的代码删掉即可。其他什么都不需要更改。

好啦，你再运行一下，现在应该是正常了，这也是我们版本1最终的代码：

看到这里，你可能会想，封装函数还要考虑到这么多，我还不如直接写呢。

请大家明确一点：当项目后面我们要实现的功能越来越复杂，如果不用函数封装独立的功能，你会发现整个代码结构会显得非常凌乱，不易读，且难以修改。这一次的练习是希望在早期就给你打好基础。

而且正如我在上一关所说，函数是减少代码冗余和增加代码复用的有效手段。

举个例子，如果这个小游戏不采用封装函数的写法，当你想多次输入双方的不同的属性运行，你需要每次都要给变量重新赋值，那代码可是又臭又长呀。请敲回车，超长代码预警~

```
import time

# 第一次 PK，给属性变量赋值
player_life = 100
player_attack = 35
enemy_life = 105
enemy_attack = 33

print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
print('------------------------')
time.sleep(1)
print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
print('------------------------')

while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        print('你发起了攻击，【敌人】剩余血量%s'%(enemy_life))
        print('敌人向你发起了攻击，【玩家】的血量剩余%s'%(player_life))
        print('------------------------')
        time.sleep(1.2)

if player_life > 0 and enemy_life <= 0:
    print('敌人死翘翘了，这局你赢了')
elif player_life <= 0 and enemy_life > 0:
    print('悲催，这局敌人把你干掉了！')
else:
    print('哎呀，这局你和敌人同归于尽了！')
print('------------------------')

# 第二次 PK，又给属性变量赋值
player_life = 120
player_attack = 36
enemy_life = 100
enemy_attack = 45
print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
print('------------------------')
time.sleep(1)
print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
```

```
print('-----------------------')

while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        print('你发起了攻击，【敌人】剩余血量%s'%(enemy_life))
        print('敌人向你发起了攻击，【玩家】的血量剩余%s'%(player_life))
        print('-----------------------')
        time.sleep(1.2)

if player_life > 0 and enemy_life <= 0:
    print('敌人死翘翘了，这局你赢了')
elif player_life <= 0 and enemy_life > 0:
    print('悲催，这局敌人把你干掉了！')
else:
    print('哎呀，这局你和敌人同归于尽了！')
print('-----------------------')

# 第三次PK ，又又又给属性变量赋值
player_life = 100
player_attack = 35
enemy_life = 100
enemy_attack = 35
print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
print('-----------------------')
time.sleep(1)
print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
print('-----------------------')

while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        print('你发起了攻击，【敌人】剩余血量%s'%(enemy_life))
        print('敌人向你发起了攻击，【玩家】的血量剩余%s'%(player_life))
        print('-----------------------')
        time.sleep(1.2)

if player_life > 0 and enemy_life <= 0:
    print('敌人死翘翘了，这局你赢了')
elif player_life <= 0 and enemy_life > 0:
    print('悲催，这局敌人把你干掉了！')
else:
    print('哎呀，这局你和敌人同归于尽了！')
print('-----------------------')
```

```
【玩家】
血量：100
攻击：35
-----------------------
【敌人】
```

```
血量：105
攻击：33
----------------------
你发起了攻击，【敌人】剩余血量70
敌人向你发起了攻击，【玩家】的血量剩余67
----------------------
你发起了攻击，【敌人】剩余血量35
敌人向你发起了攻击，【玩家】的血量剩余34
----------------------
你发起了攻击，【敌人】剩余血量0
敌人向你发起了攻击，【玩家】的血量剩余1
----------------------
敌人死翘翘了，这局你赢了
----------------------
【玩家】
血量：120
攻击：36
-----------------------
【敌人】
血量：100
攻击：45
----------------------
你发起了攻击，【敌人】剩余血量64
敌人向你发起了攻击，【玩家】的血量剩余75
----------------------
你发起了攻击，【敌人】剩余血量28
敌人向你发起了攻击，【玩家】的血量剩余30
----------------------
你发起了攻击，【敌人】剩余血量-8
敌人向你发起了攻击，【玩家】的血量剩余-15
----------------------
哎呀，这局你和敌人同归于尽了!
----------------------
【玩家】
血量：100
攻击：35
-----------------------
【敌人】
血量：100
攻击：35
----------------------
你发起了攻击，【敌人】剩余血量65
敌人向你发起了攻击，【玩家】的血量剩余65
----------------------
你发起了攻击，【敌人】剩余血量30
敌人向你发起了攻击，【玩家】的血量剩余30
----------------------
你发起了攻击，【敌人】剩余血量-5
敌人向你发起了攻击，【玩家】的血量剩余-5
----------------------
哎呀，这局你和敌人同归于尽了!
----------------------
```

这其实是同一段代码重复写了三次，因为每次都要重新赋值。而如果用函数封装的话，那只需每次在调用函数的时候传入不同的参数即可，属于“前人栽树，后人乘凉”。(看最后几行)

```
import time

# 展示角色
def show_role(player_life,player_attack,enemy_life,enemy_attack):
    print('【玩家】\n血量：%s\n攻击：%s'%(player_life,player_attack))
    print('------------------------')
    time.sleep(1)
    print('【敌人】\n血量：%s\n攻击：%s'%(enemy_life,enemy_attack))
    print('------------------------')

# 双方PK
def pk_role(player_life,player_attack,enemy_life,enemy_attack):
    while player_life > 0 and enemy_life > 0:
        player_life = player_life - enemy_attack
        enemy_life = enemy_life - player_attack
        time.sleep(1)
        print('你发起了攻击，【敌人】剩余血量'+str(enemy_life))
        print('敌人向你发起了攻击，【玩家】剩余血量'+str(player_life))
        print('------------------------')

    show_result(player_life,enemy_life)

# 打印战果
def show_result(player_life,enemy_life):
    if player_life > 0 and enemy_life <= 0:
        print('敌人死翘翘了，这局你赢了')
    elif player_life <= 0 and enemy_life > 0:
        print('悲催，这局敌人把你干掉了！')
    else:
        print('哎呀，这局你和敌人同归于尽了！')
    print('------------------------')

# （主函数）展开战斗全流程
def main(player_life,player_attack,enemy_life,enemy_attack):
    show_role(player_life,player_attack,enemy_life,enemy_attack)
    pk_role(player_life,player_attack,enemy_life,enemy_attack)

main(100,35,105,33)
main(120,36,100,45)
main(100,35,100,35)
```

【玩家】
血量：100
攻击：35

【敌人】
血量：105
攻击：33

你发起了攻击，【敌人】剩余血量70
敌人向你发起了攻击，【玩家】剩余血量67

你发起了攻击，【敌人】剩余血量35
敌人向你发起了攻击，【玩家】剩余血量34

你发起了攻击，【敌人】剩余血量0
敌人向你发起了攻击，【玩家】剩余血量1

敌人死翘翘了，这局你赢了

【玩家】
血量：120
攻击：36

【敌人】
血量：100
攻击：45

你发起了攻击，【敌人】剩余血量64
敌人向你发起了攻击，【玩家】剩余血量75

你发起了攻击，【敌人】剩余血量28
敌人向你发起了攻击，【玩家】剩余血量30

你发起了攻击，【敌人】剩余血量-8
敌人向你发起了攻击，【玩家】剩余血量-15

哎呀，这局你和敌人同归于尽了！

【玩家】
血量：100
攻击：35

【敌人】
血量：100
攻击：35

你发起了攻击，【敌人】剩余血量65
敌人向你发起了攻击，【玩家】剩余血量65

你发起了攻击，【敌人】剩余血量30
敌人向你发起了攻击，【玩家】剩余血量30

```
你发起了攻击，【敌人】剩余血量-5
敌人向你发起了攻击，【玩家】剩余血量-5
----------------------
哎呀，这局你和敌人同归于尽了！
----------------------
```

相信你能多少体会到函数的作用了吧？首先，它能减少代码冗余，多次复用代码。其次，它能分解流程，使整个代码结构变得清晰，更容易理解、维护和扩展。

好，这就是版本1.0的全部过程了。或许你现在应该休息一下，站起来喝喝水，放放松，再继续学。休息，是为了更高的效率。

你应该发现了：项目2果然是项目2，版本1.0就有一定的难度了。

不过，我们跨出了很重要的一步：从简单的代码，到封装成函数。虽然和最终的目标还差得有点距离。但我们在慢慢完善，慢慢靠近，版本1.0代码结构可以复用到后续版本里。

版本2.0：随机角色，随机属性

在版本2.0，我们的目标是：随机生成游戏的不同角色及属性。具体来说，是在一定数量的角色名单里各为双方随机选出3个角色，并为他们随机生成血量值和攻击值。

所以，我们可以先随机生成角色，再赋予属性。

我们先来分析一下【随机生成角色名称】大概的步骤，再考虑怎么用代码去实现。这其实也是"拆解思维"的一种运用：化整为零，逐个击破。 要做到随机生成三个角色，第一步要做的是：一个存放6个备选角色名的列表。有的选，再考虑怎么选。

好，现在需要设定我方和敌方共12个角色名称，暂时就由我来给出好了，这将是一场正与邪的较量。

```
#不要嫌弃我起的名字中二啊！
player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑客】','【格斗大师】','【枪弹专家】']
enemy_list = ['【暗黑战士】','【黑暗弩手】','【骷髅骑士】','【嗜血刀客】','【首席刺客】','【陷阱之王】']
```

接下来，就是要从这个存放了六个角色的列表里随机抽取三个出来。

说到随机选取，你是不是有种似曾相识的感觉？啊哈，还记得我们项目1的random.randint()函数吗？

我们提到过，random是一个强大的模块，里面装着许多随机函数，说不定其中就有一种函数能帮我们从列表中随机选出几个元素呢！

当然老师可以直接告诉你答案，不过我更想让你知道当碰到问题时，主动寻求解决方案才是王道。

每当这种时候，我们又要运用搜索大法了。我们输入关键词： 果然有的，它的语法是这样子的：

```
import random  #先调用模块
random.sample(seq, n)
#写法是random.sample()
#需要传递两个参数，第一个参数是一个序列（列表、字符串、元组），第二个参数是随机选取的元素个数
```

举个例子，可以多运行几次感受一下random.sample的用法。

```
import random
player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑客】','【格斗大师】','【枪弹专家】']
players = random.sample (player_list,3)
print(players)
#打印出来也是一个列表
```

```
['【枪弹专家】', '【格斗大师】', '【独行剑客】']
```

版本2.0需要实现的两个功能【随机生成角色】和【为角色生成属性】，我们已经完成一半啦。接下来，就是为角色添加随机属性，也就是每个生成的角色都会有对应的血量和攻击值。

所以，我们首先要考虑的问题是：属性和角色要如何对应起来。谈到【对应】，你是否想到了某种数据类型?

标准答案是：键值对应的字典。我们可以以角色名为键，以属性（血量和攻击力）为值，来存放这些数据之间的对应关系。 一个新的问题来了：由于字典的键后面只能跟一个值，那血量和攻击值这两个值要如何安放呢?

接下来，让我们先试试用创建两个字典的方式来实现"随机属性"这个功能吧。

请你先阅读下列代码和注释，代码功能是为玩家角色生成随机属性。

```
import random

# 将需要用到的固定变量一起放到代码开头，便于使用和管理。
player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑客】','【格斗大师】','【枪弹专家】']
players = random.sample(player_list,3)  # 从列表里随机选取三个元素
player_life = {}  # 建立空字典，存放我方角色的血量。
player_attack = {}  # 建立空字典，存放我方角色的攻击。

# 生成角色的属性
```

```
life = random.randint(100,180) # 从100-180随机生成整数，赋值给变量life
attack = random.randint(30,50) # 从30-50随机生成整数，赋值给变量attack
player_life[players[0]] = life # 给空字典player_life添加键值对，角色列表players的第0个元素为键，变量life为值
player_attack[players[0]] = attack # 给空字典player_attack添加键值对，角色列表players的第0个元素为键，变量attack为值
print(player_life)
print(player_attack)
```

```
{'【格斗大师】': 115}
{'【格斗大师】': 50}
```

生成完属性信息，就是要打印出来了。着重看下面第18-19行的代码，你应该还记得如何取出字典里键对应的值吧?

```
import random

player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑客】','【格斗大师】','【枪弹专家】']
players = random.sample(player_list,3)  # 从列表里随机选取三个元素
player_life = {}  # 建立空字典，存放我方角色的血量。
player_attack = {}  # 建立空字典，存放我方角色的攻击。

#循环三次
for i in range(3):
    # 生成角色的属性
    life = random.randint(100,180)
    attack = random.randint(30,50)
    player_life[players[i]] = life # 给空字典添加键值对，角色列表players的第0个元素为键，变量life为值
    player_attack[players[i]] = attack # 给空字典添加键值对，角色列表players的第0个元素为键，变量attack为值

    # 展示我方的角色信息
print('----------------- 角色信息 -----------------')
print('你的人物：')

for i in range(3):
    print('%s  血量：%d  攻击：%d'
    %(players[i],player_life[players[i]],player_attack[players[i]]))
    print('-----------------------------------------')
```

```
----------------- 角色信息 -----------------
你的人物:
【狂血战士】   血量: 180   攻击: 34
-------------------------------------------
【枪弹专家】   血量: 112   攻击: 30
-------------------------------------------
【格斗大师】   血量: 168   攻击: 34
-------------------------------------------
```

好，我方的一个角色设定就完成了。不过一方有三个角色，所以我们要用到循环，由于循环次数确定，我们可以用到for循环和range()函数。

现在请你在下面代码的基础上加入for循环，打印出三个角色的信息吧！(注意修改列表的偏移量)

别忘了今天的主角是函数，所以我们将生成和展示人物属性信息的功能封装成函数就是:

```
import random

player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑
客】','【格斗大师】','【枪弹专家】']
players = random.sample(player_list,3)
player_life = {}
player_attack = {}

# 将这个过程封装成函数
def show_role():
    for i in range(3):
        life = random.randint(100,180)
        attack = random.randint(30,50)
        player_life[players[i]] = life
        player_attack[players[i]] = attack

    # 展示我方的3个角色
    print('----------------- 角色信息 -----------------')
    print('你的人物: ')
    for i in range(3):
        print('%s  血量: %d  攻击: %d'
        %
(players[i],player_life[players[i]],player_attack[players[i]]))
    print('-------------------------------------------')

show_role()
```

```
----------------- 角色信息 -----------------
你的人物:
【格斗大师】   血量: 117   攻击: 30
【森林箭手】   血量: 124   攻击: 35
【枪弹专家】   血量: 130   攻击: 49
-------------------------------------------
```

我们可以直接运行一下，看看目前到了哪一步：

```
import random

player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑
客】','【格斗大师】','【枪弹专家】']
players = random.sample(player_list,3)
player_life = {}
player_attack = {}

# 将这个过程封装成函数
def show_role():
    for i in range(3):
        life = random.randint(100,180)
        attack = random.randint(30,50)
        player_life[players[i]] = life
        player_attack[players[i]] = attack

    # 展示我方的3个角色

    print('----------------- 角色信息 -----------------')
    print('你的人物: ')
```

好，这就是分别创建两个字典来存储血量和攻击力的方法，但是可想而知，如果要加上敌方的信息，就会有四个字典，到时自己可能会把自己弄懵了。

前面的选择题中，我们提到了另一种存储血量和攻击力的方法，就是将血量和攻击力放在一个列表或者元组里，当成是字典的值，比如说player_info = {'【狂血战士】': (135,40)}

而在上一关，我们有提到了在函数内，return返回多个值时，会直接返回一个元组。下面，我会具体展示如何利用返回值在函数间传递信息。

我们先来看一个简化版的例子，你可以跟着下面的代码和注释看一看，并推测每次print()在终端的打印结果。

```
import random

player = ['狂血战士']
#角色列表，有一个元素
```

```
dict1 = {}
#创建一个字典，存放角色属性信息

def info():
    a = random.randint(1,3)
    b = random.randint(4,6)
    return a,b  # return 多个值时，返回一个元组(a,b)

data = info()
#调用info()函数，将返回的元组(a,b)赋值给变量data
print(data)

dict1[player[0]] = data
#往空字典添加键值对，player[0]即'狂血战士'为键，data（元组）为值。
print(dict1)
print(dict1[player[0]]) # 打印出字典的值
print(dict1[player[0]][0]) # 打印出字典的值data的第0个元素
print(dict1[player[0]][1]) # 打印出字典的值data的第1个元素
```

```
(3, 6)
{'狂血战士': (3, 6)}
(3, 6)
3
6
```

不着急，你可以再仔细看两遍代码，最好在本地的编辑器上自己打一遍，把每一步都print()出来。等弄懂后，再继续往下进行。

因为我们是3V3的赛制，所以要生成三个角色的信息，就需用到for循环，同时我们将显示角色属性的功能用函数单独封装起来，就变成了这样子：

```
import random

player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑客】','【格斗大师】','【枪弹专家】']
players = random.sample(player_list,3)
player_info = {}

# 随机生成属性，并用return语句保存属性信息
def born_role():
    life = random.randint(100,180)
    attack = random.randint(30,50)
    return life,attack

# 给角色生成随机属性，并展示角色信息。
def show_role():
    for i in range(3):
```

```
        player_info[players[i]] = born_role()

    # 展示我方的3个角色
    print('----------------- 角色信息 -----------------')
    print('你的人物：')
    for i in range(3):
        print('%s  血量：%d  攻击：%d'
        %(players[i],player_info[players[i]][0],player_info[players[i]
[1]))
    print('--------------------------------------------')

show_role()
```

```
----------------- 角色信息 -----------------
你的人物：
【森林箭手】  血量：106  攻击：46
【光明骑士】  血量：149  攻击：50
【格斗大师】  血量：127  攻击：40
--------------------------------------------
```

一直都是老师在讲讲讲，是不是有点手痒了？现在就请你补充下列代码，根据提示，将敌人的属性信息也打印出来吧。

现在代码已经比较长了，我们将进入【专注模式】进行练习，请点击回车（有困难的话可以试试点"需要帮助"）。

```
# 运行代码即可。
import time,random

player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑
客】','【格斗大师】','【枪弹专家】']
enemy_list = ['【暗黑战士】','【黑暗弩手】','【暗夜骑士】','【嗜血刀客】','【首
席刺客】','【陷阱之王】']
players = random.sample(player_list,3)
enemies = random.sample(enemy_list,3)
player_info = {}
enemy_info = {}

# 随机生成两种属性
def born_role():
    life = random.randint(100,180)
    attack = random.randint(30,50)
    return life,attack  # return 多个元素时，返回一个元组（昨天课堂有讲）

# 给角色生成随机属性，并展示角色信息。
def show_role():
    for i in range(3):
```

```
        player_info[players[i]] = born_role()
        enemy_info[enemies[i]] = born_role()

    # 展示我方的3个角色
    print('----------------- 角色信息 -----------------')
    print('你的人物：')
    for i in range(3):
        print('%s  血量：%d  攻击：%d'
        %(players[i],player_info[players[i]][0],player_info[players[i]]
[1]))
    print('----------------------------------------------')
    print('电脑敌人：')

    # 展示敌方的3个角色
    for i in range(3):
        print('%s  血量：%d  攻击：%d'
        %(enemies[i],enemy_info[enemies[i]][0],enemy_info[enemies[i]]
[1]))
    print('----------------------------------------------')

show_role()
```

```
----------------- 角色信息 -----------------
你的人物：
【森林箭手】   血量：141  攻击：36
【独行剑客】   血量：161  攻击：49
【枪弹专家】   血量：117  攻击：33
----------------------------------------------
电脑敌人：
【嗜血刀客】   血量：105  攻击：42
【暗夜骑士】   血量：165  攻击：45
【陷阱之王】   血量：113  攻击：34
----------------------------------------------
```

在，2.0的两个功能【随机生成角色】和【为角色生成属性】都已经完成了。如果你清晰地明白上面的代码每一行为什么要那么写，那么，可以说你已经攻克了版本2.0了。

在正式开始挑战版本3.0之前，我们稍稍回顾一下，前2个版本，我们都应用到了哪些知识： 我们离项目大目标，你开始看到的那个游戏动图，越来越近了！这个时候，更需要你的专注。所以，请你再次评估一下自己的状态。

欲速则不达，现在请给自己5分钟或更长的放松时间。然后我们接着往下看版本3.0！

版本3.0：询问玩家出场顺序

欢迎状态回归的你，一起继续向版本3.0进发！

版本3.0是要在版本2.0的基础上，增加两大功能：1.分别询问玩家每个角色的出场顺序，根据玩家的排序来战斗 2. 开始3轮战斗，并输出单轮和最终结果。

首先我们来看如何询问玩家顺序，由于要让玩家自由选择顺序，所以要用到input()函数，依次询问，我直接给出代码，请你运行试试。

```
import random
player_list = ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑客】','【格斗大师】','【枪弹专家】']
players = random.sample(player_list,3)
print(players)
for i in range(3):
  order = input('你的'+players[i]+'要第几个上场？（请输入数字1,2,3)')
```

```
['【光明骑士】', '【独行剑客】', '【枪弹专家】']
你的【光明骑士】要第几个上场？（请输入数字1,2,3)1
你的【独行剑客】要第几个上场？（请输入数字1,2,3)3
你的【枪弹专家】要第几个上场？（请输入数字1,2,3)4
```

这里有个难点是，假设随机生成的角色列表是('【森林箭手】','【狂血战士】','【格斗大师】')，然后我依次输入了3，2，1的出场顺序，我怎么把列表的顺序调整成('【格斗大师】','【狂血战士】','【森林箭手】')呢？ 下面，为了讲清楚这一个难点，我们来把问题稍稍简化，现在list1 = ('A','B','C')，通过3次input：3、2、1，将排序变为list1=('C','B','A')，并打印出来。

不过老师也不想抹杀让你思考解决方法的机会，现在请你试一试能否实现这个效果，说不定你会想到比我更聪明的解法呢？如果实在想不出或想直接听老师讲解，可以在报错后点击跳过本题。

```
#阅读代码后，点击运行，依次输入3，2，1
list1=['A','B','C']
dict1={}
for i in range(3):
    order = int(input('你要把'+list1[i]+'放在第几位？（请输入数字1,2,3)'))
    dict1[order]=list1[i]
print(dict1)
```

```
你要把A放在第几位？（请输入数字1,2,3)3
你要把B放在第几位？（请输入数字1,2,3)2
你要把C放在第几位？（请输入数字1,2,3)1
{3: 'A', 2: 'B', 1: 'C'}
```

怎么样呢？如果能想出来真的很棒！如果不懂也没关系，确实有点难。下面老师会讲讲我的思路。

首先因为输入的顺序和列表中的元素存在一个对应的关系，所以可以新建一个字典来存储，顺序为键，元素为值，我们希望最终的格式应该为dict1 = {3:'A',2:'B',1:'C'}

可以看到dict1的键是输入的顺序，也就是变量order，值是列表list1的元素。往空字典添加键值对写成代码

好，现在我们已经得到了dict1 = {3:'A',2:'B',1:'C'}。下一步我们就是要提取dict1的值，按键的顺序组成新的列表list1 = ('C','B','A')

还记得怎么取出字典里的值，以及用append()函数往列表添加元素吧？请仔细阅读下面的代码，尤其是8-10行，然后点击运行：

```
dict1 = {3:'A',2:'B',1:'C'}
print(dict1[1])
print(dict1[2])
print(dict1[3])

list1 = []
#新建一个列表
list1.append(dict1[1])
list1.append(dict1[2])
list1.append(dict1[3])
#依次往list1新增元素：'C','B','A'
print(list1)
```

```
C
B
A
['C', 'B', 'A']
```

你留意到了吗？上面的8-10行是一个重复的结构，且dict1()方括号里的数字是从1递增到3，所以我们可以用for循环将上述代码写成：

```
dict1 = {3:'A',2:'B',1:'C'}
list1 = []

for i in range(1,4):
# i从1到3遍历
    list1.append(dict1[i])
```

所以将以上两个步骤的代码整合在一起，我们的排序代码终于呼之欲出了！

```
list1 = ['A','B','C']
dict1 = {}
for i in range(3):
    order = int(input('你要把'+list1[i]+'放在第几位？（请输入数字1,2,3)'))
    dict1[order] = list1[i]
print(dict1)

list1 = []
# 清空原本列表list1的元素
for i in range(1,4):
    list1.append(dict1[i])
print(list1)
```

```
你要把A放在第几位？（请输入数字1,2,3)3
你要把B放在第几位？（请输入数字1,2,3)2
你要把C放在第几位？（请输入数字1,2,3)1
{3: 'A', 2: 'B', 1: 'C'}
['C', 'B', 'A']
```

可以看到这里灵活地使用了我们之前的已学知识，现在就请你把代码复现出来吧，只有自己打代码才能真正理解。

同理，现在我们看回我们的小游戏，让玩家自行输入出场顺序的代码就是：

```
players = ['【狂血战士】','【森林箭手】','【光明骑士】']

order_dict = {}
# 新建字典，存储顺序
for i in range(3):
    order = int(input('你想将 %s 放在第几个上场？(输入数字1~3)' %
players[i]))
    order_dict[order] = players[i]

players = []
for i in range(1,4):
```

```
    players.append(order_dict[i])

print('\n我方角色的出场顺序是：%s、%s、%s' %
(players[0],players[1],players[2]))
```

```
你想将 【狂血战士】 放在第几个上场？(输入数字1~3)3
你想将 【森林箭手】 放在第几个上场？(输入数字1~3)2
你想将 【光明骑士】 放在第几个上场？(输入数字1~3)1

我方角色的出场顺序是：【光明骑士】、【森林箭手】、【狂血战士】
```

下一步，便是将以上代码封装成一个函数并入之前的代码中。不过在这之前，有个细节要提前说一下，和我们上一关所说的函数的作用域有关。

下面我们来对比下两个版本的代码，请阅读代码后点击运行。先来看代码1：

```
# 代码1
list1 = ['A','B','C']
#这里的list1 是全局变量，可以被其他函数使用

def order1():
    list1 = ['C', 'B', 'A']
   # orde1()里的list1是局部变量，其他函数无法使用

def use1():
    print(list1)
# 这里打印的list1是最开头的全局变量

order1()
use1()
# 打印出['A','B','C']
```

```
['A', 'B', 'C']
```

```
# 代码2
list2 = ['A','B','C']
def order2():
    global list2
    # 将list2定义为全局变量，这样后面的函数调用时，会使用这个变量list2。
    list2 = ['C', 'B', 'A']
def use2():
    print(list2)
    # 这里打印的list2是order2()里的list2
order2()
use2()
# 打印出['C', 'B', 'A']
```

```
['C', 'B', 'A']
```

因为我们要新封装一个有排序功能的函数，而排序后的新列表，会要被后面战斗过程函数所用。所以，在定义排序函数时，要记得players为全局变量，也就是在前面加上globle。

现在，我们可以正式将排序功能封装成一个函数。

```
# 排序功能，未封装函数前的代码
players = ['【狂血战士】','【森林箭手】','【光明骑士】']

order_dict = {}
for i in range(3):
    order = int(input('你想将 %s 放在第几个上场？（输入数字1~3）' %
players[i]))
    order_dict[order] = players[i]

players = []
for i in range(1,4):
    players.append(order_dict[i])

print('\n我方角色的出场顺序是：%s、%s、%s' %
(players[0],players[1],players[2]))

# 排序功能，封装成函数后(未调用)
players = ['【狂血战士】','【森林箭手】','【光明骑士】']

def order_role():
    global players  #记得global一下
    order_dict = {}
    for i in range(3):
        order = int(input('你想将 %s 放在第几个上场？（输入数字1~3）'%
(players[i])))
        order_dict[order] = players[i]

    players = []
```

```
    for i in range(1,4):
        players.append(order_dict[i])

    print('\n我方角色的出场顺序是：%s、%s、%s' %
(players[0],players[1],players[2]))
```

```
你想将 【狂血战士】 放在第几个上场？(输入数字1~3)3
你想将 【森林箭手】 放在第几个上场？(输入数字1~3)1
你想将 【光明骑士】 放在第几个上场？(输入数字1~3)2

我方角色的出场顺序是：【森林箭手】、【光明骑士】、【狂血战士】
```

```
import random

# 将需要的数据和固定变量放在开头
player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑
客】','【格斗大师】','【枪弹专家】']
enemy_list = ['【暗黑战士】','【黑暗弩手】','【暗夜骑士】','【嗜血刀客】','【首
席刺客】','【陷阱之王】']
players = random.sample(player_list,3)
enemies = random.sample(enemy_list,3)
player_info = {}
enemy_info = {}

# 随机生成角色的属性
def born_role():
    life = random.randint(100,180)
    attack = random.randint(30,50)
    return life,attack

# 生成和展示角色信息
def show_role():
    for i in range(3):
        player_info[players[i]] = born_role()
        enemy_info[enemies[i]] = born_role()

    # 展示我方的3个角色
    print('----------------- 角色信息 -----------------')
    print('你的人物：')
    for i in range(3):
        print('%s  血量：%d  攻击：%d'
        %(players[i],player_info[players[i]][0],player_info[players[i]]
[1]))
    print('---------------------------------------------')
    print('电脑敌人：')
```

```
    # 展示敌方的3个角色
    for i in range(3):
        print('%s  血量：%d  攻击：%d'
        %(enemies[i],enemy_info[enemies[i]][0],enemy_info[enemies[i]]
[1]))
    print('---------------------------------------------')
    input('请按回车键继续。\n')  # 为了让玩家更有控制感，可以插入类似的代码来切分
游戏进程。

# 角色排序，选择出场顺序。
def order_role():
    global players
    order_dict = {}
    for i in range(3):
        order = int(input('你想将 %s 放在第几个上场？(输入数字1~3)'%
(players[i])))
        order_dict[order] = players[i]

    players = []
    for i in range(1,4):
        players.append(order_dict[i])

    print('\n我方角色的出场顺序是：%s、%s、%s' %
(players[0],players[1],players[2]))
    print('敌方角色的出场顺序是：%s、%s、%s' %
(enemies[0],enemies[1],enemies[2]))

def main():
    show_role()
    order_role()

main()
```

```
----------------- 角色信息 -----------------
你的人物：
【格斗大师】   血量：161   攻击：38
【光明骑士】   血量：170   攻击：45
【独行剑客】   血量：153   攻击：35
---------------------------------------------
电脑敌人：
【黑暗弩手】   血量：142   攻击：39
【首席刺客】   血量：124   攻击：42
【暗黑战士】   血量：106   攻击：46
---------------------------------------------
请按回车键继续。
3
你想将 【格斗大师】 放在第几个上场？(输入数字1~3)2
你想将 【光明骑士】 放在第几个上场？(输入数字1~3)1
你想将 【独行剑客】 放在第几个上场？(输入数字1~3)3
```

```
我方角色的出场顺序是：【光明骑士】、【格斗大师】、【独行剑客】
敌方角色的出场顺序是：【黑暗弩手】、【首席刺客】、【暗黑战士】
```

注意37行，我加了一行input('请按回车键继续。\n')，正如注释所说，这是为了让玩家更有控制感，可以插入类似的代码来切分游戏进程。待会你试试就知道啦。

那么现在我们就把最难啃的骨头——排序函数给搞定了，现在需要你做的就是点击运行，感受我们目前的成果吧！

怎么样？真的不需要休息吗？到这里的你，真的太厉害了。

接下来是最后一个挑战：

版本4.0：3V3战斗，输出战果

至此，我们只要再加上"角色PK"的过程，同时展示战果，这个游戏便完成了。

这个过程，其实和项目1的最终代码很类似。互相PK，三局两胜。我们可以将其拆解为3部分：角色PK、打印单局战果和打印最终结果。

先来回顾一下项目1的代码：

```
player_victory = 0
enemy_victory = 0

while player_life > 0 and enemy_life > 0:
    player_life = player_life - enemy_attack
    enemy_life = enemy_life - player_attack
    print('你发起了攻击，【玩家】剩余血量%s' % player_life)
    print('敌人向你发起了攻击，【敌人】的血量剩余%s' % enemy_life)
    print('----------------------')
    time.sleep(1.2)

if player_life > 0 and enemy_life <= 0:
    player_victory += 1
    print('敌人死翘翘了，你赢了！')
elif player_life <= 0 and enemy_life > 0:
    enemy_victory += 1
    print('悲催，敌人把你干掉了！')
else:
    print('哎呀，你和敌人同归于尽了！')

if player_victory > enemy_victory :
    time.sleep(1)
    print('\n【最终结果：你赢了！】')
elif enemy_victory > player_victory:
    print('\n【最终结果：你输了！】')
else:
    print('\n【最终结果：平局！】')
```

之前最终结果的判断是靠双方胜利局数player_victory和enemy_victory的比较。

这里老师会提供一种更简单的思路，设置一个变量为score = 0，玩家赢了加一分，输了扣一分，平局加零分。最终只要判断score是大于、小于或等于0就可以了。 因为战斗过程的总体逻辑和我们在项目1所说是相似的，所以我会直接给出代码，你只需看懂就可以啦～

【前方高能预警】代码虽长，但其实不难理解，千万别慌！！！

```
# 为专注于pk_role()函数，将前面代码做了简化，是我们前面已经实现的效果（默认你已经掌握了）。
# 注意代码的缩进、循环和判断条件的设定。

import time
#PK过程，还是有暂停效果比较好!

players = ['【狂血战士】','【森林箭手】','【光明骑士】']
enemies = ['【暗黑战士】','【黑暗弩手】','【暗夜骑士】']
player_info = {'【狂血战士】':(105,35),'【森林箭手】':(105,35),'【光明骑士】':(105,35)}
enemy_info = {'【暗黑战士】':(105,35),'【黑暗弩手】':(105,35),'【暗夜骑士】':(105,35)}
input('按回车开始简化版游戏：')

# 角色PK
def pk_role():
    round = 1  # 注：round表示局数。
    score = 0
    for i in range(3):  # 一共要打三局 i依次为0,1,2
        player_name = players[i]  # 统一将下面会用到的数据先赋值给变量会更清晰更好管理
        # 提取玩家角色名称
        enemy_name = enemies[i]
        player_life = player_info[players[i]][0]
        #玩家血量是字典里的值（元组）的第0个元素，以下同理
        player_attack = player_info[players[i]][1]
        enemy_life = enemy_info[enemies[i]][0]
        enemy_attack = enemy_info[enemies[i]][1]

        # 每一局开战前展示战斗信息
        print('\n----------------- 【第%d局】 -----------------' % round)
        print('玩家角色：%s vs 敌方角色：%s ' % (player_name,enemy_name))
        print('%s 血量：%d  攻击：%d' % (player_name,player_life,player_attack))
        print('%s 血量：%d  攻击：%d' % (enemy_name,enemy_life,enemy_attack))
        print('-----------------------------------------')
        input('请按回车键继续。\n')
```

```
        # 双方血量都大于零，战斗过程会一直然后互扣血量。
        while player_life > 0 and enemy_life > 0:
            enemy_life = enemy_life - player_attack
            player_life = player_life - enemy_attack
            print('%s发起了攻击，%s剩余血量%d' %
(player_name,enemy_name,enemy_life))
            print('%s发起了攻击，%s剩余血量%d'%
(enemy_name,player_name,player_life))
            print('----------------------------------------------')
            time.sleep(1)
        else:  # 每局的战果展示，以及对分数score和局数round的影响。
            if player_life > 0 and enemy_life <= 0:
                print('\n敌人死翘翘了，你赢了！')
                score += 1  # 分数变化（1）
            elif player_life <= 0 and enemy_life > 0:
                print('\n悲催，敌人把你干掉了！')
                score += -1  # 分数变化（2）
                # 等价于score = score - 1
            else :
                print('\n哎呀，你和敌人同归于尽了！')
                score += 0  # 分数变化（3）
                #这行不写也不影响
            round += 1  # 局数+1，直到循环完3局。
    input('\n点击回车，查看比赛的最终结果\n')
    if score > 0:
        print('【最终结果：你赢了！】\n')
    elif score < 0:
        print('【最终结果：你输了！】\n')
    else:
        print('【最终结果：平局！】\n')

pk_role()
```

如果你能看懂上面的代码，老师为你激情点赞！

从功能上来看，可以说我们已经完成双方角色3V3战斗部分的代码。不过，还有一些可以优化的空间。

我们之前说过要保持函数的功能单一性，即每个功能尽量都封装成单独的函数，而pk_role()函数包含了战斗过程和战果统计两个功能。

所以，可以考虑让统计单局战果的代码独立成一个函数，在pk_role()函数内部调用即可。

需要注意的是：有一些函数不会包含到主函数里，而是作为"辅助型"函数被其他函数调用，如一些返回数值的函数。这其实和之前在show_role()函数内部调用born_role()函数有些类似。

所以，上面的代码可以改造为下列这个版本，可以着重看新增的show_result()函数，以及该函数如何在pk_role()函数内部被调用。

```
import time

players = ['【狂血战士】','【森林箭手】','【光明骑士】']
enemies = ['【暗黑战士】','【黑暗弩手】','【暗夜骑士】']
player_info = {'【狂血战士】':(105,35),'【森林箭手】':(105,35),'【光明骑
士】':(105,35)}
enemy_info = {'【暗黑战士】':(105,35),'【黑暗弩手】':(105,35),'【暗夜骑士】':
(105,35)}
input('按回车开始简化版游戏：')

# 角色PK
def pk_role():
    round = 1
    score = 0
    for i in range(3):  # 一共要打三局
        player_name = players[i]
        enemy_name = enemies[i]
        player_life = player_info[players[i]][0]
        player_attack = player_info[players[i]][1]
        enemy_life = enemy_info[enemies[i]][0]
        enemy_attack = enemy_info[enemies[i]][1]
        # 每一局开战前展示战斗信息
        print('\n----------------- 【第%d局】 -----------------' %
round)
        print('玩家角色：%s vs 敌方角色：%s ' %(player_name,enemy_name))
        print('%s 血量：%d  攻击：%d' %
(player_name,player_life,player_attack))
        print('%s 血量：%d  攻击：%d' %
(enemy_name,enemy_life,enemy_attack))
        print('-----------------------------------------')
        input('请按回车键继续。\n')
        # 开始判断血量是否都大于零，然后互扣血量。
        while player_life > 0 and enemy_life > 0:
            enemy_life = enemy_life - player_attack
            player_life = player_life - enemy_attack
            print('%s发起了攻击，%s剩余血量%d'%
(player_name,enemy_name,enemy_life))
            print('%s发起了攻击，%s剩余血量%d'%
(enemy_name,player_name,player_life))
            print('-----------------------------------------')
            time.sleep(1)
        else:  # 每局的战果展示，以及分数score和局数的变化。
            print(show_result(player_life,enemy_life)[1])
            # 调用show_result()函数，打印返回元组中的第一个元素result。
            score += int(show_result(player_life,enemy_life)[0])
            # 调用show_result()函数，完成计分变动。
            round += 1
    input('\n点击回车，查看比赛的最终结果\n')
    if score > 0:
        print('【最终结果：你赢了！】\n')
    elif score < 0:
        print('【最终结果：你输了！】\n')
```

```
    else:
        print('【最终结果：平局！】\n')

# 返回单局战果和计分法所加分数。
def show_result(player_life,enemy_life):  # 注意：该函数要设定参数，才能判断单局战果。
    if player_life > 0 and enemy_life <= 0:
        result = '\n敌人死翘翘了，你赢了！'
        return 1,result  # 返回元组(1,'\n敌人死翘翘了，你赢了！')
    elif player_life <= 0 and enemy_life > 0:
        result = '\n悲催，敌人把你干掉了！'
        return -1,result
    else :
        result = '\n哎呀，你和敌人同归于尽了！'
        return 0,result

pk_role()
```

按回车开始简化版游戏：

```
----------------- 【第1局】 -----------------
玩家角色：【狂血战士】 vs 敌方角色：【暗黑战士】
【狂血战士】 血量：105  攻击：35
【暗黑战士】 血量：105  攻击：35
---------------------------------------------
请按回车键继续。

【狂血战士】发起了攻击，【暗黑战士】剩余血量70
【暗黑战士】发起了攻击，【狂血战士】剩余血量70
---------------------------------------------
【狂血战士】发起了攻击，【暗黑战士】剩余血量35
【暗黑战士】发起了攻击，【狂血战士】剩余血量35
---------------------------------------------
【狂血战士】发起了攻击，【暗黑战士】剩余血量0
【暗黑战士】发起了攻击，【狂血战士】剩余血量0
---------------------------------------------

哎呀，你和敌人同归于尽了！

----------------- 【第2局】 -----------------
玩家角色：【森林箭手】 vs 敌方角色：【黑暗弩手】
【森林箭手】 血量：105  攻击：35
【黑暗弩手】 血量：105  攻击：35
---------------------------------------------
请按回车键继续。

【森林箭手】发起了攻击，【黑暗弩手】剩余血量70
【黑暗弩手】发起了攻击，【森林箭手】剩余血量70
---------------------------------------------
【森林箭手】发起了攻击，【黑暗弩手】剩余血量35
```

```
【黑暗弩手】发起了攻击，【森林箭手】剩余血量35
------------------------------------------
【森林箭手】发起了攻击，【黑暗弩手】剩余血量0
【黑暗弩手】发起了攻击，【森林箭手】剩余血量0
------------------------------------------

哎呀，你和敌人同归于尽了！

----------------- 【第3局】 -----------------
玩家角色：【光明骑士】 vs 敌方角色：【暗夜骑士】
【光明骑士】 血量：105  攻击：35
【暗夜骑士】 血量：105  攻击：35
------------------------------------------
请按回车键继续。

【光明骑士】发起了攻击，【暗夜骑士】剩余血量70
【暗夜骑士】发起了攻击，【光明骑士】剩余血量70
------------------------------------------
【光明骑士】发起了攻击，【暗夜骑士】剩余血量35
【暗夜骑士】发起了攻击，【光明骑士】剩余血量35
------------------------------------------
【光明骑士】发起了攻击，【暗夜骑士】剩余血量0
【暗夜骑士】发起了攻击，【光明骑士】剩余血量0
------------------------------------------

哎呀，你和敌人同归于尽了！

点击回车，查看比赛的最终结果

【最终结果：平局！】
```

回过头看，我们已经一步步写出了游戏需要的所有代码，定义了5个函数：born_role()/show_role()/order_role()/pk_role()/show_result()，让游戏具备了开头所见动图的所有功能。

如果你学有余力，想挑战一下自己，可以从零开始，尝试把完整的代码独立地码出来。如果觉得对游戏中某个功能对应的函数还不清楚的话，可以先回头再看一看，然后再来挑战。

活动活动身体，放松放松大脑，拉伸拉伸手指。如果哪里卡住了，边思考边用代码来验证自己的想法。实在卡得比较久，可往查看前面的内容。卡点解决后，建议做下笔记，方便之后温习。

对了。一个重要的提醒：这个大魔王级的项目，请务必先在本地编辑器完成（推荐Visual Studio Code和Pycharm），再复制过来提交。编辑器会帮你节省很多功夫（自动补全函数和变量名等）。

不过，对于一个刚涉猎Python的新手来说，能跟上老师的节奏，读懂每行代码已经相当不错了。所以如果你觉得有点累了，可以直接点击运行选择【跳过本题】。

```
import time,random

# 需要的数据和变量放在开头
player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑
客】','【格斗大师】','【枪弹专家】']
enemy_list = ['【暗黑战士】','【黑暗弩手】','【暗夜骑士】','【嗜血刀客】','【首
席刺客】','【陷阱之王】']
players = random.sample(player_list,3)
enemies = random.sample(enemy_list,3)
player_info = {}
enemy_info = {}

# 随机生成角色的属性
def born_role():
    life = random.randint(100,180)
    attack = random.randint(30,50)
    return life,attack

# 生成和展示角色信息
def show_role():
    for i in range(3):
        player_info[players[i]] = born_role()
        enemy_info[enemies[i]] = born_role()

    # 展示我方的3个角色
    print('------------------ 角色信息 ------------------')
    print('你的人物：')
    for i in range(3):
        print('%s  血量：%d  攻击：%d'
        %(players[i],player_info[players[i]][0],player_info[players[i]]
[1]))
    print('----------------------------------------------')
    print('电脑敌人：')

    # 展示敌方的3个角色
    for i in range(3):
        print('%s  血量：%d  攻击：%d'
        %(enemies[i],enemy_info[enemies[i]][0],enemy_info[enemies[i]]
[1]))
    print('----------------------------------------------')
    input('请按回车键继续。\n')  # 为了让玩家更有控制感，可以插入类似的代码来切分
游戏进程。

# 角色排序，选择出场顺序。
def order_role():
    global players
    order_dict = {}
    for i in range(3):
        order = int(input('你想将 %s 放在第几个上场？（输入数字1~3）'%
players[i]))
```

```
        order_dict[order] = players[i]

    players = []
    for i in range(1,4):
        players.append(order_dict[i])

    print('\n我方角色的出场顺序是：%s、%s、%s' %
(players[0],players[1],players[2]))
    print('敌方角色的出场顺序是：%s、%s、%s' %
(enemies[0],enemies[1],enemies[2]))

# 角色PK
def pk_role():
    round = 1
    score = 0
    for i in range(3):  # 一共要打三局
        player_name = players[i]
        enemy_name = enemies[i]
        player_life = player_info[players[i]][0]
        player_attack = player_info[players[i]][1]
        enemy_life = enemy_info[enemies[i]][0]
        enemy_attack = enemy_info[enemies[i]][1]

        # 每一局开战前展示战斗信息
        print('\n----------------- 【第%d局】 -----------------' %
round)
        print('玩家角色：%s vs 敌方角色：%s ' %(player_name,enemy_name))
        print('%s 血量：%d  攻击：%d' %
(player_name,player_life,player_attack))
        print('%s 血量：%d  攻击：%d' %
(enemy_name,enemy_life,enemy_attack))
        print('-----------------------------------------------')
        input('请按回车键继续。\n')

        # 开始判断血量是否都大于零，然后互扣血量。
        while player_life > 0 and enemy_life > 0:
            enemy_life = enemy_life - player_attack
            player_life = player_life - enemy_attack
            print('%s发起了攻击，%s剩余血量%d' %
(player_name,enemy_name,enemy_life))
            print('%s发起了攻击，%s剩余血量%d' %
(enemy_name,player_name,player_life))
            print('-----------------------------------------------')
            time.sleep(1)
        else:  # 每局的战果展示，以及分数score和局数的变化。
            # 调用show_result()函数，打印返回元组中的result。
            print(show_result(player_life,enemy_life)[1])
            # 调用show_result()函数，完成计分变动。
            score += int(show_result(player_life,enemy_life)[0])
            round += 1
    input('\n点击回车，查看比赛的最终结果\n')
```

```
    if score > 0:
        print('【最终结果：你赢了！】\n')
    elif score < 0:
        print('【最终结果：你输了！】\n')
    else:
        print('【最终结果：平局！】\n')

# 返回单局战果和计分法所加分数。
def show_result(player_life,enemy_life):  # 注意：该函数要设定参数，才能判断单局战果。
    if player_life > 0 and enemy_life <= 0:
        result = '\n敌人死翘翘了，你赢了！'
        return 1,result  # 返回元组(1,'\n敌人死翘翘了，你赢了！')，类似角色属性的传递。
    elif player_life <= 0 and enemy_life > 0:
        result = '\n悲催，敌人把你干掉了！'
        return -1,result
    else :
        result = '\n哎呀，你和敌人同归于尽了！'
        return 0,result

# （主函数）展开战斗流程
def main():
    show_role()
    order_role()
    pk_role()

# 启动程序（即调用主函数）
main()
```

```
----------------- 角色信息 -----------------
你的人物：
【格斗大师】   血量：101   攻击：46
【光明骑士】   血量：121   攻击：34
【狂血战士】   血量：180   攻击：37
-------------------------------------------
电脑敌人：
【暗夜骑士】   血量：180   攻击：49
【暗黑战士】   血量：103   攻击：50
【首席刺客】   血量：119   攻击：49
-------------------------------------------
请按回车键继续。

你想将 【格斗大师】 放在第几个上场？(输入数字1~3)3
你想将 【光明骑士】 放在第几个上场？(输入数字1~3)1
你想将 【狂血战士】 放在第几个上场？(输入数字1~3)2

我方角色的出场顺序是：【光明骑士】、【狂血战士】、【格斗大师】
敌方角色的出场顺序是：【暗夜骑士】、【暗黑战士】、【首席刺客】
```

```
----------------- 【第1局】 -----------------
玩家角色：【光明骑士】 vs 敌方角色：【暗夜骑士】
【光明骑士】 血量：121  攻击：34
【暗夜骑士】 血量：180  攻击：49
-------------------------------------------
请按回车键继续。

【光明骑士】发起了攻击，【暗夜骑士】剩余血量146
【暗夜骑士】发起了攻击，【光明骑士】剩余血量72
-------------------------------------------
【光明骑士】发起了攻击，【暗夜骑士】剩余血量112
【暗夜骑士】发起了攻击，【光明骑士】剩余血量23
-------------------------------------------
【光明骑士】发起了攻击，【暗夜骑士】剩余血量78
【暗夜骑士】发起了攻击，【光明骑士】剩余血量-26
-------------------------------------------

悲催，敌人把你干掉了！

----------------- 【第2局】 -----------------
玩家角色：【狂血战士】 vs 敌方角色：【暗黑战士】
【狂血战士】 血量：180  攻击：37
【暗黑战士】 血量：103  攻击：50
-------------------------------------------
请按回车键继续。

【狂血战士】发起了攻击，【暗黑战士】剩余血量66
【暗黑战士】发起了攻击，【狂血战士】剩余血量130
-------------------------------------------
【狂血战士】发起了攻击，【暗黑战士】剩余血量29
【暗黑战士】发起了攻击，【狂血战士】剩余血量80
-------------------------------------------
【狂血战士】发起了攻击，【暗黑战士】剩余血量-8
【暗黑战士】发起了攻击，【狂血战士】剩余血量30
-------------------------------------------

敌人死翘翘了，你赢了！

----------------- 【第3局】 -----------------
玩家角色：【格斗大师】 vs 敌方角色：【首席刺客】
【格斗大师】 血量：101  攻击：46
【首席刺客】 血量：119  攻击：49
-------------------------------------------
请按回车键继续。

【格斗大师】发起了攻击，【首席刺客】剩余血量73
【首席刺客】发起了攻击，【格斗大师】剩余血量52
-------------------------------------------
【格斗大师】发起了攻击，【首席刺客】剩余血量27
【首席刺客】发起了攻击，【格斗大师】剩余血量3
-------------------------------------------
【格斗大师】发起了攻击，【首席刺客】剩余血量-19
```

```
【首席刺客】发起了攻击，【格斗大师】剩余血量-46
----------------------------------------

哎呀，你和敌人同归于尽了！

点击回车，查看比赛的最终结果

【最终结果：平局！】
```

作业

```python
import random

# 出拳
punches = ['石头','剪刀','布']
computer_choice = random.choice(punches)
user_choice = ''
user_choice = input('请出拳：（石头、剪刀、布）')  # 请用户输入选择
while user_choice not in punches:  # 当用户输入错误，提示错误，重新输入
    print('输入有误，请重新出拳')
    user_choice = input()

# 亮拳
print('———战斗过程———')
print('电脑出了：%s' % computer_choice)
print('你出了：%s' % user_choice)

# 胜负

print('————结果————')
if user_choice == computer_choice:  # 使用if进行条件判断
    print('平局！')
elif (user_choice == '石头' and computer_choice == '剪刀') or
(user_choice == '剪刀' and computer_choice == '布') or (user_choice ==
'布' and computer_choice == '石头'):
    print('你赢了！')
else:
    print('你输了！')
```

```
请出拳：（石头、剪刀、布）石头
———战斗过程———
电脑出了：布
你出了：石头
————结果————
你输了！
```

```python
import random

# 出拳
punches = ['石头','剪刀','布']
computer_choice = random.choice(punches)
user_choice = ''
user_choice = input('请出拳：（石头、剪刀、布）')  # 请用户输入选择
while user_choice not in punches:  # 当用户输入错误，提示错误，重新输入
    print('输入有误，请重新出拳')
    user_choice = input()

# 亮拳
print('———战斗过程———')
print('电脑出了：%s' % computer_choice)
print('你出了：%s' % user_choice)


# 胜负
print('————结果————')
if user_choice == computer_choice:  # 使用if进行条件判断
    print('平局！')
# 电脑的选择有3种，索引位置分别是：0石头、1剪刀、2布。
# 假设在电脑索引位置上减1，对应：-1布，0石头，1剪刀，皆胜。
elif user_choice == punches[punches.index(computer_choice)-1]:
    print('你赢了！')
else:
    print('你输了！')
```

```
请出拳：（石头、剪刀、布）石头
———战斗过程———
电脑出了：布
你出了：石头
————结果————
你输了！
```

INDEX()函数

index() 函数用于找出列表中某个元素第一次出现的索引位置。 语法为：list.index(obj)，obj为object（对象）的缩写。 具体可参考右侧的代码和运行结果。

第11关 BUG

老师将bug分成了四种类型： ![%E5%9B%BE%E7%89%87.png](attachment: 一个小提醒：因为希望让大家能享受独立抓bug的乐趣，所以这一关的代码题基本没有提示，如果实在找不到可以先点击【跳过本题】，再查看老师讲解。

BUG 1：粗心

```
a = input('请输入密码：')
if a == '123456'
    print('通过')
```

这段代码的意思是：如果用户输入123456，屏幕上会打印出"通过"。但运行这段代码，终端会报错：

相信眼尖的你能发现，这段代码的问题是少了一个【英文冒号】。

仔细看报错，其中有3个关键信息。（1）line 2代表这个bug出现在第2行，所以，我们在Debug的时候，可以优先从第2行开始检查。

（2）^代表bug发生的位置，这里指出的位置是第二行末尾。（3）这一行写的是错误类型，SyntaxError指的是语法错误。

一开始可能对错误类型的英文不太熟悉，可以直接复制到百度搜索

像这样，通过理解报错信息，我们可以快速定位错误的根源。这种阅读、搜索报错信息的能力，在我们以后独立编写愈来愈复杂的程序时显得尤为重要。

我们再来看一段代码。别看这段代码只有两行，却有3处粗心错误，请你帮忙debug，让它能够顺利运行：

```
for x in range(10):
    print(x)
```

这是debug前后的代码，对比一下。

```
# 这是debug前的代码：
for x in range(10)：
      print (x)

# 这是debug后的代码：
for x in range(10):
    print(x)
```

虽然粗略地看起来差别不大，但编写代码的严谨性往往就体现在细微之处。

相信以上对你来说应该是小菜一碟，那让我们继续将挑战的难度提升。

下面这段代码的目的是：用户输入用户名和密码，当用户名为abc且密码为123时，显示登录成功，否则登录失败。用户最多可以尝试输入3次。请你修改代码，让程序能顺利运行。

要提醒你，如果光看代码看不出问题，完全可以运行后查看报错信息，再进行修改。

```
n=0

while n<3:
    username = input("请输入用户名：")
    password = input("请输入密码：")
    if username == 'abc' and password == '123':
        print("登录成功")
        break
    else:
        n=n+1
        print("输入有误")
else:
    print("你输错了三次，登录失败")
```

```
请输入用户名：ABC
请输入密码：123
输入有误
请输入用户名：abc
请输入密码：123
登录成功
```

BUG 2：知识不熟练

接下来，我们来看看第二种bug：由于知识不够熟练而引起的错误。

让我们一如既往来找茬，该代码的目的是取出列表中的'星期日'，请修改代码让它能正确运行。

```
week = ['星期一','星期二','星期三','星期四','星期五','星期六','星期日']
sunday = week[6]
print(sunday)
```

```
星期日
```

再来一道题：某学员建了一个空列表a，希望往里面增加3个值，让最后的列表变成 ['A','B','C']，但写出的代码有误。请你帮忙debug，让它能够顺利运行。

```
a = []
a.append('A')
print(a)
```

```
['A']
```

这里的问题出在append()函数，回顾课堂中append()函数的相关知识，或者搜索“python append”，我们可以知道，并没有a=append('A','B','C') 这种用法。

append()函数是列表的一个方法，要用句点.调用，且append()每次只能接受一个参数，所以正确的写法是这样：

```
a = []
a.append('A')
a.append('B')
a.append('C')
print(a)
```

由于课堂容量有限，所以这里只列举两个例子。这种bug给我们的启示是：当你发现知识点记不清或者不能确定的时候，就要及时复习或者上网搜索。不要强行写出自己不敢确定的代码，这种情况往往容易出错。

如果对某个基础知识点没有熟练的掌握，随着往后知识广度、深度以及项目难度的增加，很可能会增加大量的理解成本，所以多复习、多练习总是没有错滴。

BUG 3：思路不清

接下来我们要挑战的是“思路不清bug”，解决了这类bug，对于初学者而言，八成的问题都能自己解决了。

思路不清指的是当我们解决比较复杂的问题时，由于我们对细节和实现手段思考得不够清楚，要么导致一步错，步步错；要么虽然没有报错，但是程序没有达到我们想要的效果。

针对这一点，我先给大家推荐两个工具： 先来看print()函数。这是大家一开始就接触的函数。我们对它的功能也非常熟悉了——打印内容在屏幕上。

但其实它也能成为我们检验对错的里程碑：遇到关键步骤时print出来，看是否达到我们所期望的结果，以此来揪出错误的那一步。

‘#’ 号注释我们也学过，计算机是不会执行代码中的#号和其之后的内容的.

```
print('伊丽莎白')
#print('我属于我自己')
```

```
伊丽莎白
```

比如这个就只会执行第一行代码，而第二行代码会被计算机视而不见。

因此，当你写的代码总是不对，又弄不明白哪里不对的时候，使用#号把后面的代码注释掉，一步一步运行，可以帮助排除错误。

print()函数常和#号注释结合在一起用来debug。下面来讲一个例子：

这是一个我们在第5关的选做练习。题目是长这样的：

以下是一个同学提交的一段错误代码，大家可以运行看看（记得这里有input()函数，要在终端输入，然后点击enter）：

你可以体验到，这个程序没有达到题目要求的效果，可是又没有报错。这时就需要我们思考，问题出在哪里呢?

1-7行看不出问题，因为字典的写法挺规范的，没出现“粗心bug”。所以，问题应该出现在for循环下面的语句中。

继续看第8行：这位同学想要用for循环遍历这个字典。第9行：这位同学试图取出字典中的值。（对字典用法熟悉的人可以看出，这不符合语法规范）

但如果他自己不知道怎么回事的话，这时，就可以用注释和print()函数来帮助他看看到底是怎么回事，请看下面的第10-12行代码：

```
movie = {
    '妖猫传':['黄轩','染谷将太'],
    '无问西东':['章子怡','王力宏','祖峰'],
    '超时空同居':['雷佳音','佟丽娅']
}

name=input('你查询的演员是？ ')
for i in movie:
    actors=[i]
    print(actors) #使用print() 函数查看操作是否正确。
    #if name in actors:
        #print(name+'出演了'+i)
```

```
你查询的演员是？ 黄轩
['妖猫传']
['无问西东']
['超时空同居']
```

我们先把11-12行的代码注释掉，也就是在代码前面加一个#。（如果需要多行代码同时注释，选中代码后可以使用快捷键操作。Windows快捷键是ctrl+/，Mac为cmd+/）

我们先执行前面的代码，并且用print() 函数确定这行的操作有无问题。如果运行这段代 码，输入'黄轩'，终端会这样显示： 可见这样写取到的全部是字典的键，而非值。这时，就能意识到是这一行出了问题，他可以回看知识点，发现字典的值的取法，然后修改代码。

请你来帮他修改代码吧，回想下要如何取出字典里的值。

通过这次debug，我们掌握了解决思路不清bug的三步法： 打铁要趁热，让我们继续来实操。以下代码是一位学员制作的猜硬币游戏，一共有两次猜的机会。

```
import random

guess = ''

while guess not in ['正面','反面']:
    print('------猜硬币游戏------')
    print('猜一猜硬币是正面还是反面？ ')
    guess = input('请输入“正面”或“反面”： ')

# 随机抛硬币，0代表正面，1代表反面
toss = random.randint(0,1)

if toss == guess:
```

```
    print('猜对了！你真棒')
else:
    print('没猜对，你还有一次机会。')
    guess = input('再输一次“正面”或“反面”：')
    if toss == guess:
        print('你终于猜对了！')
    else:
        print('大失败！')
```

但是，这位学员可能没有想清楚代码的逻辑，导致这个程序有个致命问题：用户永远都不可能猜得对。那要如何把这段代码修改正确呢？请你来试一试。

```
import random
all= ['正面','反面']
guess = ''

while guess not in all:
    print('------猜硬币游戏------')
    print('猜一猜硬币是正面还是反面？')
    guess = input('请输入“正面”或“反面”：')

# 随机抛硬币，0代表正面，1代表反面

toss = all[random.randint(0,1)]

if toss == guess:
    print('猜对了！你真棒')
else:
    print('没猜对，你还有一次机会。')
    guess = input('再输一次“正面”或“反面”：')
    if toss == guess:
        print('你终于猜对了！')
    else:
        print('大失败！')
```

```
------猜硬币游戏------
猜一猜硬币是正面还是反面？
请输入“正面”或“反面”：正面
没猜对，你还有一次机会。
再输一次“正面”或“反面”：反面
你终于猜对了！
```

BUG 4：被动掉坑

被动掉坑，是指有时候你的代码逻辑上并没有错，但可能因为用户的错误操作或者是一些“例外情况”而导致程序崩溃。

我们举个例子，当运行以下代码的时候，如果输入的东西不是数字，则程序一定会报错。

```
age = input('你今年几岁了？ ')
if age < 18:
    print('不可以喝酒噢')
```

```
你今年几岁了？ 11
```

```
---------------------------------------------------------------------------
----

TypeError                                 Traceback (most recent call
last)

<ipython-input-9-8d92f44e3c67> in <module>
      1 age = input('你今年几岁了？ ')
----> 2 if age < 18:
      3     print('不可以喝酒噢')
```

```
TypeError: '<' not supported between instances of 'str' and 'int'
```

这里的“ValueError”的意思是“传入无效的参数”。因为，int()函数不能接受非数字的字符串。

对于这种“被动掉坑bug”，我们该怎么解决呢？请判断下列思路是否可行：

（1）写个条件判断——用type()函数，判断用户输入的是不是整数（2）写个while循环：如果用户输入的不是整数——让用户重输入。

其实，这个思路是行不通的。

为什么这么说呢？你输入一个数字试试就知道了：

可以发现，input()函数默认输出的数据类型是字符串，哪怕你输入的数字1，也会被转化为字符串'1'。所以type()函数并不能帮我们判断输入的到底是不是数字。

到这里，似乎只能强硬提醒用户一定要输入数字了呢？其实不然。

为了不让一些无关痛痒的小错影响程序的后续执行，Python给我们提供了一种异常处理的机制，可以在异常出现时即时捕获，然后内部消化掉，让程序继续运行。

这就是try...except...语句，具体用法如下： 让我们举个例子。刚才的报错，可以查到报错类型是“ValueError”：

现在你试试不输入整数(比如输入个abc之类的），看代码是否会报错：

```
try:
    age = int(input('请输入一个整数: '))
except ValueError:
    print('要输入整数噢')
```

```
请输入一个整数: abc
要输入整数噢
```

所以，用新学到的知识，来试一试解决之前程序的bug吧：

```
#不用修改代码，直接运行即可，尝试多输入几次非数字

while True:
    try:
        age = int(input('你今年几岁了? '))
        break
    except ValueError:
        print('你输入的不是数字! ')

if age < 18:
    print('不可以喝酒噢')
```

```
你今年几岁了? 123
```

代码要点有两个：（1）因为不知道用户什么时候才会输入正确，所以设置while循环来接受输入，只要用户输入不是数字就会一直循环，输入了数字就break跳出循环。（2）使用try……except……语句，当用户输错的时候会给予提示。

我们再来看一个例子，下列代码的目的是遍历列表中的数字，依次用6除以他们。你可以运行一下，看看报错类型是什么，然后点击跳过本题。

```
num = [1,2,0,3]
for x in num:
    print (6/x)
```

```
6.0
3.0
```

```
---------------------------------------------------------------------------
----

ZeroDivisionError                         Traceback (most recent call
last)

<ipython-input-14-bb9ce393aac6> in <module>
      1 num = [1,2,0,3]
      2 for x in num:
----> 3     print (6/x)
```

```
ZeroDivisionError: division by zero
```

可见，报错类型是ZeroDivisionError，因为小学数学告诉我们，0是不可以做除数的，所以导致后面的循环无法执行。 这时呢，你可以使用try...except语句来帮助你：如果出现ZeroDivisionError就提醒'0不能做除数'，现在请你尝试把代码补全吧～

```
num = [1,2,0,3]

try:
    for x in num:
        print (6/x)
except ZeroDivisionError:
    print('0不能被整除')
```

```
6.0
3.0
0不能被整除
```

最后，关于Python的所有报错类型，有需要的话可以在这里查阅：https://www.runoob.com/python/python-exceptions.html 好了，我们已经把四种类型的bug和解决方案都介绍了一遍，现在我们稍微回顾一下。

0. 针对粗心造成的bug，有一份自检清单帮助大家检查。
1. 针对知识点不熟造成的bug，要记得多复习，查阅笔记，针对性地做练习掌握用法。
2. 针对思维不清的bug，要多用print()函数和#注释一步步地排查错误。
3. 针对容易被忽略的例外情况从而被动掉坑的bug，可以用try...except语句让程序顺利运行。

最后，老师希望大家以后在面对bug的时候能耐心地找到问题的根源，尝试先独立思考、排查错误，逐步提升debug的能力。

如果解决不了需要求助的话，也记得要把代码完整地复制下来发给助教，方便排查错误，而不要只是截图哦。

作业

```
deposit = [100,300,900,2000,5000,0,2000,4500]

for i in range(1, len(deposit)):
    if deposit[i-1] == 0:  # 判断被除数等于0时，特殊处理。
        print('你上次存款为 0 哦！')
    else:
        times = deposit[i]/deposit[i-1]
        print('你的存款涨了%f倍'%times)
```

```
你的存款涨了3.000000倍
你的存款涨了3.000000倍
你的存款涨了2.222222倍
你的存款涨了2.500000倍
你的存款涨了0.000000倍
你上次存款为 0 哦！
你的存款涨了2.250000倍
```

```
print('\n欢迎使用除法计算器！\n')
while True:
    try:
        x = input('请你输入除数：')
        y = input('请你输入被除数：')
        z = float(x)/float(y)
        print(x,'/',y,'=',z)
        break  # 默认每次只计算一次，所以在这里写了 break。
    except ZeroDivisionError:  # 当被除数为0时，跳出提示，重新输入。
        print('0是不能做除数的！')
    except ValueError:  # 当除数或被除数中有一个无法转换成浮点数时，跳出提示，重新输入。
        print('除数和被除数都应该是整值或浮点数！')

    # 方式2：将两个（或多个）异常放在一起，只要触发其中一个，就执行所包含的代码。
    # except(ZeroDivisionError,ValueError):
    #     print('你的输入有误，请重新输入！')
    # 方式3：常规错误的基类，假设不想提供很精细的提示，可以用这个语句响应常规错误。
    # except Exception:
    #     print('你的输入有误，请重新输入！')
```

```
欢迎使用除法计算器!

请你输入除数：1
请你输入被除数：0
0是不能做除数的!
请你输入除数：1
请你输入被除数：3
1 / 3 = 0.3333333333333333
```

第12关 两种编程思维，类

两种编程思维

这种分析出解决问题所需要的步骤，然后按照这些步骤解决问题的方法，就是【面向过程】编程。

那什么是【面向对象】编程呢?

因为我们还没开始学习【类与对象】的相关知识，为了方便理解，我们举个生活化的例子。比如我们现在接到的一个需求是：做一道西红柿炒鸡蛋。

第一种方法是：自己按照炒菜的步骤亲手制作一道西红柿炒鸡蛋。

第二种方法是：制作一台炒菜机器人，然后告诉机器人做一道西红柿炒鸡蛋。

在这里，我们直接面对的是机器人，而非炒菜的过程，所以这里机器人就是我们面对的对象，这种解决问题的方法就叫做【面向对象】编程。

【面向过程】编程主要用到的是"函数"知识，将每个过程步骤打包成函数，再依次调用，还记得我们项目二用到的几个函数吗?

而【面向对象】编程主要用到的是"类和对象"知识，这也是我们这两关的学习目标。

一言以蔽之，【类】是【对象】的母板，得先有了类，我们才能制造各种"对象"。就像我们先有了产品图纸，才能制造各种产品一样。

所以，这一关我们先聚焦学习【类】，然后在下一关学习【类与对象】。

需要事先强调的是：在这两关的课堂代码中，需要取很多名称，包括各种变量名、函数名。对于英文基础不是很好的人可能会有些吃力。

所以为了让大家能把主要精力放在学习的重点上，降低理解成本。这两关的代码，不同于以往，老师会用中文来命名，便于大家理解代码含义：

```
#直接运行即可

变量1 = 2019
变量2 = '中国梦'

def 打印变量():
# 虽然可以中文命名，但括号依旧要用英文
    print(变量1)
    print(变量2)

def 两倍放大(数据):
    数据 =  数据 * 2
    return 数据

打印变量()

print(两倍放大(1000))
```

```
2019
中国梦
2000
```

```
var1 = 2019
var2 = '中国梦'

def print_var():
    print(var1)
    print(var2)
def double_num(num):
    num =  num * 2
    return num
print_var()
print(double_num(1000))
```

```
2019
中国梦
2000
```

好了，让我们开始今天的学习吧。

类是一个函数包

理解【类】最简单的方式：【类】是一个函数包。类中可以放置函数和变量，然后类中的函数可以很方便的使用类中的变量。 我们来看看类中是如何放置函数和变量的。

类中可以放置函数和变量

就像我们可以用def语句来自定义一个函数，我们用class语句来自定义一个类。

```
# 语法：创建一个名为“ClassName”的类，类名一般首字母要大写，(): 不能丢
class ClassName():
# 如定义一个名为'狗'的类，可以写成class Dog():
    # 规范：class语句后续的代码块要缩进
    def function1():
    # 定义类中的函数1
```

既然【类】是一个函数包，所以一个类中可以放置一堆函数，就像这样：（如前文所说，以下例子会用中文取名）

```
class 类A():
    def 函数1():
        print('报道！我是类A的第一个方法！')
    def 函数2():
        print('报道！我是类A的第二个方法！')
    def 函数3():
        print('报道！我是类A的第三个方法！')
```

在类中被定义的函数被称为类的【方法】，描述的是这个类能做什么。我们使用类名.函数名()的格式，就可以让类的方法运行起来。让我们来试一试：

```
class 类A():
    def 函数1():
        print('报道！我是类A的第一个方法！')
    def 函数2():
        print('报道！我是类A的第二个方法！')
    def 函数3():
        print('报道！我是类A的第三个方法！')


类A.函数1()
类A.函数2()
类A.函数3()
```

```
报道！我是类A的第一个方法！
报道！我是类A的第二个方法！
报道！我是类A的第三个方法！
```

除了函数外，在类中还可以放置一堆变量，就像这样：

```
class 类A():
    变量1 = 100
    变量2 = -5.83
    变量3 = 'abc'
```

在类中被定义的变量被称为类的【属性】。使用类名.变量名的格式，可以把类中的属性的值提取出来。让我们来试一试：（这里需要用print语句，把取出的数值打印到屏幕上）

```
class 类A():
    变量1 = 100
    变量2 = -5.83
    变量3 = 'abc'

# 这里需要用print语句，才能把提取出的数值打印到屏幕上
print(类A.变量1)
print(类A.变量2)
print(类A.变量3)
```

```
100
-5.83
abc
```

不过类中的属性（变量）也不是不能改变，使用类名.变量名的格式，可以让我们在类的外面，增加或修改类的属性：

```
class 类A():
    变量1 = 100
    变量2 = -5.83
    变量3 = 'abc'

类A.变量1 = 99
类A.变量4 = '新增一个变量'

print(类A.变量1)
print(类A.变量4)
```

```
99
新增一个变量
```

当类中放置了函数和变量，类就有了方法和属性。 下面我们来制作一个最简单的，既有方法又有属性的类：

```
class 智能机器人():
    胸围 = 33
    腰围 = 44
    臀围 = 55
    # 以上为类属性
    def 打招呼():
        print('主人你好！')
    def 卖萌():
        print('主人，求抱抱！')
    def 生气():
        print('主人，我要报警了！')
    # 以上为类方法

print('把类的属性打印出来：')
print(智能机器人.胸围)
print(智能机器人.腰围)
print(智能机器人.腰围)

智能机器人.打招呼()
智能机器人.卖萌()
```

```
把类的属性打印出来：
33
44
44
主人你好！
主人，求抱抱！
```

现在你你应该知道使用类的属性和方法了吧?

下面我们来做个练习，请为机器人再添加一个方法智能机器人.奔跑()，当运行这个方法的时候，屏幕上会打印出一串字符:我快乐地奔跑、奔跑……哎呦喂！撞墙了。

来做个练习，为机器人再添加一个属性智商 = 200，当运行print(智能机器人.智商)这个的时候，屏幕上会打印出200。请补全代码。

干得不错哟～在这里，我们为了让代码更直观，一般会把类中的函数和函数之间换行隔开，这不影响代码运行。

看到这里，你可能会有有点疑惑，类方法也是函数，那和我们之前学的单独定义函数有什么区别吗?

它们两者最大的区别，一个是它的调用格式：类.函数名()比函数名()多了一个【类.】，但更重要的是，“类”中的函数可以利用“类”中的变量（也就是类方法可以调用类属性）。

这里新出现的@classmethod是声明了下面的函数是类方法，为什么要这么写呢？这就是我们接下来要讲的：

类方法和类属性可以组合

什么意思呢？我们来看一个例子：下面的类中有两个变量和一个函数。请直接运行体验一下。然后请猜猜函数1的功能是什么？

遇到看不懂的格式没关系，老师马上会解释。

```
class 类A():
    变量1 = 100
    变量2 = 200

    @classmethod
    def 函数1(cls):
        print(cls.变量1)
        print(cls.变量2)

类A.函数1()
```

```
100
200
```

在这里，为了把类中的变量传递给类中的函数，我们需要用到3个特定格式： ① 第一个格式@classmethod的中文意思就是"类方法"，@classmethod声明了函数1是类方法，这样才能允许函数1使用类属性中的数据。

② 第二个格式cls的意思是class的缩写。如果类方法函数1想使用类属性（也就是类中的变量），就要写上cls为函数1的第一个参数，也就是把这个类作为参数传给自己，这样就能被允许使用类中的数据。

③ 第三个格式是cls.变量。类方法想使用类属性的时候，需要在这些变量名称前加上cls.。

这就好比类方法和类之间的约法三章，所以但凡有任何格式错误都会报错。

如果缺①，即缺了"@classmethod"，类方法就不能直接利用类中的属性，于是报错。（请运行代码，报错后，修改格式到正确的样式就能运行通过）

```
class 类A():
    变量1 = 100
    变量2 = 200

    def 函数1(cls):
        print(cls.变量1)
        print(cls.变量2)

类A.函数1()
```

```
---------------------------------------------------------------------------
----

TypeError                                 Traceback (most recent call
last)

<ipython-input-10-01ec3510c2a9> in <module>
      7         print(cls.变量2)
      8
----> 9 类A.函数1()
```

```
TypeError: 函数1() missing 1 required positional argument: 'cls'
```

如果缺②和③，即缺了cls和cls.变量，类方法没有和"类"沟通好数据对接，所以不能得到"类"中属性的数据，也会报错。（请运行代码，报错后，修改格式到正确的样式就能运行通过）

```
class 类A():
    变量1 = 100
    变量2 = 200

    @classmethod
    def 函数1():
        print(变量1)
        print(变量2)

类A.函数1()
```

```
---------------------------------------------------------------------------
----

TypeError                                 Traceback (most recent call
last)

<ipython-input-11-b09ec6148124> in <module>
      8         print(变量2)
      9
---> 10 类A.函数1()
```

```
TypeError: 函数1() takes 0 positional arguments but 1 was given
```

另外，当类中的函数【不需要】用到类中的变量时，就不要用@classmethod、cls、cls.三处格式，否则就是占着茅坑不拉屎，终端也会给你报错。（没错，就是这么傲娇～）

```
class 类A():
    变量1 = 100
    变量2 = 200

    #classmethod
    def 函数1():
        print('我不需要使用类属性。')
类A.函数1()
```

```
我不需要使用类属性。
```

让我们牢记正确格式。当格式正确后，我们就可以开心的玩耍了。

```
'''【正确案例】'''

class 类A():
    变量1 = 100
    变量2 = 200

    def 函数1():  # 【不需要】使用类属性
        print('我不需要使用类属性。')

    @classmethod
    def 函数2(cls):  # 【需要】使用类属性
        print(cls.变量1)
        (cls.变量2)
```

```
类A.函数1()
类A.函数2()
```

让我们做一个练习，为刚才的智能机器人，再写一个自报三围的类方法，调用方式是智能机器人.自报三围()，调用该类方法的终端显示效果如下：

```
class robot():
    胸围='33'
    腰围='44'
    臀围='55'

    @classmethod
    def f1(cls):
        print('主人，我的三围是：')
        print('胸围：'+cls.胸围)
        print('胸围：'+cls.腰围)
        print('胸围：'+cls.臀围)
        print('哈哈哈哈哈，下面粗上面细，我长得像个圆锥。')


robot.f1()
```

```
主人，我的三围是：
胸围：33
胸围：44
胸围：55
哈哈哈哈哈，下面粗上面细，我长得像个圆锥。
```

这个练习中，类方法"自报三围"通过cls的格式，取得了类属性"胸围、腰围、臀围"的值并打印了出来。这个过程中，数据是按箭头方向流转的： 在实际编程中，通过【数据流转】，我们还能让类为我们做更多事情。

给类方法传参数

类方法仅使用外部参数

我们可以给函数传递参数被函数内部使用，就像这样：

```
# 请直接运行并体验代码效果

def 加100函数(参数):
    总和 = 参数 + 100
    print('计算结果如下: ')
    print(总和)

参数 = 1
加100函数(参数)
```

```
计算结果如下:
101
```

可能有些学员现在看回中文反而一时间反应不过来，上述代码写成规范的英文格式是:

```
def add_100(num):
    sum = num + 100
    print('计算结果如下: ')
    print(sum)

num = 1
add_100(1)
```

```
计算结果如下:
101
```

类方法和函数类似，也可以传递参数。我们把上面的函数"收编"，改造成类中的方法，代码会变成这样:

```
# 请直接运行并体验代码效果

class 加100类():
    def 加100函数(参数):
        总和 = 参数 + 100
        print('计算结果如下: ')
        print(总和)

参数 = 1
加100类.加100函数(参数)
```

```
计算结果如下：
101
```

这里的加100类()中的加100函数，只使用了外部的参数，没有使用类属性，所以格式上不需要@classmethod和cls。

让我们做个练习。这里有一个函数，运行起来可以“念诗”：

```
一首诗 = ['《卜算子》','我住长江头，','君住长江尾。','日日思君不见君，','共饮长江水。']

def 念诗函数(参数):
    for i in 参数:
        print(i)

念诗函数(一首诗)
```

```
《卜算子》
我住长江头，
君住长江尾。
日日思君不见君，
共饮长江水。
```

请你把上面的代码改造成类方法：

```
一首诗 = ['《卜算子》','我住长江头，','君住长江尾。','日日思君不见君，','共饮长江水。']

class 念诗类():
    def 念诗函数(参数):
        for i in 参数:
            print(i)

念诗类.念诗函数(一首诗)
```

```
《卜算子》
我住长江头，
君住长江尾。
日日思君不见君，
共饮长江水。
```

和函数不同的是，类方法还可以利用类属性作为参数，也就是从类的内部给自己传递参数。

类方法仅使用内部参数

课堂开始的时候有一个智能机器人的例题，就是从类的内部给自己传递参数。我们运行代码体验一下：

```
# 请直接运行并体验代码效果

class 智能机器人():
    胸围 = 33
    腰围 = 44
    臀围 = 55

    @classmethod
    def 自报三围(cls):
        print('主人，我的三围是：')
        print('胸围：' + str(cls.胸围))
        print('腰围：' + str(cls.腰围))
        print('臀围：' + str(cls.臀围))
        print('哈哈哈哈哈，下面粗上面细，我长得像个圆锥。')

智能机器人.自报三围()
```

```
主人，我的三围是：
胸围：33
腰围：44
臀围：55
哈哈哈哈哈，下面粗上面细，我长得像个圆锥。
```

我们来做个练习，这是刚才的念诗类方法：

```
一首诗 = ['《卜算子》','我住长江头，','君住长江尾。','日日思君不见君，','共饮长江水。']

class 念诗类():
    def 念诗函数(参数):
        for i in 参数:
            print(i)

念诗类.念诗函数(一首诗)
```

```
《卜算子》
我住长江头，
君住长江尾。
日日思君不见君，
共饮长江水。
```

然后我们把变量一首诗从类的外部放到类的内部，请你调整代码，让念诗类()能够从内部获得参数，实现和之前相同的念诗功能。提示：可参考上述机器人的例子。

```
class 念诗类():
    一首诗 = ['《卜算子》','我住长江头，','君住长江尾。','日日思君不见君，','共饮长江水。']

    @classmethod
    def 念诗函数(cls):
        for i in cls.一首诗:
            print(i)

念诗类.念诗函数()
```

```
《卜算子》
我住长江头，
君住长江尾。
日日思君不见君，
共饮长江水。
```

当然，类方法可以同时使用内部参数和外部参数，我们接着看。

类方法同时使用内部参数和外部参数

老师写了一个加100类，你运行体验一下：

```
# 请直接运行并体验代码效果
class 加100类():
    变量 = 100

    @classmethod
    def 加100函数(cls,参数):
        总和 = cls.变量 + 参数
        print('加100函数计算结果如下：')
        print(总和)

参数 = int(input('请输入一个整数：'))
加100类.加100函数(参数)
```

```
请输入一个整数：3
加100函数计算结果如下：
103
```

这个类一共接受了两个参数，一个是用户输入的外部参数，另一个是cls这个内部参数。

另外提一下，当类方法要使用多个外部参数时，我们要多预设几个参数位置。比如：

```
# 请直接运行并体验代码效果

class 加100类():
    变量 = 100

    @classmethod
    def 加100函数(cls,参数1,参数2,参数3):
        总和 = cls.变量 + 参数1 + 参数2 + 参数3
        print('加100函数计算结果如下：')
        print(总和)

参数1 = int(input('请输入一个整数：'))
参数2 = int(input('请输入一个整数：'))
参数3 = int(input('请输入一个整数：'))

加100类.加100函数(参数1,参数2,参数3)
```

```
请输入一个整数：3
请输入一个整数：13
请输入一个整数：4
加100函数计算结果如下：
120
```

我们再做个练习吧~这里是之前的念诗代码：

```
class 念诗类():
    一首诗 = ['《卜算子》','我住长江头，','君住长江尾。','日日思君不见君，','共饮长江水。']

    @classmethod
    def 念诗函数(cls):
        for i in cls.一首诗:
            print(i)

念诗类.念诗函数()
```

你的目标是：改造这段念诗代码，让念诗类.念诗函数()可以接受外部参数，且运行效果应如下：

```
念给张三的诗：
《卜算子》
我住长江头，
君住长江尾。
日日思君不见君，
共饮长江水。
```

请你补全代码，要求取得和上面相同的运行效果。

```
class 念诗类():
    一首诗 = ['《卜算子》','我住长江头，','君住长江尾。','日日思君不见君，','共饮长江水。']

    @classmethod
    def 念诗函数(cls,参数):
        print('念给'+ 参数 +'的诗：')
        for i in cls.一首诗:
            print(i)

念诗类.念诗函数('张三')
```

```
念给张三的诗：
《卜算子》
我住长江头，
君住长江尾。
日日思君不见君，
共饮长江水。
```

到这里，给类方法传参的知识全部讲完了，让我们总结一下： 接下来我们要讲解一下，如何增加/修改类属性。

增加/修改类属性

有两种途径来增加或修改类属性。一种是从内部，用类方法去增加/修改；另一种是从外部，用类.变量 = xx直接增加/修改类属性。

从外部增加/修改属性

```
# 请直接运行并体验代码效果

class 类A():
    pass

类A.变量1 = 100
print(类A.变量1)
```

```
100
```

类A()是一个空类（里面的pass语句代表"什么都不做"），利用类A.变量1，我们在外部给类A()添加了一个类属性变量1，然后使用print语句把类属性打印了出来。

```
变量1 = 15
#这是类外部的变量1
变量2 = 'abc'

class 类A():
    变量1 = 100
    #这是类属性变量1

变量1 = 类A.变量1
类A.变量2 = 变量2

print(类A.变量1)
print(类A.变量2)
```

我们要区分两个变量，虽然名字相同，但是类外部的【变量1】和类属性【变量1】是两个不同的变量。变量1 = 类A.变量1是把类属性100重新赋值给了类外部的变量1，类属性变量1的值没有改变，还是100。

而类A.变量2 = 变量2，是为类A增加了一个类属性变量2，它的值等于外部的变量2也就是abc。

我们再做一个练习。请你先体验以下代码，然后接下来会有一道题目。

```
class 类():
    @classmethod
    def 打印类属性(cls):
        print(cls.变量)

类.变量 = input('请输入字符串：')

类.打印类属性()
```

```
请输入字符串：1
1
```

刚才你运行的代码，先利用input函数从外部接收了一个字符串，并通过赋值的方式保存为了类属性类.变量。然后利用类方法类.打印类属性()打印出了这个外部输入的字符串。

接下来，我们要做一个“幸运数翻倍”小程序，这个程序能接收外部输入的幸运数字，然后翻888倍打印出来。程序运行效果如下：

```
class 幸运数字():
    @classmethod
    def 翻倍函数(cls):
        print('好的，我把它存了起来，然后翻了888倍还给你：'+str(cls.数字*888))

幸运数字.数字=int(input('你的幸运数是多少？请输入一个整数。'))

幸运数字.翻倍函数()
```

```
你的幸运数是多少？请输入一个整数。66
好的，我把它存了起来，然后翻了888倍还给你：58608
```

到这里，如何从【类的外部】增加/修改类属性就比较清楚了，让我们再看看如何从【类的内部】增加/修改类属性。

从内部增加/修改属性

我们来体验一段代码：让类方法直接增加/修改类属性：

```
class 类():
    @classmethod
    def 增加类属性(cls):
        cls.变量 = input('请随意输入字符串：')
类.增加类属性()

print('打印新增的类属性：')
print(类.变量)
```

```
请随意输入字符串：d
打印新增的类属性：
d
```

可以发现，我们直接通过类方法类.增加并打印类属性()接收外部输入的字符串，然后新增为类属性。

掌握这个技巧后，我们再做一个练习，这里是“给张三的诗”代码：

```
class 念诗类():
    一首诗 = ['《卜算子》','我住长江头，','君住长江尾。','日日思君不见君，','共饮长江水。']

    @classmethod
    def 念诗函数(cls,参数):
        print('念给'+ 参数 +'的诗：')
        for i in cls.一首诗:
            print(i)

念诗类.念诗函数('张三')
```

现在希望你能改良这段程序，令它的运行效果如下：

```
请输入你想给谁念诗：（用户输入“张三”）
念给张三的诗：
《卜算子》
我住长江头，
君住长江尾。
日日思君不见君，
共饮长江水。
```

```
class 念诗类():
    一首诗 = ['《卜算子》','我住长江头，','君住长江尾。','日日思君不见君，','共饮长江水。']

    @classmethod
    def 增加函数(cls):
        cls.参数=input('请输入你想给谁念诗：')
        print('念给'+ cls.参数 +'的诗：')
        for i in cls.一首诗:
            print(i)

念诗类.增加函数()
```

```
请输入你想给谁念诗：zhangsan
念给zhangsan的诗：
《卜算子》
我住长江头，
君住长江尾。
日日思君不见君，
共饮长江水。
```

到这里，如何从类的内部和外部增加/修改类属性就学习完毕了： 到现在，这一关的知识就讲完了，知识量确实有点大，相信你也能感受到类并不简单。

为了让大家活学活用，接下来我会带着大家完成一个小型的课堂实操，综合使用这一关我们学过的所有知识。

不过在这之前，建议你先放松休息一下，等状态恢复了再继续回来上课。去玩儿会吧！等会儿见！

同学，你这是放松好了，还是好奇心十足，压根没去休息哇？好啦，我们继续。

课堂实操

来，想象一下，你回到了校园……

现在是人人都会写代码的时代，作为班主任的你也不例外，为了录入学生的成绩，你自己写了一段代码：

这还只是最终程序的雏形，现在可以录入一张成绩单，把一个学生的姓名和成绩存起来，请你先体验一下：（输入成绩的时候请输入数字）

```
# 请直接运行体验代码

class 成绩单():
    @classmethod
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名：')
        cls.语文_成绩 = int(input('请输入语文成绩：'))

成绩单.录入成绩单()

print(成绩单.学生姓名 + '的成绩单如下：')
print('语文成绩：'+ str(成绩单.语文_成绩))
```

```
请输入学生姓名：11
请输入语文成绩：21
11的成绩单如下：
语文成绩：21
```

后续的练习，我们会在这段代码的基础上，为class 成绩单()增加更多功能。

我们现在要做的第一步改装是，请写出一个类方法成绩单.打印成绩单()，代替那两句print语句，起到同样的运行效果。

```
class 成绩单():
    @classmethod
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名：')
        cls.语文_成绩 = int(input('请输入语文成绩：'))
    @classmethod
    def 打印成绩单(cls):
        print(cls.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(cls.语文_成绩))


成绩单.录入成绩单()
成绩单.打印成绩单()
```

```
请输入学生姓名：ww
请输入语文成绩：11
ww的成绩单如下：
语文成绩：11
```

请继续改良class 成绩单()，让它可以录入和打印的信息多一项"数学成绩"。要求改良后的代码运行效果是这样的：

```
-----代码中运行了这些类方法-----
成绩单.录入成绩单()
成绩单.打印成绩单()

---------代码运行的效果--------
请输入学生姓名：（用户输入：王明明）
请输入语文成绩：（用户输入：99）
请输入数学成绩：（用户输入：88）
王明明的成绩单如下：
语文成绩：99
数学成绩：88
```

```
class 成绩单():
    @classmethod
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名：')
        cls.语文_成绩 = int(input('请输入语文成绩：'))
        cls.数学_成绩 = int(input('请输入数学成绩：'))
    @classmethod
    def 打印成绩单(cls):
        print(cls.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(cls.语文_成绩))
        print('数学成绩：'+ str(cls.数学_成绩))
```

```
成绩单.录入成绩单()
成绩单.打印成绩单()
```

```
请输入学生姓名：啊啊
请输入语文成绩：11
请输入数学成绩：221
啊啊的成绩单如下：
语文成绩：11
数学成绩：221
```

让我们再接再厉，继续改良class 成绩单()，让它多一个类方法成绩单.打印平均分()。要求改良后的代码运行效果是这样的：

```
-----代码中运行了这些类方法-----
成绩单.录入成绩单()
成绩单.打印成绩单()
成绩单.打印平均分()

---------代码运行的效果--------
请输入学生姓名：（用户输入：王明明）
请输入语文成绩：（用户输入：99）
请输入数学成绩：（用户输入：88）
王明明的成绩单如下：
语文成绩：99
数学成绩：88
王明明的平均分是：93.5
```

```
class 成绩单():
    @classmethod
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名：')
        cls.语文_成绩 = int(input('请输入语文成绩：'))
        cls.数学_成绩 = int(input('请输入数学成绩：'))
    @classmethod
    def 打印成绩单(cls):
        print(cls.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(cls.语文_成绩))
        print('数学成绩：'+ str(cls.数学_成绩))

    @classmethod
    def 打印平均分(cls):
        平均分=(cls.数学_成绩+cls.语文_成绩)/2
        print(cls.学生姓名 + '的平均分是：' +str(平均分))

成绩单.录入成绩单()
```

```
成绩单.打印成绩单()
成绩单.打印平均分()
```

```
请输入学生姓名：ff
请输入语文成绩：123
请输入数学成绩：122
ff的成绩单如下：
语文成绩：123
数学成绩：122
ff的平均分是：122.5
```

好，作为班主任，你希望给学生的均分进行评级。请你继续改良class 成绩单()，让它多一个类方法成绩单.评级()。要求改良后的代码运行效果是这样的：

```
-----代码中运行了这些类方法-----
成绩单.录入成绩单()
成绩单.打印成绩单()
成绩单.打印平均分()
成绩单.评级()

---------代码运行的效果---------
请输入学生姓名：（用户输入：王明明）
请输入语文成绩：（用户输入：99）
请输入数学成绩：（用户输入：88）
王明明的成绩单如下：
语文成绩：99
数学成绩：88
王明明的平均分是：93.5
王明明的评级是：优
```

在这里，评级和平均分相互挂钩： 看到这个表格，你应该瞬间感知到，要用条件判断语句来写了吧！

```
class 成绩单():
    @classmethod
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名：')
        cls.语文_成绩 = int(input('请输入语文成绩：'))
        cls.数学_成绩 = int(input('请输入数学成绩：'))
        cls.平均分=int((cls.数学_成绩+cls.语文_成绩)/2)
    @classmethod
    def 打印成绩单(cls):
        print(cls.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(cls.语文_成绩))
        print('数学成绩：'+ str(cls.数学_成绩))

    @classmethod
```

```
    def 打印平均分(cls):
        平均分=int((cls.数学_成绩+cls.语文_成绩)/2)
        print(cls.学生姓名 + '的平均分是: ' +str(平均分))

    @classmethod
    def 评级(cls):
        if cls.平均分 >=90:
            print(cls.学生姓名+'的评级是: 优')
        elif cls.平均分>=80 and cls.平均分<90:
            print(cls.学生姓名+'的评级是: 良')
        elif cls.平均分>=60 and cls.平均分<80:
            print(cls.学生姓名+'的评级是: 中')
        else:
             print(cls.学生姓名+'的评级是: 差')


成绩单.录入成绩单()
成绩单.打印成绩单()
成绩单.打印平均分()
成绩单.评级()
```

```
请输入学生姓名: w
请输入语文成绩: 12
请输入数学成绩: 32
w的成绩单如下:
语文成绩: 12
数学成绩: 32
w的平均分是: 22
w的评级是: 差
```

```
class 成绩单():
    @classmethod
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名: ')
        cls.语文_成绩 = int(input('请输入语文成绩: '))
        cls.数学_成绩 = int(input('请输入数学成绩: '))

    @classmethod
    def 计算平均分(cls):
        平均分 = (cls.语文_成绩 + cls.数学_成绩)/2
        return 平均分

    @classmethod
    def 评级(cls):
        平均分 = cls.计算平均分()
        if 平均分>=90:
```

```
            print(cls.学生姓名 + '的评级是：优')
        elif 平均分>= 80 and 平均分<90 :
            print(cls.学生姓名 + '的评级是：良')
        elif 平均分>= 60 and 平均分<80 :
            print(cls.学生姓名 + '的评级是：中')
        else:
            print(cls.学生姓名 + '的评级是：差')

成绩单.录入成绩单()
成绩单.评级()
```

```
请输入学生姓名：wang
请输入语文成绩：54
请输入数学成绩：76
wang的评级是：中
```

也可以正常地得出结果，对吧？这段代码的重点在于：在def 评级(cls)中有一句cls.计算平均分()，请你翻回去确认一下。

这句代码可以调用类方法计算平均分()，在这个类方法中，用return语句保存了平均分，然后赋值给def 评级(cls)中的变量平均分。

这个程序在运行的时候，数据的流转过程如图所示： 看的明白吗？①是调用类方法评级，②是在评级内部调用了类方法计算平均分，③④是类方法计算平均分的过程，最后返回的值重新赋值给了评级中的类属性平均分。

这里有个小点要注意，def 评级(cls)中的平均分和def 计算平均分(cls)的平均分，是两个不同的变量，因为它们在不同的函数下工作，作用域不同。（可以回顾第10关“变量的作用域”相关知识）

好啦，我们最后最后再做一个练习。这里有一段残缺的代码：

```
class 成绩单():
    @classmethod
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名：')
        cls.成绩 = int(input('请输入考试成绩：'))

    @classmethod
    def 计算是否及格(cls):
        if cls.成绩 >= 60:
            return '及格'
        else:
            return '不及格'

    @classmethod
    def 考试结果(cls):
        ……
```

```
        ……

成绩单.录入成绩单()
成绩单.考试结果()
```

这段代码的目的是：输入学生姓名和成绩，能判断学生考试是否通过。补全代码后，程序会有这样的运行效果：

```
-----代码中运行了这些类方法-----
成绩单.录入成绩单()
成绩单.考试结果()

---------代码运行效果1--------
请输入学生姓名：（用户输入：王明明）
请输入考试成绩：（用户输入：60）
王明明同学考试通过啦！

---------代码运行效果2--------
请输入学生姓名：（用户输入：王明明）
请输入考试成绩：（用户输入：59）
王明明同学需要补考！
```

```
class 成绩单():
    @classmethod
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名：')
        cls.成绩 = int(input('请输入考试成绩：'))

    @classmethod
    def 计算是否及格(cls):
        if cls.成绩 >= 60:
            return '及格'
        else:
            return '不及格'

    @classmethod
    def 考试结果(cls):
        if cls.计算是否及格 == '及格':
            print(cls.学生姓名+'同学考试通过啦！')
        else:
            print(cls.学生姓名+'同学需要补考！')

成绩单.录入成绩单()
成绩单.考试结果()
```

```
请输入学生姓名：de
请输入考试成绩：56
de同学需要补考！
```

你做对了没？经历了前面的一步步实操过程，这最后一题对你来说，相信是小菜一碟啦。

到了这里，类的知识讲解上半场，就要告一段落了。从前几关的简单知识，再到经历了两个项目实操关卡，到了这里的你，应该与刚来到这里时，有了很大很大的改变吧！

不知道，你是否有逐渐感受到：编程世界很严谨，而这严谨有时又十分美妙。就像类与函数的知识，其实是一脉相承，又有了变化和丰富的过程。

最后我还要唠叨一句：为了便于课堂教学，今天的代码都是中文取名，但这种做法并不规范，大家可以在codelf网站查询英文取名：https://unbug.github.io/codelf/

作业

```
# 创建一个人事系统类
class hrSystem:
    # 创建存储员工名字的变量 name
    name = ''
    # 创建存储员工工资的变量 salary
    salary = 0
    # 创建存储员工绩效的变量 kpi
    kpi = 0
    # 定义录入员工信息的类方法
    @classmethod
    def record(cls, name, salary, kpi):
        cls.name = name
        cls.salary = salary
        cls.kpi = kpi

    # 定义打印员工信息的类方法
    @classmethod
    def print_record(cls):
        if cls.check_name():  # 以 cls.check_name() 的返回值（0或1）作为判
断条件。
            print(cls.name + '的工作信息如下：')
            print('本月工资：' + str(cls.salary))
            print('本年绩效：' + str(cls.kpi))
    # 定义根据 kpi 评奖的类方法
    @classmethod
    def kpi_reward(cls):
        if cls.kpi > 95:
            print('恭喜' + cls.name + '拿到明星员工奖杯！')
```

```
class calendar:
    # 日程表的日期
    date = '2020-08-08'
    # 事件清单，以字典形式给出，键为事件，值是安排的时间
    things = {'给父母买礼物':'9:00', '学习':'10:00', '和朋友聚会':'18:30'}

    @classmethod
    def thing_done(cls, thing):
        del cls.things[thing]

    @classmethod
    def add_thing(cls, thing, time):4
        cls.things[thing] = time

calendar.thing_done('给父母买礼物')
calendar.add_thing('写日记', '20:00')
print(calendar.things)
```

```
  File "<ipython-input-87-721ba2f2cd55>", line 14
    cls.things[thing] = time
    ^
IndentationError: unexpected indent
```

第13关 类与对象

上一关我们聚焦学习了【类】，你可能会感慨：Python里的类简直和我们人类一样复杂啊。不过我相信，如果你能很好地理解【类】，这一关就会相对轻松了。

这一关我们学习的主题是【类与对象】。

类与对象

先和大家明确一下“类”和“对象”的关系：【类】是【对象】的模板。

上一关我们提到，为了完成“西红柿炒蛋”的任务，可以制造一个“炒菜机器人”，给它下达指令。

但如果我们同时要炒出“西红柿炒蛋、宫保鸡丁、梅菜扣肉、青椒肉丝、蒜蓉油麦菜……”一系列不同的菜式，我们可以用“炒菜机器人”作为模板，复制出多个功能相同的机器人，让它们各就各位： 注意，以上说的【对象】，都是指【实例对象】。我们可以以【类】为模板，多次复制，生成多个【实例对象】。 什么叫实例对象呢？大家可以想象一下，【类】就像工厂的模具，以它为模板，造出来的成千上万的产品，才是被我们消费、购买、使用，真正融入我们生活的东西。这些产品，在Python中就叫【实例对象】。

往深了说，Python中，万事万物都可以是对象，【类】这种模板层级的本身也是【对象】，但并不是【实例对象】。

从模具变成产品，也就是从【类】变成【实例对象】的过程，就叫做【实例化】。

为了把这个事情讲得更清楚一些，最直观的就是请大家先体验一下代码。

还记得上一关“成绩单”的案例么？这是上一关的代码，只能保存和打印一份成绩单：

```
class 成绩单():
    @classmethod
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名：')
        cls.语文_成绩 = int(input('请输入语文成绩：'))
        cls.数学_成绩 = int(input('请输入数学成绩：'))

    @classmethod
    def 打印成绩单(cls):
        print(cls.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(cls.语文_成绩))
        print('数学成绩：'+ str(cls.数学_成绩))


成绩单.录入成绩单()
成绩单.打印成绩单()
```

```
请输入学生姓名：Ryan
请输入语文成绩：87
请输入数学成绩：87
Ryan的成绩单如下：
语文成绩：87
数学成绩：87
```

现在，我们已经拥有了一个“工厂模板”：被命名为成绩单的类，但如果我们需要得到多份成绩单（产品），可以把【类】实例化为多个【实例对象】，然后让每个【实例对象】各就各位。

我改写了代码，让你可以录入和打印多份不同的成绩单了。请直接运行体验代码，里面的陌生代码我马上就会解释：

```
class 成绩单():
    def 录入成绩单(self):
        self.学生姓名 = input('请输入学生姓名：')
        self.语文_成绩 = int(input('请输入语文成绩：'))
        self.数学_成绩 = int(input('请输入数学成绩：'))

    def 打印成绩单(self):
        print(self.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(self.语文_成绩))
        print('数学成绩：'+ str(self.数学_成绩))



成绩单1 = 成绩单() # 实例化，得到实例对象“成绩单1”
成绩单2 = 成绩单() # 实例化，得到实例对象“成绩单2”
成绩单3 = 成绩单() # 实例化，得到实例对象“成绩单2”
```

在上面的代码中，我们成功地录入和保存了3份成绩单。这个过程的专业描述是：把成绩单()类实例化为成绩单1、成绩单2、成绩单3三个【实例对象】。(15-17行代码是关键)

用这种思路，要录入和打印更多份成绩单也是可以做到的。

通过这个例子，我们对“类的实例化”可以有一个比较直观的体会 在这里，【实例对象】可以简称为【实例】。

如何得到【实例】，如何使用【实例】，使用【实例】和直接使用【类】有什么区别？我们在接下来的学习中可以得到这些问题的答案。

类的实例化

让我们先明确一下实例化的基本格式。要注意，当类需要被实例化后再使用时，和直接使用类的格式是不同的。

这是上一关直接使用类的格式： 这是实例化后再使用的格式： 通过对比可以看到，实例化后再使用的格式，①是空着的，意思是这里不再需要@classmethod的声明，并且在第②处，把cls替换成了self。

同时，实例化后再使用的格式，需要先赋值然后再调用（第③处） 在第④步骤，我们需要用实例名 = 类()的方式（实例名其实就是任取一个变量名），为类创建一个实例，然后再使用实例名.函数()的方式调用对应的方法。

再回顾一下“成绩单”的例题，看看实例化的格式究竟是怎么样的：

```
#直接使用类
class 成绩单():
    @classmethod
```

```
    def 录入成绩单(cls):
        cls.学生姓名 = input('请输入学生姓名：')
        cls.语文_成绩 = int(input('请输入语文成绩：'))
        cls.数学_成绩 = int(input('请输入数学成绩：'))

    @classmethod
    def 打印成绩单(cls):
        print(cls.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(cls.语文_成绩))
        print('数学成绩：'+ str(cls.数学_成绩))

成绩单.录入成绩单()
成绩单.打印成绩单()

# 实例化之后
class 成绩单():   # ①不用再写@classmethod
    def 录入成绩单(self):  # ②cls变成self
        self.学生姓名 = input('请输入学生姓名：')  # ③cls.变成self.
        self.语文_成绩 = int(input('请输入语文成绩：'))
        self.数学_成绩 = int(input('请输入数学成绩：'))

    def 打印成绩单(self):
        print(self.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(self.语文_成绩))
        print('数学成绩：'+ str(self.数学_成绩))


成绩单1 = 成绩单() # ④创建实例对象：成绩单1

成绩单1.录入成绩单() # ⑤实例化后使用
成绩单1.打印成绩单()
```

```
请输入学生姓名：Ryan
请输入语文成绩：76
请输入数学成绩：675
Ryan的成绩单如下：
语文成绩：76
数学成绩：675
请输入学生姓名：Ryan
请输入语文成绩：43
请输入数学成绩：3
Ryan的成绩单如下：
语文成绩：43
数学成绩：3
```

让我们做个练习。以下代码是直接使用类的格式，请改成实例化后再使用的格式：

```
class 智能机器人():
    胸围 = 33
    腰围 = 44
```

```
    臀围 = 55

    def 自报三围(self):
        print('主人，我的三围是：')
        print('胸围：' + str(self.胸围))
        print('腰围：' + str(self.腰围))
        print('臀围：' + str(self.臀围))
        print('哈哈哈哈哈，下面粗上面细，我长得像个圆锥。')

a = 智能机器人() # 实例化“智能机器人”类得到一个名字叫做“a”的实例
a.自报三围()
```

```
主人，我的三围是：
胸围：33
腰围：44
臀围：55
哈哈哈哈哈，下面粗上面细，我长得像个圆锥。
```

另外提一下，cls代表“类”的意思，self代表“实例”的意思，这样写是编码规范（程序员们的共识），但不是强制要求。理论上只要写个变量名占位，写什么都行，比如把self写成bbb

```
class 智能机器人():
    胸围 = 33
    腰围 = 44
    臀围 = 55

    def 自报三围(bbb):
        print('主人，我的三围是：')
        print('胸围：' + str(bbb.胸围))
        print('腰围：' + str(bbb.腰围))
        print('臀围：' + str(bbb.臀围))
        print('哈哈哈哈哈，下面粗上面细，我长得像个圆锥。')

a = 智能机器人() # 实例化“智能机器人”类得到一个名字叫做“a”的实例
a.自报三围()
```

```
主人，我的三围是：
胸围：33
腰围：44
臀围：55
哈哈哈哈哈，下面粗上面细，我长得像个圆锥。
```

不过为了不做“非主流”，我们还是乖乖地用self就好了。也就是说，当类支持实例化的时候，self是所有类方法位于首位、默认的特殊参数。

换而言之，实例化后，只要你在类中用了def语句，那么就必须在其后的括号里把第一个位置留给self。

另外，当类支持实例化的时候，就不能再直接使用类方法了，如果运行以下代码将会报错：（报错后请点击跳过按钮）

```
class 智能机器人():
    胸围 = 33
    腰围 = 44
    臀围 = 55


    def 自报三围(self):
        print('主人，我的三围是：')
        print('胸围：' + str(self.胸围))
        print('腰围：' + str(self.腰围))
        print('臀围：' + str(self.臀围))
        print('哈哈哈哈哈，下面粗上面细，我长得像个圆锥。')



#实例化后，直接使用类方法会报错
智能机器人.自报三围()
```

```
---------------------------------------------------------------------------
----

TypeError                                 Traceback (most recent call
last)

<ipython-input-6-1c781735daca> in <module>
     15
     16 #实例化后，直接使用类方法会报错
---> 17 智能机器人.自报三围()
```

```
TypeError: 自报三围() missing 1 required positional argument: 'self'
```

不过没关系。通常来说，我们都是把类实例化后再调用。（哪怕取一个和类名相同的实例名称也可以）

```
class 成绩单():


    def 录入成绩单(self):
        self.学生姓名 = input('请输入学生姓名：')
        self.语文_成绩 = int(input('请输入语文成绩：'))
        self.数学_成绩 = int(input('请输入数学成绩：'))


    def 打印成绩单(self):
```

```
        print(self.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(self.语文_成绩))
        print('数学成绩：'+ str(self.数学_成绩))


成绩单 = 成绩单()  # 【请注意这里取了一个和类名相同的实例名】

成绩单.录入成绩单()

成绩单.打印成绩单()
```

```
请输入学生姓名：Ryan
请输入语文成绩：54
请输入数学成绩：78
Ryan的成绩单如下：
语文成绩：54
数学成绩：78
```

当我们完成实例化后，对应于一个实例的属性和方法，叫"实例属性、实例方法"，不再称为"类属性、类方法"： 我们知道了如何用类生成多个实例对象，那实例的属性和方法，与类的属性和方法有什么关系呢？我们接着看。

实例属性和类属性

类和实例的关系，就像母体和复制品的关系一样。当一个类实例化为多个实例后，实例将原封不动的获得类属性，也就是实例属性和类属性完全相等。

```
# 请直接运行并体验代码

class 类():
    变量 = 100


实例1 = 类() # 实例化
实例2 = 类() # 实例化



print(类.变量)
print(实例1.变量)
print(实例2.变量)
```

```
100
100
100
```

我们可以修改类属性，这会导致所有实例属性变化（因为类是模板）。

```
class 类():
    变量=100

实例1=类()
实例2=类()
print(实例1.变量)
print(实例2.变量)

类.变量='abc'
print(实例1.变量)
print(实例2.变量)
```

```
100
100
abc
abc
```

我们也可以修改实例属性，但这不会影响到其他实例，也不会影响到类。因为每个实例都是独立的个体。

```
class 类():
    变量=100

实例1=类()
实例2=类()

print('原来的类属性；')
print(类.变量)
print('原来的实例1属性：')
print(实例1.变量)
print('原来的实例2属性：')
print(实例2.变量)

实例1.变量='abc'
print('------修改后实例1的属性------')
print(实例1.变量)
```

```
原来的类属性；
100
原来的实例1属性：
100
原来的实例2属性：
100
------修改后实例1的属性------
abc
```

新增也是一样的道理，在类中新增属性会影响到实例，但在实例中新增属性只影响这个实例自己。

体验一下新增类属性：

```
class 类():
    变量1=100

实例=类()

类.变量2='abc'

print(实例.变量1)
print(实例.变量2)
```

```
100
abc
```

体验一下新增实例属性：（将会报错，因为增加实例属性不会影响到类。报错后请点击跳过按钮）

```
class 类():
    变量1=100

实例=类()

实例.变量2='abc'

print(类.变量2)
```

```
---------------------------------------------------------------------------
----

AttributeError                            Traceback (most recent call
last)

<ipython-input-13-b0f42965146d> in <module>
      6 实例.变量2='abc'
      7
----> 8 print(类.变量2)
```

```
AttributeError: type object '类' has no attribute '变量2'
```

我们来做一个练习，请阅读以下代码，然后想一想它的运行结果将会是什么。

```
# 请思考以下代码运行结果

class 类:
    变量 = 100 #类属性

实例1 = 类() # 实例化
实例2 = 类() # 实例化

实例1.变量 = 10
类.变量 = 1

print(实例1.变量)
print(实例2.变量)
```

```
10
1
```

在这里，每个实例都是独立的个体。其中实例1.变量 = 10相当于修改了实例属性，所以类.变量 = 1不能影响到它，只能影响到实例2.变量。最终实例1.变量的值是10，实例2.变量的值是1。

弄清楚了“实例属性和类属性”，接下来我们继续看“实例方法和类方法”。

实例方法和类方法

和类属性一样，我们可以重写类方法，这会导致所有实例方法自动被重写。 在这里解释一下“重写类方法”。

“重写类方法”分成两个步骤：第一个步骤是在类的外部写一个函数，第二个步骤是把这个新函数的名字赋值给类.原始函数： 要注意的是，这里的赋值是在替换方法，并不是调用函数，所以【不要加上括号】—— 写成类.原始函数() = 新函数()是错误的。

我们来体验一下代码；

```
class 类():
    def 原始函数(self):
        print('我是原始函数！')

def 新函数(self):
    print('我是重写的新函数！')

a=类()#实例化
a.原始函数()

#用新函数替代原始函数
类.原始函数=新函数

a.原始函数()
```

```
我是原始函数！
我是重写的新函数！
```

可以看到，类方法已经被替换成了新函数，所以第二次调用a.原始函数()的效果是新函数效果。

我们来做个练习，这里有一段残缺的代码：

```
class 幸运():
    def 好运翻倍(self):
        print('好的，我把它存了起来，然后翻了888倍还给你：' + str(self.幸运
数*888))

def 新函数(self):
    print('我是重写后的新函数！')
    print('好的，我把它存了起来，然后翻了666倍还给你：' + str(self.幸运
数*666))


幸运.幸运数 = int(input('你的幸运数是多少？请输入一个整数。'))

幸运.好运翻倍=新函数

实例 = 幸运()  # 实例化
实例.好运翻倍()
```

```
你的幸运数是多少？请输入一个整数。2
我是重写后的新函数！
好的，我把它存了起来，然后翻了666倍还给你：1332
```

我们可以通过重写类方法，让实例方法发生变化，但我们不能重写实例方法，模板给的技能不是说换就能换的。 如果尝试重写实例方法将会报错：（一直运行，直到报错后，点击跳过按钮）

```
class 幸运():
    def 好运翻倍(self):
        print('好的，我把它存了起来，然后翻了888倍还给你：' + str(self.幸运数*888))

def 新函数(self):
    print('我是重写后的新函数！')
    print('好的，我把它存了起来，然后翻了666倍还给你：' + str(self.幸运数*666))


幸运.幸运数 = int(input('你的幸运数是多少？请输入一个整数。'))


实例 = 幸运()  # 实例化
实例.好运翻倍=新函数
实例.好运翻倍()
```

```
你的幸运数是多少？请输入一个整数。2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-17-c9994871a7f0> in <module>
     13 实例 = 幸运()  # 实例化
     14 实例.好运翻倍=新函数
---> 15 实例.好运翻倍()
```

```
TypeError: 新函数() missing 1 required positional argument: 'self'
```

这里，我们对“实例的属性和方法”做个总结：

初始化函数

接下来给大家介绍一个很实用的类方法，叫"初始化函数"。请你先运行一段代码：

```
class 类():
    def __init__(self):
        print('实例化成功！')


实例 = 类()
```

```
实例化成功！
```

初始化函数的意思是，当你创建一个实例的时候，这个函数就会被调用。上面的代码在执行实例 = 类()的语句时，就自动调用了**init**(self)函数。

初始化函数的写法是固定的格式：中间是"init"，这个单词的中文意思是"初始化"，然后前后都要有【两个下划线】，然后**init**()的括号中，第一个参数一定要写上self，不然会报错。

同时，这个初始化函数照样可以传递参数，我们来看个例子。

之前录入多份成绩单，我们用到了以下的代码：

```
class 成绩单():
    def 录入成绩单(self):
        self.学生姓名 = input('请输入学生姓名：')
        self.语文_成绩 = int(input('请输入语文成绩：'))
        self.数学_成绩 = int(input('请输入数学成绩：'))

    def 打印成绩单(self):
        print(cls.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(self.语文_成绩))
        print('数学成绩：'+ str(self.数学_成绩))


成绩单1 = 成绩单() # 实例化
成绩单2 = 成绩单() # 实例化
成绩单3 = 成绩单() # 实例化

成绩单1.录入成绩单()
成绩单2.录入成绩单()
成绩单3.录入成绩单()

成绩单1.打印成绩单()
成绩单2.打印成绩单()
成绩单3.打印成绩单()
```

但有了初始化函数后，我们可以直接把需要录入的信息作为参数传递给成绩单1、成绩单2、成绩单3这三个实例对象。

```
class 成绩单():
    def __init__(self,学生姓名,语文_成绩,数学_成绩):
        self.学生姓名 = 学生姓名
        self.语文_成绩 = 语文_成绩
        self.数学_成绩 = 数学_成绩

    def 打印成绩单(self):
        print(self.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(self.语文_成绩))
        print('数学成绩：'+ str(self.数学_成绩))


成绩单1 = 成绩单('张三',99,88)
成绩单2 = 成绩单('李四',64,73)
成绩单3 = 成绩单('王五',33,22)

成绩单1.打印成绩单()
成绩单2.打印成绩单()
成绩单3.打印成绩单()
```

这里的代码利用初始化函数 def **init**(self, 学生姓名 , 语文*成绩, 数学*成绩)，为每个实例自动创建了实例属性self.学生姓名、self.语文*成绩*、*self.数学*成绩。

我们来运行体验一下：

```
class 成绩单():
    def __init__(self,学生姓名,语文_成绩,数学_成绩):
        self.学生姓名 = 学生姓名
        self.语文_成绩 = 语文_成绩
        self.数学_成绩 = 数学_成绩

    def 打印成绩单(self):
        print(self.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(self.语文_成绩))
        print('数学成绩：'+ str(self.数学_成绩))


成绩单1 = 成绩单('张三',99,88)
成绩单2 = 成绩单('李四',64,73)
成绩单3 = 成绩单('王五',33,22)

成绩单1.打印成绩单()
成绩单2.打印成绩单()
成绩单3.打印成绩单()
```

```
张三的成绩单如下:
语文成绩: 99
数学成绩: 88
李四的成绩单如下:
语文成绩: 64
数学成绩: 73
王五的成绩单如下:
语文成绩: 33
数学成绩: 22
```

有了初始化函数，生成不同的实例是不是方便了很多?

接下来又是练习时间啦!

我先介绍一下题目背景。以下代码的效果是打印"九九乘法表"，我们在之前有教过，你可以运行体验一下。

```
for i in range(1,10):
    for x in range(1,i+1):
        print( '%d X %d = %d' % (i ,x ,i*x) ,end = '   ' )
    print('   ')
```

```
1 X 1 = 1
2 X 1 = 2   2 X 2 = 4
3 X 1 = 3   3 X 2 = 6   3 X 3 = 9
4 X 1 = 4   4 X 2 = 8   4 X 3 = 12   4 X 4 = 16
5 X 1 = 5   5 X 2 = 10   5 X 3 = 15   5 X 4 = 20   5 X 5 = 25
6 X 1 = 6   6 X 2 = 12   6 X 3 = 18   6 X 4 = 24   6 X 5 = 30   6 X 6 = 36

7 X 1 = 7   7 X 2 = 14   7 X 3 = 21   7 X 4 = 28   7 X 5 = 35   7 X 6 = 42
7 X 7 = 49
8 X 1 = 8   8 X 2 = 16   8 X 3 = 24   8 X 4 = 32   8 X 5 = 40   8 X 6 = 48
8 X 7 = 56   8 X 8 = 64
9 X 1 = 9   9 X 2 = 18   9 X 3 = 27   9 X 4 = 36   9 X 5 = 45   9 X 6 = 54
9 X 7 = 63   9 X 8 = 72   9 X 9 = 81
```

如果我们想打印三三乘法表，我们可以这样写

```
for i in range(1,4):
    for x in range(1,i+1):
        print( '%d X %d = %d' % (i ,x ,i*x) ,end = '   ' )
    print('   ')
```

```
1 X 1 = 1
2 X 1 = 2  2 X 2 = 4
3 X 1 = 3  3 X 2 = 6  3 X 3 = 9
```

现在我们希望把"打印乘法表"封装成一个类，让它可以通过初始化函数传递参数，需要补全的代码如下：

```
class 乘法表():
    def __init__(self,n):
        self.n=n

    def 打印(self):
        for i in range(self.n+1):
            for x in range(1,i+1):
                print( '%d X %d = %d' % (i ,x ,i*x) ,end = '  ' )
            print('  ')

三三乘法表 = 乘法表(3)
三三乘法表.打印()

五五乘法表 = 乘法表(5)
五五乘法表.打印()
```

```
1 X 1 = 1
2 X 1 = 2  2 X 2 = 4
3 X 1 = 3  3 X 2 = 6  3 X 3 = 9

1 X 1 = 1
2 X 1 = 2  2 X 2 = 4
3 X 1 = 3  3 X 2 = 6  3 X 3 = 9
4 X 1 = 4  4 X 2 = 8  4 X 3 = 12  4 X 4 = 16
5 X 1 = 5  5 X 2 = 10  5 X 3 = 15  5 X 4 = 20  5 X 5 = 25
```

这个案例，我们通过初始化函数传递参数，从而控制了三三乘法表、五五乘法表这些实例的打印行数，这里的数据流转图如下： 到这里，"类的实例化"的相关知识就学完了：

接下来我们要学习的知识是"类的继承"。

继承

说到继承，你一定脑补出了一场偶像剧里富二代继承老爹遗产，从此甩开99%同龄人，走上人生巅峰的大戏。"类的继承"也和这个有点类似，"子类"继承了"父类"的"财产"。

类的继承很大程度也是为了避免重复性劳动。比如说当我们要写一个新的类，如果新的类有许多代码都和旧类相同，又有一部分不同的时候，就可以用"继承"的方式避免重复写代码。

让我们看一个案例：有两个类，它们有许多代码一模一样，也有一部分代码不同。

```
class 成绩单_旧():
    def __init__(self,学生姓名,语文_成绩,数学_成绩):
        self.学生姓名 = 学生姓名
        self.语文_成绩 = 语文_成绩
        self.数学_成绩 = 数学_成绩

    def 打印成绩单(self):
        print(self.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(self.语文_成绩))
        print('数学成绩：'+ str(self.数学_成绩))

    def 打印平均分(self):
        平均分 = (self.语文_成绩 + self.数学_成绩)/2
        print(self.学生姓名 + '的平均分是：' + str(平均分))

class 成绩单_新():
    def __init__(self,学生姓名,语文_成绩,数学_成绩):
        self.学生姓名 = 学生姓名
        self.语文_成绩 = 语文_成绩
        self.数学_成绩 = 数学_成绩

    def 打印成绩单(self):
        print(self.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(self.语文_成绩))
        print('数学成绩：'+ str(self.数学_成绩))

    def 打印平均分(self):
        平均分 = (self.语文_成绩 + self.数学_成绩)/2
        print(self.学生姓名 + '的平均分是：' + str(平均分))

    def 打印总分(self):
        总分 = self.语文_成绩 + self.数学_成绩
        print(self.学生姓名 + '的总分是：' + str(总分))


实例_旧 = 成绩单_旧('王明明',99,88)
实例_旧.打印成绩单()
实例_旧.打印平均分()

实例_新 = 成绩单_新('王明明',99,88)
实例_新.打印成绩单()
实例_新.打印平均分()
实例_新.打印总分()
```

这两个类中，有三个类方法完全相同，但成绩单_新类多了一个类方法打印总分。 如果运用"继承"的知识，这些代码其实可以写成这样：（先请直接运行和体验代码，我接下来会讲解）

```
class 成绩单_旧():
    def __init__(self,学生姓名,语文_成绩,数学_成绩):
        self.学生姓名 = 学生姓名
        self.语文_成绩 = 语文_成绩
        self.数学_成绩 = 数学_成绩

    def 打印成绩单(self):
        print(self.学生姓名 + '的成绩单如下：')
        print('语文成绩：'+ str(self.语文_成绩))
        print('数学成绩：'+ str(self.数学_成绩))

    def 打印平均分(self):
        平均分 = (self.语文_成绩 + self.数学_成绩)/2
        print(self.学生姓名 + '的平均分是：' + str(平均分))

class 成绩单_新(成绩单_旧):
```

```
  File "<ipython-input-32-84abd11355f5>", line 16
    class 成绩单_新(成绩单_旧):
                          ^
SyntaxError: unexpected EOF while parsing
```

第18行，class 成绩单*新(成绩单*旧)就用到了类的继承，格式是class 新类(旧类） 在Python里，我们统一把旧的类称为父类，新写的类称为子类。子类可以在父类的基础上改造类方法，所以我们可以说子类继承了父类。

为了给大家展示格式，我再给大家展示一个最基本的"继承"案例。这里的子类原封不动地继承了父类，没有做任何变化，所以子类和父类的功能是一模一样的：

```
class 父类():
    def __init__(self,参数):
        self.变量=参数

    def 打印属性(self):
        print('变量的值是：'+str(self.变量))
class 子类(父类):
    pass #什么都不做


子类实例=子类(2)
子类实例.打印属性()
```

```
变量的值是：2
```

这里的格式很简单，只需使用class 子类(父类)，就能让子类继承父类的所有类方法。

如果你对运行的结果有点困惑，可以结合下面的数据流转图按步骤来参考： 下面我们来做一个练习。这里有一段残缺的代码：

```
class 基础机器人():
    def __init__(self,参数):
        self.姓名 = 参数

    def 自报姓名(self):
        print('我是' + self.姓名 + '! ')

    def 卖萌(self):
        print('主人，求抱抱！')

class 高级机器人(基础机器人):
    def 高级卖萌(self):
        print('主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒！')


安迪 = 高级机器人('安迪')

安迪.自报姓名()
安迪.卖萌()
安迪.高级卖萌()
```

```
我是安迪!
主人，求抱抱!
主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒!
```

子类除了可以定制新的类方法，还能直接覆盖父类的方法（可类比富二代男主推翻了老爹管理公司的做派），只要使用相同的类方法名称就能做到这一点。

请直接运行和体验一下代码，这里子类高级机器人重写了父类卖萌这一方法，注意看注释：

```
class 基础机器人():
    def __init__(self,参数):
        self.姓名 = 参数

    def 自报姓名(self):
```

```
        print('我是' + self.姓名 + '! ')

    def 卖萌(self):
        print('主人，求抱抱! ')
class 高级机器人(基础机器人):
    def 卖萌(self):
        print('主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒! ')

鲁宾 = 高级机器人('鲁宾')
鲁宾.自报姓名()
鲁宾.卖萌()
```

```
我是鲁宾!
主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒!
```

我们来做一个练习。这里有一段残缺的代码：

```
class 基础机器人():
    def __init__(self,参数):
        self.姓名 = 参数

    def 自报姓名(self):
        print('我是' + self.姓名 + '! ')

    def 卖萌(self):
        print('主人，求抱抱! ')

class 高级机器人(基础机器人):
    def 自报姓名(self):
        print('我是高级机器人' + self.姓名 + '! ')

    def 卖萌(self):
        print('主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒! ')

安迪 = 高级机器人('安迪')
安迪.自报姓名()
安迪.卖萌()
```

```
我是高级机器人安迪!
主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒!
```

像这样，我们能在子类中重写覆盖任意父类方法，哪怕是初始化函数**init**。我们再做个重写初始化函数**init**的练习：

```
class 基础机器人():
    def __init__(self,参数):
        self.姓名 = 参数

    def 自报姓名(self):
        print('我是' + self.姓名 + '! ')

    def 卖萌(self):
        print('主人，求抱抱! ')

class 高级机器人(基础机器人):
    def __init__(self,参数,n):
        self.姓名=参数
        self.智商=n

    def 自报姓名(self):
        print('我是高级机器人'+self.姓名+'! 智商高达'+str(self.智商)+'!')

    def 卖萌(self):
        print('主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒! ')

安迪 = 高级机器人('安迪',150)
安迪.自报姓名()
安迪.卖萌()
```

```
我是高级机器人安迪! 智商高达150!
主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒!
```

到这里，我们已经掌握了"类的继承"的基本方法：'像这样，子类从【一个父类】继承类方法，我们叫做"单继承"。还有一种更复杂的继承情况，叫"多重继承"。 顾名思义，"多重继承"就是一个子类从【多个父类】中继承类方法。格式是class 子类(父类1,父类2,.....)。

来看一个案例：

```
class 基础机器人():
    def 卖萌(self):
        print('主人，求抱抱! ')


 # 注：因为多重继承要求父类是平等的关系，所以这里的"高级机器人"没有继承"基础机器人"


class 高级机器人():
```

```
    def 高级卖萌(self):
        print('主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒！')


class 超级机器人(基础机器人,高级机器人):
    def 超级卖萌(self):
        print('pika, qiu!')


超级=超级机器人()
超级.卖萌()
超级.高级卖萌()
超级.超级卖萌()
```

```
主人，求抱抱！
主人，每次想到怎么欺负你的时候，就感觉自己全身biubiubiu散发着智慧的光芒！
pika, qiu!
```

在此，子类超级机器人同时继承了父类基础机器人和高级机器人中的类方法。

不过，多重继承有利有弊。过度使用继承容易把事情搞复杂，就像一个人有很多爸爸必定会带来诸多麻烦。

如果不是开发大型项目，不太需要用到太复杂的继承关系，所以你只需有个印象就好，我们对继承的讲解就讲到这里。

那么到现在，这一关的知识就讲完了，为了让大家活学活用，我会带着大家完成一个课堂实操，综合使用前面学过的所有知识。

课堂实操

这个项目是关于“算钱”的。

有一天，隔壁小王请你为他编写了一个出租车计费的程序，你三下五除二就完成了：

```
class 出租车():
    def 计费(self):
        公里数 = float(input('请输入行程公里数: '))
        费用 = 公里数 * 2.5
        print('费用一共是: ' + str(费用) + '元')


小王的出租车 = 出租车()
小王的出租车.计费()
```

```
请输入行程公里数: 12
费用一共是: 30.0元
```

个程序的目的是帮助出租车司机计费，每公里费用2.5元，只要输入行程公里数，就会计算出价格并打印出来。

程序不完美的地方在于，每公里费用2.5元是一个写死的数据，无法修改。

小王希望你能完善一下这个代码，也就是写出一个初始化函数，让这个类能接收参数（也就是每公里计费价格）生成不同的实例。这样，他就能把这个程序借个隔壁的隔壁，也就是司机小李用。

```
class 出租车():
    def __init__ (self,参数):
        self.每公里费用=参数

    def 计费(self):
        公里数 = float(input('请输入行程公里数: '))
        费用 = 公里数 * self.每公里费用
        print('费用一共是: ' + str(费用) + '元')


小王的出租车 = 出租车(2.5)
小王的出租车.计费()

小李的出租车 = 出租车(3)
小李的出租车.计费()
```

```
请输入行程公里数: 12
费用一共是: 30.0元
请输入行程公里数: 12
费用一共是: 36.0元
```

这里的数据流转图如下：

但是呢，这里的计费规则不太科学。司机小王对代码提出意见，他说：

小王："我这里都是前3公里15元，后面每公里2.5元，你帮我把代码改一下，让它按我的方式计费。"

请帮小王修改代码，让计费规则能实现小王的效果吧。

```
class 出租车():
    def __init__ (self,参数):
        self.每公里费用=参数

    def 计费(self):
        公里数 = float(input('请输入行程公里数：'))
        if 公里数 > 3:
            费用= 15+(公里数-3)*2.5
        else:
            费用=15

        print('费用一共是：' + str(费用) + '元')


小王的出租车 = 出租车(2.5)
小王的出租车.计费()

小李的出租车 = 出租车(3)
小李的出租车.计费()
```

```
请输入行程公里数：12
费用一共是：37.5元
请输入行程公里数：10
费用一共是：32.5元
```

过了一段时间，小王又来反馈意见：

小王："你的代码还是不够方便。我有时候想改成4公里20元，有时候想改成2公里12元，你这代码不支持我传递这些参数，能不能修改一下？"

请帮小王修改代码，让参数传递能实现小王的效果吧。

```
class 出租车():
    def __init__ (self,参数1,参数2,参数3):
        self.每公里费用=参数1
        self.初始公里数=参数2
        self.起步价=参数3

    def 计费(self):
        公里数 = float(input('请输入行程公里数：'))
        if 公里数 > self.初始公里数:
```

```
            费用= 15+(self.初始公里数-3)*self.每公里费用
        else:
            费用=self.起步价

        print('费用一共是: ' + str(费用) + '元')


小王的出租车 = 出租车(2.5,3,15)
小王的出租车.计费()
```

```
请输入行程公里数: 12
费用一共是: 15.0元
```

接下来，我们来一个难度高一些的练习：重构代码，把计费函数拆解成计费、记录行程、统计费用、结算信息四个函数：

```
class 出租车():
    def __init__(self,参数1,参数2,参数3):
        self.每公里费用 = 参数1
        self.最低公里 = 参数2
        self.最低费用 = 参数3

    def 计费(self):
        self.记录行程()
        self.统计费用()
        self.结算信息()

    def 记录行程(self):
         self.行程公里数 = float(input('请输入行程公里数: '))

    def 统计费用(self):
        if self.行程公里数<= self.最低公里:
            self.最终费用 = self.最低费用
        else:
            self.最终费用 = self.最低费用 + (self.行程公里数 - self.最低公
里) * self.每公里费用

    def 结算信息(self):
         print('费用一共是: ' + str(self.最终费用) + '元')


小王的出租车 = 出租车(2.5,3,15)
小王的出租车.计费()
```

```
请输入行程公里数：12
费用一共是：37.5元
```

小王对你的代码很满意。不过，开电动车的小李又跑来找你。

小李：“我是开电动车的，现在政府号召环保，对电动车也补贴，所有电动车计费按8折算，你能帮我写个程序么？”

你思考了一下，写出代码框架如下。

```
class 出租车():
    def __init__(self,参数1,参数2,参数3):
        self.每公里费用 = 参数1
        self.最低公里 = 参数2
        self.最低费用 = 参数3

    def 计费(self):
        self.记录行程()
        self.统计费用()
        self.结算信息()

    def 记录行程(self):
         self.行程公里数 = float(input('请输入行程公里数：'))

    def 统计费用(self):
        if self.行程公里数<= self.最低公里:
            self.最终费用 = self.最低费用
        else:
            self.最终费用 = self.最低费用 + (self.行程公里数 - self.最低公里) * self.每公里费用

    def 结算信息(self):
         print('费用一共是：' + str(self.最终费用) + '元')

class 电动车(出租车):
    def 结算信息(self):
        print('费用一共是：' + str(self.最终费用*0.8) + '元')


小王的出租车 = 出租车(2.5,3,15)
小王的出租车.计费()
小李的电动车 = 电动车(2.5,3,15)
小李的电动车.计费()
```

```
请输入行程公里数：10
费用一共是：32.5元
请输入行程公里数：10
费用一共是：26.0元
```

干得漂亮！到这里，本关的课堂实操就结束了。

通过本关的学习，我们知道了类就像是一个模版，通过这个模版可以制造出许多个实例对象，这个制造过程，我们称之为实例化。

在我们实例化的过程，可以通过初始化函数**init**给类传递不同的参数，这样的方式可以让我们得到属性不同的实例。 我们也学习了继承，有了继承之后，我们可以很方便的改装类，得到子类，这个过程就像在改装模版，我们既可以直接复制模版，又可以给模版增加一些功能，亦或是替换、强化原来模板的某些功能。

作业

练习目标：

我们会通过今天的作业，再次实践子类的继承和定制。

练习要求：

小刚需要设计一份调查问卷系统，考虑到每次调查问卷的问题都可能不同，参与人数也不同。 请你用今天学到的子类相关的知识，帮他将不同问题定制成不同子类（具体的问题见各步骤）。

```
class Survey():
    # 收集调查问卷的答案
    def __init__(self, question):
        self.question = question
        self.response = []


    # 显示调查问卷的题目
    def show_question(self):
        print(self.question)


    # 存储问卷搜集的答案
    def store_response(self, new_response):
        self.response.append(new_response)
```

```
# 请定义实名调查问卷的新类 RealNameSurvey，继承自 Survey 类
class RealNameSurvey(Survey):

    def __init__(self, question):
        Survey.__init__(self, question)
        self.response = {}
    # 由于籍贯地和名字挂钩，所以用构成为“键值对”的字典来存放。


    # 存储问卷搜集的答案（覆盖父类的类方法）
    def store_response(self, name, new_response):
        # 除了 self，还需要两个参数。
        self.response[name] = new_response  # 键值对的新增


survey = RealNameSurvey('你的籍贯地是哪？')
survey.show_question()
while True:
    response = input('请回答问卷问题，按 q 键退出：')
    if response == 'q':
        break
```

```
你的籍贯地是哪？
请回答问卷问题，按 q 键退出：周五
请回答问卷问题，按 q 键退出：反倒是
请回答问卷问题，按 q 键退出：q
```

第14关 项目实操 游戏升级

Hi，很开心，又见到你。

这一关，是我们的第3个项目实操关卡：学会用类与对象的方法编写程序。

这个项目，也是我们“PK小游戏”系列的终点。还记得我为你放的两次烟花吧:)

都到第3款小游戏了，今天我们要完成的任务，真的有些难度。需要一个高能预警：本关卡预计完成时间2小时以上。

我的建议是，抽一个完整的时间段来跟我走一遭。

我跟你提过，学Python，在做项目的过程中，是进步最快的。我没说的是：做项目总会遇到种种困难，想不通的逻辑，频频报错的代码。

如果你在今天实操的时候碰到困惑和障碍，可以停下来让自己专注地思考，复习回顾一下再继续，不必急于将这个项目完成。

毕竟，我们并不赶时间。我最大的期望，还是你能把这段时间学到的东西真正地吃下肚，用你自己喜欢的速度消化好，变成你成长中真正有用的养分。

当然，如果你能一鼓作气取得胜利，我心甚慰。先给你加油打气。

好啦，项目实操流程，希望你有印象，还是老样子： 接下来，我们就一步一步来推进这个项目的实现吧。首先是：明确这个项目的目标。

明确项目目标

这次的小游戏项目，依然是在之前项目的基础上进行改造。我希望你能在这个过程中，感受到"类与对象"的两种使用方式：

第一种方式是使用类把函数打包封装在一起，这个是我们在第12关的主要学习内容。

第二种方式是使用类生成实例对象，这个是我们在第13关主要学习的内容。

用这两种方法，我们就能大显神通，来完成这个"命中注定我克你"的游戏。

这里的"克"其实说的是"属性克制"的意思。就像我们常说水能克火，这就是一种"属性克制"。本关我们将在游戏中定义三种类型的角色：圣光骑士、暗影刺客、精灵弩手，他们三者之间也会互相克制。

我们来看下项目最终的运行效果：

明确项目目标后，接下来我们要做的就是分析过程，拆解项目。

分析过程，拆解项目

我们先回顾一下之前项目2是怎么拆分的：

那么现在请你尝试在下面的代码区将项目3也拆分成数个版本吧，在这个过程中你可以思考一下【类与对象】相关知识，可以如何运用在这个项目中。

提示：可以回看项目运行效果，以及两个项目的对比图。

```
print('''

版本1.0：
版本2.0：
版本3.0：
''')
```

以下是我的方案：

我解释一下这个拆分逻辑。

版本1.0：我们知道类能够作为函数包来使用，而上次的程序中有特别多函数，所以我们可以先用类把他们打包到一起，做成一个“游戏运行”类。打包完毕后，我们再微调一些代码，为实现新功能做好预先准备。

版本2.0：接下来我们需要为游戏创建3种角色。因为类是实例对象的模版，所以我们可以把三种角色封装成三个属性不同的类（骑士、刺客、弩手）。封装完毕后，我们还需要这三种角色能在“游戏运行”类中被调用。

版本3.0：最后一步就是要让角色之间可以相互克制。想做到这件事，最简单的方式是在刚才写的三个角色的类中，每个类添加一个“属性克制”的类方法。

好啦，开始写代码吧！

逐步执行，代码实现

首先，我们来看看版本1.0，这个版本主要是将项目2代码中的多个函数封装成类。所以我们先回顾一下【项目2最终代码】。

项目2回顾

还记得那段过百行的代码吧：

```
import time,random

# 需要的数据和变量放在开头
player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑客】','【格斗大师】','【枪弹专家】']
enemy_list = ['【暗黑战士】','【黑暗弩手】','【暗夜骑士】','【嗜血刀客】','【首席刺客】','【陷阱之王】']
players = random.sample(player_list,3)
enemies = random.sample(enemy_list,3)
player_info = {}
enemy_info = {}
```

```
# 随机生成角色的属性
def born_role():
    life = random.randint(100,180)
    attack = random.randint(30,50)
    return life,attack

# 生成和展示角色信息
def show_role():
    for i in range(3):
        player_info[players[i]] = born_role()
        enemy_info[enemies[i]] = born_role()

    # 展示我方的3个角色
    print('----------------- 角色信息 -----------------')
    print('你的人物：')
    for i in range(3):
        print('%s  血量：%s  攻击：%s'
        %(players[i],player_info[players[i]][0],player_info[players[i]]
[1]))
    print('----------------------------------------------')
    print('电脑敌人：')

    # 展示敌方的3个角色
    for i in range(3):
        print('%s  血量：%s  攻击：%s'
        %(enemies[i],enemy_info[enemies[i]][0],enemy_info[enemies[i]]
[1]))
    print('----------------------------------------------')
    input('请按回车键继续。\n')  # 为了让玩家更有控制感，可以插入类似的代码来切分
游戏进程。

# 角色排序，选择出场顺序。
def order_role():
    global players
    order_dict = {}
    for i in range(3):
        order = int(input('你想将 %s 放在第几个上场？(输入数字1~3)'%
players[i]))
        order_dict[order] = players[i]

    players = []
    for i in range(1,4):
        players.append(order_dict[i])

    print('\n我方角色的出场顺序是：%s、%s、%s' %
(players[0],players[1],players[2]))
    print('敌方角色的出场顺序是：%s、%s、%s' %
(enemies[0],enemies[1],enemies[2]))

# 角色PK
def pk_role():
    round = 1
```

```
    score = 0
    for i in range(3):  # 一共要打三局
        player_name = players[i]
        enemy_name = enemies[i]
        player_life = player_info[players[i]][0]
        player_attack = player_info[players[i]][1]
        enemy_life = enemy_info[enemies[i]][0]
        enemy_attack = enemy_info[enemies[i]][1]

        # 每一局开战前展示战斗信息
        print('\n----------------- 【第%s局】 -----------------' %
round)
        print('玩家角色：%s vs 敌方角色：%s ' %(player_name,enemy_name))
        print('%s 血量：%s  攻击：%s' %
(player_name,player_life,player_attack))
        print('%s 血量：%s  攻击：%s' %
(enemy_name,enemy_life,enemy_attack))
        print('-----------------------------------------')
        input('请按回车键继续。\n')

        # 开始判断血量是否都大于零，然后互扣血量。
        while player_life > 0 and enemy_life > 0:
            enemy_life = enemy_life - player_attack
            player_life = player_life - enemy_attack
            print('%s发起了攻击，%s剩余血量%s' %
(player_name,enemy_name,enemy_life))
            print('%s发起了攻击，%s剩余血量%s' %
(enemy_name,player_name,player_life))
            print('-----------------------------------------')
            time.sleep(1)
        else:  # 每局的战果展示，以及分数score和局数的变化。
            # 调用show_result()函数，打印返回元组中的result。
            print(show_result(player_life,enemy_life)[1])
            # 调用show_result()函数，完成计分变动。
            score += int(show_result(player_life,enemy_life)[0])
            round += 1
    input('\n点击回车，查看比赛的最终结果\n')

    if score > 0:
        print('【最终结果：你赢了！】\n')
    elif score < 0:
        print('【最终结果：你输了！】\n')
    else:
        print('【最终结果：平局！】\n')

# 返回单局战果和计分法所加分数。
def show_result(player_life,enemy_life):  # 注意：该函数要设定参数，才能判断
单局战果。
    if player_life > 0 and enemy_life <= 0:
        result = '\n敌人死翘翘了，你赢了！'
        return 1,result  # 返回元组(1,'\n敌人死翘翘了，你赢了！')，类似角色属
性的传递。
```

```
    elif player_life <= 0 and enemy_life > 0:
        result = '\n悲催，敌人把你干掉了！'
        return -1,result
    else :
        result = '\n哎呀，你和敌人同归于尽了！'
        return 0,result

# （主函数）展开战斗流程
def main():
    show_role()  # 生成和展示角色信息
    order_role()  # 角色排序，选择出场顺序
    pk_role()  # 完成角色PK，并展示PK结果

# 启动程序（即调用主函数）
main()
```

对于这一百多行代码，你是否还有印象？如果需要的话，我可以再解释解释。后面我们将基于这些代码继续添加游戏功能。

好的哟！那我们直接开始进行项目3吧。

版本1.0：写出运行游戏的类

要把项目2的最终代码封装成类，只需做这么几件事：

1和2不必多讲，我们来看看第3步：改变函数的数据传递。

在项目2中，我们使用了很多全局变量，这是因为有多个函数都需要使用这些变量。而当这些函数被封装成一个类之后，我们就不必再使用全局变量，直接把它们放在类里面，作为类属性来被不同的类方法调用。

来对比一下：

```
# 【项目2代码】
import time,random

# 需要的数据和变量放在开头
player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑客】','【格斗大师】','【枪弹专家】']
enemy_list = ['【暗黑战士】','【黑暗弩手】','【暗夜骑士】','【嗜血刀客】','【首席刺客】','【陷阱之王】']
players = random.sample(player_list,3)
enemies = random.sample(enemy_list,3)
player_info = {}
enemy_info = {}

# 【项目3代码】
import time,random
```

```
# 将变量放在类里面，作为类属性
class Game():
    player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑
客】','【格斗大师】','【枪弹专家】']
    enemy_list = ['【暗黑战士】','【黑暗弩手】','【暗夜骑士】','【嗜血刀
客】','【首席刺客】','【陷阱之王】']
    player_info = {}
    enemy_info = {}

    # 初始化函数
    def __init__(self):
        # 随机抽取3个角色名形成玩家角色列表和敌人角色列表
        self.players = random.sample(self.player_list,3)
        self.enemies = random.sample(self.enemy_list,3)
        # 然后执行游戏流程的相关函数
        self.show_role()
        self.order_role()
        self.pk_role()
```

后面的函数代码也要相应地做一些调整，主要是在类方法要加上self为第一个参数，而且在类属性前增加self.我抽一个order_role()函数的对比给大家看看：

```
# 【项目2代码】
# 角色排序，选择出场顺序。
def order_role():
    global players
    order_dict = {}
    for i in range(3):
        order = int(input('你想将 %s 放在第几个上场？(输入数字1~3)'%
players[i]))
        order_dict[order] = players[i]

    players = []
    for i in range(1,4):
        players.append(order_dict[i])

    print('\n我方角色的出场顺序是：%s、%s、%s' %
(players[0],players[1],players[2]))
    print('敌方角色的出场顺序是：%s、%s、%s' %
(enemies[0],enemies[1],enemies[2]))
```

```
# 【项目3代码】
# 角色排序，选择出场顺序。
def order_role(self):  # 在类方法中加入self作为参数
    order_dict = {}
    for i in range(3):
        order = int(input('你想将 %s 放在第几个上场？(输入数字1~3)'%
self.players[i]))
        order_dict[order] = self.players[i]
```

```
    self.players = []  # 变量前加上self.其余几处同理
    for i in range(1,4):
        self.players.append(order_dict[i])

    print('\n我方角色的出场顺序是：%s、%s、%s' %
(self.players[0],self.players[1],self.players[2]))
    print('敌方角色的出场顺序是：%s、%s、%s' %
(self.enemies[0],self.enemies[1],self.enemies[2]))
```

所有函数全部调整完毕的代码，展示给大家，运行效果和项目2的代码是一样的：

```
import time,random

class Game():
    player_list =  ['【狂血战士】','【森林箭手】','【光明骑士】','【独行剑
客】','【格斗大师】','【枪弹专家】']
    enemy_list = ['【暗黑战士】','【黑暗弩手】','【暗夜骑士】','【嗜血刀
客】','【首席刺客】','【陷阱之王】']
    player_info = {}
    enemy_info = {}

    # 初始化函数
    def __init__(self):
        # 随机抽取3个角色名形成玩家角色列表和敌人角色列表
        self.players = random.sample(self.player_list,3)
        self.enemies = random.sample(self.enemy_list,3)
        # 然后执行游戏流程的相关函数
        self.show_role()

        self.order_role()
```

你会发现，所有的类方法都加上了self为第一个参数，然后类方法在调用类属性的时候，需要用self.变量的方式。

另外，把"生成初始角色列表"和"执行游戏流程的相关函数"两个任务直接封装到了初始化函数中，所以直接game = Game()完成实例化的同时，就能启动游戏。

把项目2的最终代码封装成类后，版本1.0我们就完成了。接下来是版本2.0。

版本2.0：写出三个角色的类

根据项目目标，我们要为游戏创建三种类型的角色：光明骑士、暗影刺客、精灵弩手。游戏中的玩家队伍和敌人队伍，都由这三类角色随机组成。

我们期望达成这样的效果（开头的文字描述你自己随便写啦）：

这三类角色的差异如下：

为了得到三种类型的角色，我们就要用到“以类作为实例对象的模版”的知识。

回顾一下上一关的相关知识，这是“类的实例化”的格式：

这是“初始化函数”的格式：

```
class 类():
    def __init__(self):
        print('实例化成功！')

实例 = 类()
```

我先举个例子吧，我写了一段代码，这段代码的功能是“实例化一个固定角色”。请你来体验一下：

```
# 创建一个类，可实例化成一个固定的游戏角色

import random

class Role():

    def __init__(self):
        self.name = '【普通角色】'`
        self.life = random.randint(100,150)
        self.attack = random.randint(30,50)

a = Role()
print('实例a的角色名称是：'+ a.name)
print('实例a的血量是：'+ str(a.life))
print('实例a的攻击力是：' + str(a.attack))

b = Role()
print('实例b的角色名称是：'+ b.name)
print('实例b的血量是：'+ str(b.life))
print('实例b的攻击力是：' + str(b.attack))
```

像这样，我们先写出一个类Role()，里面有三个类属性：名称、血量、攻击力。然后用实例化的方法，就可以得到角色的实例，并使用实例.属性的方式提取出每个实例的属性值。不过我们没有对实例的name属性做区别，现在它们都是“普通角色”。

现在你来尝试一下，请写出三个角色的类，其中属性差异如下：

```
class Role():
# 父类 “【普通角色】”
# 随机生成血量和攻击

class Knight(Role):
# 子类 “【圣光骑士】” (英文是Knight)
# 血量是随机数的5倍大小，攻击力是随机数的3倍大小
```

```
class Assassin(Role):
# 子类 “【暗影刺客】” (英文是Assassin)
# 血量是随机数的3倍大小，攻击力是随机数的5倍大小

class Bowman(Role):
# 子类 “【精灵弩手】” (英文是Bowman)
# 血量是随机数的4倍大小，攻击力是随机数的4倍大小
```

请根据上面的属性差异，写出三个子类，并把每个角色的类属性也打印出来，print()语句我已经帮你写好啦。注意这次要把三个子类的name属性设置好哦。

```
import random
class Role():
    def __init__(self):
        self.name = '【普通角色】'
        self.life = random.randint(100,150)
        self.attack = random.randint(30,50)

class Knight(Role):
    def __init__(self):
        self.name = '【圣光骑士】'
        self.life = random.randint(100,150)*5
        self.attack = random.randint(30,50)*3

class Assassin(Role):
    def __init__(self):
        self.name = '【暗影刺客】'
```

在这里，每个子类都重写了一遍初始化函数，把角色名、血量、攻击力做出了区分。例如self.name = '【圣光骑士】'，是区分了子类的名称，而self.life = random.randint(100,150)*5代表5倍血量。

这样做，我们就既保证了血量和攻击力是随机数，又保证了在不同类型的角色中，血量和攻击力是有差异的。

不过，这种写法还不能体现出“继承”的思想，更程序员风格的代码是这样写的：

上面写法的第一个变化是初始化函数多了一个参数**init**(self, name='【角色】')，第二个参数是默认参数，也就是当实例化的时候不输入这个第二个参数，第二个参数就默认为'【角色】'。

上面写法的第二个变化是，利用Role.**init**(self,name)这一句代码，可以在子类中使用父类的初始化函数。

好的，现在我们写出了三种角色的子类。但是代码最后的print语句还是比较冗余，你能思考一下，如何用for循环把它们折叠起来么？（用1个for循环解决4轮的重复print）

```
……
……

a = Role()
b = Knight()
c = Assassin()
d = Bowman()

list1 = [a,b,c,d]
for i in list1:
    print(i.name + '的血量是' + str(i.life) + '; 攻击力是' +
str(i.attack))
```

在这里，我们创建了一个列表把4个实例保存起来，然后用for循环遍历即可。

给你运行体验一下： 这里，三个角色的类都准备完毕了。接下来我们要把项目2的代码和刚才写的代码拼接在一起，让游戏能3v3运行起来。

因为之前的项目代码实在太长，动辄一百多行，不方便我们实操。所以我们做一个精简版本的实操（也就是先省略很多暂时用不上的函数）。

以下有一段我差不多写好了的代码，除了49行born_role()【随机生成角色和属性】功能还没完成，请你观察这段代码，思考这个功能应该怎么写：

```
import random
import time

# 创建一个类，可实例化成具体的游戏角色
class Role():
    def __init__(self, name='【角色】'):  # 把角色名作为默认参数
        self.name = name
        self.life = random.randint(100,150)
        self.attack = random.randint(30,50)

# 创建三个子类，可实例化为3个不同类型的角色
class Knight(Role):
    def __init__(self, name='【圣光骑士】'):   # 把子类角色名作为默认参数
        Role.__init__(self,name)  # 利用了父类的初始化函数
        self.life = self.life*5  # 骑士有5份血量
        self.attack = self.attack*3    # 骑士有3份攻击力

class Assassin(Role):
    def __init__(self, name='【暗影刺客】'):
        Role.__init__(self,name)
        self.life = self.life*3
        self.attack = self.attack*5

class Bowman(Role):
    def __init__(self, name='【精灵弩手】'):
```

```
        Role.__init__(self,name)
        self.life = self.life*4
        self.attack = self.attack*4

# 创建一个类，可生成3V3并展示：可分为：欢迎语→随机生成→展示角色
class Game:
    def __init__(self):
        # 初始化各种变量
        self.players = []
        self.enemies = []
        self.score = 0
        # 执行各种游戏函数
        self.game_start()
        self.born_role()
        self.show_role()

    # 欢迎语
    def game_start(self):
        print('------------ 欢迎来到“PK小游戏” ------------')
        print('将自动生成【玩家角色】和【电脑人物】')
        input('请按回车键继续。')

    # 随机生成6个角色
    def born_role(self):
        ……

    # 展示角色
    def show_role(self):
        print('----------------- 角色信息 -----------------')
        print('你的队伍：')
        for i in range(3):
            print( '『我方』%s 血量：%s  攻击：%s'%

(self.players[i].name,self.players[i].life,self.players[i].attack))
        print('----------------------------------------------')

        print('敌人队伍：')
        for i in range(3):
            print('『敌方』%s 血量：%s  攻击：%s'%

(self.enemies[i].name,self.enemies[i].life,self.enemies[i].attack))
        print('----------------------------------------------')


gp = Game()  # 运行游戏
```

我希望补全这段代码后，游戏运行效果如下：

```
------------ 欢迎来到“PK小游戏” ------------
将自动生成【玩家角色】和【电脑人物】
```

```
请按回车键继续。
----------------- 角色信息 -----------------
你的队伍：
『我方』【暗影刺客】 血量：369  攻击：175
『我方』【精灵弩手】 血量：548  攻击：184
『我方』【圣光骑士】 血量：685  攻击：117
--------------------------------------------
敌人队伍：
『敌方』【精灵弩手】 血量：576  攻击：140
『敌方』【圣光骑士】 血量：700  攻击：132
『敌方』【圣光骑士】 血量：735  攻击：144
--------------------------------------------
```

请注意，这里的数字是随机生成的，队伍中的角色也是随机生成的（也就是队伍中的角色可以重复，比如随机生成三个【暗影刺客】都是合理的）。

你来补全这里的代码吧。这个题目稍微有些难度，如果实在想不出来，可以在报错后点击"提示"按钮。

老师的答案是这样的：

```
……

def born_role(self):
    for i in range(3):

self.players.append(random.choice([Knight(),Assassin(),Bowman()]))

self.enemies.append(random.choice([Knight(),Assassin(),Bowman()]))

……
```

self.players.append(random.choice((Knight(),Assassin(),Bowman())))这一段代码可能不易理解，但其实线索就在之前的例子中：

```
……
……

a = Role()
b = Knight()
c = Assassin()
d = Bowman()

list1 = [a,b,c,d]
for i in list1:
    print(i.name + '的血量是' + str(i.life) + '；攻击力是' +
str(i.attack))
```

这个例子中，我们用列表list1保存了4个实例对象。也就是说，实例对象可以像变量一样，被直接放到列表里。以上代码甚至可以写成这样，效果不变：

```
……
……

for i in [Role(),Knight(),Assassin(),Bowman()]:
    print(i.name + '的血量是' + str(i.life) + '；攻击力是' +
str(i.attack))
```

所以回过来看，现在使用random.choice((Knight(),Assassin(),Bowman())可以随机抽取一个类的实例对象。

然后self.players.append()可以把抽到的实例对象存在列表players中。

我们把这个函数抽取出来，运行体验一下，运行前请先阅读代码注释：

运行程序后我们会发现，每个列表中存了3个实例化对象：

图片中， <**main**.Bowman object at 0x0000023D9E1E95C0>的意思是，存了一个弓箭手的实例对象。(object的意思是"实例对象"，**main**.Bowman的意思是，当前程序文件中的"Bowman"类。连起来意思就是：这是一个Bowman类的实例对象，编号是xxxx)

现在我们知道了born_role()函数的作用，让我们这部分把代码连起来体验。按同样的道理，我们把排序、PK的代码补充完整，就能得到版本2.0的最终代码了：

```
import random
import time
# 创建一个类，可实例化成具体的游戏角色

class Role:
    def __init__(self, name):
        self.name = name
        self.life = random.randint(100,150)
        self.attack = random.randint(30,50)

# 创建三个子类，可实例化为3个不同类型的角色

class Knight(Role):

    def __init__(self, name = '【圣光骑士】'):
        Role.__init__(self,name)  # 继承了父类的初始化函数，所以，self.name
= name 不用重复写。
        self.life = self.life*5
        self.attack = self.attack*3

class Assassin(Role):
```

这里的代码绝大部分都复用了今天在版本1.0部分封装成Game类的代码，然后在这个基础上又做了两处改变。

第一处改变是把函数拆开，函数拆开后，游戏逻辑会更加清晰： 第二处改变是改写了born_role()函数，这也是刚才重点实操练习过的：

```
# 随机生成6个角色
def born_role(self):
    for i in range(3):

self.players.append(random.choice([Knight(),Assassin(),Bowman()]))

self.enemies.append(random.choice([Knight(),Assassin(),Bowman()]))
```

到这里，版本2的任务完成了。要不要先休息一下？没关系，可以先离开课堂，去做一做其他的事情，让大脑切换模式得到放松。

休息好之后，就让我们继续向版本3迈进吧！

版本3.0：为三个角色的类添加类方法

现在我们到了最后一步：实现属性克制功能。

我们对属性克制效果做个简化，让强势方遇到弱势方，攻击力就会增加50%。用具象的数值可以这样表示： 为了方便大家独立实操，我先给大家展示一个更小的案例：

用代码写出来是这样的，请仔细阅读代码及注释：

```
class Animal():  # 创建一个父类“动物”
    def __init__(self, name = '动物'):
        self.name = name
        self.attack = 100  # 动物的攻击力是100

class Cat(Animal):  # 创建一个子类“猫”
    def __init__(self, name = '猫'):
        Animal.__init__(self,name)  # 完全继承动物的初始化函数，也就是攻击力
还是100

    def fight_buff(self, opponent):  # fight_buff的意思是“战斗强化”，
opponent的意思是“对手”
        if opponent.name  == '老鼠':
            self.attack = int(self.attack * 1.5)

class Rat(Animal):  # 创建一个子类“老鼠”
    def __init__(self, name = '老鼠'):
        Animal.__init__(self,name)  # 完全继承动物的初始化函数，也就是攻击力
还是100
```

```
class Monkey(Animal):  # 创建一个子类“猴子”
    def __init__(self, name = '猴子'):
        Animal.__init__(self,name)  # 完全继承动物的初始化函数，也就是攻击力
还是100

Tom = Cat()  # 实例化一只叫做Tom的猫
Jerry = Rat()  # 实例化一只叫做Jerry的老鼠
Bobo = Monkey()  # 实例化一只叫做Bobo的猴子

print('猫的攻击力是：')
print(Tom.attack)

Tom.fight_buff(Jerry)
print('遇到老鼠，猫的攻击力是：')
print(Tom.attack)

Tom.fight_buff(Bobo)
print('遇到猴子，猫的攻击力是：')
print(Tom.attack)
```

上面代码的关键是新增了一个类法fight_buff()，这个函数需要接受一个参数opponent，也就是战斗对手的实例对象。

然后在函数中用opponent.name可以把对手的名称给提取出来，并用条件判断，如果对手的名称是'老鼠'，那就给自己的攻击力重新赋值为1.5倍（也就是增加了50%攻击力）

我的猫和老鼠例题讲完了，现在请你参照这个例题，来试试补全PK游戏代码中的类方法，要求达到以下效果：

那如何把这个效果写入游戏中呢？我们只需要在游戏Game类的pk_role()函数中加上这么两句代码：

```
self.players[i].fight_buff(self.enemies[i])
self.enemies[i].fight_buff(self.players[i])
```

有了这两句代码，游戏就可以在开始前调用角色子类对应的fight_buff()实例方法，从而获得攻击力加成，实现属性克制功能！

哇，到这里，这个游戏的全部功能就制作完毕了！

我们再为游戏代码中添加一些剧情的print语句，然后把完整的代码写出来。点击回车一起看看这段149行的代码！

```
import random
import time

# 创建一个类，可实例化成具体的游戏角色
```

```
class Role:
    def __init__(self, name='【角色】'):  # 把角色名作为默认参数
        self.name = name
        self.life = random.randint(100,150)
        self.attack = random.randint(30,50)

# 创建3个子类，可实例化为3个不同的职业

class Knight(Role):
    def __init__(self, name='【圣光骑士】'):   # 把子类角色名作为默认参数
        Role.__init__(self,name)  # 利用了父类的初始化函数
        self.life = self.life*5  # 骑士有5份血量
        self.attack = self.attack*3    # 骑士有3份攻击力

    # 职业克制关系
    def fight_buff(self, opponent,str1,str2):
        if opponent.name  == '【暗影刺客】':
            self.attack = int(self.attack * 1.5)
            print('『%s』【圣光骑士】对 『%s』【暗影刺客】说：“让无尽光芒制裁你
的堕落！”'%(str1, str2))

class Assassin(Role):
    def __init__(self, name='【暗影刺客】'):
        Role.__init__(self,name)
        self.life = self.life*3
        self.attack = self.attack*5

    # 职业克制关系
    def fight_buff(self, opponent,str1,str2):
        if opponent.name  == '【精灵弩手】':
            self.attack = int(self.attack * 1.5)
            print('『%s』【暗影刺客】对 『%s』【精灵弩手】说：“主动找死，就别怪
我心狠手辣。”'%(str1, str2))

class Bowman(Role):
    def __init__(self, name='【精灵弩手】'):
        Role.__init__(self,name)
        self.life = self.life*4
        self.attack = self.attack*4

    # 职业克制关系
    def fight_buff(self, opponent,str1,str2):
        if opponent.name  == '【圣光骑士】':
            self.attack = int(self.attack * 1.5)
            print('『%s』【精灵弩手】对 『%s』【圣光骑士】说：“骑着倔驴又如何?
你都碰不到我衣服。”'%(str1, str2))

# 创建一个类，可生成3V3并展示：可分为：欢迎语→随机生成→展示角色
class Game():
    def __init__(self):
        self.players = []  # 存玩家顺序
        self.enemies = []  # 存敌人顺序
```

```
        self.score = 0  # 比赛积分
        self.i = 0  # 记轮次
        # 依次执行以下函数
        self.game_start()  # 欢迎语
        self.born_role()  # 随机生成6个角色
        self.show_role()  # 展示角色
        self.order_role()  # 排序并展示
        self.pk_role()  # 让双方 Pk 并展示结果
        self.show_result()  # 展示最终结局

    # 欢迎语
    def game_start(self):
        print('------------ 欢迎来到“炼狱角斗场” ------------')
        print('在昔日的黄昏山脉，奥卢帝国的北境边界上，有传说中的“炼狱角斗场”。')
        print('鲜血与战斗是角斗士的归宿，金钱与荣耀是角斗士的信仰！')
        print('今日，只要你【你的队伍】能取得胜利，你将获得一笔够花500年的财
富。')
        time.sleep(2)
        print('将随机生成【你的队伍】和【敌人队伍】！')
        input('\n狭路相逢勇者胜，请按任意键继续。\n')

    # 随机生成6个角色
    def born_role(self):
        for i in range(3):

self.players.append(random.choice([Knight(),Assassin(),Bowman()]))

self.enemies.append(random.choice([Knight(),Assassin(),Bowman()]))

    # 展示角色
    def show_role(self):
        print('----------------- 角色信息 -----------------')
        print('你的队伍：')
        for i in range(3):
            print( '『我方』%s 血量：%s  攻击：%s'%

(self.players[i].name,self.players[i].life,self.players[i].attack))
        print('-------------------------------------------')

        print('敌人队伍：')
        for i in range(3):
            print('『敌方』%s 血量：%s  攻击：%s'%

(self.enemies[i].name,self.enemies[i].life,self.enemies[i].attack))
        print('-------------------------------------------')
        input('请按回车键继续。\n')

    # 排序并展示
    def order_role(self):
        order_dict = {}
        for i in range(3):
```

```
            order = int(input('你想将 %s 放在第几个上场？(输入数字1~3)'%
self.players[i].name))
            order_dict[order] = self.players[i]
        self.players = []
        for i in range(1,4):
            self.players.append(order_dict[i])
        print('\n你的队伍出场顺序是：%s、%s、%s'
        %
(self.players[0].name,self.players[1].name,self.players[2].name))
        print('敌人队伍出场顺序是：%s、%s、%s'
        %
(self.enemies[0].name,self.enemies[1].name,self.enemies[2].name))


    # 让双方 Pk 并展示结果
    def pk_role(self):
        for i in range(3):
            print('\n----------------- 【第%s轮】 -----------------' %
(i+1))
            # 每一局开战前加buff
            self.players[i].fight_buff(self.enemies[i],'我方','敌方')
            self.enemies[i].fight_buff(self.players[i],'敌方','我方')
            input('\n战斗双方准备完毕，请按回车键继续。')
            print('-----------------------------------------------')

            while self.players[i].life >0 and self.enemies[i].life>0:
                self.enemies[i].life -= self.players[i].attack
                self.players[i].life -= self.enemies[i].attack
                print('我方%s 发起了攻击，敌方%s 剩余血量 %s'%

(self.players[i].name,self.enemies[i].name,self.enemies[i].life))
                print('敌方%s 发起了攻击，我方%s 剩余血量 %s'%

(self.enemies[i].name,self.players[i].name,self.players[i].life))
                print('-----------------------------------------------')
                time.sleep(1)
            if self.players[i].life <= 0 and self.enemies[i].life> 0:
                print('\n很遗憾，我方%s 挂掉了！'% (self.players[i].name))
                self.score -= 1
            elif self.players[i].life >0 and self.enemies[i].life<= 0:
                print('\n恭喜，我方%s 活下来了。'% (self.players[i].name))
                self.score += 1
            else:
                print('\n我的天，他们俩都死了啊！')

    # 展示最终结局
    def show_result(self):
        input('\n请按回车查看最终结果。\n')
        if self.score >0:
            print('【最终结果】\n你赢了，最终的财宝都归你了！')
        elif self.score == 0:
```

```
            print('【最终结果】\n你没有胜利，但也没有失败，在夜色中灰溜溜离开了奥
卢帝国。')
        else:
            print('【最终结果】\n你输了。炼狱角斗场又多了几具枯骨。')

game = Game()
```

现在，你可以运行体验，来一起享受我们的最终成果吧：

到这里项目3的实操就结束了。这个项目确实比较复杂，难度也不小。烟花，应该已经在你的心中绽放百次。

在关卡开始的时候，我说希望你能在这个过程中，感受到"类与对象"的两种使用方式：

相信现在的你，也有了一些自己的体会。希望你也能自己独立探索，做一些自己的小项目。如果你还想做小游戏的话，也可以按课堂上的项目实操思路，自己给自己提出项目目标，然后一步一步实现。

提醒你，早期的项目目标不要太难哦，比如"猜拳游戏、抽奖游戏"是新手可以实现的，而"文字版斗地主、文字版扎金花"之类的代码将会特别复杂，新手很难驾驭，可以在有了更多编程熟练度后再尝试挑战。

等你熟练之后，说不定我们可以一起试试做一款真正有交互的图形化游戏。不要怀疑，只要你想，那会是下一个被你征服的目标。

这个项目实操关卡结束后，我们又会迎来一个知识关卡《第15关 计算机的"新华字典"》，这次的知识很简单，对于已经越过高山的你来说，真的是小case！

那我们就下一关再见啦。

第15关 编码，文件读写

今天的课主要有两大块内容，分别是编码和文件读写。

编码

我们先来看编码。编码的本质就是让只认识0和1的计算机，能够理解我们人类使用的语言符号，并且将数据转换为二进制进行存储和传输。

这种从人类语言到计算机语言转换的形式，就叫做编码表，它让人类语言和计算机语言能够一一对应起来。

要了解编码，我们还得先来聊聊二进制。由于有二进制，0和1这两个数字才能像"太极生两仪，两仪生四象，四象生八卦"一样，涵盖容纳世间所有的信息。

二进制

说起二进制，我就想起了西游记里的二进陈家庄……噢不对，是更久远的周幽王烽火戏诸侯。所以接下里我会用烽火这种古老的信息传递形式，来比喻说明计算机是怎么传输和存储数据的。

假设我们都是看守城墙的小兵，你在烽火台A上，我在烽火台B上，只要你那边来了敌人，你就点着烽火台通知我。

如果只有一个烽火台，那么只有"点着火"和"没点火"两种状态，这就像电子元件里"通电"和"没通电"的状态，所以只有0和1.

但是你光告诉我来敌人还不够啊，还得告诉我敌人的数量有多少，让我好call齐兄弟做好准备。现在问题是你要怎么通知我敌人的数量呢?

所以，我们之间就约定了特别的"暗号"，来通知彼此敌情。 现在有两座烽火台，右边为第1座，左边为第2座。我们约定，当没有烽火台被点着的时候，表示没有敌人（00）；只点着第一座烽火台的时候，表示来了一个敌人(01)；只点着第二座烽火台的时候，表示来了2个敌人。(10,逢二进一)

当两座烽火台都被点着的时候（11），就表示来了3个人。

也就是这样的对应关系：

> 二进制 - 十进制 00 - 0 01 - 1 10 - 2 11 - 3

所以两个二进制位可以表示十进制的0,1,2,3四种状态。

现在你应该可以听得懂这个笑话：世界上有10种人，懂二进制和不懂二进制的。

我们继续往下推，当有三座烽火台的时候，我们可以表示0~7八种状态（也就是2的3次方）。

以此类推，当有八座烽火台的时候，我们就能表示2的8次方，也就是256种状态，它由8个0或1组成。

> 00000000 表示状态0: 烽火全暗，一个敌人没有，平安无事，放心睡觉。 11111111 表示状态255：烽火全亮，来了255个敌人。起来打啊!

用来存放一位0或1，就是计算机里最小的存储单位，叫做【位】，也叫【比特】（bit）。我们规定8个比特构成一个【字节】（byte），这是计算机里最常用的单位。

bit和byte长得有点像，可别混淆！1 byte = 8 bit，也就是1字节等于8比特。

这些计算机单位，可与我们息息相关，你的手机"流量"，就是这么计算的： 而百兆宽带，下载速度最多能达到十多兆，是因为运营商的带宽是以比特每秒为单位的，比如100M就是100Mbit/s。

而我们常看到的下载速度KB却是以字节每秒为单位显示的，1byte = 8bit，所以运营商说的带宽得先除以8，你的百兆宽带下载速度，也就是十几兆了。

二进制居然能牵扯出这么多生活中的问题，你是否也很意外？哈哈，其实生活处处是知识呀。

好，咱们言归正传，来看让人类语言和计算机语言能够一一对应起来的【编码表】。

编码表

计算机一开始发明的时候，只是用来解决数字计算的问题。后来人们发现，计算机还可以做更多的事，正所谓能力越大，责任越大。但由于计算机只识"数"，因此人们必须告诉计算机哪个数字来代表哪个特定字符。i 于是除了0、1这些阿拉伯数字，像a、b、c这样的52个字母（包括大小写），还有一些常用的符号（例如*、#、@等）在计算机中存储时也要使用二进制数来表示，而具体用哪些二进制数字表示哪个符号，理论上每个人都可以有自己的一套规则（这就叫编码）。

但大家如果想要互相沟通而不造成混乱，就必须使用相同的编码规则。如果使用了不同的编码规则，那就会彼此读不懂，这就是"乱码"的由来。 为了避免乱码，一段世界历史就此启动。一开始，是美国首先出台了ASCII编码（读音：/'æski/），统一规定了常用符号用哪些二进制数来表示。

因为英文字母、数字再加上其他常用符号，也就100来个，因此使用7个比特位（最多表示128位）就够用了，所以一个字节中被剩下的那个比特位就被默认为0。

再后来呢，这套编码表传入欧洲，才发现这128位不够用啊。比如说法语字母上面还有注音符，这个怎么区分？得！把最后一个比特位也编进来吧。因此欧洲普遍使用一个全字节（8个比特位）进行编码，最多可表示256位，至此，一个字节就用满了！

但是前面的状态0-127位可以共用，但从状态128到255这一段的解释就完全乱套了，比如135在法语，希伯来语，俄语编码中完全是不同的符号。

当计算机漂洋过海来到中国后，问题又来了，计算机完全不认识博大精深的中文，当然也没法显示中文；而且一个字节的256位都被占满了，但中国有10万多个汉字，256位连塞牙缝都不够啊。

于是中国科学家自力更生，重写了一张编码表，也就是GB2312，它用2个字节，也就是16个比特位，来表示绝大部分（65535个）常用汉字。后来，为了能显示更多的中文，又出台了GBK标准。

不仅中国，其他国家也都搞出自己的一套编码标准，这样的话地球村村民咋沟通？日本人发封email给中国人，两边编码表不同，显示的都是乱码。

为了沟通的便利，Unicode（万国码）应运而生，这套编码表将世界上所有的符号都纳入其中。每个符号都有一个独一无二的编码，现在Unicode可以容纳100多万个符号，所有语言都可以互通，一个网页上也可以显示多国语言。

看起来皆大欢喜。但是！问题又来了，自从英文世界吃上了Unicode这口大锅饭，为迁就一些占用字节比较多的语言，英文也要跟着占两个字节。比如要存储A，原本00010001就可以了，现在偏得用两个字节：00000000 00010001才行，这样对计算机空间存储是种极大的浪费！

基于这个痛点，科学家们又提出了天才的想法：UTF-8（8-bit Unicode Transformation Format）。它是一种针对Unicode的可变长度字符编码，它可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度，而当字符在ASCII码的范围时，就用一个字节表示，所以UTF-8还可以兼容ASCII编码。

Unicode与UTF-8这种暧昧的关系一言以蔽之：Unicode是内存编码的规范，而UTF-8是如何保存和传输Unicode的手段。

将上述这段波澜壮阔、分久必合的编码史浓缩成一个表格表示，就是： 人类语言千变万化，我们有《新华字典》《牛津英语字典》这样的辞书来记录和收纳。可以说，这些编码表就是计算机世界的字典辞书，它们同样也是人类智慧的结晶。 里，我再顺便介绍下八进制和十六进制，别嫌我啰嗦啊。

因为二进制是由一堆0和1构成的，过长的数字对于人的阅读有很大障碍，为了解决这一问题，也减少书写的复杂性，我们又引入了八进制和十六进制。

为什么偏偏是16或8进制？2、8、16，分别是2的1次方、3次方、4次方。这一点使得三种进制之间可以非常直接地互相转换。

8进制是用0，1，2，3，4，5，6，7；16进制是用0，1，2，3，4，5，6，7，8，9，a，b，c，d，e，f来表示。比如说，字母K在ASCII编码表用不同进制表示的话是这样的：（你并不需要知道具体的转换规则） 接下来，我为你介绍几种编码方案在当前的使用情况。

第0，计算机是有自己的工作区的，这个工作区被称为"内存"。数据在内存当中处理时，使用的格式是Unicode，统一标准。 在Python3当中，程序处理我们输入的字符串，是默认使用Unicode编码的，所以你什么语言都可以输入。

第1，数据在硬盘上存储，或者是在网络上传输时，用的是UTF-8，因为节省空间。但你不必操心如何转换UTF-8和Unicode，当我们点击保存的时候，程序已经"默默地"帮我们做好了编码工作。

第2，一些中文的文件和中文网站，还在使用GBK，和GB2312。

基于此，有时候面对不同编码的数据，我们要进行一些操作来实现转换。这里就涉及接下来要讲的【encode】（编码）和【decode】（解码）的用法。

encode()和decode()

编码，即将人类语言转换为计算机语言，就是【编码】encode()；反之，就是【解码】decode()。它们的用法如下图所表示： 你可以尝试一下，抄写下面的代码来运行。1~2行是encode()的用法，3-4行是decode()的用法

```
print('吴枫'.encode('utf-8'))
print('吴枫'.encode('gbk'))
print(b'\xe5\x90\xb4\xe6\x9e\xab'.decode('utf-8'))
print(b'\xce\xe2\xb7\xe3'.decode('gbk'))
```

```
b'\xe5\x90\xb4\xe6\x9e\xab'
b'\xce\xe2\xb7\xe3'
吴枫
吴枫
```

将人类语言编码后得到的结果，有一个相同之处，就是最前面都有一个字母b，比如b'\xce\xe2\xb7\xe3'，这代表它是bytes（字节）类型的数据。我们可以用type()函数验证一下，请你直接点击下列程序运行。

```
print(type('吴枫'))
print(type(b'\xce\xe2\xb7\xe3'))
```

```
<class 'str'>
<class 'bytes'>
```

所谓的编码，其实本质就是把str（字符串）类型的数据，利用不同的编码表，转换成bytes（字节）类型的数据。

我们再来区分下字符和字节两个概念。字符是人们使用的记号，一个抽象的符号，这些都是字符：'1'，'中'，'a'，'$'，'¥'。

而字节则是计算机中存储数据的单元，一个8位的二进制数。

编码结果中除了标志性的字母b，你还会在编码结果中看到许多\x，你再观察一下这个例子：b'\xce\xe2\xb7\xe3'。

\x是分隔符，用来分隔一个字节和另一个字节。

分隔符还挺常见的，我们在上网的时候，不是会有网址嘛？你经常会看到网址里面有好多的%，它们也是分隔符，替换了Python中的\x。比如像下面这个：

https://www.baidu.com/s?wd=%E5%90%B4%E6%9E%AB

它的意思就是在百度里面，搜索“吴枫”，使用的是UTF-8编码。你眯着眼睛看一看上面的UTF-8编码结果和这一串网址的差异，其实它们除了分隔符以外，是一模一样的。

\xe5\x90\xb4\xe6\x9e\xab # Python编码“吴枫”的结果

%E5%90%B4%E6%9E%AB # 网址里的“吴枫”

此外，用decode()解码的时候则要注意，UTF-8编码的字节就一定要用UTF-8的规则解码，其他编码同理，否则就会出现乱码或者报错的情况，现在请你将下列字节解码成UTF-8的形式，打印出来。

```
print(b'\xe6\x88\x91\xe7\x88\xb1\xe4\xbd\xa0'.decode('utf-8'))
```

```
我爱你
```

所以以后当闷骚的程序猿丢给你这样一串代码时，不要一脸懵，或者你想向不解风情的程序员委婉地表白时，或许可以采用这种清奇的示爱方式噢。

最后我们再来看下ASCII编码，它不支持中文，所以我们来转换一个大写英文字母K。

```
print('K'.encode('ASCII'))
```

```
b'K'
```

你看到大写字母K被编码后还是K，但这两个K对计算机来说意义是不同的。前者是字符串，采用系统默认的Unicode编码，占两个字节。后者则是bytes类型的数据，只占一个字节。这也验证我们前面所说的编码就是将str类型转换成bytes类型。

编码知识虽然看起来很琐碎，但它又是非常重要的，如果不能理解这些背景知识，指不定你哪天就会遇到坑，就像隐藏在丛林中的蛇，时不时地咬你一口。而它和我们接下来要教的文件读写也有点关系。

接下来我们就来看看文件读写。

文件读写

文件读写，是Python代码调用电脑文件的主要功能，能被用于读取和写入文本记录、音频片段、Excel文档、保存邮件以及任何保存在电脑上的东西。

你可能会疑惑：为什么要在Python打开文件？我直接打开那个文件，在那个文件上操作不就好了吗？

一般来说直接打开操作当然是没问题的。但假如你有一项工作，需要把100个Word文档里的资料合并到1个文件上，一个个地复制粘贴多麻烦啊，这时你就能用上Python了。或者，当你要从网上下载几千条数据时，直接用Python帮你把数据一次性存入文件也是相当方便。

要不然怎么说，Python把我们从重复性工作中解放出来呢～

【文件读写】，是分为【读】和【写】两部分的，我们就先来瞧瞧【读文件】是怎么实现的？

读取文件

其实，真的就三步： 是不是像很久之前的那个冷笑话？“请问把大象放进冰箱需要几步？”三步：打开

冰箱，放入大象，关闭冰箱。同样地，读文件也是三步:开——读——关。

我举个例子好了，你可以跟着我在自己的电脑上操作，如果你的电脑上还没有安装Python环境，可以找助教领取安装包。

首先，我们先在桌面新建一个test文件夹，然后在文件夹里新建一个名为abc的txt文件，在里面随便写点儿什么，我写的是周芷若、赵敏。

我用编辑器Visual Studio Code打开这个文件，是这样的： 然后，你可以用VS Code新建一个open.py的Python文件，也放在test文件夹里，我们就在这里写代码。

代码怎么写呢?

【第1步-开】使用open()函数打开文件。语法是这样的:

```
file1 = open('/Users/Ted/Desktop/test/abc.txt','r',encoding='utf-8')
```

file1这个变量是存放读取的文件数据的，以便对文件进行下一步的操作。 icon open()函数里面有三个参数，对吧:

```
'/Users/Ted/Desktop/test/abc.txt'
'r'
encoding='utf-8'
```

我们一个个来看。第一个参数是文件的保存地址，一定要写清楚，否则计算机找不到。注意：我和你的文件地址是不一样的哦。

要找到你的文件地址，只需要把你要打开的文件直接拖到编辑器终端的窗口里，就会显示出文件地址，然后复制一下就好。

不过文件的地址有两种:相对路径和绝对路径，拖到终端获取的地址是绝对路径。这两种地址，Mac和Windows电脑还有点傲娇地不太一样，下面我就帮大家捋一捋。

绝对路径就是最完整的路径，相对路径指的就是【相对于当前文件夹】的路径，也就是你编写的这个py文件所放的文件夹路径!

如果你要打开的文件和open.py在同一个文件夹里，这时只要使用相对路径就行了，而要使用其他文件夹的文件则需使用绝对路径。

我们先来看Mac电脑，现在我的txt文件和py文件都放在桌面的test文件夹里。 我将txt文件拖入终端窗口，获得文件的绝对路径: 那么当我用open()函数打开的时候，就可以写成：'

```
open('/Users/Ted/Desktop/test/abc.txt')    #绝对路径
open('abc.txt')     #相对路径
#相对路径也可以写成open('./abc.txt')
```

在这种情况下，你写绝对和相对路径都是可以的。

假如现在这个txt文件，是放在test文件夹下面一个叫做word的文件夹里，绝对路径和相对路径就变成：

```
open('/Users/Ted/Desktop/test/word/abc.txt')
open('word/abc.txt')
```

我们再来看看Windows。Windows系统里，常用\来表示绝对路径，/来表示相对路径，所以当你把文件拖入终端的时候，绝对路径就变成：

```
C:\Users\Ted\Desktop\test\abc.txt
```

但是呢，别忘了\在Python中是转义字符，所以时常会有冲突。为了避坑，Windows的绝对路径通常要稍作处理，写成以下两种格式；

```
open('C:\\Users\\Ted\\Desktop\\test\\abc.txt')
#将'\'替换成'\\'

open(r'C:\Users\Ted\Desktop\test\abc.txt')
#在路径前加上字母r
```

获取文件的相对路径还有个小窍门，用VS Code打开文件夹，在文件点击右键，选择： 现在，把这行代码复制到你的open.py文件中，然后把文件地址替换成你自己的地址。

```
file1 = open('/Users/Ted/Desktop/test/abc.txt','r',encoding='utf-8')
```

好了。终于讲完了第一个参数文件地址，我们来回头看看open()的其他参数：

```
file1 = open('/Users/Ted/Desktop/test/abc.txt','r',encoding='utf-8')
```

第二个参数表示打开文件时的模式。这里是字符串 'r'，表示 read，表示我们以读的模式打开了这个文件。

你可能会疑惑，为什么打开的时候就要决定是读还是写，之后决定不行吗？这是因为，计算机非常注意数据的保密性，在打开时就要决定以什么模式打开文件。

除了'r',其他还有'w'(写入)，'a'(追加)等模式，我们稍后会涉及到。

第三个参数encoding='utf-8'，表示的是返回的数据采用何种编码，一般采用utf-8或者gbk。注意这里是写encoding而不是encode噢。

读文件的三步:开——读——关，【第1步-开】我们就讲完了，现在看【第2步-读】。

打开文件file1之后，就可以用read()函数进行读取的操作了。请看代码：

```
file1 = open('/Users/Ted/Desktop/test/abc.txt', 'r',encoding='utf-8')
filecontent = file1.read()
```

第1行代码是我们之前写的。是以读取的方式打开了文件"abc.txt"。

第2行代码就是在读取file1的内容，写法是变量file1后面加个.句点，再加个read()，并且把读到的内容放在变量filecontent里面，这样我们才能拿到文件的内容。

那么，现在我们想要看看读到了什么数据，可以用print()函数看看。请你在自己的电脑里，把剩下的代码补全，可参考下面的代码

```
file1 = open('/Users/Ted/Desktop/test/abc.txt','r',encoding='utf-8')
filecontent = file1.read()
print(filecontent)
```

然后，在编辑器窗口【右键】，选择【在终端中运行Python文件】，这时终端显示的是： 会发现，打印出了abc.txt文件里面的内容，它会读成字符串的数据形式。

【第3步-关】关闭文件，使用的是close()函数。

```
file1 = open('/Users/Ted/Desktop/test/abc.txt','r',encoding='utf-8')
filecontent = file1.read()
print(filecontent)
file1.close()
```

前3行代码你都学了，第4行：变量file1后面加个点，然后再加个close()，就代表着关闭文件。千万要记得后面的括号可不能丢。

为啥要关闭文件呢？原因有两个：1.计算机能够打开的文件数量是有限制的，open()过多而不close()的话，就不能再打开文件了。2.能保证写入的内容已经在文件里被保存好了。

文件关闭之后就不能再对这个文件进行读写了。如果还需要读写这个文件的话，就要再次 open() 打开这个文件。

我们总结一下读文件的三步：开——读——关，并奉上一张总结图。 尤其需要留意的是第二、三步，即读和关的写法。

学完了【读文件】，然后是【写文件】。

```
file1 = open('/Users/ryan/Desktop/abc.rtf','r',encoding='utf-8')
filecontent = file1.read()
print(filecontent)
```

```
{\rtf1\ansi\ansicpg936\cocoartf1671\cocoasubrtf600
{\fonttbl\f0\fnil\fcharset134 PingFangSC-Regular;\f1\fswiss\fcharset0
Helvetica;}
{\colortbl;\red255\green255\blue255;}
{\*\expandedcolortbl;;}
\paperw11900\paperh16840\margl1440\margr1440\vieww10800\viewh8400\viewk
ind0
\pard\tx566\tx1133\tx1700\tx2267\tx2834\tx3401\tx3968\tx4535\tx5102\tx5
669\tx6236\tx6803\pardirnatural\partightenfactor0

\f0\fs24 \cf0 \'c4\'e3\'ba\'c3\'b0\'a1
\f1 \
}
```

写入文件

嘻嘻，写文件也是三步：打开文件——写入文件——关闭文件。 【第1步-开】以写入的模式打开文件。

```
file1 = open('/Users/Ted/Desktop/test/abc.txt','w',encoding='utf-8')
```

第1行代码：以写入的模式打开了文件"abc.txt"。

open() 中还是三个参数，其他都一样，除了要把第二个参数改成'w'，表示write，即以写入的模式打开文件。

【第2步-写】往文件中写入内容，使用write()函数。

```
file1 = open('/Users/Ted/Desktop/test/abc.txt', 'w',encoding='utf-8')
file1.write('张无忌\n')
file1.write('宋青书\n')
```

第2-3行代码：往"abc.txt"文件中写入了"张无忌"和"宋青书"这两个字符串。\n表示另起一行。

请你原样照做，然后记得运行程序。然后当你打开txt文件查看数据： 诶？原来文件里的周芷若和赵敏去哪里了？

是这样子的，'w'写入模式会给你暴力清空掉文件，然后再给你写入。如果你只想增加东西，而不想完全覆盖掉原文件的话，就要使用'a'模式，表示append，你学过，它是追加的意思。

如果重新再来一遍的话，就要这样写：

```
file1 = open('/Users/Ted/Desktop/test/abc.txt', 'a',encoding='utf-8')
#以追加的方式打开文件abc.txt
file1.write('张无忌\n')
#把字符串'张无忌'写入文件file1
file1.write('宋青书\n')
#把字符串'宋青书'写入文件file1
```

这样的话，就会追加成功，而不会覆盖了。你可以随便试试加点什么，运行看看。 第3步-关】还是要记得关闭文件，使用close()函数，看代码：

```
file1 = open('/Users/Ted/Desktop/test/abc.txt','a',encoding='utf-8')
file1.write('张无忌\n')
file1.write('宋青书\n')
file1.close()
```

第4行代码，还是熟悉的配方，还是熟悉的味道。这样就搞定【写文件】了。

不过呢，有两个小提示：1.write()函数写入文本文件的也是字符串类型。2.在'w'和'a'模式下，如果你打开的文件不存在，那么open()函数会自动帮你创建一个

【练习时间来咯】1.请你在一个叫1.txt文件里写入字符串'难念的经' 2.然后请你读取这个1.txt文件的内容，并打印出来。

提示：先写再读。写文件分为3步，读文件也同样分为3步。文件的地址帮你写了(因为我们的网页设置，这里用的是相对路径，如果你想在本地操作，掌握上面老师讲的地址写法即可)；其他就请你来完成吧。

```
f = open('./1.txt', 'a',encoding='utf-8')
f.write('难念的经')
f.close()

f = open('./1.txt', 'r',encoding='utf-8')
neirong=f.read()
print(neirong)
f.close()
```

> 难念的经

总结

我们再来总结下写文件的三步法。 现在问题来了，如果我们想写入的数据不是文本内容，而是音频和图片的话，该怎么做呢？ 我们可以看到里面有'wb'的模式，它的意思是以二进制的方式打开一个文件用于写入。因为图片和音频是以二进制的形式保存的，所以使用wb模式就好了，这在今天的课后作业我们会用到。

这里再顺便补充一个用法，为了避免打开文件后忘记关闭，占用资源或当不能确定关闭文件的恰当时机的时候，我们可以用到关键字with，之前的例子可以写成这样：

```
# 普通写法
file1 = open('abc.txt','a')
file1.write('张无忌')
file1.close()

# 使用with关键字的写法
with open('abc.txt','a') as file1:
#with open('文件地址','读写模式') as 变量名:
    #格式：冒号不能丢
    file1.write('张无忌')
    #格式：对文件的操作要缩进
    #格式：无需用close()关闭
```

所以之后当你看到with open...as这种打开文件的语法格式也要淡定，这种还挺常见的。

正所谓“光看不写，学这Python有何用”，接下来，我们还是一起写写代码噢 (p≧w≦q)！

小练习

我们一路到这里，你应该也看出来了，本人非常喜欢《哈利波特》哈哈。

现在假设你来到了魔法世界，最近期末快到了，霍格沃兹魔法学校准备统计一下大家的成绩。

评选的依据是什么呢？就是同学们平时的作业成绩。

现在有这样一个叫scores.txt的文件，里面有赫敏、哈利、罗恩、马尔福四个人的几次魔法作业的成绩。

但是呢，因为有些魔法作业有一定难度，教授不强制同学们必须上交，所以大家上交作业的次数并不一致。 这里是文件内容，你可以在自己的电脑里新建一个scores.txt来操作。

```
罗恩 23 35 44
哈利 60 77 68 88 90
赫敏 97 99 89 91 95 90
马尔福 100 85 90
```

望你来统计这四个学生的魔法作业的总得分，然后再写入一个txt文件。注意，这个练习的全程只能用Python。

面对这个功能，请你思考30s，粗略地想想该怎么实现？然后点击enter继续。

好，一个非常粗糙的思路应该是：拿到txt文件里的数据，然后对数据进行统计，然后再写入txt文件。好，马上开始。

首先，毫无疑问，肯定是打开文件，还记得用什么函数吗?

```
file1 = open('/Users/Ted/Desktop/scores.txt','r',encoding='utf-8')
```

接着呢，就是读取文件了。一般来说，我们是用read()函数，但是在这里，我们是需要把四个人的数据分开处理的，我们想要按行处理，而不是一整个处理，所以读的时候也希望逐行读取。

因此，我们需要使用一个新函数readlines()，也就是“按行读取”。

```
file1 = open('/Users/Ted/Desktop/scores.txt','r',encoding='utf-8')
file_lines = file1.readlines()
file1.close()
print(file_lines)
```

用print()函数打印一下，看看这种方法读出来的内容是咋显示的： 你看到了，readlines() 会从txt文件取得一个列表，列表中的每个字符串就是scores.txt中的每一行。而且每个字符串后面还有换行的\n符号。

这样一来，我们就可以使用for循环来遍历这个列表，然后处理列表中的数据，请看第五行代码：

```
file1 = open('/Users/ryan/Desktop/scores.txt','r',encoding='utf-8')
file_lines = file1.readlines()
file1.close()

for i in file_lines:     #用for...in...把每一行的数据遍历
    print(i)             #打印变量i
```

```
罗恩 23 35 44

哈利 60 77 68 88 90

赫敏 97 99 89 91 95 90

马尔福 100 85 90
```

好，现在我们要把这里每一行的名字、分数也分开，这时需要我们使用split()来把字符串分开，它会按空格把字符串里面的内容分开。

看上图第一行的罗恩 23 35 44，它将被分为('罗恩', '23', '35', '44')。

```
file1 = open('/Users/ryan/Desktop/scores.txt','r',encoding='utf-8')
file_lines = file1.readlines()
file1.close()

for i in file_lines:    #用for...in...把每一行的数据遍历
    data =i.split()     #把字符串切分成更细的一个个的字符串
    print(data)         #打印出来看看
```

```
['罗恩', '23', '35', '44']
['哈利', '60', '77', '68', '88', '90']
['赫敏', '97', '99', '89', '91', '95', '90']
['马尔福', '100', '85', '90']
```

显然，对比上面两个终端的图，split()又把每一行的内容分成了一个个的字符串，于是变成了一个个列表。

split()是我们没有学过的对字符串的处理方法，在这里，老师想插一句，对数据类型的处理是有很多种方法的，但我们不可能一次学完，而应该学习最基础必要的知识，然后在需要用到新知识时，再继续学。

split()是把字符串分割的，而还有一个join()函数，是把字符串合并的。

```
a=['c','a','t']
b=''
print(b.join(a))
c='-'
print(c.join(a))
```

```
cat
c-a-t
```

join()的用法是str.join(sequence)，str代表在这些字符串之中，你要用什么字符串连接，在这里两个例子，一个是空字符串，一个是横杠，sequence代表数据序列，在这里是列表a。

在这里只是为了让大家理解join()，不需要记忆，之后再用再看就好。

回到哈利波特的那一步： 这4个列表的第0个数据是姓名，之后的就是成绩。我们需要先统计各人的总成绩，然后把姓名和成绩放在一起。

还是可以用for...in...循环进行加法的操作，请看第8行的代码：

```
file1 = open('/Users/ryan/Desktop/scores.txt','r',encoding='utf-8')
file_lines = file1.readlines()
file1.close()

for i in file_lines:
    data =i.split()
    sum = 0                         #先把总成绩设为0
    for score in data[1:]:          #遍历列表中第1个数据和之后的数据
        sum = sum + int(score) #然后依次加起来，但分数是字符串，所以要转换
    result = data[0]+str(sum)   #结果就是学生姓名和总分
    print(result)
```

```
罗恩102
哈利383
赫敏561
马尔福275
```

好，接下来就是把成绩写入一个空的列表，因为这样才有助于我们之后写入一个txt文件。

```
file1 = open('/Users/ryan/Desktop/scores.txt','r',encoding='utf-8')
file_lines = file1.readlines()
file1.close()

final_scores = []                           #新建一个空列表

for i in file_lines:
    data =i.split()
    sum = 0
    for score in data[1:]:
        sum = sum + int(score)
    result = data[0]+str(sum)+'\n'          #后面加上换行符，写入的时候更清晰。

    final_scores.append(result)#每统计一个学生的总分，就把姓名和总分写入空列表


winner = open('/Users/ryan/Desktop/winner.txt','w',encoding='utf-8')
winner.writelines(final_scores)
winner.close()
```

15行的代码是打开一个叫winner.txt的文件。(如果电脑中不存在winner.txt的话，这行代码会帮你自动新建一个空白的winner.txt)

16行的代码是以writelines()的方式写进去，为什么不能用write()？因为final_scores是一个列表，而write()的参数必须是一个字符串，而writelines()可以是序列，所以我们使用writelines()。

17行代码是关闭文件。这个，你记得不要把括号丢掉就好。

这个小练习里，我们增加了一些新的函数用法，并不是很难，只是对你来说暂时还有些陌生，但它们与我们学过的函数其实都有联系，或是用法大有相似之处，你只需要熟悉它，等要用的时候翻出来就好啦。

而且，相比于我们已经翻越过的几座大山，也就是前几个项目实操关卡，这个小练习应该只能算一个轻轻松松就能越过的小山坡，对吧?

第16关 模块

本关要学习的知识是"模块"。虽然说是新知识，但我们在前面的课程经常cue到它，而且比起"类和对象"，模块的知识相对简单一些。

开门见山，我们先来谈一谈什么是模块。

什么是模块

之前我们已经学过，类可以封装方法和属性，就像这样： 用书里的话说：模块是最高级别的程序组织单元。这句话的意思是，模块什么都能封装，就像这样： 在模块中，我们不但可以直接存放变量，还能存放函数，还能存放类。 更独特的是，定义变量需要用赋值语句，封装函数需要用def语句，封装类需要用class语句，但封装模块不需要任何语句。

之所以不用任何语句，是因为每一份单独的Python代码文件（后缀名是.py的文件）就是一个单独的模块。

如果你使用过vscode或pycharm等编程工具编写python程序，每次都需要先创建一个后缀名为.py的Python程序文件，才能运行程序： 在平时的课堂教学中，其实我们每次运行的代码，本质上都是在运行一个名为main.py的程序文件：（只不过被隐藏在了终端里） 像这样：每一个单独的py文件，本质上都是一个模块。

封装模块的目的也是为了把程序代码和数据存放起来以便再次利用。如果封装成类和函数，主要还是便于自己调用，但封装了模块，我们不仅能自己使用，文件的方式也很容易共享给其他人使用。

所以，我们使用模块主要有两种方式，一种是自己建立模块并使用，另外一种是使用他人共享的模块。

使用自己的模块

建立模块，其实就是在主程序的py文件中，使用import语句导入其他py文件。

我们看一个小例子，这个例子中有两个模块，一个是test.py文件，另一个是main.py文件。 请你阅读两个文件中的代码，并直接运行main.py文件。

```
import test  # 导入test模块
print(test.a)
print(test.hi())
# 直接点击运行按钮即可，无需任何修改
```

是不是什么都没有出现？现在请你点击重做，输入print(test.a) 和test.hi() 然后点击运行，你就会发现main.py文件借用并运行了test.py文件里的代码。

这就是import语句起作用了。具体来说，模块相关的常用语句主要有3个，我们接下来一个个来看。

import语句

我们使用import语句导入一个模块，最主要的目的并不是运行模块中的执行语句，而是为了利用模块中已经封装好的变量、函数、类。

```
a = '我是模块中的变量a'

def hi():
    a = '我是函数里的变量a'
    print('函数“hi”已经运行！')

class Go1:  # 如果没有继承的类，class语句中可以省略括号，但定义函数的def语句括号不能省
    a = '我是类1中的变量a'
    @classmethod
    def do1(cls):
        print('函数“do1”已经运行！')

class Go2:
    a = '我是类2中的变量a'
    def do2(self):
        print('函数“do2”已经运行！')

print(a)  # 打印变量“a”

hi()  # 调用函数“hi”

print(Go1.a)  # 打印类属性“a”
Go1.do1()  # 调用类方法“Go1”

A = Go2()  # 实例化“Go2”类
```

```
print(A.a)  # 打印实例属性“a”
A.do2()  # 调用实例方法“do2”
```

```
我是模块中的变量a
函数“hi”已经运行!
我是类1中的变量a
函数“do1”已经运行!
我是类2中的变量a
函数“do2”已经运行!
```

麻雀虽小，五脏俱全。这段代码中基本上展现了所有的调用方式。

现在我们要做的是把这段代码拆分成两个模块，把封装好的变量、函数、类，放到test.py文件中，把执行相关的语句放到main.py文件中。

你是否注意到，当我们导入模块后，要使用模块中的变量、函数、类，需要在使用时加上模块.的格式。请阅读主程序main.py的代码注释：

```
# 这是主程序main.py
# 请阅读代码注释

import test  # 导入test模块

print(test.a)  # 使用“模块.变量”调用模块中的变量

test.hi()  # 使用“模块.函数()”调用模块中的函数

print(test.Go1.a)    # 使用“模块.类.变量”调用模块中的类属性
test.Go1.do1()  # 使用“模块.类.函数()”调用模块中的类方法

A = test.Go2()  # 使用“变量 = 模块.类()”实例化模块中的类
print(A.a)  # 实例化后，不再需要“模块.”
A.do2()  # 实例化后，不再需要“模块.”
```

到这里，我们来做个练习。下面有一段代码：

```
sentence = '从前有座山，'

def mountain():
    print('山里有座庙，')

class Temple:
    sentence = '庙里有个老和尚，'
    @classmethod
    def reading(cls):
        print('在讲故事，')

class Story:
    sentence = '一个长长的故事。'
```

```
    def reading(self):
        print('讲的什么故事呢？')

for i in range(10):
    print(sentence)
    mountain()
    print(Temple.sentence)
    Temple.reading()
    A = Story()
    print(A.sentence)
    A.reading()
    print()
```

请你把这段代码拆分到两个模块中，其中执行语句放到main.py文件，封装好的变量、函数和类放到story.py文件。

```
# 【文件：story.py】

sentence = '从前有座山，'

def mountain():
    print('山里有座庙，')

class Temple:
    sentence = '庙里有个老和尚，'
    @classmethod
    def reading(cls):
        print('在讲故事，')

class Story:
    sentence = '一个长长的故事。'
    def reading(self):
        print('讲的什么故事呢？')


# 【文件：main.py】

import story

for i in range(10):
    print(story.sentence)
    story.mountain()
    print(story.Temple.sentence)
    story.Temple.reading()
    A = story.Story()
    print(A.sentence)
    A.reading()
    print()
```

import语句还有一种用法是import…as…。比如我们觉得import story太长，就可以用import story as s语句，意思是为“story”取个别名为“s”。

上面的案例中，main.py文件可以写成这样：

```
import story as s

for i in range(10):
    print(s.sentence)
    s.mountain()
    print(s.Temple.sentence)
    s.Temple.reading()
    A = s.Story()
    print(A.sentence)
    A.reading()
    print()
```

另外，当我们需要同时导入多个模块时，可以用逗号隔开。比如import a,b,c可以同时导入“a.py，b.py，c.py”三个文件。

好，学完了import语句，我们接着学习from … import …语句。

from … import … 语句

from … import …语句可以让你从模块中导入一个指定的部分到当前模块。格式如下： 我们来看一个例子：

```
# 【文件：test.py】
def hi():
    print('函数“hi”已经运行！')

# 【文件：main.py】
from test import hi  # 从模块test中导入函数“hi”
hi()  # 使用函数“hi”时无需加上“模块.”前缀
```

运行一下试试

```
from test import hi # 从模块test中导入函数“hi”
hi()  # 使用函数“hi”时无需加上“模块.”前缀
```

当我们需要从模块中同时导入多个指定内容，也可以用逗号隔开，写成from xx模块 import a,b,c的形式。我们再运行一个小案例。

```
from test import a,hi,Go1,Go2
print(a)  # 打印变量“a”
hi()  # 调用函数“hi”
print(Go1.a)  # 打印类属性“a”
Go1.do1()  # 调用类方法“Go1”
A = Go2()  # 实例化“Go2”类
print(A.a)  # 打印实例属性“a”
A.do2()  # 调用实例方法“do2”
```

对于from … import …语句要注意的是，没有被写在import后面的内容，将不会被导入。

比如以下代码将会报错，因为使用了没有被导入的函数：

```
from test import hi # 从模块test中导入函数“hi”
hi()


# 以下语句将会导致报错，因为并没有导入test模块，只是导入test模块中的函数“hi”

test.hey()
```

当我们需要从模块中指定所有内容直接使用时，可以写成【from xx模块 import *】*的形式*，代表“模块中所有的变量、函数、类”，我们再运行一个小案例。

```
from test import *


print(a)  # 打印变量“a”

hi()  # 调用函数“hi”
print(Go1.a)  # 打印类属性“a”
Go1.do1()  # 调用类方法“Go1”

A = Go2()  # 实例化“Go2”类
print(A.a)  # 打印实例属性“a”
A.do2()  # 调用实例方法“do2”
```

不过，一般情况下，我们不要为了图方便直接使用【from xx模块 import *】的形式。

这是因为，模块.xx的调用形式能通过阅读代码一眼看出是在调用模块中的变量/函数/方法，而去掉模块.后代码就不是那么直观了。

到这里我们学完了from … import …语句，再来做一个练习吧。请看以下代码：

```
#  【文件：story.py】
```

```
sentence = '从前有座山，'

def mountain():
    print('山里有座庙，')

class Temple:
    sentence = '庙里有个老和尚，'
    @classmethod
    def reading(cls):
        print('在讲故事，')

class Story:
    sentence = '一个长长的故事。'
    def reading(self):
        print('讲的什么故事呢？')

# 【文件：main.py】

import story

for i in range(10):
    print(story.sentence)
    story.mountain()
    print(story.Temple.sentence)
    story.Temple.reading()
    A = story.Story()
    print(A.sentence)
    A.reading()
    print()
```

题目要求：在main.py文件导入story模块，将类Temple的属性'庙里有个老和尚，'打印出来。

直接使用Temple类的属性，我们需要在导入的时候指定Temple类（不能直接指定类属性）。老师的答案是这样的：

```
# 【文件：story.py】

sentence = '从前有座山，'

def mountain():
    print('山里有座庙，')

class Temple:
    sentence = '庙里有个老和尚，'
    @classmethod
    def reading(cls):
        print('在讲故事，')

class Story:
```

```
    sentence = '一个长长的故事。'
    def reading(self):
        print('讲的什么故事呢？')


#  【文件：main.py】

from story import Temple

print(Temple.sentence)
```

学会了import语句和from … import …语句后，你就能愉快地导入模块了。我们接着学习下一个语句。

if **name** == '**main**'

为了解释什么是if **name** == '**main**'，我先给大家讲解一个概念"程序的入口"。

对于Python和其他许多编程语言来说，程序都要有一个运行入口。

在Python中，当我们在运行某一个py文件，就能启动程序 ——— 这个py文件就是程序的运行入口。

拿我们刚才的课堂练习为例： 更复杂的情况，我们也可以运行一个主模块，然后层层导入其他模块：
但是，当我们有了一大堆py文件组成一个程序的时候，为了【指明】某个py文件是程序的运行入口，我们可以在该py文件中写出这样的代码：

```
#  【文件：xx.py】

代码块 ①……

if __name__ == '__main__':
    代码块 ②……
```

这句话的意思是这样的： 这里的if **name** == '**main**' 就相当于是 Python 模拟的程序入口。Python 本身并没有规定这么写，这是一种程序员达成共识的编码习惯。

第一种情况：加上这句话后，程序运行效果不会变化，我们来试试：

```
import story

if __name__ == '__main__':
    print(story.sentence)
    story.mountain()
    print(story.Temple.sentence)
    story.Temple.reading()

    A = story.Story()
    print(A.sentence)
    A.reading()
    print()
```

我们解释了“当xx.py文件被直接运行时，代码块②将被运行”，再解释一下“xx.py文件作为模块是被其他程序导入时，代码块②不被运行。”

我们来运行体验两段代码。这是第一段： 第一段代码没有使用if **name** == '**main**'，所有语句都会被运行。

这是第二段：（如果你看到运行结果什么都没有，那就是对的，请点击继续按钮）

现在我们运行代码的时候，会发现if **name** == '**main**'下的语句不会被执行。这是因为B.py文件并不是我们现在的程序运行入口，它是被A.py文件导入的。

关于这一个点目前你只需有个印象即可。接下来，我们来看看如何“使用他人的模块”。

使用他人的模块

在之前的项目实操中，我们常常用到这样的语句：

```
import time

print('第一句话，过两秒出现第二句。')
time.sleep(2)
print('第二句话。')
```

```
第一句话，过两秒出现第二句。
第二句话。
```

```
import random

a = random.randint(0,100)  # 随机从0-100之间抽取一个数字
print(a)
```

```
62
```

这两个例子中的第一句代码import time和import random其实就是在导入time和random模块。

初探借用模块

time模块和random模块是Python的系统内置模块，也就是说Python安装后就准备好了这些模块供你使用。

此外，Python作为一门胶水语言，一个强大的优势就是它拥有许多第三方的模块可以直接拿来使用。

如果是第三方编写的模块，我们需要先从Python的资源管理库下载安装相关的模块文件。

下载安装的方式是打开终端，Windows用户输入pip install + 模块名；苹果电脑输入：pip3 install + 模块名，点击enter即可。（需要预装python解释器和pip）

比如说，爬虫时我们会需要用到requests这个库（库是具有相关功能模块的集合），就需要在终端输入pip3 install requests(Mac用户)的指令。 第三方模块的使用我们会在之后的其他课程具体介绍，今天我们主要来学习Python的内置模块。

如果内置模块是用Python语言编写的话，就能找到py文件： 我们用命令random.**file**找出了random模块的文件路径，就可以去打开查看它的代码：

```
rom warnings import warn as _warn
from types import MethodType as _MethodType, BuiltinMethodType as
_BuiltinMethodType
from math import log as _log, exp as _exp, pi as _pi, e as _e, ceil as
_ceil
from math import sqrt as _sqrt, acos as _acos, cos as _cos, sin as _sin
from os import urandom as _urandom
from _collections_abc import Set as _Set, Sequence as _Sequence
from hashlib import sha512 as _sha512
import _random

__all__ =
["Random","seed","random","uniform","randint","choice","sample",
           "randrange","shuffle","normalvariate","lognormvariate",
           "expovariate","vonmisesvariate","gammavariate","triangular",
           "gauss","betavariate","paretovariate","weibullvariate",
           "getstate","setstate", "getrandbits",
           "SystemRandom"]
```

```
NV_MAGICCONST = 4 * _exp(-0.5)/_sqrt(2.0)
TWOPI = 2.0*_pi
LOG4 = _log(4.0)
SG_MAGICCONST = 1.0 + _log(4.5)
BPF = 53        # Number of bits in a float
RECIP_BPF = 2**-BPF

class Random(_random.Random):
    VERSION = 3     # used by getstate/setstate

    def __init__(self, x=None):
        self.seed(x)
        self.gauss_next = None

    def seed(self, a=None, version=2):
        if a is None:
            try:
                # Seed with enough bytes to span the 19937 bit
                # state space for the Mersenne Twister
                a = int.from_bytes(_urandom(2500), 'big')
            except NotImplementedError:
                import time
                a = int(time.time() * 256) # use fractional seconds

        if version == 2:
            if isinstance(a, (str, bytes, bytearray)):
                if isinstance(a, str):
                    a = a.encode()
                a += _sha512(a).digest()
                a = int.from_bytes(a, 'big')

        super().seed(a)
        self.gauss_next = None
……
……
```

由于代码太长，我们就不全部展示了。不过可以看到，random模块的源代码是这样的结构： 我们熟悉的函数random.choice(list)，功能是从列表中随机抽取一个元素并返回。它的代码被找到了：

```
def choice(self, seq):
    """Choose a random element from a non-empty sequence."""
    try:
        i = self._randbelow(len(seq))
    except ValueError:
        raise IndexError('Cannot choose from an empty sequence')
    return seq[i]
```

另一个熟悉的函数random.randint(a,b)，功能是在a到b的范围随机抽取一个整数。它的代码也被找到了：

```
def randint(self, a, b):
    """Return random integer in range [a, b], including both end
points."""
    return self.randrange(a, b+1)
```

像这样，通过阅读源代码我们能找到所有能够使用的变量、函数、类方法。

虽然你可以通过看源代码的方式来理解这个模块的功能。但如果你想要高效地学会使用一个模块，看源代码并不是最佳选项。我们接着谈谈“如何自学模块”。

如何自学模块

学习模块的核心是搞清楚模块的功能，也就是模块中的函数和类方法有什么作用，以及具体使用案例长什么样。

用自学“random”模块为例，如果英文好的同学，可以直接阅读官方文档：https://docs.python.org/3.6/library/random.html

或者也可以直接百度搜索： 搜到教程后，我们重点关注的是模块中的函数和类方法有什么作用，然后把使用案例做成笔记（还记得第8关谈到的如何做学习笔记么？）。

例如random模块的关键知识（也就是比较有用的函数和类方法），可以做成这样的笔记：

```
import random  # 调用random模块

a = random.random()  # 随机从0-1之间抽取一个小数
print(a)

a = random.randint(0,100)  # 随机从0-100之间抽取一个数字
print(a)

a = random.choice('abcdefg')  # 随机从字符串/列表/字典等对象中抽取一个元素（可能会重复）
print(a)

a = random.sample('abcdefg', 3) # 随机从字符串/列表/字典等对象中抽取多个不重复的元素
print(a)

items = [1, 2, 3, 4, 5, 6]  # “随机洗牌”，比如打乱列表
random.shuffle(items)
print(items)
```

```
0.1616700776898432
36
f
['c', 'g', 'f']
[4, 3, 6, 1, 2, 5]
```

另外，我们还可以使用dir()函数查看一个模块，看看它里面有什么变量、函数、类、类方法。

```
import random
print(dir(random))
```

```
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random',
'SG_MAGICCONST', 'SystemRandom', 'TWOPI', '_BuiltinMethodType',
'_MethodType', '_Sequence', '_Set', '__all__', '__builtins__',
'__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', '_acos', '_bisect', '_ceil', '_cos', '_e',
'_exp', '_inst', '_itertools', '_log', '_os', '_pi', '_random',
'_sha512', '_sin', '_sqrt', '_test', '_test_generator', '_urandom',
'_warn', 'betavariate', 'choice', 'choices', 'expovariate',
'gammavariate', 'gauss', 'getrandbits', 'getstate', 'lognormvariate',
'normalvariate', 'paretovariate', 'randint', 'random', 'randrange',
'sample', 'seed', 'setstate', 'shuffle', 'triangular', 'uniform',
'vonmisesvariate', 'weibullvariate']
```

这就像是查户口一样，可以把模块中的函数（函数和类方法）一览无余地暴露出来。对于查到的结果"**xx**"结构的，它们是系统相关的函数，我们不用理会，直接看全英文的函数名即可。

这样查询的好处是便于我们继续搜索完成自学。比如我们在列表中看到一个单词"seed"，我们就可以搜一搜random.seed的用法： 甚至不是模块，我们也可以用这种方式自学：dir(x)，可以查询到x相关的函数，x可以是模块，也可以是任意一种对象。

```python
a=''
print('字符串：')
print(dir(a))

a = []  # 设置一个列表
print('列表：')
print(dir(a))    # 把列表相关的函数展示出来

a = {}  # 设置一个字典
print('字典：')
print(dir(a))  # 把字典相关的函数展示出来
```

```
字符串：
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold',
'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii',
'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
列表：
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__',
'__imul__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append',
'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
字典：
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear',
'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem',
'setdefault', 'update', 'values']
```

好，我们回到模块上。再次总结一下模块的学习方法，其实可以归纳成三个问题： 例如数据分析需要用到pandas和NumPy模块，网页开发要用到Django模块等等，这些大型模块最好还是在课程上系统学习，避免散乱的学习形不成知识体系。

以目前大家的水平来说，有一些很实用，又不难学的模块，已经足够我们探索了。在今天的最后，我想手把手带大家自学一个模块。

学习csv模块

今天想带大家学习的模块是csv模块。之所以教大家这个模块，是因为这个模块既简单又实用。

csv是一种文件格式，你可以把它理解成"简易版excel"。学会了csv模块，你就可以用程序处理简单的电子表格了。 如果要手动新建csv文件，我们可以先新建一个excel表格，然后选择另存为"csv"格式即可。 同样的，当我们有了一张csv格式的表格，我们也可以选择另存为"excel"格式。

对csv文件的介绍就到这里。下面继续学习如何用csv模块读写csv文件。

我们使用import语句导入csv模块，然后用dir()函数看看它里面有什么东西：

```
import csv

# dir()函数会得到一个列表，用for循环一行行打印列表比较直观
for i in dir(csv):
    print(i)
```

```
Dialect
DictReader
DictWriter
Error
OrderedDict
QUOTE_ALL
QUOTE_MINIMAL
QUOTE_NONE
QUOTE_NONNUMERIC
Sniffer
StringIO
_Dialect
__all__
__builtins__
__cached__
__doc__
__file__
__loader__
__name__
__package__
__spec__
__version__
excel
excel_tab
```

```
field_size_limit
get_dialect
list_dialects
re
reader
register_dialect
unix_dialect
unregister_dialect
writer
```

同时，我们可以搜索到csv模块的官方英文教程：

https://docs.python.org/3.6/library/csv.html

中文教程：https://yiyibooks.cn/xx/python_352/library/csv.html#module-csv

如果你要通过阅读这份教程来学习csv模块的话，最简单的方式是先看案例（拉到教程的最后），遇到看不懂的函数，再倒回来查具体使用细节。 现在我们也跟着案例动手试试如何读取csv文件，可见open()后面跟了两个参数，用csv.reader(文件变量)创建一个reader对象。

我们新建了一个名为test.csv的文件，里面的内容是这样： 然后我们运行learn_cse.py文件的几行代码，你就能print出csv文件中的每一行信息。

```
import csv
with open("test.csv",newline = '')  as f:
    reader = csv.reader(f)
    #使用csv的reader()方法，创建一个reader对象

    for row in reader:
    #遍历reader对象的每一行
        print(row)

print("读取完毕！")
```

我们可以看到，终端输出的每一行信息都是一个列表。

我们来做一个练习，下面有一个csv文件： 以上就是读取csv文件的写法，接下来我们来看如何往csv格式文件写入数据。

写入数据的方式是这样的： 先创建一个变量名为writer（也可以是其他名字）的实例，创建方式是writer = csv.writer(x)，然后使用writer.writerow(列表)就可以给csv文件写入一行列表中的内容。

另外关于open函数的参数，也就是图中的'a'，我们来复习一下： 我们来做一个练习，还是这个商品列表的csv文件： 你来试试用writerow()方法为它追加写入两行列表吧：('4', '猫砂', '25', '1022', '886')、('5', '猫罐头', '18', '2234', '3121')。

```
import csv
with open('test.csv','a', newline='',encoding='utf-8') as f:
    writer  = csv.writer(f)
    writer.writerow(['4', '猫砂', '25', '1022', '886'])
    writer.writerow(['5', '猫罐头', '18', '2234', '3121'])
```

到这里，最基本的csv表格读取和录入方法我们就已经学会了。csv模块虽然比random模块稍微复杂一点点，但按照模块三问（这模块有哪些函数可用？有哪些属性或方法可用？使用格式是什么？）的学习方式，我们一样可以学会它的基本用法。

这里先卖一个关子。后面的两个实操项目，我们会展示如何应用csv模块的相关知识。

最后，对今天的知识做个总结：

```

```