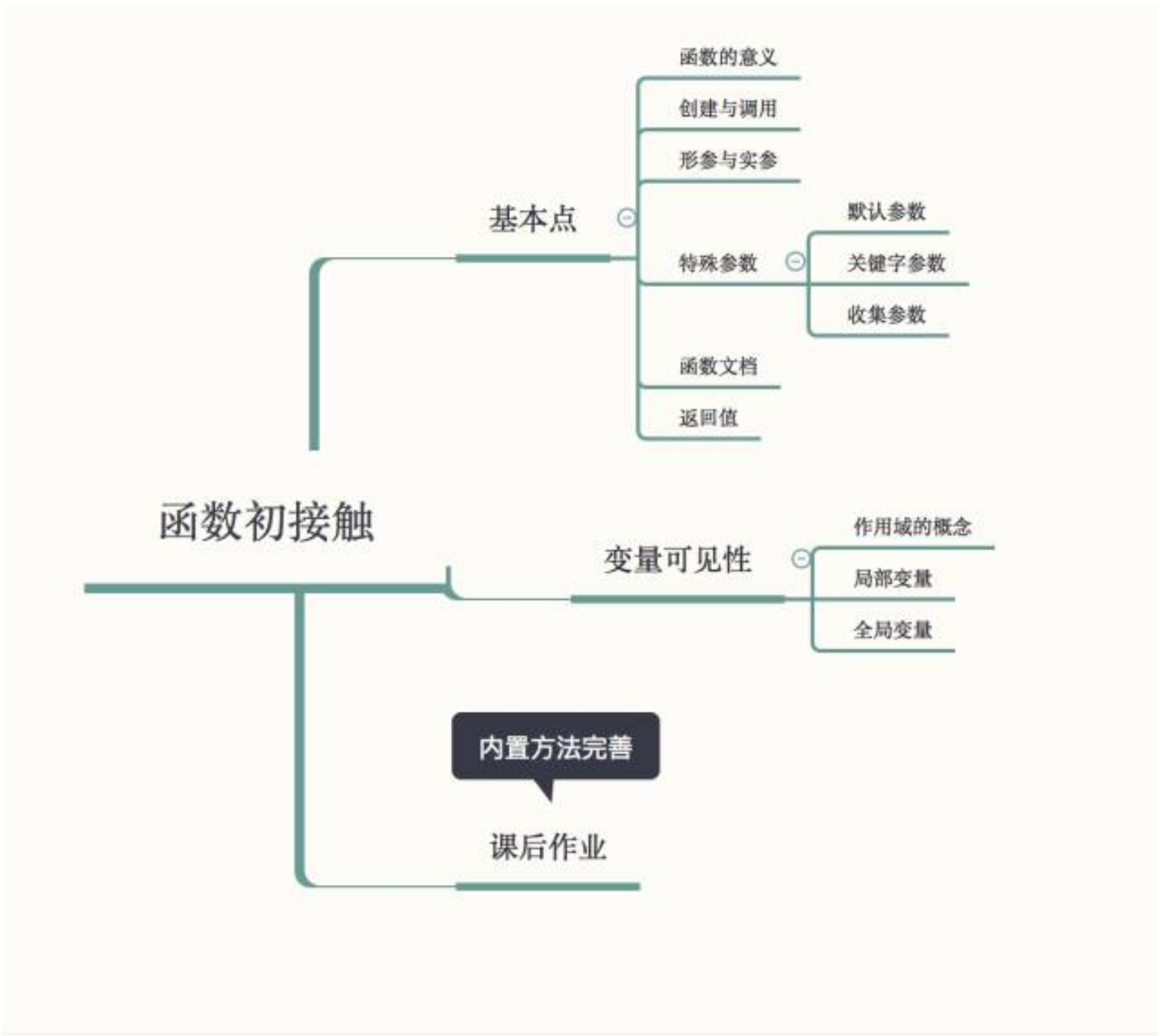
Python第五课函数（上）－小明同学的一天

Original 2016-10-29 哈哈小 菜猿观世界

加关注这种话银家怎么好意思说出口嘛

通过本节的学习，你将了解以下内容：



一，基本点

1，函数的意义

对于函数的意义是什么这个问题，本应该放在最后解答。

但哈哈小想偷懒，在这稍提一下就完事了。前面几节课接触过的一些内置方法，也就是我将要讲到的函数，像字符串七七八八一大推，还有上节课讲的序列一些好用的内置方法，想必大家通过对这些函数的使用，对函数的用处也能感知一二。还有，通过学节的学习，相信大家对函数意义的理解自有一番长进。

虽然，前面我们多次用到函数，但是我们用到的都是python小伙伴预先就给我们定义好的。到目前为止，我们只了解到有的函数好像要传参数，有的好像有返回值。至于函数是怎么被定义的，怎么实现的等等这些具体问题，我们还是懵懵懂懂。

好吧，接下来，我们就对函数来一探究竟。

2，创建与定义

今天哈哈小决定换一种方式来对本节的知识点进行讲解。

于是就把昨天隔壁老张家正在读三年级的小明同学与我分享的某一篇日记回忆起来，转述如下：

清晨时分，已是旭日高照，更令人惊喜的是，昨天还是雾浓雨霏，现在已是天朗气清，碧洁如洗。

这次，久违的明净偕凛冽的寒风一同到来，尽管这可能只是短暂的停留，但久别重逢，也甚是令人舒畅不已。

这时，闹钟作响，小明就此开始了新的一天，如同往日。

三年级的小明同学起床后要做的第一件事就是洗漱。

等等，哈哈小这里有话说，如果把小明同学这个事件的具体过程用程序来描述，第一感觉会是这样：

```
print("开始刷牙\n")
print("开始洗脸\n")
print("洗漱完毕！\n")
```

但是，既然本节学的是函数，当然要换成下面这种方式：

```
>>> def doFirstThing():
print("开始刷牙\n")
print("开始洗脸\n")
print("洗漱完毕！\n")
```

这是什么意思？ 第一次见到，好的，重点来了，划起啊。

在python中定义一个函数的语法：

```
def 函数名():
函数体
```

语法定式看起来虽简单的，但作进一步解释也是必要的。def为python的关键字，define的前三个字母，表示定义，后面接函数名，连起来就是define 某个函数，并给它取个名。如此地接近人类语言的编程语言不仅高级的傲娇，而且在助于理解上也毫不含糊。

第一行的：符号以及缩紧取代大括号等这些规范，同python其它像if，while等语句块。函数体就是写上这个函数所要执行的功能。

到现在为止，好像函数只是定义了，但是好像没被使用，去使用函数我们叫做调用。就好像，print这个函数可以直接为我们所用，就是因为它事先已经被系统定义在那，我们要其发挥真正的功能和效果，就需要去调用它。

所以，依print画doFirstThing，要去调用这个叫做doFirstThing的函数，很简单，一句代码即可。

```
>>> doFirstThing()
开始刷牙
开始洗脸
洗漱完毕！
>>>
```

讲到这里有的同学要发问了，并不感觉用函数这种方式，代码没简洁到哪里去。哈哈小回应：要是小明明天，后天，大后天要洗漱的时候，你就知道好到那里去了，因为只要一个函数被定义后，就可以不断地复

用，这样程序的结构性和简洁性都会得到提高。

3，实参与形参

这时又一同学发问了，多次使用，我使用循环不就行了，使用几次循环几次，何必这么麻烦?

这位同学的问题非常好，哈哈小的回应是：

要是小明的同学小红，小松，小鹿也要洗漱这么办?

看函数怎么应对这个问题：

```
>>> def doFirstThing(name):
print(name+"开始刷牙\n")
print(name+"开始洗脸\n")
print(name+"洗漱完毕！\n")

>>> doFirstThing('小明')
小明开始刷牙
小明开始洗脸
小明洗漱完毕!
>>> doFirstThing('小鹿')
小鹿开始刷牙
小鹿开始洗脸
小鹿洗漱完毕!
>>>
```

谁要洗漱就传谁，第一种方式表示无能为力，不知所措。

我们来分析一下上面的代码，函数名后面小括号里的name表示为该函数的设定一个参数，如果要设定多个参数，多个参数之间用逗号隔开，例如doFirstThing(name，time)。这里还需要明确两个概念，函数定义时设定的参数叫做形式参数，简称形参，顾名思义，他们只是一种形式的存在。后面调用函数传入的参数像'小明''小鹿'这些叫做实际参数，简称实参，故名再思义，这些参数才是含有具体的实实在在的活生生的具体值。

4，特殊参数

哈哈小不妨多絮叨几句，除了上面提到的几乎所有编程语言而适用之的参数，我们再来看看在python中，还可以对参数做哪些特殊操作。

1）默认参数－为形参赋初值

假如哈哈小定义带了name和time两个参数的doFirstThing函数。就小明及其同学洗漱这件事，因为几乎每天都要上学，所以洗漱时间可以是不变的，假设是早上七点。如果是这种情况，每次调用都去给time传值，那显得有点多余多事。

没关系，python提供的默认参数机制能解决这个问题，具体是这样做的：

```
>>> def doFirstThing(name，time='7:00'):
print(name+time+"开始刷牙\n")
print(name+time+"开始洗脸\n")
print(name+time+"洗漱完毕！\n")
```

```
>>> doFirstThing('小明')
小明7:00开始刷牙
小明7:00开始洗脸
小明7:00洗漱完毕!
>>> doFirstThing('小红','8:00')
小红8:00开始刷牙
小红8:00开始洗脸
小红8:00洗漱完毕!
>>>
```

time就此变成了默认参数，默认值为‘7:00’，语法是：time＝'7:00'。对于小明而言，是个坚强boy，能保证每天准时起床，准时刷牙，所以没有必要多此一举去给time再传值。但是小红不知咋地，当时可能是闹钟没响或者昨天作业做的太晚，今天就没能准时起来，晚了一点，所以只好另给时间了，这样原先默认的7:00这个时间就只好被覆盖了。

默认参数其实我们之前也有多次遇到，不知道大家还记不记得sorted这个方法。

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

sorted(iterable, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customise the sort order, and the
    reverse flag can be set to request the result in descending order.

>>>
```

由上图可以看出，sorted函数有两个默认参数，key和reverse，默认值分别为None和False。

2）关键字参数

小鹿也来一次吧：

```
>>> doFirstThing('8:00','小鹿')
8:00小鹿开始刷牙
8:00小鹿开始洗脸
8:00小鹿洗漱完毕！
>>>
```

哎呀，犯糊涂了，不小心把顺序敲错了。这现在仅仅是两个参数，都免不了犯错，如果有十几个参数或更多，这个问题该怎么应对？

关键字参数说他能解决，我们来看看。

```
>>> doFirstThing(time='8:00',name='小鹿')
小鹿8:00开始刷牙
小鹿8:00开始洗脸
小鹿8:00洗漱完毕！
>>>
```

简单，so easy。只要把参数值和形参名对应好，这样小鹿的妈妈再也不用担心小鹿考试的时候把b看成6了。

3）收集参数

既然小明与小鹿是同一时间点洗漱（或者还有很多别的同学），我们何不调用一次，把事件过程整合起来表现？ 如果真能做到，那我们的代码又会简化不少。

收集参数就是专门做这种事的，找他保没错。

```
>>> def doFirstThing(*name,time='7:00'):
print("三年级二班现在在洗漱的人:",len(name),end="\n")
print(name[0]＋","+name[1]+time+"开始刷牙\n")
print(name[0]+","+name[1]+time+"开始洗脸\n")
print(name[0]+","+name[1]+time+"洗漱完毕！\n")

>>> doFirstThing('小明','小鹿')
三年级二班现在在洗漱的人: 2
小明,小鹿7:00开始刷牙
小明,小鹿7:00开始洗脸
小明,小鹿7:00洗漱完毕！
>>>
```

name参数前面多了一个＊，表示该参数为收集参数，也就是可变参数。在本例中，该可变参数传入两个值，分别是小明和小鹿。在函数体中，可以使用len，name［0］等方式对name进行操作，好像与我们前面学过的序列很相似。其实没错，收集参数的原理就是，将所有传入的参数打包成一个元组，并放在可变参数名字中。如果后面还有自己的其他的参数，要用可变参数或默认参数，如：

```
doFirstThing('小明','小鹿',time＝'8:00')
```

5，函数文档

python提供的对参数操作的强大功能，终于使我们的函数趋于完美，尽如人意了。

接下来还有一点收尾工作，完成之后，可以说，就完美了。

不知道大家有没有注意到，上文给大家展示的help（print）的输出内容中，print(...) 下方有一长串文字描述，这就是说函数文档。

我们也可以用这种方式查看函数文档：

```
>>> print.__doc__
"print(value, ..., sep=' ', end='\\n', file=sys.stdout, flush=False)\n\nPrints the values to a stream, or to sys.stdout by default.\nOptional keyword arguments:\nfile:  a file-like object (stream); defaults to the current sys.stdout.\nsep:   string inserted between values, default a space.\nend:   string appended after the last value, default a newline.\nflush: whether to forcibly flush the stream."
>>>
```

函数文档，只是起到对函数文档说明的作用，这个函数的功能是什么啊，有多少个参数啊，分别是什么意思啊等等这些信息都有可能包含在函数文档中。记住函数文档跟注释很相似，对代码的执行不会产生任何的影响。

为方面以后更明白地使用我们写的这个doFirstThing函数，现在哈哈小想对这个函数做一个文档说明，做法是这样的。

设置函数文档

```
>>> def doFirstThing(*name,time='7:00'):
  '这是一个三年二班所用同学7:00起床做的第一件事的描述函数'
print("三年级二班现在在洗漱的人:",len(name),end="\n")
print(name[0]＋","+name[1]+time+"开始刷牙\n")
print(name[0]+","+name[1]+time+"开始洗脸\n")
print(name[0]+","+name[1]+time+"洗漱完毕！\n")
```

查看一下

```
>>> doFirstThing.__doc__
'这是一个三年二班所用同学7:00起床做的第一件事的描述函数'
>>>
```

只需内容引号包裹，放函数体第一行，简单，完成。

6，返回值

返回值，函数中又一个重要的概念。我们通过调用洗漱函数，洗刷函数就只是执行了我们给定的事件，发挥了一种功能。但是好像并没有返回任何值，真实的情况是这样的吗？我们不妨来看看：

```
>>> return_value＝doFirstThing('小明','小鹿')
三年级二班现在在洗漱的人: 2
小明,小鹿7:00开始刷牙
小明,小鹿7:00开始洗脸
小明,小鹿7:00洗漱完毕！
>>> print(return_value)
None
>>>
```

出乎所料，定义一个叫return_value的变量，然后打印出来，最后神奇般地打印出了None，

说明以上函数也是有返回值。在这里哈哈小要告诉大家的是，在python中，任何函数都有返回值，在不指定返回值的情况下，默认情况返回None。

给函数指定返回值，可以用以下方式：

据当时小明陈述，洗刷完毕，早餐过后，看时间所剩无几，甚恐迟到，于是就飞快地赶往学校。

干好掐着点感到教师，第一节课是能让小明醒醒脑的数学课。

当时课上有道计算圆面积的题令小明百思不得其解。

如果用一个函数来描述这一问题，会是这样的：

```
>>> def getCircleArea(radius,pi=3.14):
return pi*radius*radius
```

可以看到一个默认的单词return，即返回的意思。我们如果要对一个函数做返回值操作，就是用return关键字来设定的，记住return是一个函数结束的标志，就是不管return后面还有没有为执行的语句，当在函数体中遇

到return语句时，函数立马结束执行，而不再去执行后面的在函数体中还未执行的语句。

当时小明告诉我题目给出的半径为3，我们来调用试试看：

```
>>> area=getCircleArea(3)
>>> print('%.2f'%area)
28.26
>>>
```

area变量去接收函数的返回值，通过print语句保留两位小数将面积最终打印出。

接下来，小明上的课分别为语文课和英文课，在这两节课上，有几道默写题使小明不知所措。

语文默写：我达达的马蹄是美丽的错误，我不是归人，_____。

单词默写：奶奶：_____。

我们就干脆，连同上一道，来帮小明一并解答了。

```
>>> def answer():
return 28.25,'而是过客','grandmather'
>>> ans=answer()
>>> ans
(28.25, '而是过客', 'grandmather')
```

神奇的python居然做到了多个值的返回。上例显示的结果我们可以看出，python返回多个值时，是把这些值打包成一个元组放回的。

对于return语句，有无小括号无关紧要。

```
return （28.25,'而是过客','grandmather'）
```

同时你想它以列表打包的形式返回，你也可以这样写：

```
return [28.25,'而是过客','grandmather']
```

二，变量可见性

一天紧张而忙碌的学习结束之后，小明在回家路中，遇到几家服装店。小明寻思着，天气转凉，想起了对自己疼爱有加且年过古稀的奶奶一直缺着一件保暖衣，何不趁着这个时机为奶奶买件衣服。小明摸摸口袋，发现囊中羞涩，毕竟对于小学生来讲，身上能有的零用钱也不多。正苦恼于此，小明突然想起沿着这条街道直走100米东北角的一家店铺正巧在搞促销活动，何不去哪看看。

来到店铺，小明结合自己袋中数目，挑选了一件尺寸合适，价格合理， 质量不耐的衣服。

等等，这里哈哈小又要颇为扫兴地插上一番话了。

小明所购衣服的具体情况可用下面的程序来描述：

```
def getClothesPrice(price,rate):
    result=price*rate
    return result

original_price=float(input("original price:"))
rate=float(input("discount rate:"))
final_price=getClothesPrice(original_price,rate)
print("the final price:",final_price)
```

运行结果为：

```
original price:80
discount rate:0.8
the final price: 64.0
>>>
```

从程序的结果可以看出，我们定义了得到一个折扣衣服价格的函数。衣服原价时80，折扣率是0.8，最终衣服的价格为64。

1.局部变量

上例中，出现了许多变量。而这些变量的位置和可见范围正是哈哈小接下来要重点强调的。

像result变量，其位于函数体内，我们把这样一类变量叫做局部变量。既然是局部变量，依着其意思，大家应

该也能猜出其可见范围。局部变量在局部定义，故只能局部可见。如果我们试图在函数体外面将result变量打印出来，即print("the final price:",final_price)之后，添加一句print("the result:",result)。程序运行最后一行时将会报错。

原因在于，当程序在执行final_price=getClothesPrice(original_price,rate)这一行时，函数将会被创建到python的栈内存中（函数数据包括result变量这时会封装存在在内存中）。但是，当函数执行完成后，函数内容将不再存在内存中（python机制会这么做的）。所以，最后再试图去寻找result变量的时候，程序将会给你一个错误信息，说找不到这个变量了。

2.全局变量

与局部变量相对应，全局变量理解起来就好多了。

上例中的original_price，rate，final_price等函数外的这些定义在全局的变量，我们把这些变量叫做全局变量。全局变量的可见性是全局可见。

需要注意的是，全局变量可以在函数体中去使用，但是不要试图去修改它。

如果我在getClothesPrice 函数第一行添加这样一句：

print("the orignial price:",original_price)

尽管original_price是全局变量，但最后original_price还是会正常打印出来的，因为它是全局的，在函数中也是可见的。

如果我是试图去修改全局变量，看看会发生什么：

```python
def getClothesPrice(price,rate):
    orignal_price=60
    result=price*rate
    return result

original_price=float(input("original price:"))
rate=float(input("discount rate:"))
final_price=getClothesPrice(original_price,rate)
print("the final price:",final_price)
print("the orignal price:",original_price)
```

运行结果：

```
original price:80
discount rate:0.8
the final price: 64.0
the original price: 80.0
>>>
```

可以看到，尽管orignal_price在函数体的第一行被修改成了60，最后我在函数外打印original_price，其值依然是80.0，保持我们最初输入的值不变。这是因为，如果在函数中修改orignal_price，python机制会去创建一个同名的变量。尽管名字一样，但是意义大不一样，函数中original_price可以说是被创建出来的新的一个局部变量，所以函数外面的那个原本就存在的全局original_price变量其值就没有发生改变。

尽管在局部去修改一个全局变量是可行的，但是哈哈小强烈在这建议，在以后的python编程中，不要这么做。

“如果你的系统中大量使用全局变量，那么你的系统已经成功得到一个畸形的系统，它处于一个神秘的稳定状态！你看着这台机器，机器也看着你，相对无言，心中发毛”

话说当时小明把衣服交到奶奶手中，真是令奶奶感动不已，奶奶轻抚着小明的头，连连赞道：小明长大懂事了，小明长大懂事了...

小明也因此心花怒放，在与家人交流共处许久后，就独自一人屁颠屁颠地跑到书桌前完成一天所学之输出，所想之记录。

三，课后作业

上节课我们初步探索了内置方法的实现，在学过今天函数的部分知识之后，相信大家对内置方法的完整实现已经足够清晰明朗了。

哈哈小在这还是举一例，对与sorted（list1）这个内置方法，哈哈小会想到其中的一种实现是这样的：

```
def mySorted(num):
    for i in range(0,len(num)):
        mindex=i
        for j in range(i,len(num)):
            if num[mindex]>num[j]:
                mindex=j
            temp=num[i]
            num[i]=num[mindex]
            num[mindex]=temp
    return num
```

运行结果：

```
>>> a=mySorted([2,3,4,34,-3])
>>> a
[-3, 2, 3, 4, 34]
>>>
```

bingo，完全没问题。还有其他的像reverse（list1），min（tuple1）,list(str1)等等，纸上得来终觉浅，要知此事须躬行，所以赶紧去尝试吧。

结束语：我们不想给你带来多少多少的知识，我们只想尽绵薄之力给你带来可能的智慧的启迪。

想要获取课后习题答案，可以进入菜猿编程论坛网站（www.caiyuan1024.com）python板块提问寻要，点击阅读全文直达网站主页。

END

Read more