

## 2014 年辛星 Python 教程第一版

### PYTHON 3

编辑时间：2014 年 6 月 3 日星期二，如果您在三年之后看到了本教程，那就可以扔掉了，因为您绝对可以从网上搜索到它的更新的版本。（其实它的更新效率还是很高的）

说明：本教程是 Python3。

建议：百度搜索“辛星 python”可以获得更多更全的学习资料，我会上传到百度文库、百度网盘、CSDN、新浪爱问等大多数可以免费下载的地方去的。

版权：版权完全归属作者个人所有，可自由印刷复印传阅，但是必须注明版权，版权仍然归属作者本人。

附注：由于是第一版本，可能会存在很多这样或那样的问题，我会认真听取大家的意见，在后面的版本中做的更好。

说明：相比那些动辄数百页甚至一千多页的教材，它是小巧且简洁的，因为它没有啰嗦的教条的介绍。

相比那些只有五六页的教程，它是全面的，因为它更加深入细节。

但是它仍然有自己不全面的地方，比如异常等还没有包含进来，这也是为了保持简洁基础的特色。

---

## 目录:

### 第 0 部分: 了解 Python 及搭建环境

第 0 节: 认识 Python.....	5
第 1 节: 了解编程语言.....	7
第 2 节: 开始运行 Python.....	8

### 第 1 部分: 数据类型

第 0 节: 变量和注释以及数据类型简介.....	13
第 1 节: Python 中的布尔类型和空型.....	15
第 2 节: 数值类型.....	18
第 3 节: 列表.....	20
第 4 节: 元组.....	23
第 5 节: 集合.....	24
第 6 节: 字典.....	26
第 7 节: 字符串.....	27
第 8 节: 表达式.....	29

### 第 2 部分: 语句和语法

第 0 节: 语句的简介.....	31
第 1 节: 赋值语句.....	33
第 2 节: 条件语句.....	34
第 3 节: 循环语句.....	38
第 4 节: 循环语句中的其他语句.....	41

### 第 3 部分：函数、模块和命名空间

第 0 节：认识函数·····	47
第 1 节：函数的定义和调用·····	47
第 2 节：模块·····	52
第 3 节：命名空间·····	56

### 第 4 部分：面向对象

第 0 节：面向对象的思想·····	60
第 1 节：类·····	63
第 2 节：类的实例·····	66
第 3 节：类和实例的进一步解释·····	68
第 4 节：继承·····	72
第 5 节：多态·····	75
第 6 节：封装·····	76
第 7 节：最后的总结·····	77

结语·····	79
---------	----

## 第 0 部分：了解 Python 及环境搭建

### \*\*\*\*\*主要任务\*\*\*\*\*

- ✧ 了解下 Python 的优缺点。
- ✧ 搭建可以运行的环境。

## 第 0 节：认识 Python

### 第 1 节：了解编程语言

### 第 2 节：开始运行 python

## 第 0 节：认识 Python

虽然我很烦写这种东西，但是还是不得不介绍一下。Python 翻译成汉语的意思是“毒蛇”，但是这是一个可爱的毒蛇。

### \*\*\*\*\*Python 的优势\*\*\*\*\*

1. Python 的开发效率更快，实现相同的功能，python 代码的数量往往只有 C++或者 Java 代码的五分之一到三分之一。
2. Python 的跨平台性很好，相对于 C 和 C++来说，Python 的绝大多数程序都不需要改变即可在所有主流计算机平台上运行。
3. 强大的标准库。相对于早期语言的标准库很小，但是第三方库很多的情形。Python 的标准库很强大，而且还有着丰富的第三方库，它们涵盖了网站开发、数值计算、串口读写、游戏开发等各个方面。
4. 强大的集成功能。Python 被誉为“胶水语言”，可以和诸如 C 和 C++这样的语言编写的程序互相调用，从而发挥出各种编程语言的强项，而不是单纯的使用一种语言去解决所有的问题。

### \*\*\*\*\*Python 的缺点\*\*\*\*\*

1. Python 经过了数次优化，各方面均得到了质的提升。
2. 与 C 或者 C++这样的编译语言相比，它的运行速度不够快。

#### \*\*\*\*\*Python 的经典案例\*\*\*\*\*

1. YouTube 视频分享服务大部分都是用 Python 编写的。
2. EVE 这款游戏大量的使用了 Python。
3. Maya 提供了 Python API。

#### \*\*\*\*\*Python 能干什么 \*\*\*\*\*

1. 网络开发: Python 非常适合 web 开发, 而且它也有海量的框架方便快速开发。
2. 系统编程: 主要是和操作系统服务有关的相关任务, 是 Linux 下标志性语言之一, 也是很多系统管理员的首选编程语言。
3. 界面编程: 也就是 GUI 程序。Python 内置了 Tk, 还有大量的第三方库。
4. 数值计算和科学计算: 比如通过 NumPy 等扩展包来辅助实现更多的功能。
5. 游戏、图像、人工智能、XML、机器人等其他领域。

#### \*\*\*\*\*Python 的特点\*\*\*\*\*

1. 面向对象。很多语言都支持面向对象编程, python 也无疑做到了这一点。
2. 免费。这使得我们使用 Python 的时候很少会涉及商业上的纠纷。
3. 可移植。Python 的实现使用 C 编写的, 可以运行在所有的主流平台中。且 Python 语言实现的标准库和模块也都考虑到了跨平台。
4. 简单易学并且功能强大。

---

## \*\*\*\*\*Python 历史\*\*\*\*\*

1. Python 虽然给人一种新兴语言的感觉，但是它的历史比 Java 还要悠久。
2. Python 最早开始于 1989 年底，与 1991 年诞生了第一个公开发行人版。
3. 它的创始人 Guido van Rossum，

## 第 1 节：了解编程语言

### \*\*\*\*\*语言的类型\*\*\*\*\*

说明：下面的看不懂也没关系，当您看完这本书之后，下面的就会看明白了。

- 按时间分类：机器语言属于第一代语言，直接用 0 和 1 进行编程，由于过于古老，和恐龙一起消失了。第二代语言就是汇编语言了，直接写机器码，我想学习过汇编语言的都会深感它的痛苦，真正的痛苦还不是在写那些烦长的代码，而是一旦出现了错误进行调试的时候，那叫一个头大啊。第三代语言的主流语言就是 C、C++、Java、C#，当然还有 Python 和 PHP 等语言，Java 因为是一门完全面向对象的语言，也被称之为“三代半语言”，至于第四代语言，我也很期待。
- 按是否可以直接执行分类。可以分为解释型语言和编译型语言，当然如果严格来分的话，还可以加上标记型语言，比如 xml 和 xhtml 等等。编译型语言比如 C、C++ 等等，都是可以直接生成一个 exe 应用程序的，

直接双击打开，我们使用的大多数软件都是这种类型。而解释型语言需要一个解释器，比如 Java 需要一个 JRE 来解释运行时的一些东西，我们的 Python 就是解释型语言，所以，我们平时只需要双击.py 文件就可以运行了，但是如果我们没有安装 python 的机器上双击是没有效果的。再次提出 Java 是半解释半编译型语言，因此，java 的流行和它的复杂性也不是没关系。

- 按数据的定义角度来分。语言可以分为强类型语言和弱类型语言。如果说一个变量的类型没有强制改变它就是不能改变的，那这个语言就是强类型语言，反之就可以理解为弱类型语言了。这点我们到数据类型篇的时候再说吧。

## 第 2 节：开始运行 Python

### \*\*\*\*\*运行方式\*\*\*\*\*

- Python 的运行方式有两种：第一种是通过交互式的运行方式，通过“开始”->“所有程序”->“Python3.x”->“IDLE”运行。第二种是我们写好了 Python 文件然后双击运行即可。
- 上述的第一种方式通常是利用了 Python shell，它可以更加方便的调试小段的程序，并且支持语法高亮等一些特点，这是很多其他语言所不具备的。
- 我们打开 IDLE，可以看到有三个尖括号，即“>>>”，它是提示符，我们可以在后面输入命令或者表达式。比如我们输入 1+1，按下回车，那么下一行会自动出现一个 2。



- 如果我们在提示符后面输入 `print('helloworld')` 然后回车，那么下一行也会出现 `helloworld`。这就是我们的第一个应用程序。注意必须使用半角的英文标点符号，负责 shell 无法识别。
- 上面的 `print` 是什么意思呢？其实 python 自带了一个帮助系统，这一点也是远远优越于其他语言的。我们在提示符后面输入 `help()` 然后回车，此时进入了帮助模式，发现三个尖括号变成了 `help>`，此时我们在后面输入 `print` 然后回车，会发现下面回给出一些提示信息，我们可以阅读它来更好的理解。这里可以输入的东西很多，包括各种我们想知道的且和 python 有关的信息，但是我们胡乱输入比如“`xinguimeng`”是没作用的，大家可以试试。
- 要从上述的帮助模式退出回到正常的交互模式，只需要在 `'help>'` 后面输入 `help()` 然后回车即可。
- 我们将会有很长一段时间使用这种交互模式，而且这是很多主流语言比如 `c`, `cpp`, `java`, `csharp`, `php` 都么有的，很好用奥。

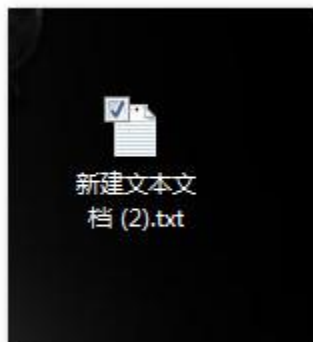
\*\*\*\*\*编辑器和 IDE\*\*\*\*\*

- 关于编辑器，也看各自的爱好，如果大神坚持用记事本，我也不说什么了。不过正常点的编辑器可以用 `EditPlus`, `Notepad++`, `Vim`, `Emacs` 什么的。
- IDE 翻译成汉语即“集成开发环境”，包括了编辑器，编译器等一切能包含的东西，可以说是能有的全有了。
- IDE 方面可以使用 `Eclipse`（需要下载插件），`PyDev`, `KomodoIDE` 等等。我还是推荐一下 `Sublime Text` 吧，谁用谁知道。

\*\*\*\*\*非交互式运行\*\*\*\*\*

- 很多时候我们不能再交互式模式下运行，那么应该怎么办呢？
- 我们可以先新建一个无格式的文本文件，例如是记事本文件，然后重命名为 `xxx.py`，这里的 `xxx` 可以自由填写，但是后缀名改为 `.py`。
- 然后双击运行打开，就可以了。如果大家出现一闪而过的情况，说明我们的程序过短，可以在最后一行加上 `input()` 来让程序接收键盘输入，这样大家可以有更多的时间查看。
- 实例：

第一步先新建一个记事本文件。



第二步重命名为 `first.py`，即修改后缀名。



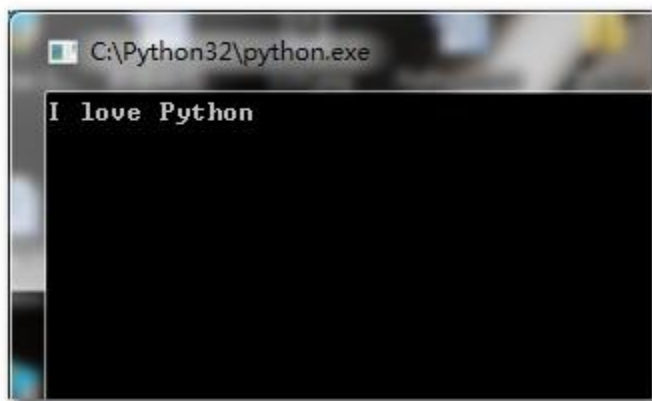
第三步单击右键，用自己喜欢的编辑器打开：



第四步开始编写代码（一定要注意这里使用英文标点符号，否则可能会出现各种各样的错误，大家使用 Ctrl+空格即可以调成英文输入 模式）：



第五步保存并且关闭该文件，然后双击打开：



## 第 1 部分：数据类型

\*\*\*\*\*主要任务\*\*\*\*\*

- 认识 Python 中的变量，以及了解数据类型。
- 自己会灵活使用变量和运算符。
- 了解什么是表达式

第 0 节：变量和注释以及数据类型简介

第 1 节：Python 中的布尔类型和空型

第 2 节：数值类型

第 3 节：列表

第 4 节：元组

第 5 节：集合

第 6 节：字典

第 7 节：字符串

第 8 节：表达式

第 0 节：变量和注释以及数据类型简介

\*\*\*\*\*变量\*\*\*\*\*

- ◆ 我们难免需要处理很多数据，这些在 python 中就是变量。
- ◆ 变量有一个名字和一个值，变量的名字一旦确定，就无法更改了，就像我们的身份证号一样，变量的值可以更改。
- ◆ 变量名用字母开头，后面跟若干字母或者数字或者下划线。比如 xin, gui, meng, xinxing 都是合法的变量名。

- ◆ 在 python 中没有“声明变量”这一说法，因为所有变量都不加声明的使用的。
- ◆ 变量的赋值用等号，比如我们让变量 `xin` 取值为 3，那么需要做的是：`xin = 3` 即可。

#### \*\*\*\*\*数据类型\*\*\*\*\*

- ☆ Python 中的每个数据都有一个类型，但是我们在使用一个变量的时候并不需要指定它的类型。
- ☆ Python 的数据类型有：布尔类型 (booleans)，数值类型 (包括整数，浮点数和复数)，字符串 (strings)，字节 (bytes)，字节数组 (bytes array)，列表 (lists)，元组 (tuples)，集合 (sets)，字典 (dictionaries)。

#### \*\*\*\*\*注释的引入\*\*\*\*\*

- 有时候我们不仅仅需要写代码，还需要写点东西来说明我们的代码时做什么的，或者说需要标记一些东西。
- 或者说我们有时候想让一段代码失效，这个时候就需要“注释”来起作用了。

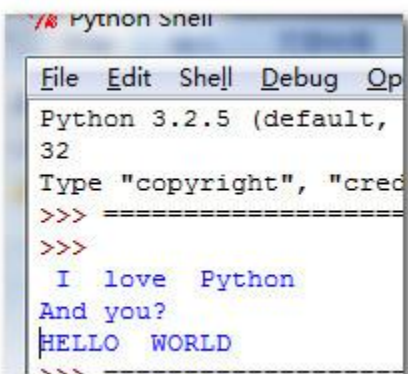
#### \*\*\*\*\*Python 中的注释\*\*\*\*\*

- 单行注释：常用的是“#”，也被称为 shell 风格或者 unix 风格的注释，只要我们使用了#，之后的内容直到这一行结束都不会起作用。
- 多行注释：如果我们要书写多行注释，即一行代码写不完，那我们可以开头用三个单引号开始，结束也使用三个单引号，这中间的内容会被当做注释来用。
- 说明：多行注释在编写函数、类或者模块的时候还有其独特的用法。
- 看下面的代码示例：



```
one.py - D:/Users/Administrator/Desktop/one.py
File Edit Format Run Options Windows Help
#这里已经变红了，它之后的内容可以随便写，欢迎百度
print(" I love Python")

print("And you?")
'''
这里我们使用了多行注释
下面的print会失效
print("Me too")
这些内容都不会起作用
'''
print("HELLO WORLD")
```

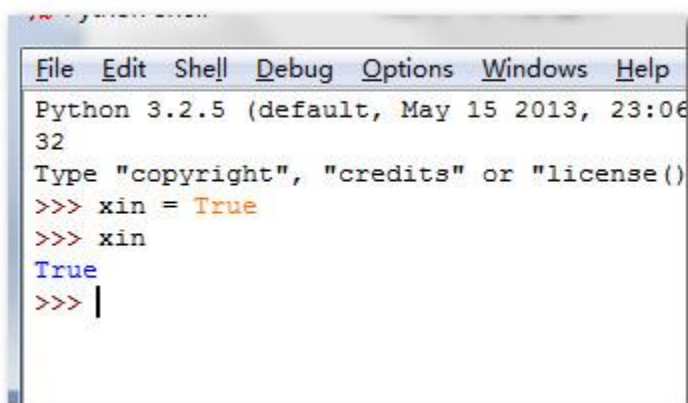


```
Python Shell
File Edit Shell Debug Op
Python 3.2.5 (default,
32
Type "copyright", "cred
>>> =====
>>>
I love Python
And you?
HELLO WORLD
>>>
```

## 第 1 节：python 中的布尔类型和空型

## \*\*\*\*\*布尔类型\*\*\*\*\*

- 布尔类型只有两个值，一个为真，一个为假，用于逻辑运算。为真的是 True，为假的是 False。
- 由于 Python2 的遗留问题，True 的结果为 1，False 为 0，这个不用管。
- 下面我们让一个变量赋值为真，然后我们查看它的值：



```
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06
32
Type "copyright", "credits" or "license()"
>>> xin = True
>>> xin
True
>>> |
```

## \*\*\*\*\*空型\*\*\*\*\*

- None 是 python 的一个特殊常量，表示没有。
- 它是一个空值，它不同于 False，False 表示假，但是 None 表示不知道。
- None 是唯一的空值，它的类型是 NoneType。

## \*\*\*\*\*逻辑运算\*\*\*\*\*


- ✧ 说到逻辑运算，可能会让初学者 一头雾水，但是逻辑运算的重要性也不言而喻。
- ✧ 所谓逻辑运算，即两个布尔变量进行运算（“与运算”和“或运算”），它得到的结果也是布尔变量。
- ✧ 与运算：两个布尔变量做“与”运算，它只有两个变量都为真的时候结果才为真，只要有一个变量取值为假，结果就为假。



- ✧ 或运算：该运算一旦有一个参与运算的布尔变量为真，结果就为真，如果参与运算的两个变量均为假，结果为假。
- ✧ 非运算：该运算直接取它的相反情况，它只能跟一个布尔变量，如果原变量为真，则进行运算后为假，如果原变量为假，则运算后变量为真。

\*\*\*\*\*逻辑运算符\*\*\*\*\*

- 上面说完了逻辑运算，那么想要在 Python 中表示这种运算，急需逻辑运算符出场了。
- 与运算我们通常用 “and” 表示，或运算用 “or” 表示，非运算用 “not” 表示。
- 下面是代码示例：



```
Python 3.2.5 (default, May 15 2013, 23:06:03)
32
Type "copyright", "credits" or "license()" fo
>>> a = True
>>> b = False
>>> a and b
False
>>> a or b
True
>>> not a
False
>>> |
```

## 第 2 节：数值类型

### \*\*\*\*\*数值类型\*\*\*\*\*

- ✧ 从上一节看到，数值类型本身就很庞大。包括整数，浮点数，还包括复数。
- ✧ 在 Python2 中，int 即整型和 long 即长整型是不同的数据类型，而且 int 也会因平台的不同而略有不同，但是 python3 中，整型只有一个，有点类似于 python2 的旧的 long。
- ✧ 浮点数可以理解为通常意义下的小数。但是 python 有一个模块专门支持分数，即 fraction 模块，这里先无视它，感觉并不算太常用。
- ✧ 其他计算机语言对于浮点数区分很严格，把浮点数分为单精度浮点数和双精度浮点数，即 float 和 double，但是在 python 下，我们这里不做区分。

- ✧ 下面的例子看出 xin 这个变量先被赋值为 7，然后查看，然后被赋值为 2，然后查看，最后被 赋值为 2.4，

```
>>> xin = 7
>>> xin
7
>>> xin = 2
>>> xin
2
>>> xin = 2.4
>>> xin
2.4
>>> |
```

然后查看：

\*\*\*\*\*数值的类型转换\*\*\*\*\*

- 我们可以强制数据类型的转换，比如用 `int(2.3)` 来得到 2，用 `int(-2.4)` 来得到 2.
- 下面是强制类型转换的例子：

```
>>> int(-2.5)
-2
>>> float(2)
2.0
>>> |
```

\*\*\*\*\*数值的运算\*\*\*\*\*

- ✧ 我们数学上学过的一些常见的运算符都是支持的，比如加减乘除用 `+-*/` 来表示。
- ✧ `/`运算符执行的是浮点数的除法，即两个数即使都是整数，得到的也是浮点数，在 Python2 中和 Python3 中有一定的变化。
- ✧ `//`对得到的结果进行取整，即高斯函数。
- ✧ `**`运算符是幂次运算符，如 `11**2` 即 11 的平方，为 121.
- ✧ `%`运算符是取模运算符，也可以叫做求余数运算符。比如 `11%2`，因为商为 5，余数为 1，所以得到的是 1.

### 第 3 节： 列表

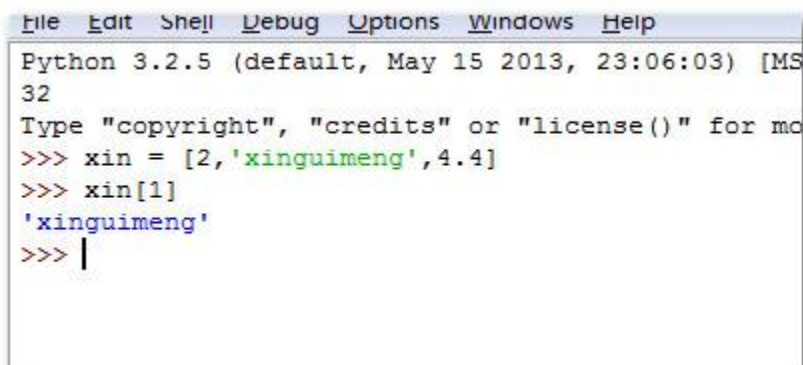
#### \*\*\*\*\*创建列表\*\*\*\*\*

- ✧ 列表创建非常简单，使用中括号包括一系列以逗号分隔的值即可了。
- ✧ 列表是元素的有序集合，注意它们是有顺序的，因此可以通过序号来查看。
- ✧ 列表有点类似其他语言中的数组，但是它不要求该列表中的值都相同的数据类型。
- ✧ 下面代码定义了一个列表（可以看出这个列表中既有字符串，也有整数和浮点数）：

```
>>> xin = ['xin',2,'xinguimeng',5.2]
>>> xin
['xin', 2, 'xinguimeng', 5.2]
>>> |
```

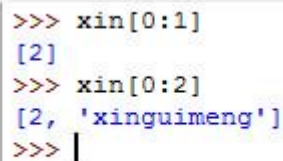
#### \*\*\*\*\*索引和切片\*\*\*\*\*

- ✓ 列表是有索引的，索引从 0 开始，然后逐次向上加一，比如我创建了一个索引 xin，然后查看它的第二个元素：



```
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MS
32
Type "copyright", "credits" or "license()" for mo
>>> xin = [2, 'xinguimeng', 4.4]
>>> xin[1]
'xinguimeng'
>>> |
```

- ✓ 索引也可以是负数，此时它是从后向前数的，具体大家可以自行测试。
- ✓ 所谓列表的切片，即中括号内不仅指定了头还指定了尾，记住要用冒号。



```
>>> xin[0:1]
[2]
>>> xin[0:2]
[2, 'xinguimeng']
>>> |
```

- ✓ 下面代码查看上述列表的切片：
- ✓ 如果切片的左索引不写，则默认为 0，右索引同理。
- ✓ 如果切片的左索引和右索引都不写，则默认是整个列表。

\*\*\*\*\*想列表中增加元素的四种方法\*\*\*

- ✧ 比如我先定义一个列表：xin = [2, 'xinguimeng', 4.4]
- ✧ 第一种方法：使用+号进行连接。如 xin = xin + [ 'a' ]
- ✧ 第二种方法：使用 append 方法，该参数接受的是列表元素。如 xin.append(True)
- ✧ 第三种方法：使用 extend 方法，需要注意的是，它接受的参数是一个列表，如 xin.extend(2)

- ✧ 第四种方法：使用 insert 方法，注意需要指定插入的序号。如 `xin.insert(1, 'I love xinguimeng')`
- ✧ 提前说一下把，列表是以类的形式实现的，创建一个列表就是将一个类实例化。

\*\*\*\*\*列表的操作\*\*\*\*\*

- 可以用 in 判断一个元素是否在一个列表中。如判断 'c' 是否在列表 xin 中，可以用 'c' in xin，示

```
>>> xin = [2, 'b']
>>> 'c' in xin
False
>>>
```

例：

- 可以用 index 函数判断一个元素在列表中的位置。如

```
>>> xin = [2, 'b']
>>> 'c' in xin
False
>>> xin.index('b')
1
>>> |
```

:

- 还有个 count 函数来计算某个元素出现的次数。

\*\*\*\*\*列表的元素的删除\*\*\*\*\*

- ✧ 可以用 del 从列表中删除元素，del 不算是函数，算是一个语法。
- ✧ 示例：`del xin[1]`
- ✧ 当某个元素被删除后，索引的编号会重新从 0 开始编排一次。
- ✧ 还可以使用 pop 函数，如果参数为空，则默认删除最后一个，该函数的返回值是删除的那个值。
- ✧ 对空列表使用 pop 将会发生例外。

- ✧ 空列表转换为布尔值的时候，会当做 False 用，非空列表为真。

## 第 4 节：元组

\*\*\*\*\*元组和列表的关系\*\*\*\*\*

- 记住一句话就可以了：元组是不可改变的列表。
- 而且列表在赋值的时候用中括号，元组用小括号。

\*\*\*\*\*元组的用法\*\*\*\*\*

- 元组的元素也有序号，也有切片，也可以使用 pop 等函数。
- 元组只是不能删除和添加元素，仅此而已。

\*\*\*\*\*元组的好处\*\*\*\*\*

- ◆ 元组的速度比列表快。
- ◆ 元组可以当做字典键来用。
- ◆ 元组可以实现一次赋多个值。比如：

```
>>> xin = ('a',2,9)
>>> (x,y,z) = xin
>>> x
'a'
>>> y
2
>>> z
9
>>> |
```

## 第 5 节： 集合

### \*\*\*\*\*集合的定义\*\*\*\*\*

- ✧ 集合是装有独特值的无序的袋子。
- ✧ 因此集合的三个特征为：集合内的值不能重复，集合内的值没有编号。
- ✧ 因此集合可以概括为三个特征：确定性、无序性、不可重复。
- ✧ 集合的值的类型没有规定。

### \*\*\*\*\*集合的创建\*\*\*\*\*

- 创建集合使用大括号。如 `xin = { 'a' , 2 }`
- 也可以把列表强制转换为集合。如 `xin` 为一个列表，则 `gui = set(xin)` 则 `gui` 为一个集合。
- 集合的内部实现是通过类完成的。
- 由于从 Python2 继承下来的古怪规定，不能用两个花括号创建一个空的集合，因为这实际上创建的是一个空的字典。

### \*\*\*\*\*向集合中添加元素\*\*\*\*\*

- ◆ 向集合中增加元素使用 `add` 函数, `add` 函数添加的是元素。如 `xin.add(2)`
- ◆ 可以用 `len` 函数来看一个集合的元素个数，如 `len(xin)`
- ◆ 向集合中添加元素还可以使用 `update` 函数，该函数向集合中添加的是集合，但是集合的元素不可能是一个集合，



必须是元素。因为在添加的时候，会把被添加的集合拆分为不同的元素，然后逐个添加。

\*\*\*\*\*从集合中删除元素\*\*\*\*\*

- ✓ `discard` 函数接受一个单值作为参数，如果该集合中没有该值，不会报错。如 `xin.discard(2)`
- ✓ `remove` 函数也接受一个单值作为参数，如果该集合中没有该值，将会引发异常，即一个 `KeyError`。
- ✓ `clear` 函数会清空该集合，如 `xin.clear()`。
- ✓ `pop` 函数会从集合中删除该值，并且会弹出它，这里 `pop` 出来的数据是随机的。
- ✓ 对空集合使用 `pop` 函数会报错。

\*\*\*\*\*集合的运算\*\*\*\*\*

- ◇ `in` 运算：判断一个元素是否在集合中。如： `2 in xin` 说明：这里 `xin` 是一个集合。
- ◇ `union` 运算：对两个集合求并集。如 `xin.union(gui)` 说明：这里 `xin` 和 `gui` 都是集合。
- ◇ `intersection` 方法：对两个集合求交集。
- ◇ `difference` 方法：相当于求差集。
- ◇ `symmetric_difference` 方法：相当于求对称差。
- ◇ `issubset` 方法：判断前者是否是后者的子集。
- ◇ `issuperset` 方法：判断前者是否是后者的超集。

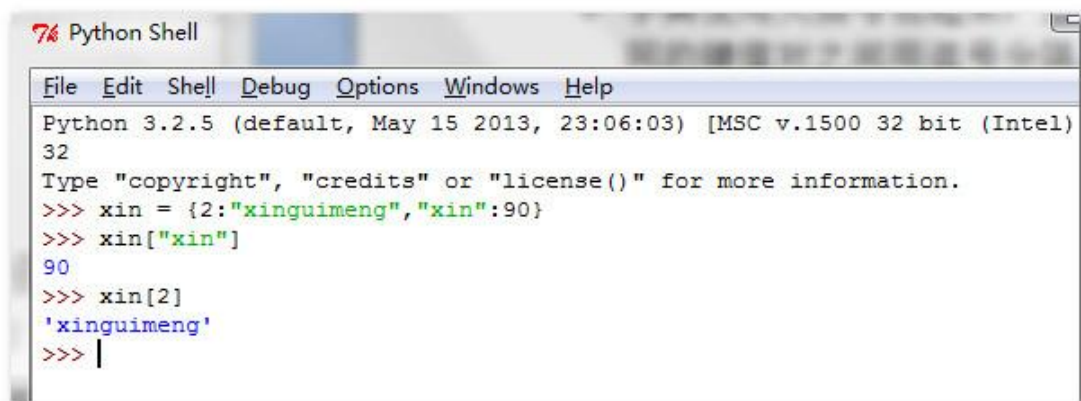
## 第 6 节：字典

### \*\*\*\*\*字典的定义\*\*\*\*\*

- 字典是键值对的无需集合。
- 向字典添加一个键的同时，必须为该键增添一个值。
- 字典有点类似于哈希表。

### \*\*\*\*\*字典的创建\*\*\*\*\*

- ✧ 字典使用大括号包起来，且键和值之间用冒号连接，不同的键值对之间用逗号分隔。
- ✧ 可以通过键来找到相应的值。



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)]
32
Type "copyright", "credits" or "license()" for more information.
>>> xin = {"2": "xinguimeng", "xin": 90}
>>> xin["xin"]
90
>>> xin[2]
'xinguimeng'
>>> |
```

### \*\*\*\*\*字典的特点\*\*\*\*\*

- 字典中不允许有重复的键。
- 在字典中修改键值对和新建键值对的语法是一样的，关键是看该键是否在字典的键中存在。

- 字典的值可以是各种数据类型，可以是字典。

\*\*\*\*\*字典的说明\*\*\*\*\*

- len 函数会返回字典的键值对的数目。
- 可以用 in 看键是否在字典中。

## 第 7 节：字符串

\*\*\*\*\*Unicode\*\*\*\*\*

- ✧ 在之前，可能很少人会在将字符串之前先提一下 unicode，但是随着全球化的今天，不提这个就没法混啦。
- ✧ Unicode 编码系统为表达任意语言的任意字符而设计出来的。它使用 4 字节的数字来表达每个字母、符号或者文字。
- ✧ Unicode 编码可以表示世界上所有的字符，但是如果没一个字符都用四个字节的的话，对很多语言而言，太过于浪费了。于是，期待着新的实现方案。
- ✧ 很多方案，比如 UTF-82，UTF-32，UTF-16 等等，但是最优秀的无疑还是 UTF-8。

---

**\*\*\*\*\*字符串的表示\*\*\*\*\***

- 字符串就是把一堆字符用单引号或者双引号包围起来，但是不能单引号和双引号同时用。比如 ‘xinguimeng’ 和 ” xiaohei ” 都是可以的，但是 ‘xinxing “是不可可以的。
- 三个引号可以用于文档字符串，也可以用于超过了一行的字符串。
- 字符串可以说是 python 里面最常见的类型了，而且它也很简单。

**\*\*\*\*\*字符串的操作\*\*\*\*\***

- 关于字符串的操作是如此的复杂，操作字符串的函数也是非常的多。
- 字符串的操作包括拼接、查找、删除等很多复杂操作。
- 字符串也支持切片，操作和列表完全一样。

**\*\*\*\*\*字节和字符\*\*\*\*\***

- ✧ 一个不可变的 unicode 编码的字符序列是字符串，即 string。
- ✧ 一串由 0 到 255 之间的数字组成的序列叫做 bytes 对象。
- ✧ bytes 对象是不可变的，它的单个字节也不能改变。
- ✧ Python3 中不会隐含地将 bytes 转化为字符串。
- ✧ Python2 里面会假定我们使用 ASCII 编码，但是 Python3 的源码会默认用 UTF-8 进行编码。

**\*\*\*\*\*字符串运算符\*\*\*\*\***

- 这里还是不得不提关于字符串的运算符。

- 字符串支持用 “+” 号进行字符串的拼接。即我们想把 ab 和 cd 拼接的时候，可以直接用 “ab” + “cd” 的形式。
- 代码示例：



```
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC
32
Type "copyright", "credits" or "license()" for mor
>>> "ab" + "cd"
'abcd'
>>> x = 'I love'
>>> y = "xinxing"
>>> x + y
'I lovexinxing'
>>> |
```

## 第 8 节：表达式

\*\*\*\*\*表达式\*\*\*\*\*

- ✓ 其实表达式我们接触过了，只是没有明确的指出这个概念。
- ✓ 表达式可以理解为用变量和运算符组成的式子。
- ✓ 一个正确的表达式通常可以产生一个值，且这个值可以用赋值运算来给变量赋值。
- ✓ 比如  $a = 2 + 4$

\*\*\*\*\*运算符\*\*\*\*\*

- ◆ 其实我们已经接触过一些运算符了。

- ◆ 下面是一个表格，或者说是一个图片，它列出了所有的运算符：

运算符	描述
<code>x+y, x-y</code>	加、减，"+"号可重载为连接符
<code>x*y, x**y, x/y, x%y</code>	相乘、求平方、相除、求余，"*"号可重载为重复，"%"号可重载为格式化
<code>&lt;, &lt;=, &gt;, &gt;=, ==, &lt;&gt;, !=</code>	比较运算符
<code>+=, -=, *=, /=, %=, **=, &lt;&lt;=, &gt;&gt;=, &amp;=, ^=,  =</code>	自变运算符
<code>x y</code>	按位或
<code>x^y</code>	按位异或
<code>x&amp;y</code>	按位与
<code>~x</code>	按位取反
<code>x&lt;&lt;y, x&gt;&gt;y</code>	x向左或向右移y位
<code>is, is not</code>	等同测试
<code>in, not in</code>	是否为成员测试
<code>or, and, not</code>	逻辑运算符
<code>x[i], x[i:j], x.y, x(...)</code>	索引，分片，限定引用，函数调用
<code>(...), [...], {...}, '...'</code>	元组，列表，字典，转化为字符串

## 第 2 部分：语句语法篇

\*\*\*\*\*主要任务\*\*\*\*\*

- ✧ 会书写常见的语句。
- ✧ 能熟练掌握条件语句、循环语句以及 break 和 continue 语句。
- ✧ 稍微深入的重新了解下赋值语句。

## 第 0 节：语句的简介

### 第 1 节：赋值语句

### 第 2 节：条件语句

### 第 3 节：循环语句

### 第 4 节：循环语句中的其他语句

## 第 0 节：语句的简介

\*\*\*\*\*语句\*\*\*\*\*

- 百度百科给出的解释是：构成程序的元素。
- 我认为这个解释过于简略，但是我也很难给它严格的定义，也只能大致的描述一下。
- 语句就是可以独立完成一定任务的代码。

\*\*\*\*\*语句之间的分割\*\*\*\*\*

- ✧ 基本上各种语言之间都会有标点符号来对语言进行划分，避免出现混乱。
- ✧ 比如下面是同一段文字，因为标点符号的划分而导致的歧义：第一种划分“我没有了你，就不活了”，第二种划分“我没有了，你就不活了？”。这是截然不同的两个意思。
- ✧ 如果大家对 C 家族的语言比较熟悉的话，那么应该很分

号，在 C 语言中，不同的语句用分号进行分隔开，彼此区分。

- ✧ 但是在 Python 中，我们通常是一行写一个语句，即我们用行来划分不同的语句。

#### \*\*\*\*\*冒号的缩进\*\*\*\*\*

- 如果大家熟悉 C 家族语言，那么肯定会很熟悉分号和大括号对。但是在 Python 中，我们用冒号和缩进来表示层次。
- 在 Python 中关于缩进和换行有着严格的规定，因为没有多余的符号来划分层次。
- 这一点随着学习的深入大家会逐渐了解的，最后我们再总结。

#### \*\*\*\*\*语句的种类\*\*\*\*\*

- 语句的种类很多，比如有赋值语句，比如有流程控制语句，比如还有 class 语句等等。
- 我们将会在这一篇学习主要的语句，然后随着大家水平的提高，我们会学习完所有的语句的。

#### \*\*\*\*\*执行流程\*\*\*\*\*

- ◆ 如果你学习过其他编程语言，肯定听说过“流程控制语句”，它们通常用来控制语句的执行流程。
- ◆ 顺序结构：语句都是自上而下执行的，这也就是顺序结构，通常我们写的基本都是顺序结构。
- ◆ 选择结构：关于选择，也就是判断，我们根据一个或者一些条件是否成立，来判断程序的执行流程。
- ◆ 循环结构：有时候，一段代码可能要反复执行才能达到相应的效果，这个时候我们就需要循环结构来重复执行某些过程。



## 第 1 节：赋值语句

### \*\*\*\*\*赋值语句的复习\*\*\*\*\*

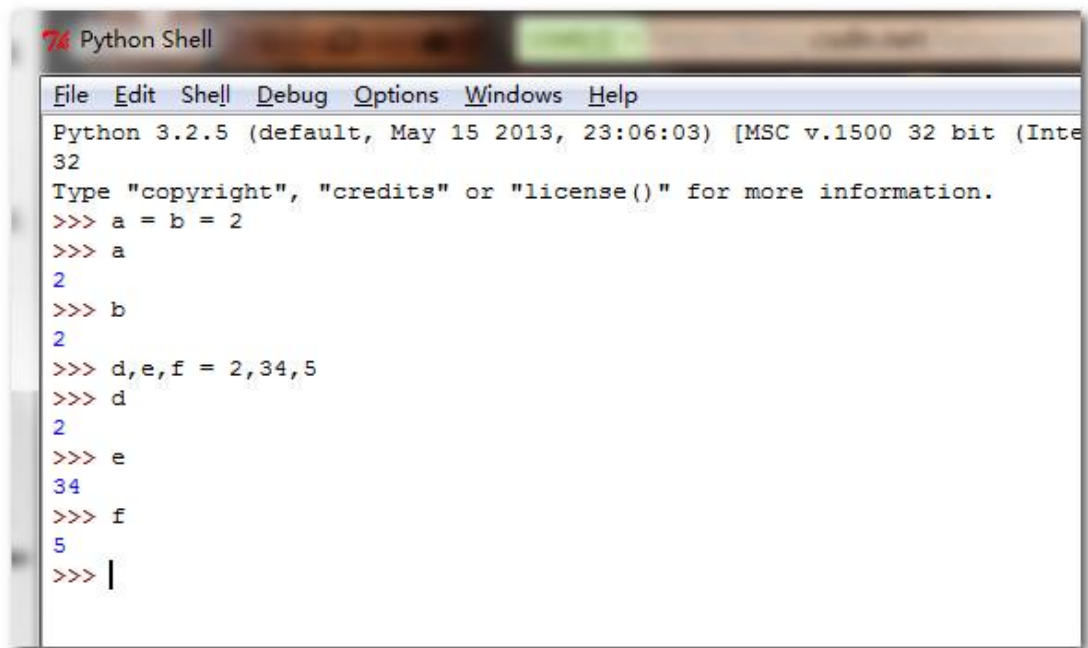
- ✧ 我们在学习数据类型的时候已经学习过赋值语句了，应该说是用“=”来表示赋值。
- ✧ 变量名在首次赋值的时候会被创建，而且不允许只声明变量不赋值。

### \*\*\*\*\*赋值号的简洁版\*\*\*\*\*

- 有时候我们需要大量的书写诸如“`a = a + 2`”这样的语句，那么我们可以用“`a += 2`”来代替它，即“`+=`”是先做加法后作赋值的简写形式。
- 同理的还有“`*=`”、“`/=`”、“`-=`”等等。

### \*\*\*\*\*赋值的一些简单形式\*\*\*\*\*

- ◆ 虽然我认为它们并不是很重要，但是毫无疑问它们可以在一定程度上减少我们的代码书写量。
- ◆ 连续赋值：比如 `a = b = 2`，则会把 `a` 和 `b` 同时取值为 2。
- ◆ 同时赋值：比如 `d, e, f = 2, 34, 5` 则会把 `d` 赋值为 2，`e` 赋值为 34，`f` 赋值为 5。
- ◆ 下面是在 IDLE 在交互模式下给出的操作示例：

A screenshot of a Python Shell window. The title bar says "Python Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The text area shows the following content:

```
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)]
Type "copyright", "credits" or "license()" for more information.
>>> a = b = 2
>>> a
2
>>> b
2
>>> d,e,f = 2,34,5
>>> d
2
>>> e
34
>>> f
5
>>> |
```

◆

## 第 2 节：条件语句

## \*\*\*\*\*Python 中的选择语句\*\*\*\*\*

- ✧ Python 中没有 C 家族中的 switch、case 语句，则在 Python 中的选择语句只有 if elif else 语句。
- ✧ 在 Python 中，一定要注意是怎么对齐的，因为在 Python 中这点尤其重要。

## \*\*\*\*\*if 语句\*\*\*\*\*

- if 翻译为汉语即“如果”，它进行判断，它的格式为：  
if 条件：

    如果条件成立执行的部分

其他语句，注意这里的其他语句要和 if 保持对齐

- 这个语句的意思是：如果条件取值为 True，那么则执行后面的语句，否则跳过去执行后面的其他语句。
- 这里的条件可以用小括号括起来，但是在简单的情形下通常是不括起来的。
- 实例（下例中的 a 直接被赋值为 True，则直接打印文字）：



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:00)
32
Type "copyright", "credits" or "license()"
>>> a = True
>>> if a :
    print("我是辛贵猛")

我是辛贵猛
>>>
```

## \*\*\*\*\*if, elif, else 语句

\*\*\*\*\*

- ✓ 有时候我们可能要进行不止一个判断，如果把多个判断写到一个判断语句中，那么就需要 `elif` 的帮助了。
- ✓ `else` 的意思是“否则”，是对上述所有情况的否定。
- ✓ 语法格式：

`if` 第一个条件：

如果 `if` 对应的条件成立，则执行它

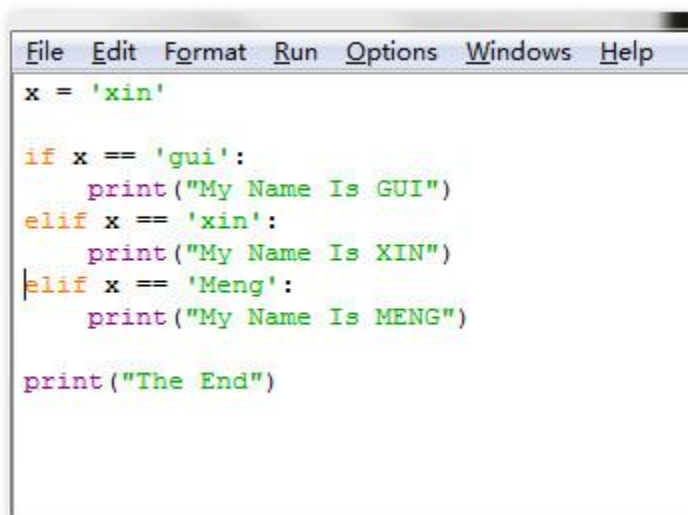
`elif` 第二个条件：

如果该条件成立，则执行它

`else`：

如果上述条件都不成立，则执行它。

- ✓ 下面是代码实例，它的 `x` 取值为“xin”，那么当执行到下面的选择语句的时候，它会执行第二部分对应的语句，然后跳出该条件语句，执行最后的 `print` 函数。
- ✓ 代码和执行结果如下：



```
File Edit Format Run Options Windows Help
x = 'xin'

if x == 'gui':
    print("My Name Is GUI")
elif x == 'xin':
    print("My Name Is XIN")
elif x == 'Meng':
    print("My Name Is MENG")

print("The End")
```



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:0
32
Type "copyright", "credits" or "license
>>> ===== RE
>>>
My Name Is XIN
The End
>>>
```

### 第 3 节：循环语句

\*\*\*\*\*循环语句\*\*\*\*\*

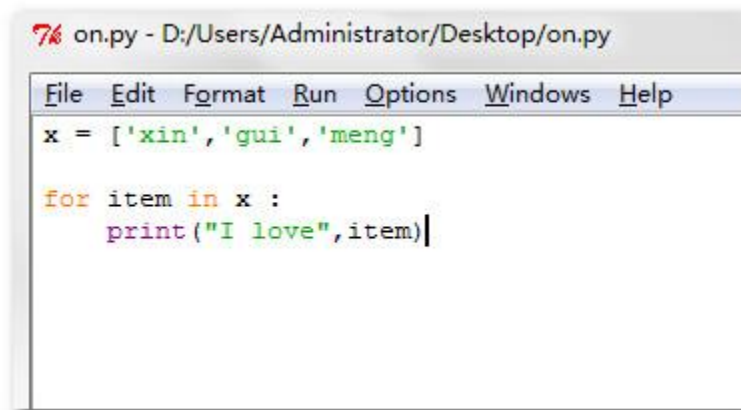
- ◇ 所谓循环，就是不断的做一件事，直到达到某结果。
- ◇ 在 Python 中，循环分为 for 循环和 while 循环。
- ◇ Python 中的 for 循环和 while 循环都和 C 语言家族的语法有点区别，我也谈不上好坏，总之，这是两种不同的思维习惯。

\*\*\*\*\*for 循环\*\*\*\*\*

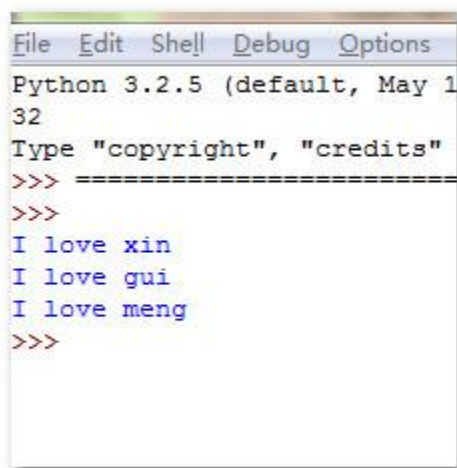
- for 循环是一个通用的序列迭代器，可以遍历任何有序的序列对象内的元素，for 语句通常用于列表、元组、字典等。
- 下面是 for 循环的语法：  
for 单个元素的代表 in 想要遍历的对象：

在遍历的时候的语句

- 可能大家会感觉有点迷茫，我们先看一个实例：

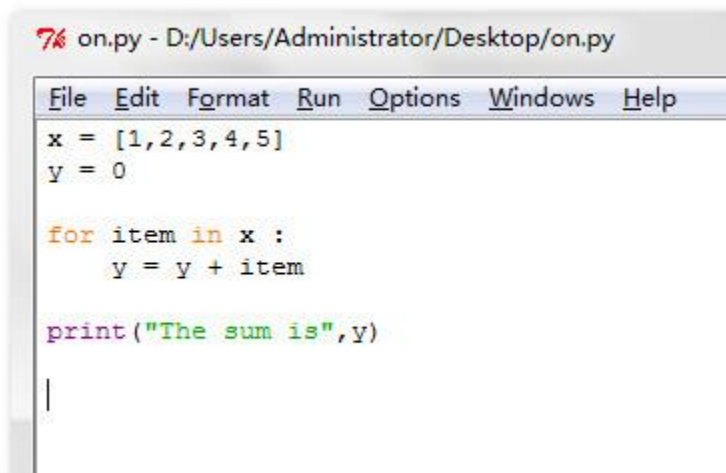
A screenshot of a Python IDE window titled 'on.py - D:/Users/Administrator/Desktop/on.py'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Windows', and 'Help'. The code editor contains the following Python code:

```
x = ['xin', 'gui', 'meng']  
  
for item in x :  
    print("I love", item)
```



```
File Edit Shell Debug Options
Python 3.2.5 (default, May 1
32
Type "copyright", "credits"
>>> =====
>>>
I love xin
I love gui
I love meng
>>>
```

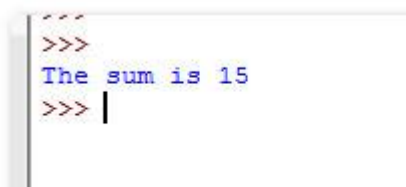
- 对上述例子的分析：这里是对 `x` 这个列表进行循环，而这里的 `item` 则是对 `x` 这个列表的一个代表，当 `item` 取值为第一个元素，即 `xin` 的时候，输出为 `I love xin`，当 `item` 取值为第二个元素的值，即 `gui` 的时候，输出为 `I love gui`，当 `item` 取值为第三个元素的时候，同理可得。
- 我们再看一个例子把：



```
on.py - D:/Users/Administrator/Desktop/on.py
File Edit Format Run Options Windows Help
x = [1,2,3,4,5]
y = 0

for item in x :
    y = y + item

print("The sum is",y)
|
```



```
>>>
The sum is 15
>>> |
```

下面是执行结果：

- 我来分析一下上述代码的作用：x 列表中有几个数字，y 先取值为 0，然后 item 逐次取值为 x 列表中的元素，然后与 y 做和，这样，最终的 y 就是 x 这个列表中的所有元素的和。

\*\*\*\*\*while 循环\*\*\*\*\*

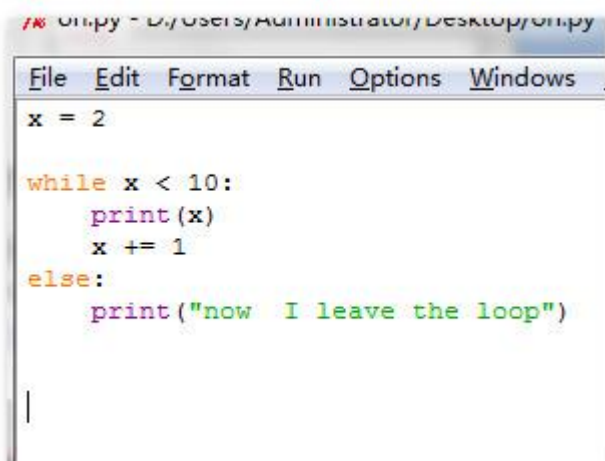
- ✧ 上面我们的两个例子都是用列表写的，有时候我们没有列表，也需要做循环，那么该怎么办呢？
- ✧ 我们此时需要借助 while 循环，while 翻译为汉语为“直到”，即它会一直做某件事直到发生另外一件事为止。
- ✧ while 循环的语法格式：
- while 条件：

    如果条件满足，则一直执行该语句

else:

    如果条件不满足则执行该语句

- ✧ 说明：它很像 if 和 else 语句，只是在 if 语句中，如果条件满足，只会执行一次，而 while 语句，如果条件满足，则会一直执行。
- ✧ 下面是一段实例和执行后的结果：

A screenshot of a Python IDE window titled 'on.py - D:/Users/Administrator/Desktop/on.py'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', and 'Windows'. The code editor contains the following Python code:

```
x = 2

while x < 10:
    print(x)
    x += 1
else:
    print("now I leave the loop")

|
```

执行后的结果为：



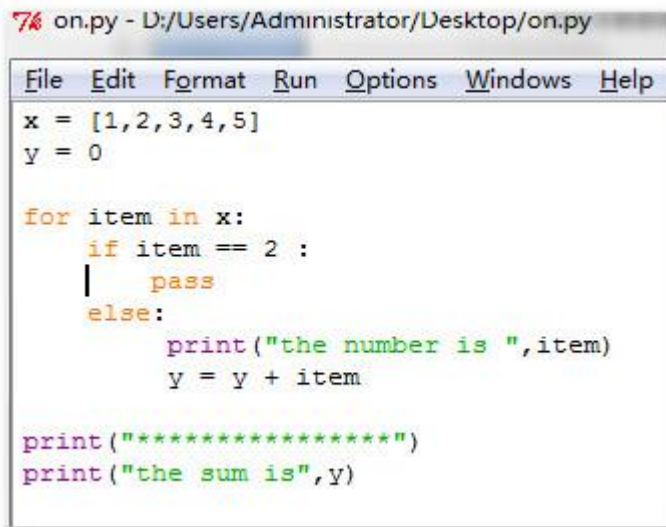
```
>>> =====
>>>
2
3
4
5
6
7
8
9
now I leave the loop
>>>
```

- ✧ 分析：上述代码首先定义了变量  $x$  为 2，而 `while` 后面的判断为只要  $x$  小于 10，那么就会一直打印该数值，然后并且把它加一，而一旦  $x$  取值为 10 的时候，则跳出该循环，执行后面的 `else` 语句，即打印 “now I leave the loop”。

## 第 4 节：循环语句中的其他语句

\*\*\*\*\*pass 语句\*\*\*\*\*

- 有时候我们会什么也不做，但是因为语法的需要，我们使用的一个占位符。
- 可能大家初次听到它可能会有点迷茫，比如我定义一个循环，我对一个列表进行循环，我输出这个列表的元素，并且对它们进行求和，当执行到列表元素为 2 的时候，我什么也不做，接着向下执行。
- 下面是代码：

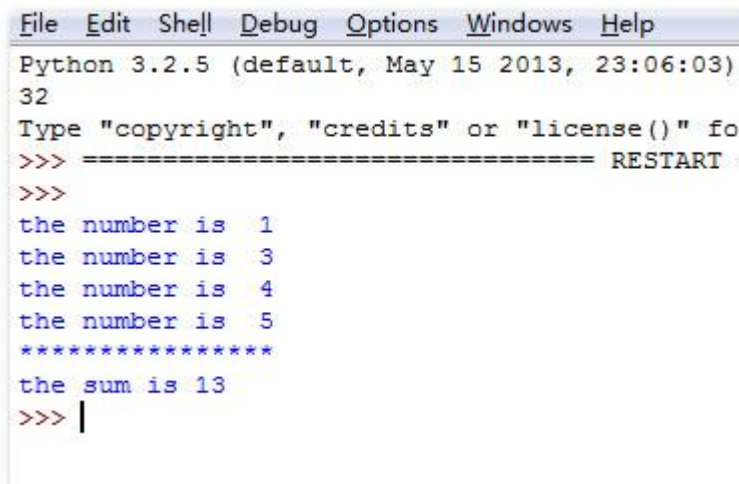


```
on.py - D:/Users/Administrator/Desktop/on.py
File Edit Format Run Options Windows Help
x = [1,2,3,4,5]
y = 0

for item in x:
    if item == 2 :
        pass
    else:
        print("the number is ",item)
        y = y + item

print("*****")
print("the sum is",y)
```

下面是运行结果：



```
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03)
32
Type "copyright", "credits" or "license()" for
>>> ===== RESTART =====
>>>
the number is 1
the number is 3
the number is 4
the number is 5
*****
the sum is 13
>>> |
```

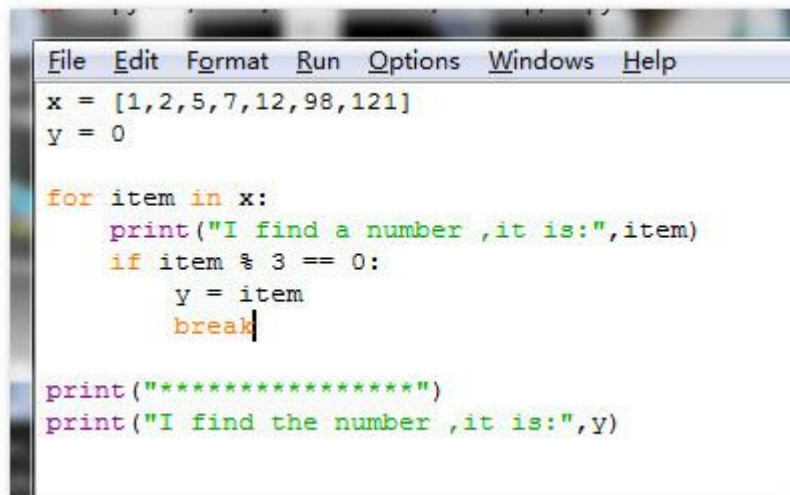
- 可能这个例子并不好，但是它确实可以说明确实有这样的情况存在。

\*\*\*\*\*break 语句\*\*\*\*\*

- break 翻译为汉语即“打破”、“打断”，它的作用通常为跳出循环。
- 有时候我们并不想让整个循环执行完毕，这时候我们可以使用 break 来跳出循环，继续向下执行。
- 比如我在一个列表里查找数据，我会把我找到的数据都打印出来，但是如果我找到了可以被三整除的数之

后，那么我就退出整个循环。

➤ 代码示例：

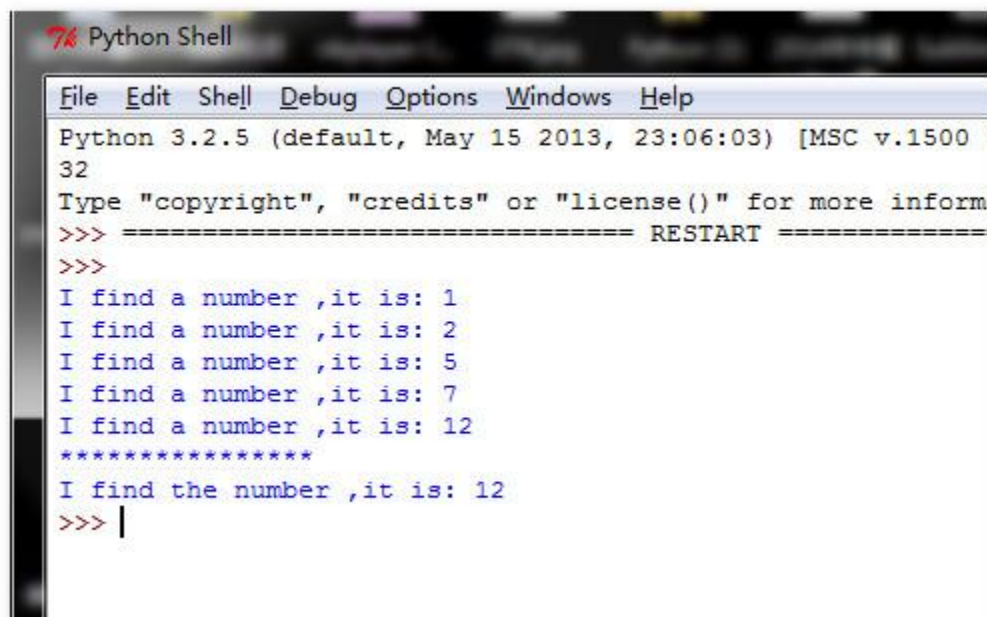


```
File Edit Format Run Options Windows Help
x = [1,2,5,7,12,98,121]
y = 0

for item in x:
    print("I find a number ,it is:",item)
    if item % 3 == 0:
        y = item
        break

print("*****")
print("I find the number ,it is:",y)
```

运行结果：

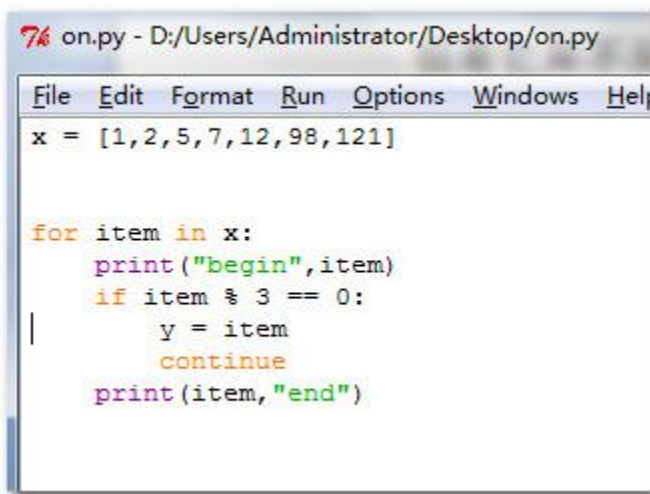


```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32
Type "copyright", "credits" or "license()" for more informa
>>> ===== RESTART =====
>>>
I find a number ,it is: 1
I find a number ,it is: 2
I find a number ,it is: 5
I find a number ,it is: 7
I find a number ,it is: 12
*****
I find the number ,it is: 12
>>> |
```

- 代码说明：我定义了一个列表 `x`，然后我对这个列表进行循环遍历，我会打印它的每一个元素，但是，如果我遇到了能被三整除的 `12`，那么我直接跳出该循环，继续向下执行。

## \*\*\*\*\*continue 语句\*\*\*\*\*

- ◆ continue 翻译为汉语即“继续”，它也通常用于循环中，但是它并不从循环中跳出，而是结束当变量取值为当前值的循环，继续下一次循环。
- ◆ 可能直接描述有点不清楚，大家根据下面的代码和对代码的分析仔细看看就一目了然了。
- ◆ 下面是代码：



```
on.py - D:/Users/Administrator/Desktop/on.py
File Edit Format Run Options Windows Help
x = [1,2,5,7,12,98,121]

for item in x:
    print("begin",item)
    if item % 3 == 0:
        y = item
        continue
    print(item,"end")
```

下面是执行结果：



```
File Edit Shell Debug Options Window
Python 3.2.5 (default, May 15 2011)
Type "copyright", "credits" or "help()"
>>> =====
>>>
>>> begin 1
>>> 1 end
>>> begin 2
>>> 2 end
>>> begin 5
>>> 5 end
>>> begin 7
>>> 7 end
>>> begin 12
>>> begin 98
>>> 98 end
>>> begin 121
>>> 121 end
>>>
```

- ◆ 分析：我定义了一个列表 `x`，下面我对这个列表进行循环遍历，对于每一个元素 `item`，我都会先输出 `begin item`，然后输出 `item end`，但是大家注意当 `item` 取值为 12 的时候，它只是输出了 `begin 12`，由于接下来的 `continue` 语句，使得它接下来的 `12 end` 无法被执行。

\*\*\*\*\*比较 break 和 continue\*\*\*\*\*

- ✧ 我记得有一个人总结的很好：`break` 是到此为止，`continue` 是再来一次。
- ✧ 即 `break` 语句直接从循环跳出去了，整个循环也随之破裂。
- ✧ `continue` 并不会使循环结束，但是它会使本轮的循环到此为止，并且进入下一轮的循环。

## 第 3 部分：函数、模块和命名空间

### 第 0 节：认识函数

### 第 1 节：函数的定义和调用

### 第 2 节：模块

### 第 3 节：命名空间

## 第 0 节：认识函数

\*\*\*\*\*无处不在的函数\*\*\*\*\*

- 我们使用任何一门编程语言(除了 ASM 那些特别古老的语言)，都离不开函数这个概念。
- 函数可以理解为完成特定任务的一个语句组。
- 比如我们经常使用的 `print()`，它就是一个函数，它的功能就是把括号内的东西打印到屏幕上。

## 第 1 节：函数的定义和调用

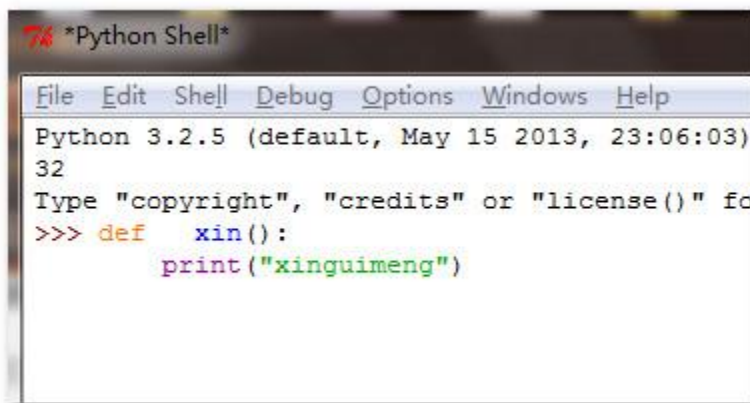
\*\*\*\*\*函数的定义\*\*\*\*\*

- 定义一个函数比较简单，语法如下：

def 函数名(参数列表):

函数体

- 即使我们的函数不需要任何参数，那么小括号也不能省略。
- 接下来的实例中我们定义一个函数 xin，它不需要接受任何参数，它的作用就是打印一个字符串“xinguimeng”：



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03)
32
Type "copyright", "credits" or "license()" fo
>>> def xin():
        print("xinguimeng")
```

\*\*\*\*\*函数的调用\*\*\*\*\*

- ✧ 要调用一个函数，其实我们已经调用过了，还记得我们使用 print () 函数吗。
- ✧ 调用一个函数，只需要使用函数名和小括号，如果需要参数，则需要添加相应的参数。
- ✧ 下面是调用我们刚才写的函数 xin ()：

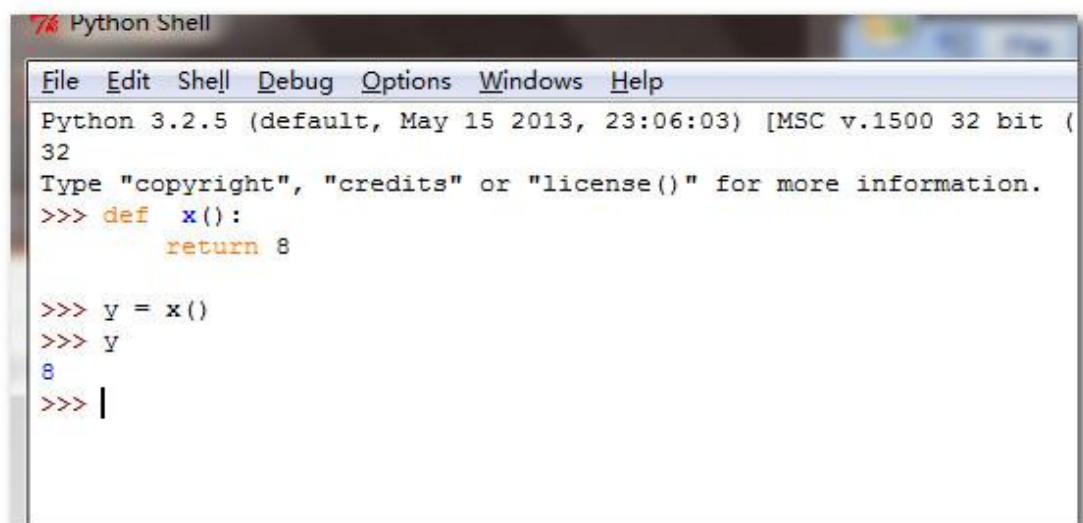




```
Python 3.2.5 (default, May 15 2013, 23:03:32)
Type "copyright", "credits" or "license()" for more
>>> def xin():
>>>     print("xinguimeng")
>>>
>>> xin()
xinguimeng
>>>
```

\*\*\*\*\*函数的返回值\*\*\*\*\*

- 有时候函数可以返回一些数据，此时可以使用 `return` 加上要返回的数据。
- 值得注意的是，在函数体内碰到了 `return` 语句，则函数立即执行结束。
- 在调用函数的时候，我们可以使用一个赋值语句，把函数的返回值赋值给相应的变量。
- 比如下面的实例中，我编写一个 `x` 函数，它返回 8，我另一个变量去接收这个返回的数据：



```
Python Shell
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (
32
Type "copyright", "credits" or "license()" for more information.
>>> def x():
>>>     return 8
>>>
>>> y = x()
>>> y
8
>>> |
```

---

**\*\*\*\*\*函数的参数\*\*\*\*\***

- 函数也可以接受参数，如果是多个参数，必须用逗号分隔开，以作区分。
- 函数在声明参数的时候只需要写出参数名称即可，不需要考虑参数类型。
- 函数在调用的时候，往往也应该传入相应的参数，最好是按照和函数定义的参数顺序一致。
- 下面的例子我们定义了一个 `add` 函数，它接收两个参数，并把他们的和返回，我们调用的时候也最好传入两个参数：

```
>>> def add(x,y):  
        return x+y  
  
>>> z = add(2,7)  
>>> z  
9  
>>>
```

- 说明：函数的参数有所谓的“形参”和“实参”的说法。在函数定义的时候，这个参数是形式参数，即此时的 `x` 和 `y` 都是形参。在函数执行或者说函数被调用的时候，这个时候的参数都是具体的值，即实际参数。

**\*\*\*\*\*函数参数的默认值 \*\*\*\*\***

- ✓ 函数的参数可以设置默认值，这样我们调用的时候就可以不指定参数的数值了。
- ✓ 指定参数的默认值在函数的定义的时候用等号加上默认值的形式指定，如果函数有默认值，原则上是从右向左依次指定默认值。

- ✓ 即不能出现左边的参数有默认值，但是右边的参数没有默认值。
- ✓ 下面的函数即有默认值，所以调用的时候只传一个值即可：

```
>>> def xin(x,y = 7):  
    return x+y  
  
>>> m = xin(2)  
>>> m  
9  
>>>
```

\*\*\*\*\*函数的关键字调用\*\*\*\*\*

- ✧ 有所谓的“关键字调用”，这个方式对于参数比较多的函数或者是参数顺序容易记乱的时候特别好用。
- ✧ 它是在调用的时候指定参数的值，此时就可以不必考虑函数定义的时候的参数的前后顺序了。

✧ 下面的例子说明了这一点：

```
File Edit Shell Debug Options Windows H  
Python 3.2.5 (default, May 15 2013, 23  
32  
Type "copyright", "credits" or "licens  
>>> def sub(x,y):  
    return x-y  
  
>>> z = sub(x = 5,y = 3)  
>>> z  
2  
>>> m = sub(y = 3,x = 5)  
>>> m  
2  
>>> s = sub(5,3)  
>>> s  
2  
>>> t = sub(3,5)  
>>> t  
-2  
>>> |
```

#### \*\*\*\*\*结构化的编程思路\*\*\*\*\*

- ✧ 虽然面向对象的编程思想已经深入人心，但是不可否认，面向过程的结构化编程的思想仍然是一个非常伟大的思想。
- ✧ 函数起到了这样的作用，它可以把一个较大的任务拆分成若干小任务，每一个小任务就是一个函数，然后通过调用不同的函数来解决它。
- ✧ 函数起到的这种“大事化小”的思想尤为可贵。

#### \*\*\*\*\*函数库\*\*\*\*\*

- 通常，现在很少提到“函数库”了，通常都是提“类库”或者是“框架”。
- 我们可以通过使用前人写的函数来极大的丰富自己的功能。
- 我们后面会涉及到模块，到时候大家就可以看到我们是如何在前人的基础上更快更好的做出自己想要的东西。

## 第二节 模块

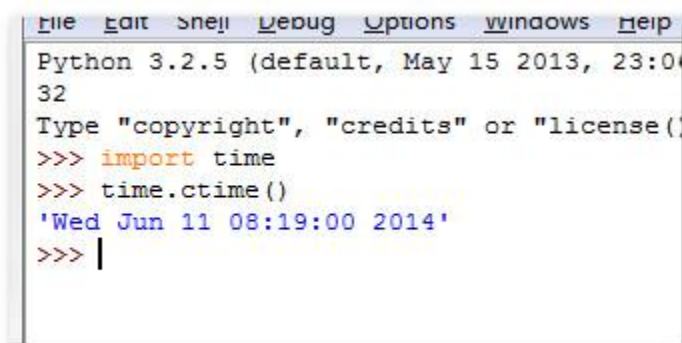
#### \*\*\*\*\*模块的引入 \*\*\*\*\*

- ✧ 上一节我们学习了函数，而且 Python 官方已经内置了很多函数，我们在下载 Python 的时候，把它们也一起下载了下来。

- ✧ 但是，要使用这些函数，例如 print 函数，是不需要涉及到模块的，但是，要使用一些高级的函数，就需要模块的介入了。
- ✧ 所谓模块，可以是一个包含所有我们定义的函数和变量的文件，即我们经常写的.py 文件，也可以是一个文件夹。

\*\*\*\*\*使用系统自带的模块 \*\*\*\*\*

- 要导入一个模块，需要使用“import ”关键字，然后写上要导入的模块名即可。
- 然后就可以调用模块内的方法或者是变量了，注意，调用模块内的方法或者变量的时候，需要在前面加上模块名并且紧跟着一个点号。
- 系统自带的模块往往被称作“标准库”，其实 Python 自带了一个庞大的标准库，后面我会专门出一套文章，专门介绍它的模块的。
- 例如我导入了系统内置的 time 模块，并且调用了其 ctime 方法，打印出来了当前的时间。
- 代码如下：

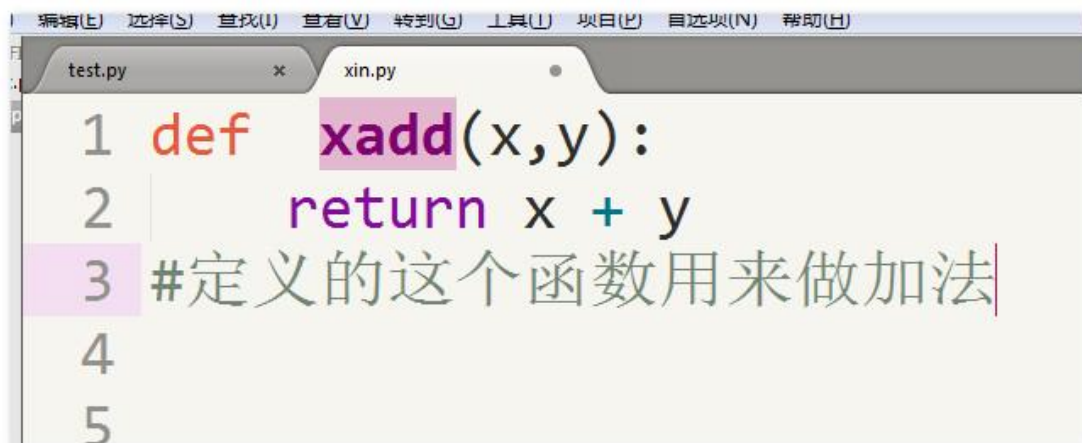


```
Python 3.2.5 (default, May 15 2013, 23:04:32)
Type "copyright", "credits" or "license()"
>>> import time
>>> time.ctime()
'Wed Jun 11 08:19:00 2014'
>>> |
```

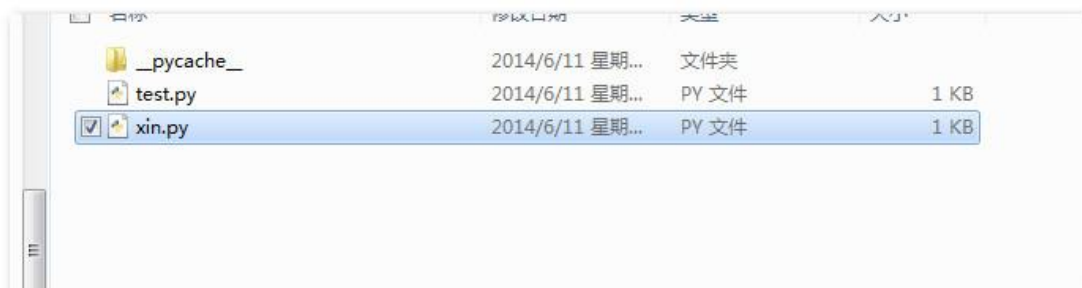
## \*\*\*\*\*创建自己的模块\*\*\*\*\*

- 要创建自己的模块，非常的简单，我们只需要写一个 py 文件，比如我们写一个 xin.py 文件，那么我们就创建了一个 xin 模块，导入的时候直接写“import xin”即可。
- 比如我们在 xin 模块里面定义了一个函数叫 xadd，它接受两个参数 x 和 y，那么我们在另一个 py 文件里调用它的时候，需要使用 xin.xadd 而不是直接的 xadd 来调用它。
- 下面我用图的方式给大家介绍：

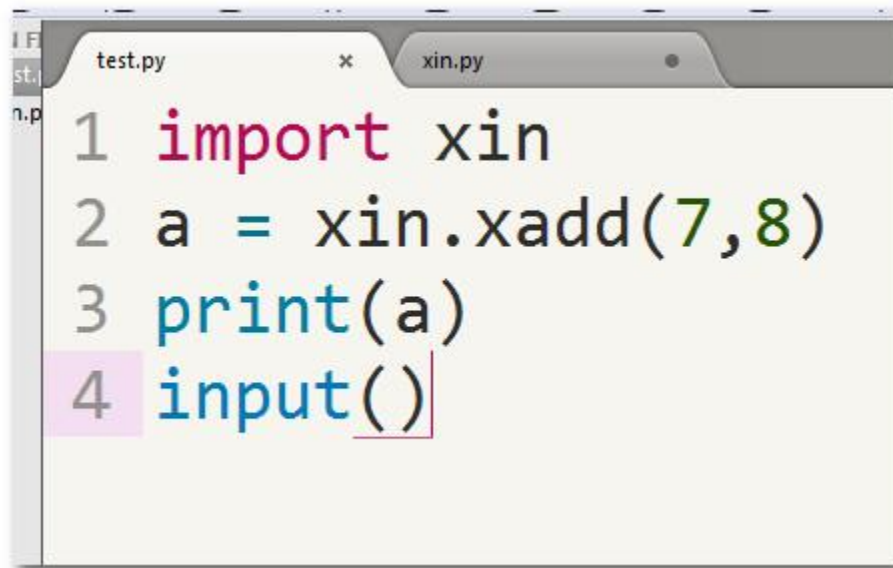
第一张是 xin.py 的内容：



第二张是这两个文件的存放位置：

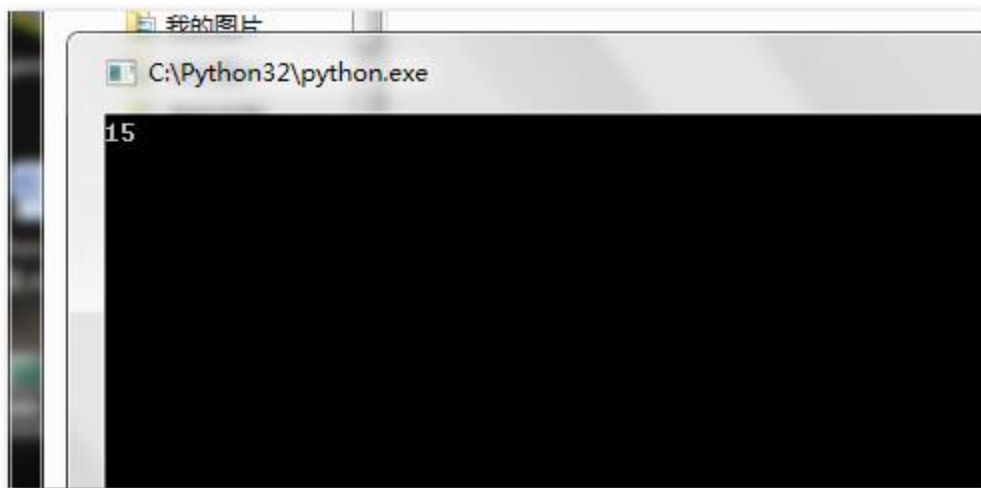


第三张是 test.py 的内容：



```
1 import xin
2 a = xin.xadd(7,8)
3 print(a)
4 input()
```

第四张是 test.py 的执行结果：



- 说明：那个\_\_pycache\_\_是缓存文件夹，是运行程序的时候自动生成的，不用理会它。
- 这里的 test.py 直接双击打开即可，它会自动调用 python.exe 打开它。
- 至于这里的 input（）函数，是为了防止程序一闪而过，因为程序执行完之后会自动关闭，而加入了 input(), 它会等待用户的输入。

- 一定要注意调用另一个模块的函数的时候需要加上模块并且紧跟着一个原点。

### 第三节 命名空间

#### \*\*\*\*\*命名空间的引入\*\*\*\*\*

- 如果读者学习过 C++，那么一定会对“using namespace std”这句话非常熟悉，没错，namespace 就是命名空间。
- 之所以会有命名空间，是为了区分同名的变量。比如两个变量有着相同的名字，那么该怎么办呢？会不会冲突呢？
- 答案是不一定的，如果它们属于不同的命名空间，那么就不会冲突。

#### \*\*\*\*\*命名空间\*\*\*\*\*

- ✧ 命名空间是从命名到对象的映射，当前的命名空间主要是通过字典来实现的，实现过程我们就不用太关心了。
- ✧ 我们可以把命名空间当做一个容器，在这个容器中可以装许多标识符，不同的容器中的同名标识符是不会相互冲突的。
- ✧ 命名空间通常和作用域一起出现，所谓作用域，即一个变量起作用的区域，在该区域之外，该变量是不可见的，或者说，对它进行操作是没有任何意义的。

#### \*\*\*\*\*命名空间内的变量和函数\*\*\*\*\*

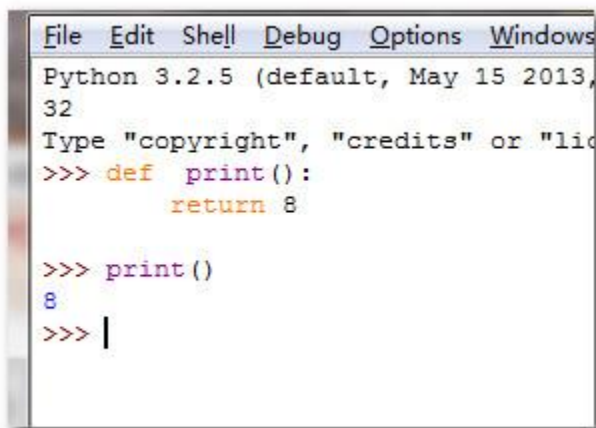
- 命名空间内的变量不要重名，否则前面的同名变量将会失效，因为系统会默认这两个是一个变量，而 后面的会修改前面的数值。



- 如果我们变量和函数都取名不重，那么完全可以不考虑命名空间。

\*\*\*\*\*常见的几个命名空间\*\*\*\*\*

- 第一种是函数定义过程中出现的命名空间，这个命名空间我们通常表示为 `local`，简记为“L”，函数定义过程中出现的形参都属于这个命名空间。
- 注意不同的函数中的变量是属于不同的命名空间的，因为它们属于不同的函数。比如 `add` 函数中的 `x` 和 `sub` 函数中的 `x` 是不会混乱的，因为它们属于不同的命名空间。
- 函数里面再次定义函数的时候，内层函数的变量所属的命名空间为 `enclosing function`，简写为 `E`。
- 第二种是当我们产生一个模块的时候，也会产生一个命名空间，不同模块的同名变量也不会混淆，因为它们属于不同的命名空间，这里我们通常表示为 `global`，即“G”。
- 第三种即 `builtin`，可以记为“B”，是 `python` 解释器启动后自动拥有的命名空间，这一层我们通常可以忽略不去考虑。但是比如在这个命名空间中已经有了 `print` 函数，那我们如果再定义 `print` 函数就会发生混乱了。
- 如下我重定义了 `print` 函数，导致无法使用系统自带的 `print` 函数了：



```
File Edit Shell Debug Options Windows
Python 3.2.5 (default, May 15 2013,
32
Type "copyright", "credits" or "lic
>>> def print():
        return 8

>>> print()
8
>>> |
```



#### \*\*\*\*\*命名空间的三个原则\*\*\*\*\*

- 第一个原则即标识符在被赋值的时候就已经决定了它所处的命名空间。
- 第二个是函数定义会产生自己的命名空间。
- 第三个是 Python 搜索标识符的顺序是“LEGB”。即对于一个变量名，能在 L 层找到就不继续向下找了，如果 L 层找不到，去 E 层找，然后是 G，最后是 B。

## 第 4 部分：面向对象

说明：“面向对象”是一种编程思想，自提出以后，便已席卷了整个软件工业，现在主流编程语言基本都已经支持面向对象编程（除了 C 和汇编这种特别古老的语言，Pascal 已经不算是主流工业编程语言了）。

\*\*\*\*\*主要任务\*\*\*\*\*

- 理解面向对象的思想。
- 能够编写自己的类，并且能够熟练实例化。

### 第 0 节：面向对象的思想

### 第 1 节：类

### 第 2 节：类的实例

第 3 节：类和实例的进一步解释

第 4 节：继承

第 5 节：多态

第 6 节：封装

第 7 节：最后的总结

## 第 0 节：面向对象的思想

### \*\*\*\*\*面向对象的三大宗旨\*\*\*\*\*

- 如果你学过其他语言，并且接触过面向对象，那么一定可以很轻松的说出来，它们就是：封装、继承、多态。
- 而且，我特别建议大家学一下 Java 的面向对象，可能是因为它是我学到的第一门编程语言，也可能是以为 Java 是一个纯面向对象的语言，在我接触到了一些编程语言之后（比如 Java，C++，Javascript，PHP，Python 等等），我感觉 Java 的面向对象设计的更好一点，当然，我没有说 Python 设计的不好。
- 面向对象不仅仅是引入了一个编程的方式，更重要的是引入了一个编程的思想。

### \*\*\*\*\*各个语言对面向对象的理解\*\*\*\*\*

- ✧ 不得不说，现在很多编程语言都支持面向对象，但是如果深入的研究一下，会发现它们之间的区别相当的大。
- ✧ 首先不说 Javascript 中的各种伪对象，单就 Java 和 C++来看，两者就有很大的区别：Java 不支持多继承，但是提供接口，C++支持多继承。Java 没有虚函数等一系列概念，但是 C++都支持。C++没有包的机制，但是 Java 支持包机制。C++有头文件，而 Java 不支持头文件。C++中不建议把类的方法的实现写入类定义中，但是 Java 中我们通常这么做。
- ✧ 然后比较一下 Java 和 Python：Python 中没有严格意义上的构造函数，它有一个初始化函数，但是 Java 中必须得有构造函数。Java 中类的属性和对象的属性是一一对应的，但是 Python 中实例却可以定义自己的属性。Python 中类的属性可以在类之外定义，且 Python 中的

类也是对象，类的实例也是对象，这对于 Java 来说简直是不能想象的。在 Python 中，class 是一个赋值语句，它会执行一个赋值运算，但是在 Java 中，它只是一个声明，并未涉及任何的运算。诸如此类还是太多了，大家可以在学习完毕之后进行总结。

#### \*\*\*\*\*类的思想\*\*\*\*\*

- “类”即一类事物，在面向对象编程中，一切事物都可以当做类，比如所有的车子可以组成车类，所有的红烧肉也可以组成红烧肉类，所有的人也可以组成人类。
- “类”有属性和方法，属性即类的一些特点，比如车类里面可以有颜色属性，重量属性等等，红烧肉也有重量和价格等属性，人也会有年龄和身高等属性。
- 类的方法即该类可以执行的功能，在编程中即类可以执行的代码，比如车类可以有启动、移动和停止等方法，人类可以有吃饭、睡觉、娱乐等方法。

#### \*\*\*\*\*类的实例\*\*\*\*\*

- ✧ 在其他编程语言中，类的实例即类的对象。
- ✧ 我们上面说的车类，但是我们找到一个具体的汽车的时候，它就是一个车类的实例，它有自己的特定的颜色，自己的特定的重量。
- ✧ 实例是个体，而类是整体。实例是特定的一个，而类则是对所有的实例的统称。

#### \*\*\*\*\*一切皆对象\*\*\*\*\*

- 现在，大家可能不太理解这句话，但是请记住：一切皆对象。

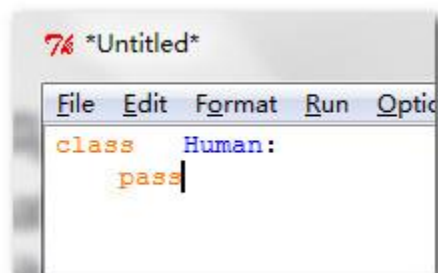
- 在 Python 中，类是对象，类的实例也是对象，模块是对象，函数也是对象。。。。所有的一切都是对象。

## 第一节 类

\*\*\*\*\*类的定义\*\*\*\*\*

✧ 类的定义使用 class 语句，具体步骤是：在写完关键字 class 之后，就跟上类的名字，然后冒号回车，下面可以定义一些类的属性，然后就是类的各种方法。

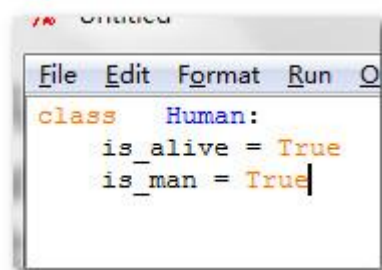
✧ 我们直接看下面的例子：



- ✧ 上面的例子中我们定义了一个类，但是这个类什么都没有做，它只是简单的拥有自己的类名。
- ✧ 我们也可以继续对这个类进行扩充，使它拥有自己的一些东西。

#### \*\*\*\*\*类的属性\*\*\*\*\*

- 类可以拥有自己的属性，属性可以理解为特殊的变量，因为这个变量是属于这个类的，即类的每一个实例也都拥有它。
- 要在类里面定义属性，通常在定义类名之后就开始书写了。
- 下面的例子里我们会定义两个属性，它们是两个布尔类型的变量，切都初始化值为 True。



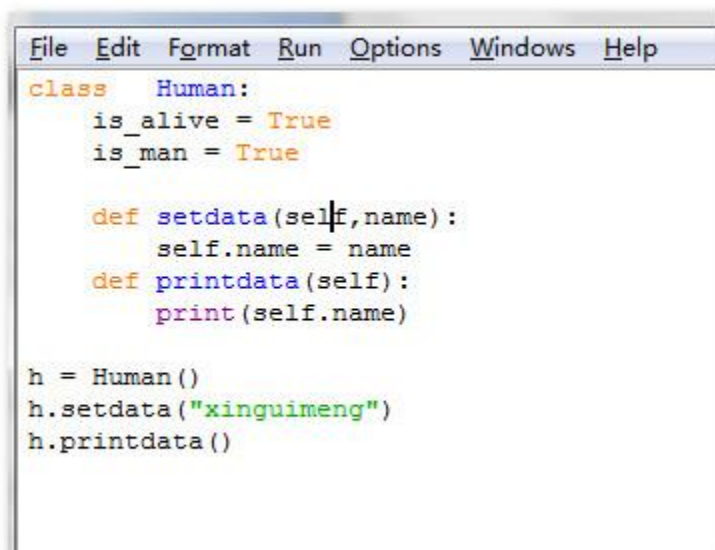
- 下面图片说明：

#### \*\*\*\*\*类的方法的定义和调用\*\*\*\*\*

- 类的方法其实也就是函数，但是我们是定义在类里面，为了和普通的函数做一个简单的区分，我们称之为“方法”。
- 我们通常的函数可以有参数，也可以没有参数。



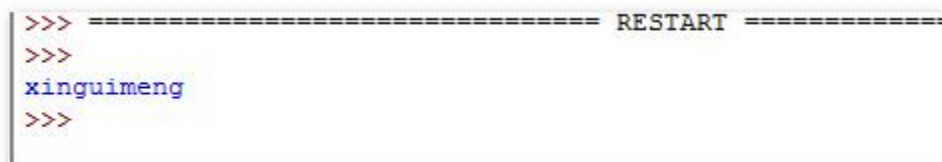
- 但是类里面的方法通常都会有参数，而且习惯上第一个参数为 `self`，它指向调用该方法的实例，有点像 C++ 或者 Java 里面的 `this`，后面的参数就随意了。
- 我们在调用的时候并不需要指定 `self`，我们只需要匹配其余的参数即可。
- 下面的例子中，我们在 `setdata` 方法中有两个参数，但是调用的时候只给一个即可，在 `printdata` 方法中有一个参数，但是调用的时候则无需给出参数。
- 注意调用的时候是用类的实例后面跟上一个圆点的。  
(貌似我们还没有讲类的实例)
- 下面是代码和调用后的代码：



```
File Edit Format Run Options Windows Help
class Human:
    is_alive = True
    is_man = True

    def setdata(self, name):
        self.name = name
    def printdata(self):
        print(self.name)

h = Human()
h.setdata("xinguimeng")
h.printdata()
```



```
>>> ===== RESTART =====
>>>
xinguimeng
>>>
```

\*\*\*\*\*一个特殊的方法\*\*\*\*\*

☆ 通常，在类中会有一个 `__init__` 方法，它是自动执行的。

- ✧ 它是在实例化完成之后自动调用的，它并不是某些编程语言中的构造方法，因为构造方法是在实例化之前调用的，而它则是在实例化完成之后调用，但是它们往往有异曲同工之妙。
- ✧ 下面我们在介绍类的实例化的时候还会用到它。

\*\*\*\*\*类的一般形式\*\*\*\*\*

- 这里说其实有点早，因为我们还没有涉及到类的继承等一些问题。
- 这里先列出一般的定义一个类的形式：

```
class 类名 (父类 1, 父类 2……):  
    data1 = value1  
    data2 = value2  
    .....  
  
    #即类的各种属性以及初值  
    def  method1(self, .....):  
        .....  
  
    def  method2(self, .....):  
        .....
```

## 第二节 类的实例

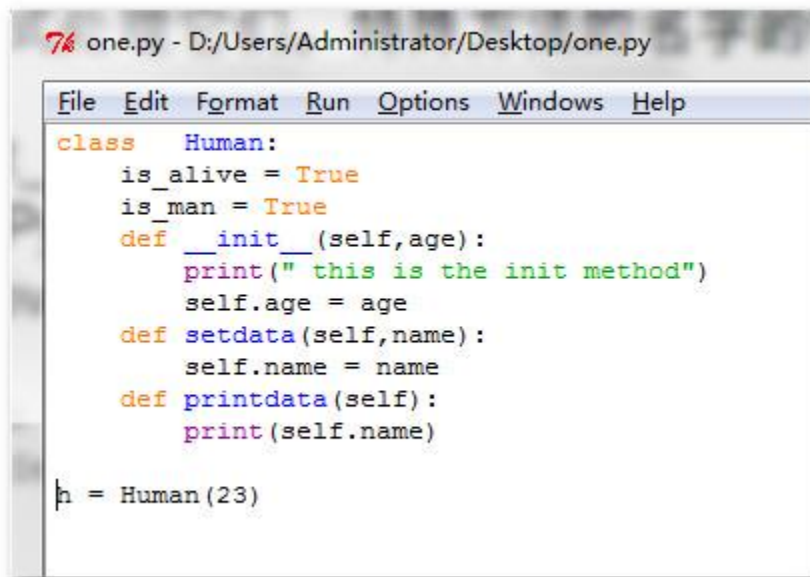
---

\*\*\*\*\*说法的变化\*\*\*\*\*

- ◆ 可能在不同的语言中，说法会稍微有点变化，在 Java 中，我们通常会说类的一个实例为一个对象，但是，在 Python 中，我们通常会说类对象和实例对象。
- ◆ 这也是说法上的一点不同，但是有一个说法是通用的，那就是类的实例。

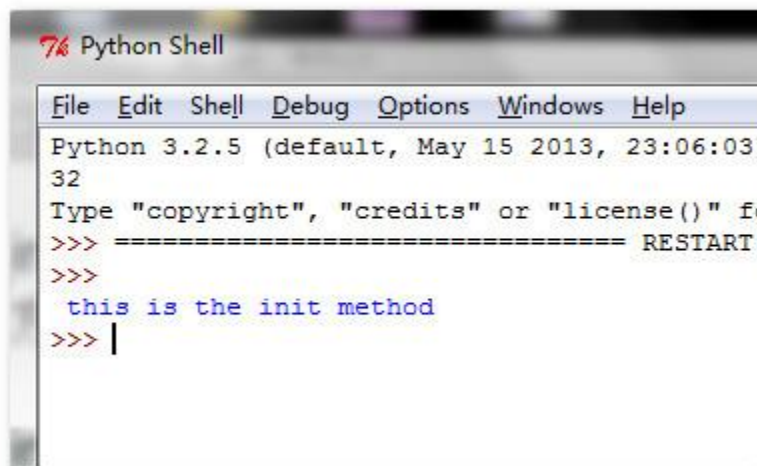
\*\*\*\*\*类如何实例化\*\*\*\*\*

- 类的实例化需要类名，通常我们在实例化之前会有一些设置，正是这些设置，说明了各个对象之间的区别。
- 这些设置我们通常在类中写一个函数，即上一节中提到的 `__init__`，它会在实例化之后被直接调用，不需要我们自动调用。
- 我们的实例化过程使用赋值号，然后接类名，括号里的参数就和 `__init__` 这个方法的参数保持一致就可以了，另外需要注意的是这里的 `self` 参数无需给出。
- 下面是一个完整的代码示例，可以看到在实例化完成之后，`__init__` 方法是得到了执行的，因为有些信息被打印了出来，而它们是属于 `__init__` 方法的：



```
one.py - D:/Users/Administrator/Desktop/one.py
File Edit Format Run Options Windows Help
class Human:
    is_alive = True
    is_man = True
    def __init__(self, age):
        print(" this is the init method")
        self.age = age
    def setdata(self, name):
        self.name = name
    def printdata(self):
        print(self.name)

h = Human(23)
```



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03)
32
Type "copyright", "credits" or "license()" f
>>> ===== RESTART
>>>
>>> this is the init method
>>> |
```

\*\*\*\*\*实例调用类的方法\*\*\*\*\*

✧ 其实在之前的代码中我们演示了实例调用类的方法的例子，此处我们进一步说明。

- ✧ 类的所有实例都可以调用该方法，下面我们说一下调用形式。
- ✧ 比如类 Human 有个方法为 getname，假设它不需要我们传递参数即可调用，而我们有一个类 Human 的实例 h，则我们想调用该方法的时候需要调用如下代码：  
h.getname()。

### 第三节：类和实例的进一步解释

#### \*\*\*\*\*二者的区别\*\*\*\*\*

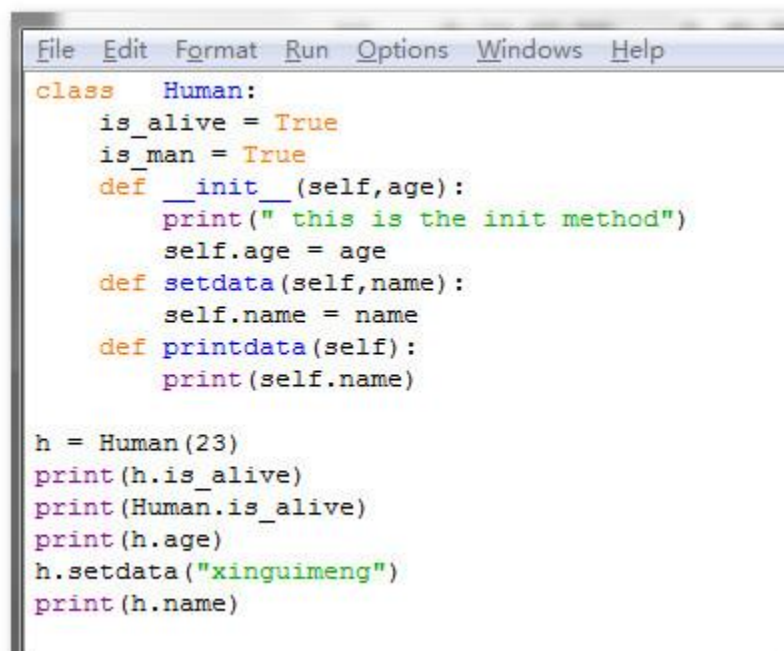
- ✧ 类是一个概括性的概念，它是对一类事物的集中表示，可以在它里面定义属性和方法。
- ✧ 实例是类的一个具体化，即我们可以根据抽象的类来得到具体的一个实例。
- ✧ 比如我们说“鸟类”是一个抽象的概念，而我们在自家的房子里看到自己喂养的宠物鸟的时候，那么我们的这个鸟就是“鸟类”的一个实例。
- ✧ 类的定义需要用 class 语句的，而实例则需要用我们定义的类得到，具体方法参考上一节。

#### \*\*\*\*\*二者的属性\*\*\*\*\*

- 类有属性，并且类的属性可以被所有的实例所共同拥有，就像所有的鸟类都有翅膀一样，我们随便拿出一只鸟，它也有翅膀（但是有翅膀不一定会飞，比如鸡有翅膀，但是并不会飞，但是鸡属于鸟类）。

- 比如我们定义一个类，类名为 Human，我们可以定义两个属性，比如为 `is_alive` 和 `is_man`，并且我们为这两个属性赋值为 `True`，那么这个类的所有实例都拥有了两个属性：`is_alive` 和 `is_man`。
- 实例的属性我们通常在 `__init__` 方法中定义，比如我们可以通过 `self.age` 来定义某对象中的 `age` 属性，这个 `age` 是属于一个特定的对象的，比如我的 `age` 是 23，但是不代表你的 `age` 是 23。比如我可以有“学法语的年数”这个属性，且取值为 2，但是很多人就可以没有这个属性。
- 下面的这段代码，定义的 Human 类的两个属性，注意下面的实例 `h`，可以用 `h.is_alive` 和 `Human.is_alive` 来查看类属性，但是不能用 `Human.age` 来查看属性，因为 `age` 这个属性时属于实例的，在类里面它没有一个统一的值可以供我们查看，而且对于实例的属性 `name`，大家可以通过对比在给该属性赋值前后的变化看到该属性的变化，

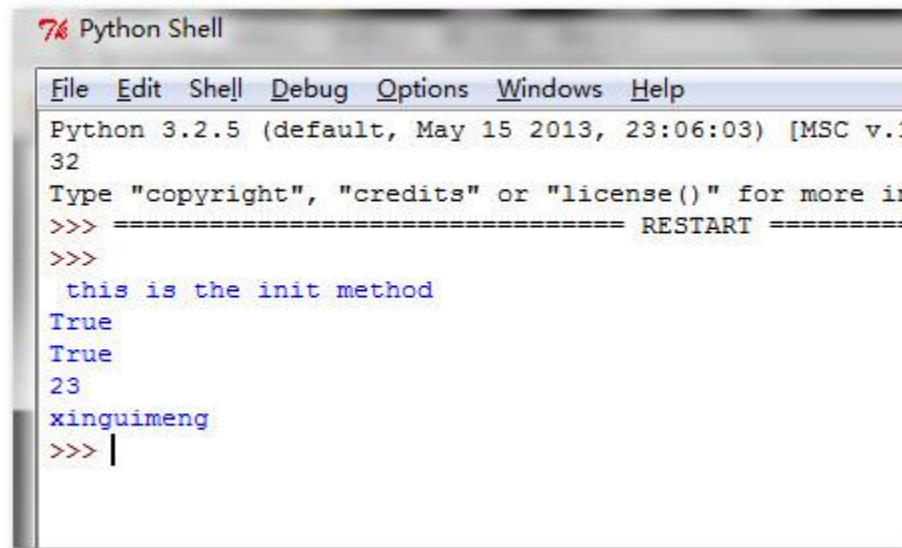
首先是代码示例：



```
File Edit Format Run Options Windows Help
class Human:
    is_alive = True
    is_man = True
    def __init__(self, age):
        print(" this is the init method")
        self.age = age
    def setdata(self, name):
        self.name = name
    def printdata(self):
        print(self.name)

h = Human(23)
print(h.is_alive)
print(Human.is_alive)
print(h.age)
h.setdata("xinguimeng")
print(h.name)
```

然后是执行之后的控制台界面：

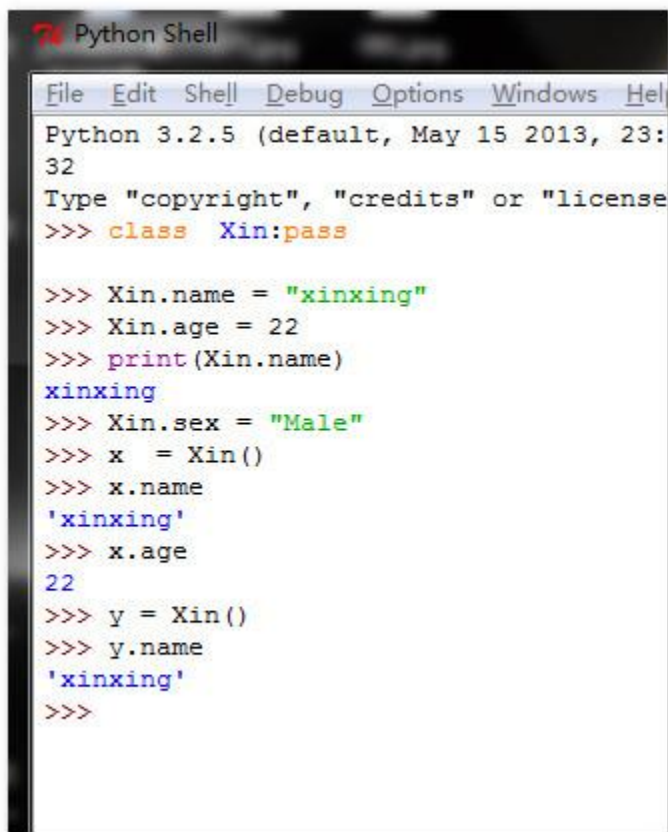


```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1310 64-bit (AMD64)]
32
Type "copyright", "credits" or "license()" for more
>>> ===== RESTART =====
>>>
>>> this is the init method
True
True
23
xinguimeng
>>> |
```

- 通过上面，我们也可以看出，类的属性在类中定义，实例的属性可以在类的方法中定义。
- 它们的引用都是在名字后面跟一个原点，然后在接上属性即可查看该值。

#### \*\*\*\*\*类的属性的进一步说明\*\*\*\*\*

- ✓ 和 Java 等编程语言不同，Python 中类的属性可以在类的定义外部进行定义并赋值。
- ✓ 下面的示例中，我们定义了一个类 Xin, 但是它什么都没有，只有一个名字，然后我们在下面先后指定了它的 name 属性，age 属性，sex 属性，在这中间我们还打印了它的属性，然后 x 和 y 是类 Xin 的两个实例，它们当然也拥有 name 属性和 age 属性。
- ✓ 下面是代码（我直接在交互式模式下的代码演示）：



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:32)
Type "copyright", "credits" or "license()"
>>> class Xin:pass

>>> Xin.name = "xinxing"
>>> Xin.age = 22
>>> print(Xin.name)
xinxing
>>> Xin.sex = "Male"
>>> x = Xin()
>>> x.name
'xinxing'
>>> x.age
22
>>> y = Xin()
>>> y.name
'xinxing'
>>>
```

\*\*\*\*\*一句话总结\*\*\*\*\*

- 类是“一类事物的统称”，实例是“一个特定的事物”。
- 实例属性是由方法内 `self` 属性进行赋值运算产生的。
- 类属性是通过 `class` 在定义类的时候通过赋值语句产生的，且它们通常从类定义的第二行开始。

\*\*\*\*\*实例的进一步说明\*\*\*\*\*

- 我们通过类来得到类的实例。
- 每个实例对象都会继承类的属性，并且还可以在方法内通过对 `self` 做赋值运算会产生每个实例自己的属性。



## 第四节：继承

### \*\*\*\*\*继承的重要性\*\*\*\*\*

- 如果没有继承，就不会出现复杂的类层次结构，也难以利用已经定义过的类。
- 如果没有继承，那么类之间的关系将会难以描述，是继承让我们更加清楚它们之间的关系。

### \*\*\*\*\*类继承的引入\*\*\*\*\*

- ◇ 比如我们定义了一个类来描述“鸟类”，但是我们同时需要“鸡类”“鸽类”等更加具体的类。
- ◇ 此时，我们如果直接定义“鸡类”和“鸽类”，那么可能会写很多冗余的代码，因为这些代码会同样在“鸟类”中同时存在，那么我们如何利用“鸟类”中的代码呢？
- ◇ 我们可以使用继承机制，继承就是利用已经写好的类来产生新的类。

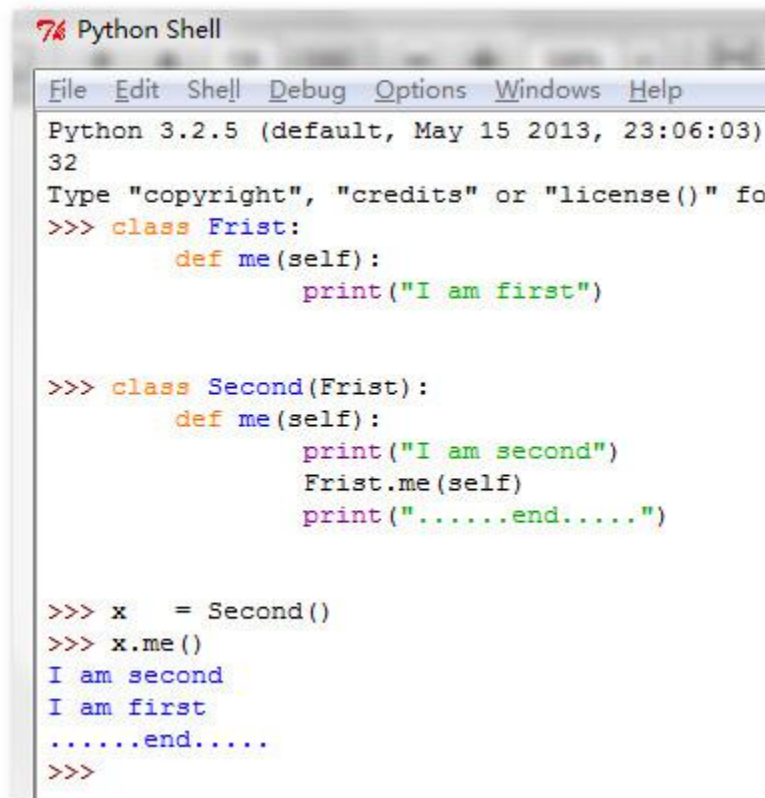
### \*\*\*\*\*类的继承\*\*\*\*\*

- 如果我们有一个类 A，我们通过继承产生了类 B 和类 C，那么我们可以说 B 和 C 是 A 的子类，而 A 是类 B 和类 C 的父类，或者说是超类，或者说是基类。
- 子类会从其父类中继承属性，就像实例会拥有其类的属性一样，类也会继承其父类中所定义的所有属性名称。

- 还记得我们第一节中类的一般形式中的括号中列出的父类列表吗？就是这里说到的继承。

\*\*\*\*\*编写子类\*\*\*\*\*

- ✧ 编写一个类的子类只需要指定它的父类即可，剩下的和写一个普通的类很相似。
- ✧ 如果我们需要在子类中调用父类中的方法，则可以使用 Super，比如在子类的\_\_init\_\_方法中调用父类的\_\_init\_\_方法，那么可以在子类的\_\_init\_\_方法中这么写：父类名.\_\_init\_\_(self, ……)



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03)
32
Type "copyright", "credits" or "license()" fo
>>> class Frist:
    def me(self):
        print("I am first")

>>> class Second(Frist):
    def me(self):
        print("I am second")
        Frist.me(self)
        print(".....end.....")

>>> x = Second()
>>> x.me()
I am second
I am first
.....end.....
>>>
```

✧

\*\*\*\*\*子类 and 父类的矛盾\*\*\*\*\*

- 在上面一小节中，我们看到子类 and 父类中都有一个方法叫做 me，这就可能引发命名冲突。

- 即我们的 `x` 是子类的实例，则它调用的是子类的 `me` 方法，而不是父类的 `me` 方法，注意的是，子类 `Second` 的 `me` 方法中的 `self` 参数是必须的，否则将会报错。

#### \*\*\*\*\*重载的引入\*\*\*\*\*

- ✧ 在涉及到继承的时候，就已经需要引入重载的概念。
- ✧ 比如说子类和父类中有相同名称的函数，比如都叫 `getname`，比如父类中打印一个字母 “Father”，子类中打印一个 “Sun”。
- ✧ 当我们实例化父类的时候，执行该函数，会打印出 `Father`，实例化子类的时候，执行该函数，会打印出 `Sun`。

#### \*\*\*\*\*特殊属性和方法\*\*\*\*\*

- 我们在之前曾经接触过一个特殊方法，即 `__init__`，它会自动执行，是实例对象初始化后自动执行的。
- 其他的一些特殊属性，比如 `__class__` 属性会得到该类的名字，`__bases__` 属性会得到该类的父类名的元组，`__add__` 方法可以用于重载加法运算符，`__str__` 方法可以用于把类转化为字符串。

#### \*\*\*\*\*抽象类\*\*\*\*\*

- 大多数其他编程语言的书籍在提到继承的时候都会顺便讲一下抽象类。

- 抽象类即“抽象的类”，具体点来说，就是不能实例化的类，它通常是包含抽象方法的。它的作用就是强制要求子类进行具体化，这样之后才可以实例化。

## 第五节：多态

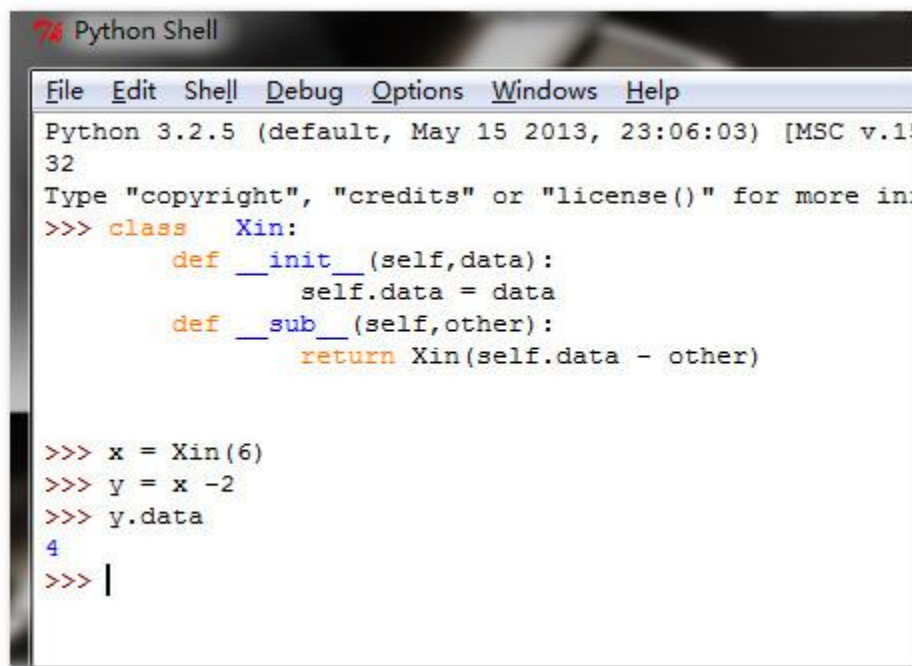
\*\*\*\*\*什么是多态\*\*\*\*\*

- 说到“多态”，我不知道大家的第一反应是什么，我的第一反应确实高中的时候组成氨基酸的多肽。好吧，开个玩笑。
- 所谓“多态”，即多种形态。在面向对象中，不乏多态的表现。
- 第一种表现：一个类可以有多个实例。即一个类可以被多次的实例化，这是多态的一种表现。
- 第二种表现：一个类可以有多个子类。即一个父类可以有很多个子类，这是多态的第二种表现。
- 第三种表现：函数可以重载。即父类的方法可以被子类中的同名方法所“覆盖”，这也是函数多态的一种体现。
- 第四种表现：就是运算符重载，这个我感觉并不算太重要，毕竟它只是提供了一个简便方式而已。

\*\*\*\*\*运算符重载\*\*\*\*\*

- ◇ 我们学过一些运算符，比如+、-、\*、/、=等等。
- ◇ 但是这些运算符往往只能用于内置的数据类型，对于我们最近涉及到的类和实例，如果我们也要以相同的方式使用，就可以使用运算符重载。

- ✧ 比如我们要实现 “+” 这个运算符，只需要在类里面重写 `__add__` 方法即可，如果要实现 “-”，则重写 `__sub__` 方法即可等等。
- ✧ 我个人感觉并不是很重要，因此我只是给出一个例子，让大家看一下运算符重载的具体实现：



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.150
32
Type "copyright", "credits" or "license()" for more in
>>> class Xin:
    def __init__(self,data):
        self.data = data
    def __sub__(self,other):
        return Xin(self.data - other)

>>> x = Xin(6)
>>> y = x - 2
>>> y.data
4
>>> |
```

## 第六节：封装

\*\*\*\*\*类的不同实现\*\*\*\*\*

- ✧ 在 Java 等语言中，比如有 `public`、`protected`、`default`、`private` 等权限修饰符来修饰类的属性或者方法，使得类外部对类内的属性获取的控制能力非常强大。

- ✧ 在 Java 等语言中，类外部通常是无法直接修改类的属性的，它们往往只能调用类的实例的方法。
- ✧ 但是 Python 中，我们上面也看到了，在类的外面不仅可以修改类的属性，甚至可以定义类的属性。
- ✧ 从这方面来说，Python 中的封装确实做的不够严谨，但是，对于脚本语言来说，这也使得它更加灵活。

#### \*\*\*\*\*封装\*\*\*\*\*

- 以下摘自百度百科：在程序上，隐藏对象的属性和实现细节，仅对外公开接口，控制在程序中属性的读和修改的访问级别；将抽象得到的数据和行为（或功能）相结合，形成一个有机的整体，也就是将数据与操作数据的源代码进行有机的结合，形成“类”，其中数据和函数都是类的成员。
- 但是 Python 中的封装做的并不是很严格，即类外面可以访问类的属性。
- 虽然我们可以通过使用伪私有属性，即在类的属性前面加上两个下划线的形式，但是我始终感觉 Python 在这方面做的并不严谨，也就不给大家介绍了。
- 直截了当的说，我并不欣赏 Python 的这一点，可能很多 Python 的粉丝又要过来给我说这样的好处是什么，其实我也知道，但是一点是必须承认的：它不够严谨。

## 第七节： 最后的总结

#### \*\*\*\*\*关于版本区别\*\*\*\*\*

- 不得不承认，对于 Python 在面向对象上的一些决定是和版本相关的，如果大家使用了不同的版本，那么可能会得到不同的结果。

- 比如 Python2.2 中引入的“新式类”，在 Python3 中已经没有了相关概念，因为我们的类都默认继承自 `object`，我们使用的类都是新式类。
- 比如 Python3 中没有了“无绑定方法”的概念，即如果类的方法中没有 `self` 参数，它会当做一个普通的函数，不会被当做类的方法。

\*

#### \*\*\*\*\*设计模式\*\*\*\*\*

- ◇ 接下来，我想如果任何一位读者如果想要在软件领域有所建树，如果想要深入面向对象编程，那么设计模式是必须熟悉的。
- ◇ 设计模式是一种思想，但是它在不同的语言中表现的却并不是很一样，可能会有点细微差别，不过，这不重要。

#### \*\*\*\*\*结束语\*\*\*\*\*

- 接下来的路还有很长，希望大家好好努力吧。
- 任重而道远，不可不努力。

结语

\*\*\*\*\*送给初学者\*\*\*\*\*

- ✧ 如果你是初学者，你可能会感觉读完本书之后，并不向自己想象的那样可以很快的搞定一个软件。
- ✧ 或者你可能还感觉读完本书之后感觉收获不大，感觉只是学了一堆语法和约定。
- ✧ 那么别急，去搜索“辛星 python 实战”，到时候你会发现原来自己有了一定的基础之后，开发应用软件是如此的轻松。

\*\*\*\*\*送给其他语言转向 Python 的\*\*\*\*\*

- 首先，我和大家一样，我一开始学习的是 Java，也是那样，先学习 Javase，然后学习 Javaee。
- 后来，我又学习了 C++，PHP 等语言。
- 我并不认为 Python 比他们好多少，但是 Python 还是有着它无可替代的优点-----灵活且迅速，开发效率高。
- 相比于 Java 和 C++的笨重，Python 的轻巧灵活的优势一下子就显现出来了。

\*\*\*\*\*未来展望\*\*\*\*\*

- 我希望有更多的人参与到这个计划中来，把这个教材写成一本真正的优秀的教材。
- 我是第一次写，可能里面会有一些问题，个别地方可能也特别不到位，在后续版本中会弥补这些缺点的。
- 最后衷心的祝愿大家可以使用 Python 可以快乐的编程，快乐的工作或者学习。
- 本人致力于游戏方面的开发，如果您也对游戏情有独钟，可以联系我奥。