说明

1. 教程将专注于Python基础，语法基于Python 2.7 (我会提醒Python 3.x中有变化的地方，以方便读者适应3.X的情况)。测试环境为Linux, 将不会使用到标准库之外的模块。

2. 我将专注于python的主干，以便读者能以最快时间对python形成概念。

3. Linux命令行将以 $ 开始，比如 $ls, $python

4. python命令行将以 >>> 开始，比如 >>>print 'Hello World!'

5. 注释会以 # 开始

建议

1. 将教程中的命令敲到python中看看效果

2. 你可以在了解之后立即去查看相关更完备的内容 (比如查阅官方文档)

简单的‘Hello World!’

**Python命令行**

假设你已经安装好了Python, 那么在Linux命令行输入:

$python

将直接进入python。然后在命令行提示符>>>后面输入:

>>>print('Hello World!')

可以看到，随后在屏幕上输出:

Hello World!

print是一个常用函数，其功能就是输出括号中得字符串。

（在Python 2.x中，print还可以是一个关键字，可写成print 'Hello World!'，但这在3.x中行不通 ）

**写一段小程序**

另一个使用Python的方法，是写一个Python程序。用文本编辑器写一个.py结尾的文件，比如说*hello.py*

在*hello.py*中写入如下，并保存:

print('Hello World!')

退出文本编辑器，然后在命令行输入:

$python hello.py

来运行hello.py。可以看到Python随后输出

Hello World!

**脚本**

我们还可以把Python程序hello.py改成一个可执行的脚本，直接执行：

#!/usr/bin/env python

print('Hello World!')

需要修改上面程序的权限为可执行：

chmod 755 hello.py

然后再命令行中，输入

./hello.py

就可以直接运行了

**总结**

print

命令行模式: 运行Python，在命令行输入命令并执行。

程序模式: 写一段Python程序并运行。

简单的数据类型以及赋值

### 变量不需要声明

Python的变量不需要声明，你可以直接输入：

>>>a = 10

那么你的内存里就有了一个变量a， 它的值是10，它的类型是integer (整数)。 在此之前你不需要做什么特别的声明，而数据类型是Python自动决定的。

>>>print(a)

>>>print(type(a))

那么会有如下输出：

10

<class 'int'>

这里，我们学到一个内置函数type(), 用以查询变量的类型。

### 回收变量名

如果你想让a存储不同的数据，你不需要删除原有变量就可以直接赋值。

>>>a = 1.3

>>>print(a,type(a))

会有如下输出

1.3 <class 'float'>

我们看到print的另一个用法，也就是print后跟多个输出，以逗号分隔。

### 基本数据类型

a=10         # int 整数

a=1.3        # float 浮点数

a=True       # 真值 (True/False)

a='Hello!'   # 字符串。字符串也可以用双引号。

以上是最常用的数据类型。此外还有分数，字符，复数等其他类型，有兴趣的可以学习一下。

### 总结

变量不需要声明，不需要删除，可以直接回收适用。

type(): 查询数据类型

整数，浮点数，真值，字符串

### sequence 序列

sequence(序列)是一组有顺序的元素的集合

(严格的说，是对象的集合，但鉴于我们还没有引入“对象”概念，暂时说元素)

序列可以包含一个或多个元素，也可以没有任何元素。

我们之前所说的基本数据类型，都可以作为序列的元素。元素还可以是另一个序列，以及我们以后要介绍的其他对象。

序列有两种：tuple（定值表； 也有翻译为元组） 和 list (表)

>>>s1 = (2, 1.3, 'love', 5.6, 9, 12, False)         # s1是一个tuple

>>>s2 = [True, 5, 'smile']                          # s2是一个list

>>>print(s1,type(s1))

>>>print(s2,type(s2))

tuple和list的主要区别在于，一旦建立，tuple的各个元素不可再变更，而list的各个元素可以再变更。

一个序列作为另一个序列的元素

>>>s3 = [1,[3,4,5]]

空序列

>>>s4 = []

### 元素的引用

序列元素的下标从0开始：

>>>print(s1[0])

>>>print(s2[2])

>>>print(s3[1][2])

由于list的元素可变更，你可以对list的某个元素赋值：

>>>s2[1] = 3.0

>>>print(s2)

如果你对tuple做这样的操作，会得到错误提示。

所以，可以看到，序列的引用通过s[<int>]实现， int为下标

### 其他引用方式

范围引用： 基本样式[下限:上限:步长]

>>>print(s1[:5])             # 从开始到下标4 （下标5的元素 不包括在内）

>>>print(s1[2:])             # 从下标2到最后

>>>print(s1[0:5:2])          # 从下标0到下标4 (下标5不包括在内)，每隔2取一个元素 （下标为0，2，4的元素）

>>>print(s1[2:0:-1])         # 从下标2到下标1

从上面可以看到，在范围引用的时候，如果写明上限，那么这个上限本身不包括在内。

尾部元素引用

>>>print(s1[-1])             # 序列最后一个元素

>>>print(s1[-3])             # 序列倒数第三个元素

同样，如果s1[0:-1], 那么最后一个元素不会被引用 （再一次，不包括上限元素本身）

### 字符串是元组

字符串是一种特殊的元组，因此可以执行元组的相关操作。

>>>str = 'abcdef'

>>>print(str[2:4])

### 总结

tuple元素不可变，list元素可变

序列的引用 s[2], s[1:8:2]

字符串是一种tuple

Python的运算符和其他语言类似

（我们暂时只了解这些运算符的基本用法，方便我们展开后面的内容，高级应用暂时不介绍）

### 数学运算

>>>print 1+9        # 加法

>>>print 1.3-4      # 减法

>>>print 3\*5        # 乘法

>>>print 4.5/1.5    # 除法

>>>print 3\*\*2       # 乘方

>>>print 10%3       # 求余数

### 判断

判断是真还是假，返回True/False

>>>print 5==6               # =， 相等

>>>print 8.0!=8.0           # !=, 不等

>>>print 3<3, 3<=3          # <, 小于; <=, 小于等于

>>>print 4>5, 4>=0          # >, 大于; >=, 大于等于

>>>print 5 in [1,3,5]       # 5是list [1,3,5]的一个元素

（还有is, is not等, 暂时不深入）

### 逻辑运算

True/False之间的运算

>>>print True and True, True and False      # and, “与”运算， 两者都为真才是真

>>>print True or False                      # or, "或"运算， 其中之一为真即为真

>>>print not True                           # not, “非”运算， 取反

可以和上一部分结合做一些练习，比如：

>>>print 5==6 or 3>=3

### 总结

数学 +, -, \*, /, \*\*, %

判断 ==, !=, >, >=, <, <=, in

逻辑 and, or, not

### 缩进

Python最具特色的是用缩进来标明成块的代码。我下面以if选择结构来举例。if后面跟随条件，如果条件成立，则执行归属于if的一个代码块。

先看C语言的表达方式（注意，这是C，不是Python!）

if ( i > 0 )

{

x = 1;

y = 2;

}

如果i > 0的话，我们将进行括号中所包括的两个赋值操作。括号中包含的就是块操作，它隶属于if。

在Python中，同样的目的，这段话是这样的

if i > 0:

x = 1

y = 2

在Python中， 去掉了i > 0周围的括号，去除了每个语句句尾的分号，表示块的花括号也消失了。

多出来了if ...之后的:(冒号), 还有就是x = 1 和 y =2前面有四个空格的缩进。通过缩进，Python识别出这两个语句是隶属于if。

Python这样设计的理由纯粹是为了程序好看。

### if语句

写一个完整的程序，命名为ifDemo.py。这个程序用于实现if结构。

i = 1

x = 1

if i > 0:

x = x+1

print x

$python ifDemo.py  # 运行

程序运行到if的时候，条件为True，因此执行x = x+1,。

print x语句没有缩进，那么就是if之外。

如果将第一句改成i = -1，那么if遇到假值 (False), x = x+1隶属于if, 这一句跳过。 print x没有缩进，是if之外，不跳过，继续执行。

这种以四个空格的缩进来表示隶属关系的书写方式，以后还会看到。强制缩进增强了程序的可读性。

复杂一些的if选择：

i = 1

if i > 0:  
    print 'positive i'

i = i + 1

elif i == 0:  
    print 'i is 0'

i = i \* 10

else:  
    print 'negative i'

i = i - 1

print 'new i:',i

这里有三个块，分别属于if, elif, else引领。  
Python检测条件，如果发现if的条件为假，那么跳过后面紧跟的块，检测下一个elif的条件； 如果还是假，那么执行else块。  
通过上面的结构将程序分出三个分支。程序根据条件，只执行三个分支中的一个。

整个if可以放在另一个if语句中，也就是if结构的嵌套使用：

i = 5

if i > 1:

print 'i bigger than 1'

print 'good'

if i > 2:

print 'i bigger than 2'

print 'even better'

if i > 2 后面的块相对于该if缩进了四个空格，以表明其隶属于该if，而不是外层的if。

### 总结

if语句之后的冒号

以四个空格的缩进来表示隶属关系, Python中不能随意缩进

if  <条件1>:

    statement

elif <条件2>:

    statement

elif <条件3>：

    statement

else:

    statement

### for循环

for循环需要预先设定好循环的次数(n)，然后执行隶属于for的语句n次。

基本构造是

for 元素 in 序列:

statement

举例来说，我们编辑一个叫forDemo.py的文件

for a in [3,4.4,'life']:

print a

这个循环就是每次从表[3,4.4,'life'] 中取出一个元素（回忆：表是一种序列），然后将这个元素赋值给a，之后执行隶属于for的操作(print)。

介绍一个新的Python函数range()，来帮助你建立表。

idx = range(5)

print idx

可以看到idx是[0,1,2,3,4]

这个函数的功能是新建一个表。这个表的元素都是整数，从0开始，下一个元素比前一个大1， 直到函数中所写的上限 （不包括该上限本身）

(关于range()，还有丰富用法，有兴趣可以查阅， Python 3中， range()用法有变化，见评论区)

举例

for a in range(10):

print a\*\*2

### while循环

while的用法是

while 条件:

statement

while会不停地循环执行隶属于它的语句，直到条件为假(False)

举例

while i < 10:

print i

i = i + 1

### 中断循环

continue   # 在循环的某一次执行中，如果遇到continue, 那么跳过这一次执行，进行下一次的操作

break      # 停止执行整个循环

for i in range(10):

if i == 2:   
 continue

print i

当循环执行到i = 2的时候，if条件成立，触发continue, 跳过本次执行(不执行print)，继续进行下一次执行(i = 3)。

for i in range(10):

if i == 2:

break

print i

当循环执行到i = 2的时候，if条件成立，触发break, 整个循环停止。

### 总结

range()

for 元素 in 序列:

while 条件:

continue

break

函数最重要的目的是方便我们重复使用相同的一段程序。

将一些操作隶属于一个函数，以后你想实现相同的操作的时候，只用调用函数名就可以，而不需要重复敲所有的语句。

**函数的定义**

首先，我们要定义一个函数, 以说明这个函数的功能。

def square\_sum(a,b):

c = a\*\*2 + b\*\*2

return c

这个函数的功能是求两个数的平方和。

首先，def，这个关键字通知python：我在定义一个函数。square\_sum是函数名。

括号中的a, b是函数的参数，是对函数的输入。参数可以有多个，也可以完全没有（但括号要保留）。

我们已经在循环和选择中见过冒号和缩进来表示的隶属关系。

c = a\*\*2 + b\*\*2        # 这一句是函数内部进行的运算

return c               # 返回c的值，也就是输出的功能。Python的函数允许不返回值，也就是不用return。

return可以返回多个值，以逗号分隔。相当于返回一个tuple(定值表)。

return a,b,c          # 相当于 return (a,b,c)

在Python中，当程序执行到return的时候，程序将停止执行函数内余下的语句。return并不是必须的，当没有return, 或者return后面没有返回值时，函数将自动返回None。None是Python中的一个特别的数据类型，用来表示什么都没有，相当于C中的NULL。None多用于关键字参数传递的默认值。

**函数调用和参数传递**

定义过函数后，就可以在后面程序中使用这一函数

print square\_sum(3,4)

Python通过位置，知道3对应的是函数定义中的第一个参数a， 4对应第二个参数b，然后把参数传递给函数square\_sum。

（Python有丰富的参数传递方式，还有关键字传递、表传递、字典传递等，基础教程将只涉及位置传递）

函数经过运算，返回值25, 这个25被print打印出来。

我们再看下面两个例子

[复制代码](javascript:void(0);)

a = 1

def change\_integer(a):

a = a + 1

return a

print change\_integer(a)

print a

#===(Python中 "#" 后面跟的内容是注释，不执行 )

b = [1,2,3]

def change\_list(b):

b[0] = b[0] + 1

return b

print change\_list(b)

print b

[复制代码](javascript:void(0);)

第一个例子，我们将一个整数变量传递给函数，函数对它进行操作，但原整数变量a不发生变化。

第二个例子，我们将一个表传递给函数，函数进行操作，原来的表b发生变化。

对于基本数据类型的变量，变量传递给函数后，函数会在内存中复制一个新的变量，从而不影响原来的变量。（我们称此为值传递）

但是对于表来说，表传递给函数的是一个指针，指针指向序列在内存中的位置，在函数中对表的操作将在原有内存中进行，从而影响原有变量。 （我们称此为指针传递）

**总结**

def function\_name(a,b,c):

statement

return something  # return不是必须的

函数的目的： 提高程序的重复可用性。

return     None

通过位置，传递参数。

基本数据类型的参数：值传递

表作为参数：指针传递

练习:

写一个判断闰年的函数，参数为年、月、日。若是是闰年，返回True

Python使用类(class)和对象(object)，进行面向对象（object-oriented programming，简称OOP）的编程。

面向对象的最主要目的是提高程序的重复使用性。我们这么早切入面向对象编程的原因是，Python的整个概念是基于对象的。了解OOP是进一步学习Python的关键。

下面是对面向对象的一种理解，基于分类。

**相近对象，归为类**

在人类认知中，会根据属性相近把东西归类，并且给类别命名。比如说，鸟类的共同属性是有羽毛，通过产卵生育后代。任何一只特别的鸟都在鸟类的原型基础上的。

面向对象就是模拟了以上人类认知过程。在Python语言，为了听起来酷，我们把上面说的“东西”称为对象（object）。

先定义鸟类

class Bird(object):

have\_feather = True

way\_of\_reproduction = 'egg'

我们定义了一个类别（class），就是鸟（Bird）。在隶属于这个类比的语句块中，我们定义了两个变量，一个是有羽毛（have\_feather），一个是生殖方式（way\_of\_reproduction）,这两个变量对应我们刚才说的属性（attribute）。我们暂时先不说明括号以及其中的内容，记为问题1。

假设我养了一只小鸡，叫summer。它是个对象，且属于鸟类。使用前面定义的类：

summer = Bird()

print summer.way\_of\_reproduction

通过第一句创建对象，并说明summer是类别鸟中的一个对象，summer就有了鸟的类属性，对属性的引用是通过 对象.属性（object.attribute） 的形式实现的。

可怜的summer，你就是个有毛产的蛋货，好不精致。

**动作**

日常认知中，我们在通过属性识别类别的时候，有时根据这个东西能做什么事情来区分类别。比如说，鸟会移动。这样，鸟就和房屋的类别区分开了。这些动作会带来一定的结果，比如移动导致位置的变化。

这样的一些“行为”属性为方法（method）。Python中通过在类的内部定义函数，来说明方法。

[复制代码](javascript:void(0);)

class Bird(object):  
    have\_feather = True  
    way\_of\_reproduction = 'egg'  
 def move(self, dx, dy):

position = [0,0]  
 position[0] = position[0] + dx

position[1] = position[1] + dy  
 return position  
  
summer = Bird()

print 'after move:',summer.move(5,8)

[复制代码](javascript:void(0);)

我们重新定义了鸟这个类别。鸟新增一个方法属性，就是表示移动的方法move。（我承认这个方法很傻，你可以在看过下一讲之后定义个有趣些的方法）

（它的参数中有一个self，它是为了方便我们引用对象自身。方法的第一个参数必须是self，无论是否用到。有关self的内容会在下一讲展开）

另外两个参数，dx, dy表示在x、y两个方向移动的距离。move方法会最终返回运算过的position。

在最后调用move方法的时候，我们只传递了dx和dy两个参数，不需要传递self参数（因为self只是为了内部使用）。

我的summer可以跑了。

**子类**

类别本身还可以进一步细分成子类

比如说，鸟类可以进一步分成鸡，大雁，黄鹂。

在OOP中，我们通过继承(inheritance)来表达上述概念。

[复制代码](javascript:void(0);)

class Chicken(Bird):

way\_of\_move = 'walk'

possible\_in\_KFC = True

class Oriole(Bird):

way\_of\_move = 'fly'

possible\_in\_KFC = False

summer = Chicken()

print summer.have\_feather

print summer.move(5,8)

[复制代码](javascript:void(0);)

新定义的鸡（Chicken）类的，增加了两个属性：移动方式（way\_of\_move），可能在KFC找到（possible\_in\_KFC）

在类定义时，括号里为了Bird。这说明，Chicken是属于鸟类（Bird）的一个子类，即Chicken继承自Bird。自然而然，Bird就是Chicken的父类。Chicken将享有Bird的所有属性。尽管我只声明了summer是鸡类，它通过继承享有了父类的属性（无论是变量属性have\_feather还是方法属性move）

新定义的黄鹂(Oriole)类，同样继承自鸟类。在创建一个黄鹂对象时，该对象自动拥有鸟类的属性。

通过继承制度，我们可以减少程序中的重复信息和重复语句。如果我们分别定义两个类，而不继承自鸟类，就必须把鸟类的属性分别输入到鸡类和黄鹂类的定义中。整个过程会变得繁琐，因此，面向对象提高了程序的可重复使用性。

（回到问题1, 括号中的object，当括号中为object时，说明这个类没有父类（到头了））

将各种各样的东西分类，从而了解世界，从人类祖先开始，我们就在练习了这个认知过程，面向对象是符合人类思维习惯的。所谓面向过程，也就是执行完一个语句再执行下一个，更多的是机器思维。通过面向对象的编程，我们可以更方便的表达思维中的复杂想法。

**总结**

将东西根据属性归类 ( 将object归为class )

方法是一种属性，表示动作

用继承来说明父类-子类关系。子类自动具有父类的所有属性。

self代表了根据类定义而创建的对象。

建立对一个对象： 对象名 = 类名()

引用对象的属性： object.attribute

9

我们熟悉了对象和类的基本概念。我们将进一步拓展，以便能实际运用对象和类。

**调用类的其它信息**

上一讲中提到，在定义方法时，必须有self这一参数。这个参数表示某个对象。对象拥有类的所有性质，那么我们可以通过self，调用类属性。

[复制代码](javascript:void(0);)

class Human(object):

laugh = 'hahahaha'

def show\_laugh(self):

print self.laugh

def laugh\_100th(self):

for i in range(100):

self.show\_laugh()  
  
li\_lei = Human()            
li\_lei.laugh\_100th()

[复制代码](javascript:void(0);)

这里有一个类属性laugh。在方法show\_laugh()中，通过self.laugh，调用了该属性的值。

还可以用相同的方式调用其它方法。方法show\_laugh()，在方法laugh\_100th中()被调用。

通过对象可以修改类属性值。但这是危险的。类属性被所有同一类及其子类的对象共享。类属性值的改变会影响所有的对象。

**\_\_init\_\_()方法**

\_\_init\_\_()是一个特殊方法(special method)。Python有一些特殊方法。Python会特殊的对待它们。特殊方法的特点是名字前后有两个下划线。

如果你在类中定义了\_\_init\_\_()这个方法，创建对象时，Python会自动调用这个方法。这个过程也叫初始化。

class happyBird(Bird):

def \_\_init\_\_(self,more\_words):

print 'We are happy birds.',more\_words

summer = happyBird('Happy,Happy!')

这里继承了Bird类，它的定义见上一讲。

屏幕上打印：

We are happy birds.Happy,Happy!

我们看到，尽管我们只是创建了summer对象，但\_\_init\_\_()方法被自动调用了。最后一行的语句(summer = happyBird...)先创建了对象，然后执行：

summer.\_\_init\_\_(more\_words)

'Happy,Happy!' 被传递给了\_\_init\_\_()的参数more\_words

**对象的性质**

我们讲到了许多属性，但这些属性是类的属性。所有属于该类的对象会共享这些属性。比如说，鸟都有羽毛，鸡都不会飞。

在一些情况下，我们定义对象的性质，用于记录该对象的特别信息。比如说，人这个类。性别是某个人的一个性质，不是所有的人类都是男，或者都是女。这个性质的值随着对象的不同而不同。李雷是人类的一个对象，性别是男；韩美美也是人类的一个对象，性别是女。

当定义类的方法时，必须要传递一个self的参数。这个参数指代的就是类的一个对象。我们可以通过操纵self，来修改某个对象的性质。比如用类来新建一个对象，即下面例子中的li\_lei, 那么li\_lei就被self表示。我们通过赋值给self.attribute，给li\_lei这一对象增加一些性质，比如说性别的男女。self会传递给各个方法。在方法内部，可以通过引用self.attribute，查询或修改对象的性质。

这样，在类属性的之外，又给每个对象增添了各自特色的性质，从而能描述多样的世界。

[复制代码](javascript:void(0);)

class Human(object):

def \_\_init\_\_(self, input\_gender):

self.gender = input\_gender

def printGender(self):

print self.gender

li\_lei = Human('male') # 这里，'male'作为参数传递给\_\_init\_\_()方法的input\_gender变量。  
print li\_lei.gender

li\_lei.printGender()

[复制代码](javascript:void(0);)

在初始化中，将参数input\_gender，赋值给对象的性质，即self.gender。

li\_lei拥有了对象性质gender。gender不是一个类属性。Python在建立了li\_lei这一对象之后，使用li\_lei.gender这一对象性质，专门储存属于对象li\_lei的特有信息。

对象的性质也可以被其它方法调用，调用方法与类属性的调用相似，正如在printGender()方法中的调用。

**总结**

通过self调用类属性

\_\_init\_\_(): 在建立对象时自动执行

类属性和对象的性质的区别

10

从最初的“Hello World”，走到面向对象。该回过头来看看，教程中是否遗漏了什么。

我们之前提到一句话，"Everything is Object". 那么我们就深入体验一下这句话。

需要先要介绍两个内置函数，dir()和help()

dir()用来查询一个类或者对象所有属性。你可以尝试一下

>>>print dir(list)

help()用来查询的说明文档。你可以尝试一下

>>>print help(list)

(list是Python内置的一个类，对应于我们之前讲解过的列表)

**list是一个类**

在上面以及看到，表是Python已经定义好的一个类。当我们新建一个表时，比如：

>>>nl = [1,2,5,3,5]

实际上，nl是类list的一个对象。

实验一些list的方法：

>>>print nl.count(5)       # 计数，看总共有多少个5

>>>print nl.index(3)       # 查询 nl 的第一个3的下标

>>>nl.append(6)            # 在 nl 的最后增添一个新元素6

>>>nl.sort()               # 对nl的元素排序

>>>print nl.pop()          # 从nl中去除最后一个元素，并将该元素返回。

>>>nl.remove(2)            # 从nl中去除第一个2

>>>nl.insert(0,9)          # 在下标为0的位置插入9

总之，list是一个类。每个列表都属于该类。

Python补充中有list常用方法的附录。

**运算符是特殊方法**

使用dir(list)的时候，能看到一个属性，是\_\_add\_\_()。从形式上看是特殊方法（下划线，下划线）。它特殊在哪呢？

这个方法定义了"+"运算符对于list对象的意义，两个list的对象相加时，会进行的操作。

>>>print [1,2,3] + [5,6,9]

运算符，比如+, -, >, <, 以及下标引用[start:end]等等，从根本上都是定义在类内部的方法。

尝试一下

>>>print [1,2,3] - [3,4]

会有错误信息，说明该运算符“-”没有定义。现在我们继承list类，添加对"-"的定义

[复制代码](javascript:void(0);)

class superList(list):

def \_\_sub\_\_(self, b):

a = self[:]     # 这里，self是supeList的对象。由于superList继承于list，它可以利用和list[:]相同的引用方法来表示整个对象。

b = b[:]

while len(b) > 0:

element\_b = b.pop()

if element\_b in a:

a.remove(element\_b)

return a

print superList([1,2,3]) - superList([3,4])

[复制代码](javascript:void(0);)

内置函数len()用来返回list所包含的元素的总数。内置函数\_\_sub\_\_()定义了“-”的操作：从第一个表中去掉第二个表中出现的元素。如果\_\_sub\_\_()已经在父类中定义，你又在子类中定义了，那么子类的对象会参考子类的定义，而不会载入父类的定义。任何其他的属性也是这样。

（教程最后也会给出一个特殊方法的清单）

定义运算符对于复杂的对象非常有用。举例来说，人类有多个属性，比如姓名，年龄和身高。我们可以把人类的比较（>, <, =）定义成只看年龄。这样就可以根据自己的目的，将原本不存在的运算增加在对象上了。

**下一步**

希望你已经对Python有了一个基本了解。你可能跃跃欲试，要写一些程序练习一下。这会对你很有好处。

但是，Python的强大很大一部分原因在于，它提供有很多已经写好的，可以现成用的对象。我们已经看到了内置的比如说list，还有tuple等等。它们用起来很方便。在Python的标准库里，还有大量可以用于操作系统互动，Internet开发，多线程，文本处理的对象。而在所有的这些的这些的基础上，又有很多外部的库包，定义了更丰富的对象，比如numpy, tkinter, django等用于科学计算，GUI开发，web开发的库，定义了各种各样的对象。对于一般用户来说，使用这些库，要比自己去从头开始容易得多。我们要开始攀登巨人的肩膀了。

谢谢你的关注，

欢迎来到Python的世界。

**总结**

len() dir() help()

数据结构list(列表)是一个类。

运算符是方法

2.1