老齐（qiwsir）著
路小磊 整理

声明

这个“零基础学Python”并不是我写的，原内容来自于这里，我只是将其Github的MarkDown内容整理成了Gitbook格式方便阅读。

我已经联系作者qiwsir，同意我整理，欢迎大家多多支持原作者。

原作者：老齐（qiwsir）

原作者Github：https://github.com/qiwsir

我：Looly

我的Github：https://github.com/looly

我的邮箱：loolly@gmail.com

为什么要开设此栏目

这个栏目的名称叫做“零基础学Python”。

现在网上已经有不少学习python的课程，其中也不乏精品。按理说，不缺少我这个基础类型的课程了。但是，我注意到一个问题，不管是课程还是出版的书，大多数是面向已经有一定编程经验的人写的或者讲的，也就是对这些朋友来讲，python已经不是他们的第一门高级编程语言。据我所知，目前国内很多大学都是将C之类的做为学生的第一门语言。

然而，在我看来，python是非常适合做为学习高级语言编程的第一门语言的。有一本书，名字叫《与孩子一起学编程》，这本书的定位，是将python定位为学习者学习的第一门高级编程语言。然而，由于读者对象是孩子，很多“成年人”不屑一顾，当然，里面的讲法与“实战”有

```
#看到“>>>”符号，表示python做好了准备，当代你向她发出指令，让她做什么事情
>>>

#print，意思是打印。在这里也是这个意思，是要求python打印什么东西
>>> print

#"Hello,World"是打印的内容，注意，量变的双引号，都是英文状态下的。引号不是打印内容，它相当于一个包裹，把
>>> print "Hello, World"

#上面命令执行的结果。python接收到你要求她所做的事情：打印Hello,World，于是她就老老实实地执行这个命令，
Hello, World
```

祝贺，伟大的程序员。

> 笑一笑：有一个程序员，自己感觉书法太烂了，于是立志继承光荣文化传统，购买了笔墨纸砚。在某天，开始练字。将纸铺好，拿起笔蘸足墨水，挥毫在纸上写下了两个打字：Hello World

从此，进入了程序员行列，但是，看官有没有感觉，程序员用的这个工具，就是刚才打印Hello,World的那个cmd或者shell，是不是太简陋了？你看美工妹妹用的Photoshop，行政妹妹用的word，出纳妹妹用的Excel，就连坐在老板桌后面的那个家伙还用一个PPT装逼自己都不相信的新理念呢，难道我们伟大的程序员，就用这么简陋的工具写化腐朽世代码吗？

当然不是。软件是谁开发的？程序员。程序员肯定会先为自己打造好用的工具，这也叫做近水楼台先得月。

IDE就是程序员的工具

集成开发环境

IDE的全称是：Integrated Development Environment，简称IDE，也稱爲Integration Design Environment、Integration Debugging Environment，翻译成中文叫做“集成开发环境”，在台湾那边叫做“整合開發環境”。它是一種輔助程式開發人員開發軟體的應用軟體。

下面就直接抄维基百科上的说明了：

> IDE通常包括程式語言編輯器、自動建立工具、通常還包括除錯器。有些IDE包含編譯器／直譯器，如微软的Microsoft Visual Studio，有些则不包含，如Eclipse、SharpDevelop等，这些IDE是通过调用第三方编译器来实现代码的编译工作的。有時IDE還會包含版本控制系統和一些可以設計圖形用户界面的工具。許多支援物件導向的現代化IDE還包括了類別瀏覽器、物件檢視器、物件結構圖。雖然目前有一些IDE支援多種程式語言（例如Eclipse、NetBeans、Microsoft Visual Studio），但是一般而言，IDE主要還是針對特定的程式語言而量身打造（例如Visual Basic）。

对这两个的定义，不用死记硬背，google一下。记住爱因斯坦说的那句话：书上有的我都不记忆（是这么的说？好像是，大概意思，反正我也不记忆）。后半句他没说，我补充一下：忘了就google。

> 定义1：类似4、-2、129486655、-988654、0这样形式的数，称之为整数 定义2：类似4.0、-2.0、2344.123、3.1415926这样形式的数，称之为浮点数

比较好理解，整数，就是小学学过的整数；浮点数，就是小数。如果整数写成小数形式，比如4写成4.0，也就变成了浮点数。

爱学习，就要有探索精神。看官在网上google一下整数，会发现还有另外一个词：长整数（型）。顾名思义，就是比较长的整数啦。在有的语言中，把这个做为单独一类区分开，但是，在python中，我们不用管这个了。只要是整数，就只是整数，不用区分长短（以前版本区分），因为区分没有什么意思，而且跟小学学过的数学知识不协调。

还有一个问题，需要向看官交代一下，眼前可能用不到，但是会总有一些人用这个来忽悠你，当他忽悠你的时候，下面的知识就用到了。

整数溢出问题

这里有一篇专门讨论这个问题的文章，推荐阅读：整数溢出

对于其它语言，整数溢出是必须正视的，但是，在python里面，看官就无忧愁了，原因就是python为我们解决了这个问题，请阅读拙文：大整数相乘

ok!看官可以在IDE中实验一下大整数相乘。


```
>>> 123456789870987654321122343445567678890098876*1233455667789990099876543332387665443345566
152278[illegible]771[illegible]352756287004435[illegible]5[illegible]7[illegible]7[illegible]2[illegible]277562328362032444339019158937017801601677976183816L
```

看官是幸运的，python解忧愁，所以，选择学习python就是珍惜光阴了。

上面计算结果的数字最后有一个L，就表示这个数是一个长整数，不过，看官不用管这点，反正是python为我们搞定了。

在结束本节之前，有两个符号需要看官牢记（不记住也没关系，可以随时google，只不过记住后使用更方便）

- 整数，用int表示，来自单词：integer
- 浮点数，用float表示，就是单词：float

可以用一个命令：type(object)来检测一个数是什么类型。

```
>>> from __future__ import division
>>> 5/2
2.5
>>> 9/2
4.5
>>> 9.0/2
4.5
>>> 9/2.0
4.5
```

注意了，引用了一个模块之后，再做除法，就不管什么情况，都是得到浮点数的结果了。

这就是轮子的力量。

关于余数

前面计算5/2的时候，商是2，余数是1

余数怎么得到？在python中（其实大多数语言也都是），用 `%` 符号来取得两个数相除的余数.

实验下面的操作：


```
>>> 5%2
1
>>> 9%2
1
>>> 7%3
1
>>> 6%4
2
>>> 5.0%2
1.0
```

符号：%，就是要得到两个数（可以是整数，也可以是浮点数）相除的余数。

前面说python有很多人见人爱的轮子（模块），她还有丰富的内建函数，也会帮我们做不少事情。例如函数 `divmod()`

```
>>> divmod(5,2)  #表示5除以2，返回了商和余数
(2, 1)
>>> divmod(9,2)
(4, 1)
>>> divmod(5.0,2)
(2.0, 1.0)
```

写两个大字：Hello,World

Hello,World 是面向世界的标志，所以，写任何程序，第一句一定要写这个，因为程序员是面向世界的，绝对不畏缩在某个局域网内，所以，所以看官要会科学上网，才能真正与世界Hello。

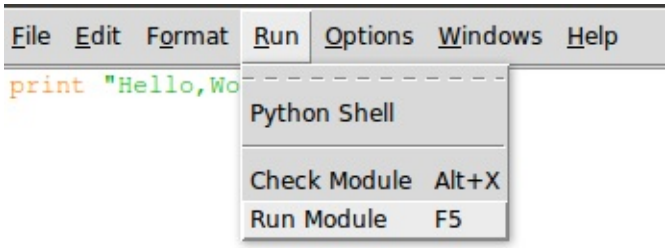
直接上代码，就这么一行即可。

```
print "Hello,World"
```

如下图的样式

前面说过了，程序就是指令的集合，现在，这个程序里面，就一条指令。一条指令也可以成为集合。

注意观察，菜单上有一个RUN，点击这个菜单，在下拉的里面选择Run Moudle



会弹出对话框，要求把这个文件保存，这就比较简单了，保存到一个位置，看官一定要记住这个位置，并且取个文件名，文件名是以.py为扩展名的。

都做好之后，点击确定按钮，就会发现在另外一个带有>>>的界面中，就自动出来了Hello,World两个大字。

成功了吗？成功了也别兴奋，因为还没有到庆祝的时候。

在这种情况系，我们依然是在IDLE的环境中实现了刚才那段程序的自动执行，如果脱离这个环境呢？

下面就关闭IDLE，打开shell(如果看官在使用苹果的 Mac OS 操作系统或者某种linux发行版的操作系统，比如我使用的是ubuntu)，或者打开cmd(windows操作系统的用户，特别提醒用windows的用户，使用windows不是你的错，错就错在你只会使用鼠标点来点去，而不想也不会使用命令，更不想也不会使用linux的命令，还梦想成为优秀程序员。)，通过命令的方式，进入到你保存刚才的文件目录。

下图是我保存那个文件的地址，我把那个文件命名为105.py，并保存在一个文件夹中。

```
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ ls
105.py
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$
```

然后在这个shell里面，输入：python 105.py

上面这句话的含义就是告诉计算机，给我运行一个python语言编写的程序，那个程序文件的名称是105.py

我的计算机我做主。于是它给我乖乖地执行了这条命令。如下图：

```
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ ls
105.py
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ python 105.py
Hello,World
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$
```

还在沉默？可以欢呼了，德国队7:1胜巴西对，列看官中，不管是德国队还是巴西队的粉丝，都可以欢呼，因为你在程序员道路上迈出了伟大的第二步。顺便预测一下，本届世界杯最终冠军应该是：中国队。（还有这么扯的吗？）

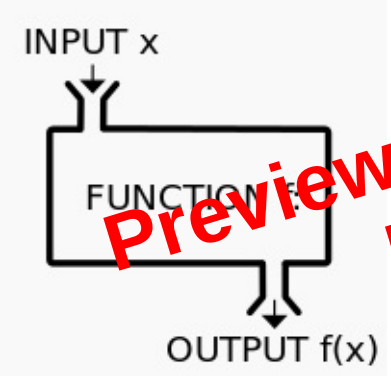
永远强大的函数

函数，对于人类来讲，能够发展到这个数学思维层次，是一个飞跃。可以说，它的提出，直接加快了现代科技和社会的发展，不论是现代的任何科技门类，乃至于经济学、政治学、社会学等，都已经普遍使用函数。

下面一段来自维基百科（在本教程中，大量的定义来自维基百科，因为它真的很百科）：函数词条

> 函数这个数学名词是莱布尼兹在1694年开始使用的，以描述曲线的一个相关量，如曲线的斜率或者曲线上的某一点。莱布尼兹所指的函数现在被称作可导函数，数学家之外的普通人一般接触到的函数即属此类。对于可导函数可以讨论它的极限和导数。此两者描述了函数输出值的变化同输入值变化的关系，是微积分学的基础。 中文的“函数”一词由清朝数学家李善兰译出。其《代数学》书中解释：“凡此變數中函（包含）彼變數者，则此爲彼之函數”。

函数，从简单到复杂，各式各样。前面提供的维基百科中的函数词条，里面可以做一个概览。但不管什么样子的函数，都可以用下图概括：



有初中数学水平都能理解一个大概了。这里不赘述。

本讲重点说明用python怎么来做一个函数用一用。

深入理解函数

在中学数学中，可以用这样的方式定义函数：y=4x+3，这就是一个一次函数，当然，也可以写成：f(x)=4x+3。其中x是变量，它可以代表任何数。

```
当x=2时，代入到上面的函数表达式：
f(2) = 4*2+3 = 11
所以：f(2) = 11
```

```
>>> a=3
>>> y
8
>>> y=3*a+2
>>> y
11
```

特别注意，如果没有先a=2，就直接下函数表达式了，像这样，就会报错。

```
>>> y=3*a+2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  NameError: name 'a' is not defined
```

注意看错误提示，a是一个变量，提示中告诉我们这个变量没有定义。显然，如果函数中要使用某个变量，不得不提前定义出来。定义方法就是给这个变量赋值。

建立实用的函数

上面用命令方式建立函数，还不够“正规化”，那么就来写一个.py文件吧。

在IDLE中，File->New window

然后输入如下代码：


```
#codin :utf-8

def add_function(a,b):
    c = a+b
    print c

if __name__=="__main__":
    add_function(2,3)
```

然后将文件保存，我把她命名为106-1.py，你根据自己的喜好取个名字。

然后我就进入到那个文件夹，运行这个文件，出现下面的结果，如图：

```
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ ls
105-1.py  105.py  106-1.py
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ python 106-1.py
5
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$
```

你运行的结果是什么？如果没有得到上面的结果，你就非常认真地检查代码，是否跟我写的完全一样，注意，包括冒号和空格，都得一样。冒号和空格很重要。

变量连接到字符串

前面讲过变量了，并且有一个钓鱼的比喻。如果忘记了，请看前一章内容。

其实，变量不仅可以跟数字连接，还能够跟字符串连接。

```
>>> a=5
>>> a
5
>>> print a
5
>>> b="hello,world"
>>> b
'hello,world'
>>> print b
hello,world
```

还记得我们曾经用过一个type命令吗？现在它还有用，就是检验一个变量，到底跟什么类型联系着，是字符串还是数字？

```
>>> type(a)
<type 'int'>
>>> type(b)
<type 'str'>
```


程序员们经常用一种简单的说法，把a称之为整型变量，意思就是它能够或者已经跟数字连着呢；把b叫做字符（串）型变量，意思就是它能够或者已经跟字符串连着呢。

对字符串的简单操作

对数字，有一些简单操作，比如四则运算就是，如果3+5，就计算出为8。那么对字符串都能进行什么样的操作呢？试试吧：

```
>>> "py"+"thon"
'python'
```

跟我那个不为大多数人认可的发现是一样的，你还不认可吗？两个字符串相加，就相当于把两个字符串连接起来。(别的运算就别尝试了，没什么意义，肯定报错，不信就试试)

```
>>> "py"-"thon"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

特别说明，编号是从左边开始，第一个是0。

能不能从右边开始编号呢？可以。这么人见人爱的python难道这点小要求都不满足吗？

```
>>> a[-1]
'd'
>>> a[11]
'd'
>>> a[-12]
'H'
>>> a[-3]
'l'
```

看到了吗？如果从右边开始，第一个编号是-1,这样就跟从左边区分开了。也就是a[-1]和a[11]是指向同一个字符。

不管从左边开始还是从右边开始，都能准确找到某个字符。看官喜欢从哪边开始就从哪边开始，或者根据实际使用情况，需要从哪边开始就从哪边开始。

字符串截取

有了编号，不仅仅能够找出某个字符，还能在字符串中取出一部分来。比如，从“hello,world”里面取出“llo”。可以这样操作

```
>>> a[2:5]
'llo'
```


这就是截取字符串的一部分，注意：所截取部分的第一个字符（l）对应的编号是(2)，从这里开始；结束的字符是(o)，对应编号是（4），但是结束的编号要增加1,不能是4,而是5.这样截取到的就是上面所要求的了。

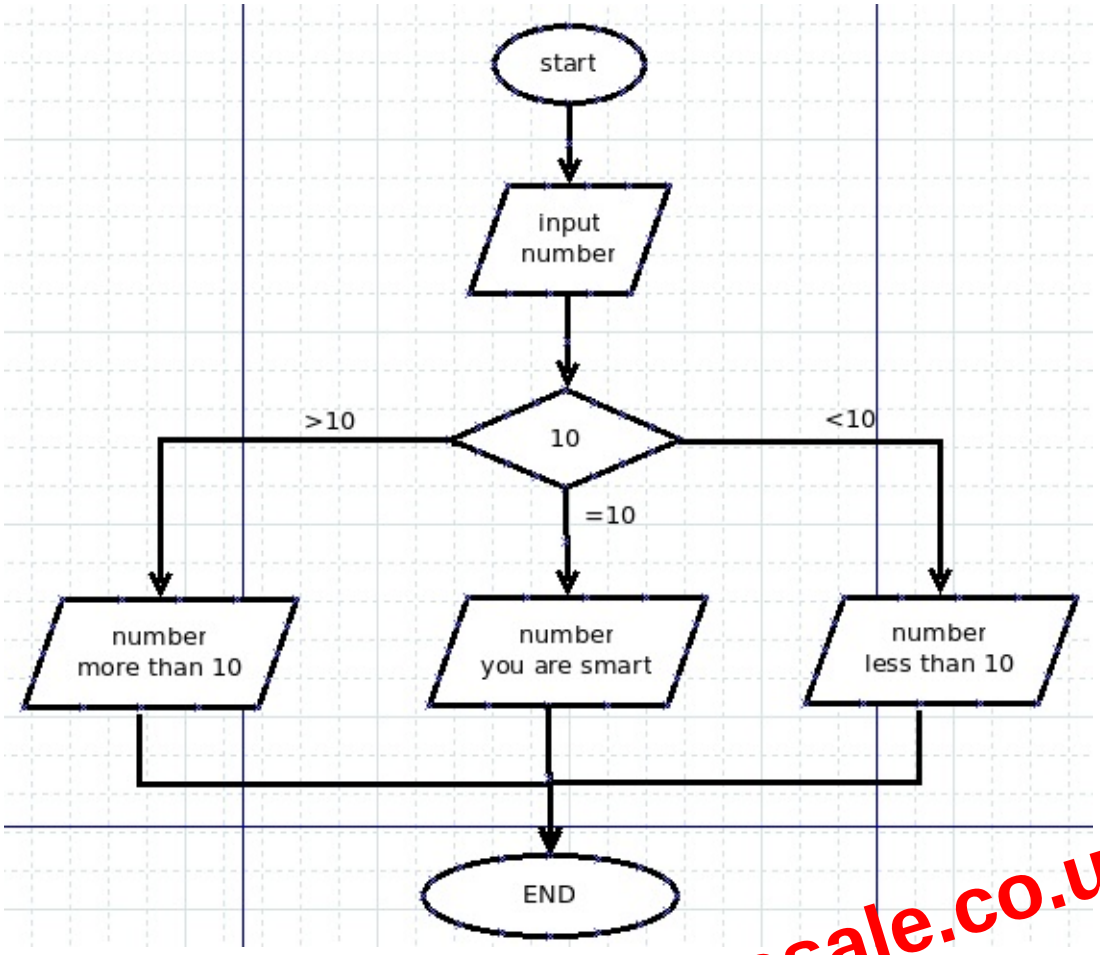
试一试，怎么截取到",wor"

也就是说，截取a[n,m]，其中n<m，得到的字符是从a[n]开始到a[m-1]

有几个比较特殊的

```
>>> a[:]    #表示截取全部
'Hello,world'
>>> a[3:]   #表示从a[3]开始，一直到字符串的最后
'lo,world'
>>> a[:4]   #表示从字符串开头一直到a[4]前结束
'Hell'
```

去掉字符串两头的空格



理解了流程图中的含义，就开始写代码，代码实例如下：


```
#! /usr/bin/env python
#coding:utf-8

print "请输入任意一个整数数字："

number = int(raw_input())    #通过raw_input()输入的数字是字符串
                             #用int()将该字符串转化为整数

if number == 10:
    print "您输入的数字是：%d"%number
    print "You are SMART."
elif number > 10:
    print "您输入的数字是：%d"%number
    print "This number is more than 10."
elif number < 10:
    print "您输入的数字是：%d"%number
    print "This number is less than 10."
else:
    print "Are you a human?"
```

特别提醒看官注意，前面我们已经用过raw_input()函数了，这个是获得用户在界面上输入的信息，而通过它得到的是字符串类型的数据。可以在IDLE中这样检验一下：

要通过系统搜索路径寻找python解释器。不同系统，可能解释器的位置不同，所以这种方式能够让代码更将拥有可移植性。对了，以上是对Unix系列操作系统而言。对与windows系统，这句话就当不存在。

```
>>> la
[1, 2, 3, 'a', 'b', 'c']
>>> lb
['qiwsir', 'python']
>>> la[len(la):]=lb
>>> la
[1, 2, 3, 'a', 'b', 'c', 'qiwsir', 'python']
```

list.extend(L) 等效于 list[len(list):] = L，L是待并入的list

联想到到上一讲中的一个list函数list.append(),这里的extend函数也是将另外的元素（只不过这个元素是列表）增加到一个已知列表中，那么两者有什么不一样呢？看下面例子：

```
>>> lst = [1,2,3]
>>> lst.append(["qiwsir","github"])
>>> lst
[1, 2, 3, ['qiwsir', 'github']]  #append的结果
>>> len(lst)
4

>>> lst2 = [1,2,3]
>>> lst2.extend(["qiwsir","github"])
>>> lst2
[1, 2, 3, 'qiwsir', 'github']   #extend的结果
>>> len(lst2)
5
```


append是整建制地追加，extend是个体化扩编。

list中某元素的个数

上面的len(L)，可得到list的长度，也就是list中有多少个元素。python的list还有一个操作，就是数一数某个元素在该list中出现多少次，也就是某个元素有多少个。官方文档是这么说的：

> list.count(x)
>
> Return the number of times x appears in the list.

一定要不断实验，才能理解文档中精炼的表达。

先实验list.remove(x)，注意看上面的描述。这是一个能够删除list元素的方法，同时上面说明告诉我们，如果x没有在list中，会报错。

```
>>> all_users
['python', 'http://', 'qiwsir', 'github', 'io', 'algorithm']
>>> all_users.remove("http://")
>>> all_users       #的确是把"http://"删除了
['python', 'qiwsir', 'github', 'io', 'algorithm']

>>> all_users.remove("tianchao")        #原list中没有“tianchao”，要删除，就报错。
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

注意两点：

- 如果正确删除，不会有任何反馈。没有消息就是好消息。
- 如果所删除的内容不在list中，就报错。注意阅读报错信息：x not in list

看官是不是想到一个问题？如果能够在删除之前，先判断一下这个元素是不是在list中，在就删，不在就不删，不是更智能吗？

如果看官想到这里，就是在编程的旅程上一进步。python的确让我们这么做。

```
>>> all_users
['python', 'qiwsir', 'github', 'io', 'algorithm']
>>> "python" in all_users   #这里用in来判断一个元素是否在list中，在则返回True，否则返回False
True

>>> if "python" in all_users:
...     all_users.remove("python")
...     print all_users
... else:
...     print "'python' is not in all_users"
...
['qiwsir', 'github', 'io', 'algorithm']     #删除了"python"元素

>>> if "python" in all_users:
...     all_users.remove("python")
...     print all_users
... else:
...     print "'python' is not in all_users"
...
'python' is not in all_users        #因为已经删除了，所以就没有了。
```

上述代码，就是两段小程序，我是在交互模式中运行的，相当于小实验。

另外一个删除list.pop([i])会怎么样呢？看看文档，做做实验。

```
>>> welcome_str
'Welcome you'
>>> welcome_str[0]+"E"+welcome_str[2:]  #从新生成一个str
'WElcome you'
>>> welcome_str                         #对原来的没有任何影响
'Welcome you'
```

其实，在这种做法中，相当于从新生成了一个str。

多维list

这个也应该算是两者的区别了，虽然有点牵强。在str中，里面的每个元素只能是字符，在list中，元素可以是任何类型的数据。前面见的多是数字或者字符，其实还可以这样：

```
>>> matrix = [[1,2,3],[4,5,6],[7,8,9]]
>>> matrix = [[1,2,3],[4,5,6],[7,8,9]]
>>> matrix[0][1]
2
>>> mult = [[1,2,3],['a','b','c'],'d','e']
>>> mult
[[1, 2, 3], ['a', 'b', 'c'], 'd', 'e']
>>> mult[1][1]
'b'
>>> mult[2]
'd'
```


以上显示了多维list以及访问方式，在多维的情况下，里面的list也跟一个前面元素一样对待。

list和str转化

str.split()

这个内置函数实现的是将str转化为list。其中str=""是分隔符。

在看例子之前，请看官在交互模式下做如下操作：

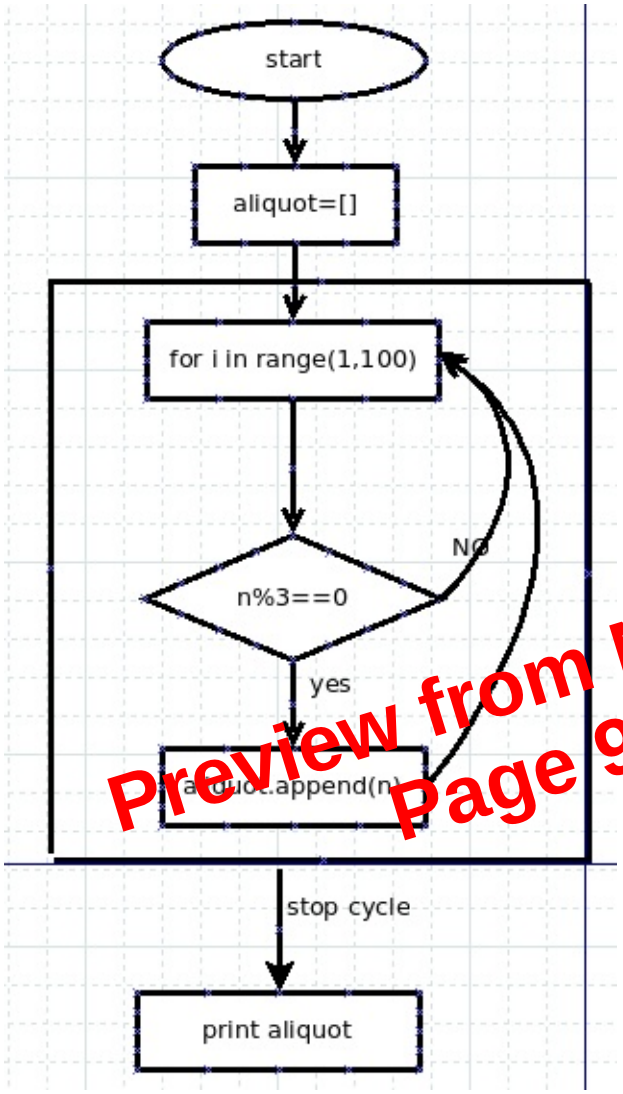
```
>>>help(str.split)
```

得到了对这个内置函数的完整说明。特别强调：这是一种非常好的学习方法

例：找出100以内的能够被3整除的正整数。

分析：这个问题有两个限制条件，第一是100以内的正整数，根据前面所学，可以用range(1,100)来实现；第二个是要解决被3整除的问题，假设某个正整数n，这个数如果能够被3整除，也就是n%3(%是取余数)为0.那么如何得到n呢，就是要用for循环。

以上做了简单分析，要实现流程，还需要细化一下。按照前面曾经讲授过的一种方法，要画出问题解决的流程图。



下面写代码就是按图索骥了。

代码：

```
>>> for i in range(len(week)):
...     print week[i]+' is '+str(i)     #注意，i是int类型，如果和前面的用+连接，必须是str类型
...
monday is 0
sunday is 1
friday is 2
```

python中提供了一个内置函数enumerate，能够实现类似的功能

```
>>> for (i,day) in enumerate(week):
...     print day+' is '+str(i)
...
monday is 0
sunday is 1
friday is 2
```

算是一个有意思的内置函数了，主要是提供一个简单快捷的方法。

官方文档是这么说的：

> Return an enumerate object. sequence must be a sequence, an iterator, or some other object which supports iteration. The next() method of the iterator returned by enumerate() returns a tuple containing a count (from start which defaults to 0) and the values obtained from iterating over sequence

顺便抄录几个例子，供看官欣赏，最好实验一下

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

在这里有类似(0,'Spring')这样的东西，这是另外一种数据类型，待后面详解。

下面将enumerate函数和list解析联合起来，同时显示，在进行list解析的时候，也可以包含进函数（关于函数，可以参考的章节有：初始强大的函数，重回函数）。

```
>>> def treatment(pos, element):
...     return "%d: %s"%(pos,element)
...
>>> seq = ["qiwsir","qiwsir.github.io","python"]
>>> [ treatment(i, ele) for i,ele in enumerate(seq) ]
['0: qiwsir', '1: qiwsir.github.io', '2: python']
```

看官也可以用小话题大函数中的lambda函数来写上面的代码：

```
>>> t
(1, '23', [123, 'abc'], ('python', 'learn'))
>>> tls = list(t)                          #tuple-->list
>>> tls
[1, '23', [123, 'abc'], ('python', 'learn')]

>>> t_tuple = tuple(tls)                   #list-->tuple
>>> t_tuple
(1, '23', [123, 'abc'], ('python', 'learn'))
```

tuple用在哪里？

既然它是list和str的杂合，它有什么用途呢？不是用list和str都可以了吗？

在很多时候，的确是用list和str都可以了。但是，看官不要忘记，我们用计算机语言解决的问题不都是简单问题，就如同我们的自然语言一样，虽然有的词汇看似可有可无,用别的也能替换之,但是我们依然需要在某些情况下使用它们.

一般认为,tuple有这类特点,并且也是它使用的情景:

- Tuple 比 list 操作速度快。如果您定义了一个值的常量集，并且唯一要用它做的是不断地遍历它，请使用 tuple 代替 list。
- 如果对不需要修改的数据进行“写保护”，可以使代码更安全。使用 tuple 而不是 list 如同拥有一个隐含的 assert 语句，说明这一数据是常量。如果必须要改变这些值，则需要执行 tuple 到 list 的转换(需要使用一个特殊的函数)。
- Tuple 可以在 dictionary 中被用做 key，但是 list 不行。实际上，事情要比这更复杂。Dictionary key 必须是不可变的。Tuple 本身是不可改变的，但是如果您有一个 list 的 tuple，那就认为是可变的了，用做 dictionary key 就是不安全的。只有字符串、整数或其它对 dictionary 安全的 tuple 才可以用作 dictionary key。
- Tuples 可以用在字符串格式化中，后面会用到。

```
>>> s1
set(['q', 'i', 's', 'r', 'w'])
>>> s1[1] = "I"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support item assignment

>>> s1
set(['q', 'i', 's', 'r', 'w'])
>>> lst = list(s1)
>>> lst
['q', 'i', 's', 'r', 'w']
>>> lst[1] = "I"
>>> lst
['q', 'I', 's', 'r', 'w']
```

上面的探索中,将set和list做了一个对比,虽然说两者都能够做原处修改,但是,通过索引编号(偏移量)的方式,直接修改,list允许,但是set报错.

那么,set如何修改呢?

更改set

还是用前面已经介绍过多次的自学方法,把set的有关内置函数找出来,看看都可以对set做什么操作.

```
>>> dir(set)
['__and__', '__class__', '__cmp__', '__contains__', '__delattr__', '__doc__', '__eq__', '
```

为了看的清楚,我把双划线__开始的先删除掉(后面我们会有专题讲述这些):

> 'add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection',
> 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove',
> 'symmetric_difference', 'symmetric_difference_update', 'union', 'update'

然后用help()可以找到每个函数的具体使用方法,下面列几个例子:

增加元素

```
>>> help(set.update)
update(...)
    Update a set with the union of itself and others.

>>> s1
set(['a', 'b'])
>>> s2
set(['github', 'qiwsir'])
>>> s1.update(s2)       #把s2的元素并入到s1中.
>>> s1                  #s1的引用对象修改
set(['a', 'qiwsir', 'b', 'github'])
>>> s2                  #s2的未变
set(['github', 'qiwsir'])
```

删除

```
>>> help(set.pop)
pop(...)
    Remove and return an arbitrary set element.
    Raises KeyError if the set is empty.

>>> b_set
set(['[1,2,3]', 'h', 'o', 'n', 'p', 't', 'qiwsir', 'y'])
>>> b_set.pop()     #从set中任意选一个删除,并返回该值
'[1,2,3]'
>>> b_set.pop()
'h'
>>> b_set.pop()
'o'
>>> b_set
set(['n', 'p', 't', 'qiwsir', 'y'])

>>> b_set.pop("n")  #如果要指定删除某个元素,报错了.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: pop() takes no arguments (1 given)
```


set.pop()是从set中任意选一个元素,删除并将这个值返回.但是,不能指定删除某个元素.报错信息中就告诉我们了,pop()不能有参数.此外,如果set是空的了,也报错.这条是帮助信息告诉我们的,看官可以试试.

要删除指定的元素,怎么办?

```
>>> help(set.remove)

remove(...)
    Remove an element from a set; it must be a member.

    If the element is not a member, raise a KeyError.
```

set.remove(obj)中的obj,必须是set中的元素,否则就报错.试一试:

```
>>> a_set
set(['i', 'a', 'qiwsir'])
>>> a_set.remove("i")
>>> a_set
set(['a', 'qiwsir'])
>>> a_set.remove("w")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'w'
```

跟remove(obj)类似的还有一个discard(obj):

```
>>> help(set.discard)

discard(...)
    Remove an element from a set if it is a member.

    If the element is not a member, do nothing.
```


与help(set.remove)的信息对比,看看有什么不同.discard(obj)中的obj如果是set中的元素,就删除,如果不是,就什么也不做,do nothing.新闻就要对比着看才有意思呢.这里也一样.

```
>>> a_set.discard('a')
>>> a_set
set(['qiwsir'])
>>> a_set.discard('b')
>>>
```

在删除上还有一个绝杀,就是set.clear(),它的功能是:Remove all elements from this set.(看官自己在交互模式下help(set.clear))

假设两个集合A、B

- A是否等于B，即两个集合的元素完全一样

在交互模式下实验

```
>>> a
set(['q', 'i', 's', 'r', 'w'])
>>> b
set(['a', 'q', 'i', 'l', 'o'])
>>> a == b
False
>>> a != b
True
```

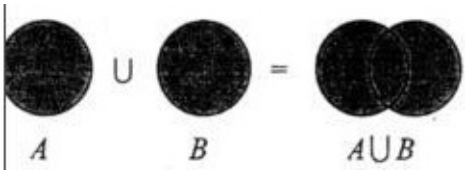
- A是否是B的子集，或者反过来，B是否是A的超集。即A的元素也都是B的元素，但是B的元素比A的元素数量多。

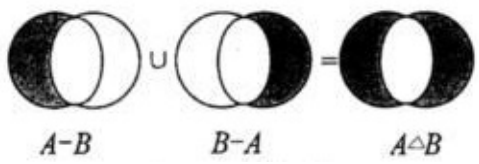
实验一下

```
>>> a
set(['q', 'i', 's', 'r', 'w'])
>>> c
set(['q', 'i'])
>>> c<a      #c是a的子集
True
>>> c.issubset(a)    #或者用这种方法，判断c是否是a的子集
True
>>> a.issuperset(c)  #判断a是否是c的超集
True

>>> b
set(['a', 'q', 'i', 'l', 'o'])
>>> a<b      #a不是b的子集
False
>>> a.issubset(b)    #或者这样做
False
```


- A、B的并集，即A、B所有元素，如下图所示






```
>>> a
set(['q', 'i', 's', 'r', 'w'])
>>> b
set(['a', 'q', 'i', 'l', 'o'])
>>> a.symmetric_difference(b)
set(['a', 'l', 'o', 's', 'r', 'w'])
```

以上是集合的基本运算。在编程中，如果用到，可以用前面说的方法查找。不用死记硬背。

```
>>> l1
[1, 2, 3]
>>> l2
[1, 2, 3]
>>> l1 == l2     #两个相等，是指内容一样
True
>>> l1 is l2     #is 是比较两个引用对象在内存中的地址是不是一样
False            #前面的检验已经说明，这是两个东东

>>> l3 = l1     #顺便看看如果这样，l3和l1应用同一个对象
>>> l3
[1, 2, 3]
>>> l3 == l1
True
>>> l3 is l1     #is的结果是True
True
```

某些对象，有copy函数，通过这个函数得到的对象，是一个新的还是引用到同一个对象呢？看官也可以做一下类似上面的实验，就晓得了。比如：

```
>>> l1
[1, 2, 3]
>>> l2 = l1[:]
>>> l2
[1, 2, 3]
>>> l1[0] = 22
>>> l1
[22, 2, 3]
>>> l2
[1, 2, 3]

>>> adict = {"name":"qiwsir","web":"qiwsir.github.io"}
>>> bdict = adict.copy()
>>> bdict
{'web': 'qiwsir.github.io', 'name': 'qiwsir'}
>>> adict["email"] = "qiwsir@gmail.com"
>>> adict
{'web': 'qiwsir.github.io', 'name': 'qiwsir', 'email': 'qiwsir@gmail.com'}
>>> bdict
{'web': 'qiwsir.github.io', 'name': 'qiwsir'}
```


不过，看官还有小心有点，python不总按照前面说的方式出牌，比如小数字的时候

```
>>> import random
>>> items = [1, 2, 3, 4, 5, 6]
>>> random.shuffle(items)
>>> items
[3, 2, 5, 6, 4, 1]
```

有点多了。不过，本次实验中，值用到了random.randint()即可。多出来是买一送一的（哦。忘记了，没有人买呢，本课程全是白送的）。

关键技术点之一已经突破。可以编程了。再梳理一下流程。画个图展示：

（备注：这里我先懒惰一下吧，看官能不能画出这个程序的流程图呢？特别是如果是一个初学者，流程图一定要自己画哦。刚才看到网上一个朋友说自己学编程，但是逻辑思维差，所以没有学好。其实，画流程图就是帮助提高逻辑思维的一种好方式，请画图吧。）

图画好了，按照直观的理解，下面的代码是一个初学者常常写出来的（老鸟们不要喷，因为是代表初学者的）。

```
#!/usr/bin/env python
#coding:utf-8

import random

number = random.randint(1,100)

print "请输入一个100以内的自然数："

input_number = raw_input()

if number == int(input_number):
    print "猜对了，这个数是："
    print number
else:
    print "错了。"
```


上面的程序已经能够基本走通，但是，还有很多缺陷。

最明显的就是只能让人猜一次，不能多次。怎么修改，能够多次猜呢？动动脑筋之后看代码，或者看官在自己的代码上改改，能不能实现多次猜测？

另外，能不能增强一些友好性呢，让用户知道自己输入的数是大了，还是小了。

根据上述修改想法，新代码如下：