- 引言
  - 如何在自己的电脑上运行Python
    - 本教程是给谁写的?
    - 要选哪个版本呢?
    - 在自己的windows上安装Python
    - 打印个 "Hello World"
    - 练习
  - 变量，数字
    - 变量是啥
      - 如何给变量起名:
      - Python 是一门动态语言哦
      - 不要小瞧给变量赋值 这一简单的动作
    - 数字
      - 数字还能分不同的类型?
      - 我能变成你，你能变成我
      - 数字的运算符有哪些
    - 练习
  - Python中的"数组" -- 列表，元组
    - 序列：我是啥?
      - 序列:我有啥(操作)?
    - 序列的小弟之一 -- 列表 list
      - 如何生成列表
      - 列表有哪些操作
      - 列表的方法
    - 序列的小弟之二 -- 元组 tuple
      - 如何生成元组
      - 元组的操作
    - 练习
  - 字符串：我也是一种序列
    - 字符串是啥?
    - 索引 -- 也可以称之为下标
    - 字符串：我有哪些常用方法
      - 字符串的格式化

        - 字符串的拼接
        - 字符串的切分
        - 字符串的替换
        - 字符串的查找
        - 字符串修饰
        - 字符串的大小写转换
        - 字符串的判定
    - 练习
- key + value --> dict (字典)
    - 创建字典的两种方法
    - key 有啥特性
    - dict 增删改查如何实现
    - dict ---> str
    - 字典有哪些常用方法
    - 练习
- 集合 set--无序的不重复的序列
    - 如何创建 `set`
    - `set` 的常用方法
    - 赋值魔法
    - 练习
- 最最最基本的分支和循环
    - 分支
    - 语句块
        - 又一种数据类型：布尔值 (bool)
        - 如何进行条件判断
    - if -- else 的一种新用法
    - 聊聊运算符
    - 循环 --重复做同一件事
    - GET新技能 -- 列表推导式
    - 练习
- 函数
    - 函数是什么
    - 如何定义函数
    - 参数详解

        - 神秘的作用域
        - 没有名字的函数 -- 匿名函数
        - 练习
    - 让我们来谈谈对象吧
        - 如何定义类
        - 如何实例化类
        - 面向对象编程三大特性之继承
        - 练习
    - 遇到错误了咋办 -- 异常处理
        - 如何定义异常情况
        - 异常也能主动出发的哦
        - 一些重要的异常类
        - 让我们捕捉异常
        - 练习
    - 模块
        - 自己也能建立模块并给别人使用
        - 导入模块还有其他的方式
        - 很重要的 `__name__` 变量
        - 多个模块 --> 包
        - 库
            - 介绍一些常用的标准库
        - 练习
    - Open the File
        - file 的基本操作方法
        - 练习

引言

在经济寒冬下，面对越来越多的测试同学咨询职场困惑以及测试技术转型之道，相对于动辄好几万的测试培训费，我觉得有义务带领大家一起成长，一起踩坑，学习测试开发贵在找到合适的方向，活跃的学习氛围，以及敢于挑战的自信；所以鲲鹏老师利用业余时间编写了python基础测试开发手册，python在测试领域几乎已经成为人人必会的脚本语言，熟练掌握python也是求职找工作的必备技能。我们会从最简单知识点开始，一步步带大家由浅入深学习，这本手册会让大家以最快的方式入门python测试开发，只学习互联网企业中最常用的模块，并附带大量的课后习题，后续我还会根据大家的学习情况编写自动化测试框架代码送给大家，还请大家关注我们的微信公众号 猿桌派，我们会在公众号内及时推送更新的内容。大家共勉！！！

如何在自己的电脑上运行Python

本教程是给谁写的?

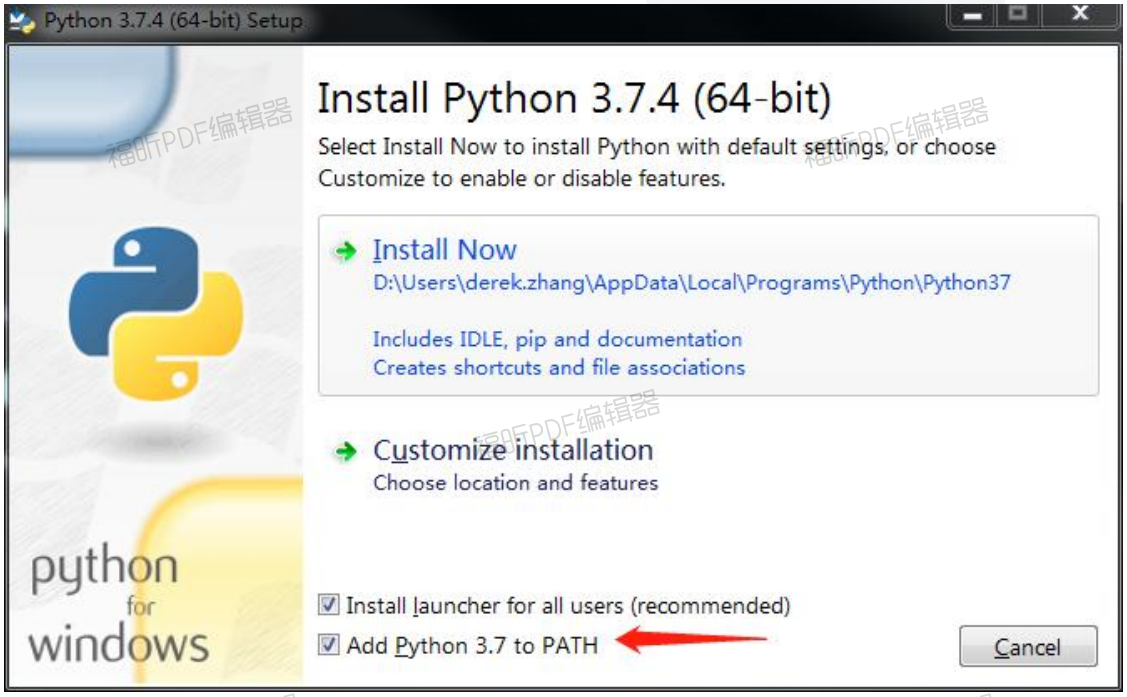
本教程适合想从零开始学习且无任何Python知识基础的同学。本教程不会花篇幅来介绍Python的历史，优缺点之类的内容，而是直接从Python的基础语法开始讲解，所以相关内容大家有兴趣的话可以自行百度。

要选哪个版本呢?

Python有两个版本，分别是2.x 和 3.x，但这两个版本是不兼容的，目前3.x版本普及度越来越高，且2.X 版本在2020.1.1已终止支持，所以本教程讲以3.7版本为基础作为讲解。

在自己的windows上安装Python

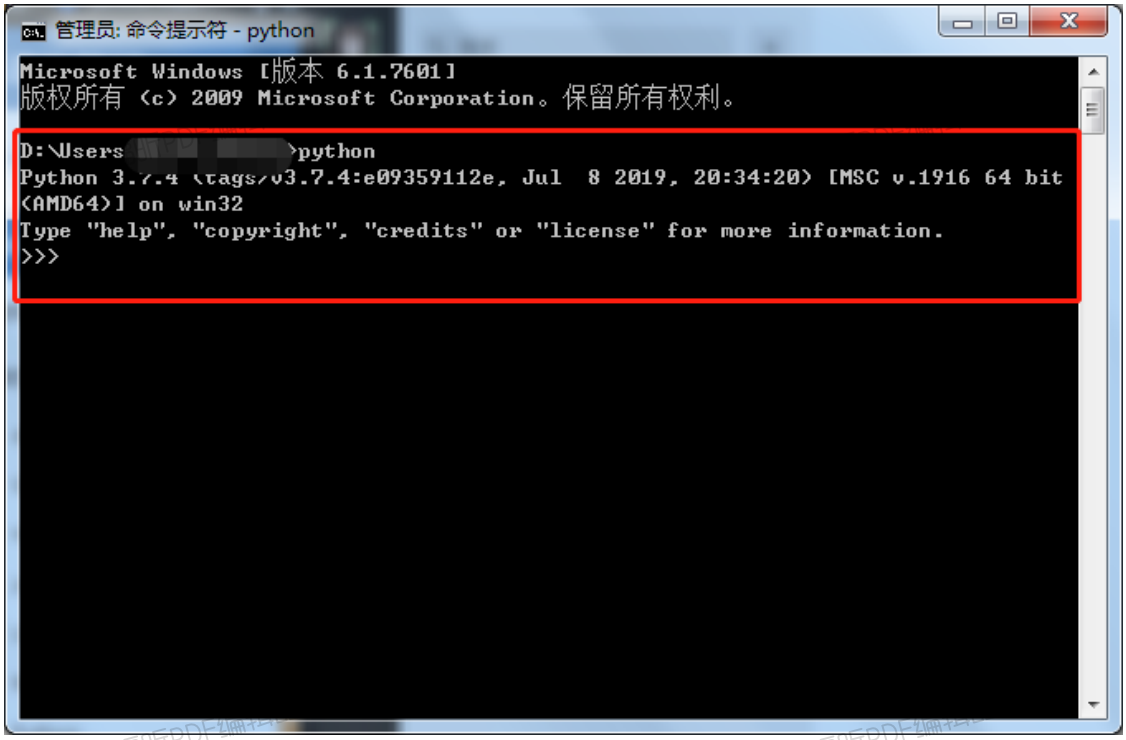
根据你的windows版本(32位/64位)从Python官网(https://www.python.org/)下载对应版本，一路傻瓜安装即可，注意下图步骤勾上 `Add Python 3.8 to PATH` 。



> 如安装时没有勾选图中所示，则安装后得手动添加环境变量。

打印个 "Hello World"

安装成功后，打开命令行提示窗口，输入python，如果出现下图所示，就说明安装成功。



我们看到的 `>>>` 就表示我们已经进入Python 的交互环境中了，你输入Python代码，回车后就会得到执行结果，我们输入 `print('hello world')` 后回车，屏幕上就会显示 `hello world` ,如图所示：

```
(devops) ~ python
Python 3.7.3 (default, Mar 27 2019, 09:23:15)
[Clang 10.0.1 (clang-1001.0.46.3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>>
```

> 在Python 中，如果要打印什么信息，就用内置的函数 `print()` 就可以了

如果想退出交互模式，则输入 `exit()` 回车即可

练习

1. 成功安装python3.x
2. 能够进入python 交互模式

变量，数字

变量是啥

如何给变量起名：

变量其实是通过定义的一个标记用来调用内存中的值，这个标记的名称就是变量名。只有符合下面条件的变量名才是有效的变量名，否则就会报错。

1. 不能使用关键字()
2. 第一个字符不能为数字
3. 变量名只能为 字母，数字 ，_ 的组合

> Python 的关键字有`['and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while', 'with', 'yield']`

Python 是一门动态语言哦

在Python中定义变量不需要申明类型，如定义一个数字类型的变量a:

```
a = 1
```

定义一个字符串类型的变量b

```
b = 'hello'
```

> 变量本身类型不固定，称为动态语言，与之对应的是 静态语言，如java定义变量时要申明类型 int a = 1。

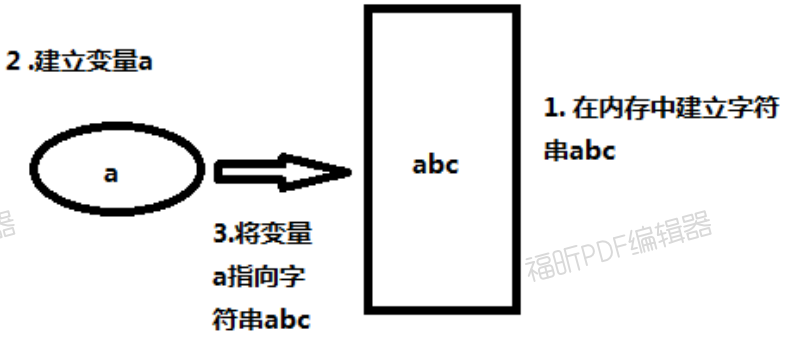
不要小瞧给变量赋值 这一简单的动作

我们在程序中定义了变量a并对他进行赋值，`a='abc'`

这简单的一行代码Python解释器进行了3步操作：

1. 在内存中建立创建了 值为'abc' 一个字符串
2. 在内存中创建了一个名为 a 的变量
3. 将 变量 a 指向 字符串 经过这上面3步操作，a 就代表了'abc'

```
>>> a = 'abc'
>>> a
'abc'
```



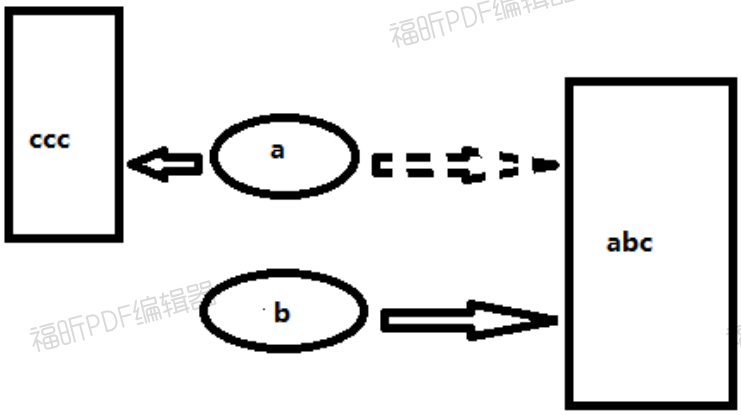
如果再编写以下代码

```
b = a
a = 'ccc'
```

那么 b 是等于 ccc 呢 ， 还是 abc 呢？答案是 abc，

```
>>> a = 'abc'
>>> a
'abc'
>>> b = a
>>> a = 'ccc'
>>> b
'abc'
```

以上代码我们用下图展示其逻辑:



我们可以看出，当我们把一个变量a赋值给另一个变量b，这个操作实际上是把变量b指向变量a所指向的数据。

数字

数字还能分不同的类型?

数字是 Python中比较常用的数据类型，数字有可以分为：

1. 整型 int
   如 `1,2,3`
2. 浮点型 float
   如 `2.1,3.5`
3. 复数 complex(不常用)

> 复数由实数部分和虚数部分构成，可以用a + bj,或者complex(a,b)表示， 复数的实部a和虚部b都是浮点型
> Python 内置的type()函数可以用来查询变量所指向的对象类型

```
# 注意python 的这种赋值方式,Python 可以同时为多个变量赋值
>>> a,b,c = 2,20.1,4+3j
>>> type(a)
<type 'int'>
>>> type(b)
<type 'float'>
>>> type(c)
<type 'complex'>
```

我能变成你，你能变成我

Python 中数字的类型是可以相互转化的

1. 通过类型函数来转化
   类型函数有 `int(),float(),complex()`
   我们通过类型函数来改变上面定义的3个变量的类型，示例如下:

```
>>> print(type(int(b)))
<type 'int'>
>>> print(type(float(a)))
<type 'float'>
>>> print(type(complex(b)))
<type 'complex'>
```

2. 通过运算来进行转化
   示例如下:

```
>>> print(type(3/2),3/2)
<type 'int'> 1
>>> print(type(3/2.0),3/2.0)
<type 'float'> 1.5
```

值得注意的是，类型的转化会带来数据精度上的变化

```
>>> a = 2.1
>>> type(a)
<class 'float'>
>>> int(a)
2
```

> print() 函数如果要打印出多个值，可以用','隔开

数字的运算符有哪些

数字的运算主要包括 `+,-,*,/,%,//,**`
示例如下：

```
>>> print(1+2) #加
3
>>> print(3-1) #减
2
>>> print(5*6) #乘
30
>>> print(6/3) #除   返回结果为 浮点型
2.0
>>> print(5%3) #取余
2
>>> print(3//2.0) #整除
1.0
>>> print(2**3) #幂
8
```

练习

1. 练习把一个数字赋值给一个变量，并打印出来,并尝试集中错误的变量命名方式，看看都会报什么错。
2. 8/3 , 8//3 , -8//3 分别是多少?

Python中的"数组" -- 列表，元组

序列：我是啥?

在介绍列表和元组前，先引入两个概念：

1. 数据结构
   即通过某种方式组织在一起的元素的集合。
2. 序列
   最基本的数据结构， 序列中每个元素会被分配一个序号，即元素的位置，也称为 索引 从左往右，从 0 开始。

> Python 中有6中内置序列，字符串，列表，元组，buffer对象，xrange对象，Unicode字符串，我们会通过列表来展示序列的共有特性。 也就是说这些特性，不管是 字符串，列表，还是元组 ，buffer对象都具有。

我们这边用列表来展示序列的共有特性，先简单介绍一下列表的形式，即用[]来表示，中间的元素用`,`隔开。如`[1,2,3]`

1. 序列种可以包换不同类型的元素

```
# 既有字符串，又有数字
a = ['zhang',23]
```

2. 序列种可以包含其他序列

```
a = ['zhang' ,23]
data  = [a,33]
print (data)   ----->[['zhang', 23], 33]
```

序列:我有啥(操作)?

1. 索引 需要注意的是，数使用负数作为索引时，Python 会从右边开始技术，最后一个元素编号是 -1，另外，字符串字面值就可以直接使用索引。看下面示例：

```
>>> a = ['a','b','c','d']
>>> print(a[0])
a
>>> print(a[1])
b
>>> print(a[-1])
d
>>> print(a[-2])
c


greeting= 'hello'
print (greeting[0])  --->h
print (greeting[-1]) --->o
print (greeting[-4]) --->e
print ('hello'[1])   --->e
```

2. 分片 使用分片操作来访问一定范围内的元素，如果分片中左边的索引比右边的索引晚出现在序列中，结果就是一个空的序列(步长为1).

```
nums = [1,2,3,4,5,6,7,8,9,10]
```

`nums[3:6]` 表示 索引从 3 开始，但不到6，左闭右开

```
print (nums[3:6])  --->[4, 5, 6]
print (nums[0:1])  --->[1]
print (nums[-3:-1]) -->[8, 9]
print (nums[-3:])  --> [8, 9, 10]
print (nums[3:])   --> [4,5,6,7,8,9,10]
print (nums[:])    -->[1,2,3,4,5,6,7,8,9,10]
nums[-3:0] -->[]
#可以指定步长
nums[0:10:1] -->[1,2,3,4,5,6,7,8,9,10]
nums[0:10:2] -->[1,3,5,7,9]
#步长也可以是负数，这样就从右往左取
nums[8:3:-1]  -->[9,8,7,6,5]
```

3. 序列相加 需要注意的是，两种相同类型的序列才可以相加。否则会报错

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

>>> 'hello' + 'zhang'
'hellozhang'
>>> [1,2,3] + 'hello'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "str") to list
```

4. 序列相乘

```
>>> 'python' *3
'pythonpythonpython'

>>> [1,2,3] *2
[1, 2, 3, 1, 2, 3]
```

5. 成员资格 判断某个元素是否在序列中

```
>>> permission = 'yw'
>>> 'y' in permission
True

>>> 'x' in  permission
False

>>> user = ['zhang','wang']
>>> 'zhang' in user
True

>>> 'li' in user
False
```

6. 长度，最大值，最小值

```
>>> nums  = [63,81,11]
>>> len(nums)
3
>>> max(nums)
81
>>> min(nums)
11
```

以上就是序列中(部分常见的)的共有特性，也就是说列表，元组，字符串都具有上面的特性。

序列的小弟之一 -- 列表 list

列表(list)是序列的一种，是由 `[]` 包围的，以“，”隔开的元素的集合,可以随时添加或删除其中的元素。

如何生成列表

1. 由 `list()` 函数生成，示例如下:

```
>>> list('abc')
['a', 'b', 'c']

>>> type(list('abc'))
<type 'list'>
```

> 注意 list() 中参数必须是可迭代的，在这，你可以暂时理解为字符串即可。

2. 由 `[]` 直接定义产生，示例如下:

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

>>> type([1,2,3,4])
<type 'list'>
```

列表有哪些操作

这一部分涉及到列表的增删改查，我们直接看示例代码：

```
>>> a = [1,2,3]  #将列表[1,2,3] 赋值给变量a
#  修改列表中索引1的值
>>> a[1] = 5
>>> a
[1, 5, 3]

#删除元素
>>> del a[2]
>>> a
[1, 5]

#分片赋值
>>> name = list('perl')
>>> name
['p', 'e', 'r', 'l']
>>> name[2:] = list('ar')
>>> name
['p', 'e', 'a', 'r']

#分片赋值可以代替原序列中长度不同的值
>>> name = list('Perl')
>>> name
['P', 'e', 'r', 'l']
>>> name[1:] = list('ython')
>>> name
['P', 'y', 't', 'h', 'o', 'n'],

#插入
>>> nums = [1,5]
>>> nums[1:1] = [2,3,4]
>>> nums
[1, 2, 3, 4, 5]
#删除
>>> nums[1:4] = []
>>> nums
[1, 5]
```

列表的方法

1. `append()` 在列表的末尾增加新的对象

```
#           语法
#               l.append(object)
#           参数
#               object
#                   想要添加的元素
#           返回值
#               None
#           注意
#               会直接修改原数组

>>> lst = [2,'23']
>>> lst.append(3)
>>> lst
[2, '23', 3]
```

2. `count()` 统计某个元素在列表中出现的次数,不存在的返回0

```
>>> x = [[1,2],1,3,1,2]
>>> x.count(1)
2
>>> x.count([1,2])
1
>>> x.count(5)
0
```

3. `extend()` 可以在列表的末尾一次性追加另一个序列的多个值，也就是可以用新列表扩展原有列表 ,与列表的连接作用不同，连接作用返回一个新的列表， `extend()` 是修改了被扩展的列表

```
#     extend
#         作用
#             往列表中, 扩展另外一个可迭代序列
#         语法
#             l.extend(iterable)
#         参数
#             iterable
#                 可迭代集合
#                     字符串
#                     列表
#                     元组
#                     ...
#         返回值
#             None
#         注意
#             会直接修改原数组


>>> a =[1,2,3]
>>> b= [4,5,6]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6]


>>> a= [1,2,3]
>>> b = [4,5,6]
>>> a + b
[1, 2, 3, 4, 5, 6]
>>> a
[1, 2, 3]
```

4. `index()` 从列表中找出某个值第一个匹配项的索引位置 找不到会抛错!

```
>>> a = ['we','are','yong']
>>> a.index('are')
1
>>> a.index('aa')
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    a.index('aa')
ValueError: 'aa' is not in list
```

5. `insert()` 将对象插入到列表指定的索引位置 --修改了 原列表的值 却返回 None 值

```
#     insert
#         作用
#             往列表中，追加一个新的元素
#             在指定索引前面
#         语法
#             l.insert(index, object)
#         参数
#             index
#                 索引，到时会插入到这个索引之前
#             object
#                 想要添加的元素
#         返回值
#             None
#         注意
#             会直接修改原数组

>>> nums = [1,2,3,4,5]
>>> a =  nums.insert(3,'zhang')
>>> print(a)
None
>>> nums
[1, 2, 3, 'zhang', 4, 5]
```

6. `pop()` 移除列表中最后一个元素，并返回该元素的值

```
#     pop
#         作用
#             移除并返回列表中指定索引对应元素
#         语法
#             L.pop(index=-1)
#         参数
#             index
#                 需要被删除返回的元素索引
#                 默认是-1
#                     也就对应着列表最后一个元素
#         返回值
#             被删除的元素
#         注意
#             会直接修改原数组
#             注意索引越界

>>> nums = [1,2,3,4,5]
>>> nums.pop()
5
>>> nums
[1, 2, 3, 4]
```

7. `reverse()` 将列表中的原色反向存放,修改了 原列表的值 却返回 None 值

```
>>> x = [1,2,3]
>>> print(x.reverse())
None
>>> x
[3, 2, 1]
```

8. `remove()` 用于移除列表中的第一个匹配项，如果没有匹配报错，改变了列表返回 None，同 `reverse`(元素反向)

```
#   remove
#         作用
#             移除列表中指定元素
#         语法
#             L.remove(object)
#         参数
#             object
#                 需要被删除的元素
#         返回值
#             None
#         注意
#             会直接修改原数组
#             如果元素不存在
#                 会报错
#             若果存在多个元素
#                 则只会删除最左边一个
#             注意循环内删除列表元素带来的坑

>>> x = ['to', 'be','or' 'not','to','be']
>>> x.remove('be')
>>> x
['to', 'ornot', 'to', 'be']
>>> x.remove('too')
Traceback (most recent call last):
  File "<pyshell#68>", line 1, in <module>
    x.remove('too')
ValueError: list.remove(x): x not in list
```

9. `sort()` 排序，修改原列表的值，返回None。

```
>>> x = [2,1,4,7,5]
>>> x.sort()
>>> x
[1, 2, 4, 5, 7]
```

序列的小弟之二 -- 元组 tuple

元组也是序列的一种，和列表相似，元素以`,`分割，以`()`包围，但不能修改。

如何生成元组

1. `tuple()` 函数

```
>>> tuple('abc')
('a', 'b', 'c')
```

2. 直接由 `()` 定义产生

```
type((1,2,3))
<type 'tuple'>
```

3. 用逗号隔开一些值，就自动创建了元组

```
>>> a = 1,2,3
>>> type(a)
<type 'tuple'>
```

元组的操作

元组的操作与列表相似，但切记，元组不能被改变，所以，列表中某些改变列表的方法在元组中不适用。

练习

1. 通过分片来赋值一个list
2. 如何通过分片来赋值?

```
a = [1,2,3,4,5]
a[:3] = ['a','b',c]
print(a)  --> ?
```

3. 弄清列表的引用 和 复制

```
>>> info = [10,20,30,40]
>>> info1 = info
>>> info1[2]= 50
>>> info1
[10, 20,50, 40]
>>> info   ---?

>>> info = [10,20,30,40]
>>> info2 = info[:]
>>> info2[2] = 50
>>> info2
[10, 20, 50, 40]
>>> info  ----?
```

4. 通过分片创建一个与原列表index 反转的列表
5. 在列表一次性追加多个值 ， 用三种方法实现
6. 一个列表a， `a[::-1]` 与 `a.reverse()` 有什么区别
7. 元组练习
    - 定义一个元组，有10个元素，分别是1,2,3,4,57,'a','b','c','d',2
    - 输出第1~3个元素的值
    - 调用方法，查询出 2 出现的次数
    - 查出值是 2 的序列号
    - 转化成列表

字符串：我也是一种序列

字符串是啥?

字符串是一个有序的，不可修改的，元素以引号包围的序列。-- 注意，还是一个序列 定义方式：

1. 单引号

```
>>> print(type('hello'))
<class 'str'>
```

2.双引号

```
>>> print(type("hello"))
<class 'str'>
```

3.三个单引号或三个双引号

```
>>> print(type('''hello'''))
<class 'str'>

>>> print(type("""hello"""))
<class 'str'>
```

这里需要注意的是

- 单引号 和 双引号要区分开，
    i. 如果字符中有单引号，那么可以用双引号来定义该字符串 `a = "what's you name"`
    ii. 如果字符中有单引号,且偏要用单引号来定义该字符串，那么可以用转义符 \, `a = 'waht\'s you name'`
    iii. 如果字符中有双引号，那么可以用单引号来定义该字符串 `print 'hello "good" boy'`
- 三引号和非三引号的区别 区别在于三引号可以定义带有换行的多行字符串，所以三引号通常会用来作为长注释，多行注释。

```
>>>print("""hello
world""")
hello
world
```

- 字符串是不可变的

```
>>> a = 'abc'
>>> a[1] = 'd'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

索引 -- 也可以称之为下标

序列中有介绍，我们这变就不介绍了。 字符串里的每一个个体都被称之为该字符串的一个元素

字符串：我有哪些常用方法

字符串的格式化

字符串格式化适用字符串格式化操作符即百分号% 来实现

```
>>> format = 'hello ,%s,%s enough for you'
>>> values = ('haha','zhang')
>>> print(format % values)
hello ,haha,zhang enough for you
```

字符串的拼接

方法	解释
join	按照自定义方法连接列表为字符串
+	将两个字符串拼接起来
*	重复

```
>>> l = ['a','b','c','d']
>>> print(''.join(l))
abcd
>>> print('m'.join(l))
ambmcmd
# +,* 在序列中的例子中演示过
```

字符串的切分

方法	解释
split	切分，可以指定切分次数和对象，默认以空格切分
splitlines	行切分字符串

```
>>> a = 'i am a badyboy'
>>> a.split()
['i', 'am', 'a', 'badyboy']
>>> a.split('',2)

Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    a.split('',2)
ValueError: empty separator
>>> a.split('a')
['i ', 'm ', ' b', 'dyboy']
>>> a.split('a',2)
['i ', 'm ', ' badyboy']



>>> a = '''i'm a boy
i'm 20
i am in shanghai'''
>>> a.splitlines()
["i'm 20", 'i am in shanghai']
```

> 这边要注意当以空格为切分符时，不能指定次数

字符串的替换

方法	解释
replace	从左到右替换指定的元素，可以指定替换的个数，默认是全部替换

```
>>> a = 'hello boy'
>>> a.replace('o','a')
'hella bay'
>>> a.replace('o','a',1)
'hella boy'
```

字符串的查找

方法	解释
count	计数功能，返回指定字符在字符串当中的个数
find	查找，返回从左第一个指定字符的索引，找不到返回-1
rfind	查找，返回从右第一个指定字符的索引，找不到返回-1
index	查找，返回从左第一个指定字符的索引，找不到报错
rindex	查找，返回从右第一个指定字符的索引，找不到报错

```
>>> a  = 'hello zhang'
>>> a.count('h')
2
>>> a.count('w')
0
>>> a.find('h')
0
>>> a.find('w')
-1
>>> a.rfind('h')
7
>>> a.rfind('w')
-1
>>> a.index('h')
0
>>> a.index('w')

Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    a.index('w')
ValueError: substring not found
>>> a.rindex('h')
7
>>> a.rindex('w')

Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    a.rindex('w')
ValueError: substring not found
```

字符串修饰

方法	解释
center	字符串在指定长度居中，默认以空格填充，也可指定字符填充，左短右长
ljust	字符串在指定长度左对齐，默认以空格填充，也可指定字符填充
rjust	字符串在指定长度右对齐，默认以空格填充，也可指定字符填充
zfill	将字符串填充到指定的长度，不足地方用0从左开始补充
format	按照顺序，将后面的参数传递给前面的大括号
strip	默认去除两边的空格，去除内容可以指定
lstrip	默认去除左边的空格，去除内容可以指定
rstrip	默认去除右边的空格，去除内容可以指定

```
>>> a = 'hello world'
>>> print(a)
hello world
>>> print(a.center(20))
    hello world
>>> print(a.center(20,'*'))
****hello world*****

>>> print(a.ljust(23,'%'))
hello world%%%%%%%%%%%%

>>> print(a.rjust(20,'#'))
#########hello world

>>> print(a.zfill(20))
000000000hello world

>>> print('my name is {},age is {}'.format('derek',28))
my name is derek,age is 28

>>> b = '    123   '
>>> print(b)
    123
>>> print(b.strip())
123
>>> c = '***555***'
>>> print(c)
***555***
>>> print(c.strip('*'))
555
```

字符串的大小写转换

方法	解释
upper	将字符串当中的所有字母转换为大写
lower	将字符串当中的所有字母转换为小写
swapcase	将字符串当中的所有字母大小写互换
title	将字符串中的单词首字母大写，其他字母都小写
capitalize	将字符串的首字母大写

```
>>> a = 'abcDEFg'
>>> print(a.upper())
ABCDEFG

>>> print(a.lower())
abcdefg

>>> print(a.swapcase())
ABCdefG

>>> b = 'this is a boy'
>>> print(b.title())
This Is A Boy

>>> print(b.capitalize())
This is a boy
```

字符串的判定

方法	解释
isalnum	判断字符串是否完全由数字和字母组成
isalpha	判断字符串是否完全由字母组成
isdigit	判断字符串是否完全由数字组成
isupper	判断字符串是否完全是大写
islower	判断字符串是否完全是大写
isspace	判断字符串是否完全由空格组成
startswith	判断字符串的开头字符，可以截取判断
endswith	判断字符串的结尾字符，可以截取判断

```
>>> a = 'sdfsd234'
>>> a.isalnum()
>>> b = 'sdf23_se'
>>> b.isalnum()
False
>>> c = 'saf'
>>> c.isalpha()
True
>>> c = '23ad'
>>> c.isalpha()
False

>>> d = '123'
>>> d.isdigit()
True
>>> d= 's234'
>>> d.isdigit()
False

>>> a = 'adb34dsfbc'
>>> a.startswith('ab')
False
>>> a.startswith('ad')
True
>>> a.endswith('c')
True
>>> a.endswith('be')
False
```

练习

1. 将一个英文语句以单词为单位逆序排放。例如“I am a boy”，逆序排放后为“boy a am I”（华为面试题）
2. 有如下变量 x = ' zhoU',实现如下功能：
   - 移除 x 变量对应的值两边的空格
   - 判断 x 变量对应的值是否以 "zh"开头和以"U"结尾，并输出结果
   - 将 x 变量对应的值中的 “ol” 替换为 “x”，并输出结果
   - 将 x 变量对应的值根据 “h” 分割，并输出结果
   - 输出 x 变量对应的值中 “U” 所在索引位置
   - 输出 x 变量中索引为2 he 3 的字符

key + value --> dict (字典)

上一讲的 `string` 方法比较多，大家需要多多的联系，才能做到熟能生巧。这一讲，我们主要来介绍字典(dict)。

字典(dict)是Python的一种内置的数据结构。在其他语言中也称为map,使用键-值（key-value）存储，可以通过查找某个特定的词语（键 key），从而找到他的定义（值 value）。

创建字典的两种方法

1. 直接定义

```
>>> phonebook = {'zhang':'231','wang':'123'}
>>> type(phonebook)
<type 'dict'>
```

2. 通过dict函数

```
>>> items  = [('name','zhang'),('age',2)]
>>> d  = dict(items)
>>> d
{'age': 2, 'name': 'zhang'}
>>> type(d)
<type 'dict'>
>>> d["name"]
'zhang'

#dict 函数也可以通过关键字参数来创建字典
>>> d = dict(name ="zhang",age=2)
>>> d
{'age': 2, 'name': 'zhang'}
```

key 有啥特性

字典中的值可以是何的 python 对象，但键不行，键必须是不可变得。关于字典的键有以下两点值得注意：

1. 如果同一个键被赋值两次，后一个值会被记住

```
>>> c ={'age':18,'name':'derek','age':20}
>>> c['age']
20
```

2. 键是不可变得，所以数字，字符串或元组可以作为键，而用列表就不行，如下实例：

```
>>> a = {[234,23]:'aa'}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'

>>> a = {(234,2):'aa'}
>>> a
{(234, 2): 'aa'}
```

dict 增删改查如何实现

```
>>> phonebook = {'zhang':'231','wang':'123'}
#返回phonebook中键值对的数量
>> len(phonebook)
2
# 返回键zhang值
>>> phonebook['zhang']
'231'
>>> phonebook['wang']
'123'
# 将V值赋值到 wang键上
>>> phonebook['wang'] =  666
>>> phonebook
{'wang': 666, 'zhang': '231'}
#若键不在字典中，则自动添加到字典中，和序列不同
>>> phonebook['li'] = '888'
>>> phonebook
{'li': '888', 'wang': 666, 'zhang': '231'}
#删除键wang
>>> del phonebook['wang']
>>> phonebook
{'zhang': '231','li': '888'}
# 检查字典中是否存在 键
>>> 'wang' in phonebook
False
>>> 'zhang' in phonebook
True
# del 也可以删除整个字典 ,删除后，变量也不存在了
>>> del phonebook
>>> phonebook
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'phonebook' is not defined
```

dict ---> str

在每个转换说明符中的%后加上键（用括号），后面再跟上其他说明元素。

```
>>> phonebook = {'zhang':'123','wang':'555'}
>>> "zhang's phone number is %(zhang)s." % phonebook
"zhang's phone number is 123."
# 注意说明元素类型必须与 键所对应的值的类型一致，否则报错

>>> "zhang's phone number is %(wang)d." % phonebook

Traceback (most recent call last):
  File "<pyshell#65>", line 1, in <module>
    "zhang's phone number is %(wang)d." % phonebook
TypeError: %d format: a number is required, not str
```

字典有哪些常用方法

1. `clear` -- 清除字典中的所有项。无返回值，或者可以理解为 返回None

```
>>> d = {}
>>> d['name'] = 'zhang'
>>> d['age'] = 23
>>> d
{'age': 23, 'name': 'zhang'}
>>> return_value = d.clear()
>>> print return_value
None
```

2. `copy` --返回一个具有相同键-值的新字典
   当简单的值替换的时候，原始字典和复制过来的字典之间互不影响，但是当添加，删除等修改操作的时候，两者之间会相互影响。 ---可以理解为浅复制

```
>>> d = {'age': 23, 'name': 'zhang'}
>>> c = d.copy()
>>> c
{'age': 23, 'name': 'zhang'}
>>> c['age'] = 18
>>> c
{'age': 18, 'name': 'zhang'}
>>> d
{'age': 23, 'name': 'zhang'}

>>> d = {'name' : ['An','Assan']  }
>>> c = d.copy()
>>> c
{'name': ['An', 'Assan']}
>>> c['name'].append('body')
>>> c
{'name': ['An', 'Assan', 'body']}
>>> d
{'name': ['An', 'Assan', 'body']}
```

> copy对于一个复杂对象的子对象并不会完全复制，什么是复杂对象的子对象呢？就比如序列里的嵌套序列，字典里的嵌套序列等都是复杂对象的子对象。对于子对象，python会把它当作一个公共镜像存储起来，所有对他的复制都被当成一个引用，所以说当其中一个引用将镜像改变了之后另一个引用使用镜像的时候镜像已经被改变了。原文链接
> https://blog.csdn.net/myordry/article/details/80320353

`deepcopy` 可以避免上面出现的情况，我们可以理解为完全复制然后变成一个新的对象，复制的对象和被复制的对象没有任何关系，彼此之间无论怎么改变都相互不影响。

```
>>> from copy import deepcopy
>>> d = {'name' : ['An','Assan']}
>>> c = deepcopy(d)
>>> c
{'name': ['An', 'Assan']}
>>> c['name'].append('su')
>>> c
{'name': ['An', 'Assan', 'su']}
>>> d
{'name': ['An', 'Assan']}
```

3. `fromkeys` --使用给定的键建立新的字典，每个键默认对应的值为`None`

```
>>> {}.fromkeys(['name','age'])
{'age': None, 'name': None}
#也可以直接在类型函数dict 上引用方法
>>> dict.fromkeys(['name','age'])
{'age': None, 'name': None}
#设置默认值
>>> dict.fromkeys(['name','age'],'unkonw')
{'age': 'unkonw', 'name': 'unkonw'}
```

4. get --更宽松的访问字典项的方法，如果键不存在，返回None

```
>>> d = {}
>>> print d.get('a')
None
#定义默认值，若果不存在的话不返回None，返回默认值
>>> d.get('a','N/A')
'N/A'

>>> d['a']

Traceback (most recent call last):
  File "<pyshell#119>", line 1, in <module>
    d['a']
KeyError: 'a'
>>>
```

5. items itmes -- 将字典所有项以列表方式返回 列表中每一项都会表示为（键，值）对的形式，但没有什么次序

```
>>> d = {'titile':'i am a tiger','age':10}
>>> d.items()
[('age', 10), ('titile', 'i am a tiger')]
```

6. keys keys -- 将字典中的键以列表形式返回，

```
>>> d = {'titile':'i am a tiger','age':10}
>>> d.keys()
['age', 'titile']
```

7. pop --用来获得对应的键的值，并删除这对键值对

```
>>> d
{'age': 10, 'titile': 'i am a tiger'}
>>> d.pop('age')
>>> d
{'titile': 'i am a tiger'}
#不存在键，报错
>>> d.pop('name')

Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    d.pop('name')
KeyError: 'name'
```

8. popitem --弹出随机的项，并删除

```
>>> d
{'age': 10, 'titile': 'i am a tiger'}
>>> d.popitem()
('age', 10)
>>> d
{'titile': 'i am a tiger'}
```

9. setdefault --类似于get方法，能够获得给定键相关联的值，还能在字典中不含有给定键的情况下设定相应的键值

```
>>> d = {}
>>> d.setdefault('name','N/A')
>>> d
{'name': 'N/A'}
>>> d['name'] = 'zhang'
>>> d.setdefault('name','N/A')
>>> d
{'name': 'zhang'}
```

10. update --利用一个字典更新另外一个字典 ， 被更新的字典 会加到 原字典中，键相同的话会被覆盖

```
>>> d = {'title':'zhang','age':10}
>>> x = {'age':20}
>>> d.update(x)
>>> d
{'age': 20, 'title': 'zhang'}
>>> y = {'sex':'male'}
>>> d.update(y)
>>> d
{'age': 20, 'sex': 'male', 'title': 'zhang'}
```

练习

集合 set --无序的不重复的序列

集合（set）是一个无序的不重复元素序列，集合中的元素不能重复,且集合是无序的，不能通过索引和分片进行操作。

如何创建 `set`

1. set() 创建一个集合，需要提供一个序列（可迭代的对象）作为输入参数：

```
#字符串
>>> set('abc')
set(['a', 'c', 'b'])
#列表
>>> set(['a','b','c'])
set(['a', 'c', 'b'])
#元组
>>> set(('a','b','c'))
set(['a', 'c', 'b'])

# 集合中的元素不重复
>>> set('aabc')
set(['a', 'c', 'b'])

#整数
>>> set(123)

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    set(123)
TypeError: 'int' object is not iterable
```

2. {}

```
>>> {1,2,3}
{1, 2, 3}
>>> a = {1,2,3}
>>> type(a)
<class 'set'>
```

但注意 利用 {} 来创建集合不能创建空集合，因为 {} 是用来创造一个空的字典

`set` 的常用方法

1. `set` 的添加和删除，更新

```
>>> a = set('abc')
>>> a
set(['a', 'c', 'b'])

#添加元素
>>> a.add('d')
>>> a
set(['a', 'c', 'b', 'd'])

#重复添加无效果
>>> a.add('d')
>>> a
set(['a', 'c', 'b', 'd'])

#删除元素
>>> a.remove('c')
>>> a
set(['a', 'b', 'd'])

#update 把要传入的元素拆分，作为个体传入到集合中
>>> a.update('abdon')
>>> a
set(['a', 'b', 'd', 'o', 'n'])
```

2. `set`的集合操作符

python符号	含义
减号（-）	差集，相对补集
&	交集
|	合集，并集
!=	不等于
==	等于
in	是成员关系
not in	不是成员关系

```
>>> a = set('abc')
>>> b = set('cdef')
>>> a&b
set(['c'])
>>> a | b
set(['a', 'c', 'b', 'e', 'd', 'f'])
>>> a -b
set(['a', 'b'])
>>> 'a' in a
True
>>> 'e' in a
False
>>> a != b
True
>>> a == b
False
```

集合还有不可变集合`frozenset`,用的不多，有兴趣的同学可以自行学习下。

基础数据结构我们就讲解到这里，本讲的其余部分我们将介绍几个知识点.

赋值魔法

1. 序列解包

```
#多个赋值同时进行
>>> x,y,z = 1,2,3
>>> print x,y,z
1 2 3

#变量互换
>>> x,y = y,x
>>> print x,y
2 1
```

上面的例子做的其实是将 多个值得序列解开，然后放到变量的序列中也称为 序列解包 或 可迭代解包 ， 可以用下面的代码形象的表示

```
>>> values = 1,2,3
>>> values
(1, 2, 3)
>>> x,y,z = values
>>> print x,y,z
1 2 3
```

所解包的序列中的元素数据必须和赋值号= 左边的变量数完全一致，否则会报错

```
>>> x,y = 1,2,3

Traceback (most recent call last):
  File "<pyshell#37>", line 1, in <module>
    x,y = 1,2,3
ValueError: too many values to unpack
```

2. 链式赋值 将同一个值付给多个变量的捷径

```
a = b = somefunciton()

等价于

b = somefunciton()
a = b
```

3. 增量赋值

```
>>> x = 2
>>> x+=2    # x = x+2
>>> x *=3   # x= x*3
>>> x
12
```

练习

1. 输入一段字符串，打印所有输入过的字符串，但重复的只打印一次，(不要求打印的顺序与输入顺序一致) ,并打印出出现的次数.
   如：输入 abbccddabs
   打印：a:2
   b:3
   c:2 d:2 s:1

最最最基本的分支和循环

我们前面主要讲了Python的基本数据类型，接下来我们就要来介绍Python的 条件判断 和循环了。条件 和循环无论在哪种语言中都是最最最基本的语法之一，务必熟练掌握！

分支

语句块

在介绍 条件判断 和 循环之前我们来介绍下语句块。
语句块是指在条件为真/假（条件语句）或者执行多次（循环语句）的一组语句。
在代码前放置空格来缩进语句就可以创建语句块，并且块中的语句都缩进同样的量。
用下面的伪代码来做下示例：

```
this is the first line
this is the second line:
    this is the block line one
    this is the block line two
    this is the block line three
this is the third line
```

在python 中用冒号（:）来标识语句块的开始，语句块中的每一个语句缩进同样的量，当语句的缩进退回到之前的缩进量时，表明该语句块结束。
条件语句就是让程序选择是否执行语句块。

又一种数据类型：布尔值 （bool）

那条件语句是怎么样让程序选择是否执行语句块的呢 -- 是根据条件语句返回的布尔值(True or False / 真 or 假) 布尔值 只有 两个值 `True`,`False`
在Python 中，下面的值作为布尔表达式（返回布尔值）时，会被解释器看做假(False) `False , None ,0 ,"" ,(),[],{}` 其他值都都被解释为真(True),

```
# True  和  False 属于 bool 型
>>> print type(True),type(False)
<type 'bool'> <type 'bool'>

>>> Ture
True
>>> False
False

# True 和 False 在和数字做运算是 分别代表 1 和  0
>>> True == 1
True

>>> True == 2
False

>>> False == 0
True

>>> False == 1
False


>>> True  + False + 42
43
```

布尔函数可以用转换其他值，看示例：

```
>>> True == 2
False
>>> True == 1
True
>>> bool('hello')
True
>>> bool('')
False
>>> bool(23)
True
>>> bool(0)
False
>>> bool([])
False
>>> bool([213,23])
True
```

所有值都可以被用作是 布尔值，所以几乎不需要对它们进行显示转换，也可以说是 python 自动转换这些值。

如何进行条件判断

Pyhton 通过 `if` , `else` , `elif` 来进行条件判断执行

- `if`

```
#实现条件执行，如果条件(在if和冒号之间的布尔表达式)盘点为真，那么后面的语句块就会被执行，如果条件为假，语句块就不会被执行
a = input('pls enter the number:')
if a > 2:
    print 'bigger than 2'

#输入3 打印
>>>
pls enter the number:3
bigger than 2

# 输入2 不打印
pls enter the number:2
```

> Python 中的 `input()` 函数用来接收外部传进来的值

- else 子句 使用else增加一种选择，else不能独立纯在，只能作为if语句的一部分

```
a = input('pls enter the number:')
if a > 2:
    print 'bigger than 2'
else:
    print 'not bigger than 2'

>>>
pls enter the number:3
bigger than 2

>>>
pls enter the number:1
not bigger than 2
```

- elif 子句 --相当于 switch 需要检查多个条件，可以使用elif。

```
a = input('pls enter the number:')
if a > 2:
    print 'bigger than 2'
elif a < 2:
    print 'not bigger than 2'
else:
    print 'a ==2'



>>>
pls enter the number:0
not bigger than 2

>>>
pls enter the number:3
bigger than 2

>>>
pls enter the number:2
a ==2
```

- 嵌套语句块

```
name = raw_input("pls enter the name:")
if name.endswith('z'):
    if name.startswith('m'):
        print 'hello m'
    elif name.startswith('l'):
        print 'hello l'
    else:
        print'z'
else:
    print 'hello stranger'
```

if -- else 的一种新用法

在java 或者其他语言中，有类似于 (condition) ? a :b 这样的语法，表示如果condition 为真，返回a，反之，返回b。我们称之为三元运算。
但在Python 中，则没有这样的表达，但Python 借助于 `if -- else` 可以时间同样的效果。示例如下：
a,b两个数，a>b 返回 'more',否则，返回'less'

```
>>> a,b = 1,2
>>> c = 'more' if a>b else 'less'
>>> print(c)
less
```

python 还有另一种写法：

```
>>> c = {True:'more',False:'less'}[a>b]
>>> c
'less'
```

大家可以好好体会下。

聊聊运算符

让我们回到条件本身，因为他才是条件执行的前提。

- 比较运算符 在 条件中最基本的运算符就是比较运算符了，它们都返回布尔值 `True` 或者 `Fasle`

表达式	描述
x == y	x 等于y
x < y	x 小于 y
x > y	x 大于y
x <= y	x小于等于y
x >= y	x大于等于y
x != y	x 不等于 y
x is y	x 和y 是同一个对象
x is not y	x 和 y 是不同对象
x in y	x 是 y 的成员
x not in y	x 不是y 的成员

- 布尔运算符
  连接两个布尔值

运算符	说明
and	两者都为真才为真
or	两者有一真即为真
not	取反

循环 --重复做同一件事

- while 循环
  while 循环用来在任何条件为真的情况下重复执行一个语句块。

```
x = 0
while x <=100:
    print x
    x += 1
```

- for 循环 for 循环可以为一个集合（序列和其他可迭代对象）的每一个元素执行一个语句块。 在这里我们可以先把可迭代对象理解为 序列。

```
nums  = [0,1,2,3,4,5,6,7,8,9]
for num in nums:
    print num
```

- 循环遍历字典元素

```
d = {'x':1,'y':2,'z':3}
#遍历字典中的键
for key in d:
    print key,'is',d[key]

# 遍历字典中的值
d = {'x':1,'y':2,'z':3,'a':11}
for value in d.values():
    print value
# 遍历键和值
d = {'x':1,'y':2,'z':3,'a':11}
for key,value in d.items():
    print key,value
```

- 跳出循环 braak 跳出并停止循环 continue 跳出该次循环，并不停止循环

```
# num =2 跳出并终止循环
nums = [1,2,3]
for num in nums:
    if num == 2:
        break
    print num

--------------------------------
>>>
1

# num =2 跳出循环，并进行下一次循环
nums = [1,2,3]
for num in nums:
    if num == 2:
        continue
    print num
--------------------------------
>>>
1
3
```

- 循环中的else 语句 但循环中有break/continue 时， 没有调用break/continue 时执行

```
nums = [1,4,3]
for num in nums:
    if num == 2:
        continue
    print num
else:
    print 'no 2'
------------------------
1
4
3
no 2
```

大家也可以尝试着把列表中的4 改成 2试试。

GET新技能 -- 列表推导式

我们上面学习了`for`循环，利用`for`,我们可以快速的构建列表，这就是我们将要将的列表推导式。先举个例子： 有个列表[1,2,3],我们要将他中的每个元素加1，组成另一个列表，常见做法如下：

```
>>> a = [1,2,3]
>>> b = []
>>> for i in a:
...     b.append(i+1)
...
>>> b
[2, 3, 4]
```

我们用列表推导式用一行代码能起到同样的效果。-

```
>>> c = [i+1 for i in a]
>>> c
[2, 3, 4]
```

`[i+1 for i in a]`就一行代码起到了同样的作用。列表推导式能极大的简化代码，但是会增加可读性难度。其基本格式如下：

```
variable = [out_exp_res for out_exp in input_list if out_exp == 2]
 out_exp_res:    列表生成元素表达式，可以是有返回值的函数。
 for out_exp in input_list:    迭代input_list将out_exp传入out_exp_res表达式中。
 if out_exp == 2:    根据条件过滤哪些值可以。
```

如上面的例子，要排除偶数，只对奇数操作,那么可以这样：

```
>>> d  = [i+1 for i in a if i % 2 > 0]
>>> d
[2, 4]
```

如果对每个元素要进行的操作很复杂，那么可以传入一个函数，示例如下：

```
>>> def fun(i):
        return i*2

>>> e = [fun(i) for i in a if i % 2 > 0]
>>> e
[2, 6]
```

有了列表推导式，那么有没有字典推导式，集合推导式呢？ 这个作为作业，大家自己练习。

练习

1. 猜数

- 给定一个定值，比如说 99
- 让用户输入数字，给用户三次机会，如果三次之内猜对了，显示猜测正确
- 如果三次之内没有猜对，退出循环，显示'stupid'

2. 自己推到出 字典推导式，集合推导式的简单示例。

函数

函数是什么

函数是组织好的，可重复使用的，用来实现某些特定功能的代码段。用通俗的话来讲就是一段实现了具体功能的并且能重复调用的代码块。

如何定义函数

1. 用`def`来定义函数，格式如下：

```
>>> def hello(name):
...     print('hello',name)
...
>>> hello
<function hello at 0x0000000003D02708>
>>> hello('derek')
hello derek
```

上面的例子，函数名为`hello`,他实际上相当于一个变量，指向了函数体在内存中的位置。这样，`hello`就代表了这个而函数体，代码中`hello`是指这个函数对象，`hello()`才是调用这个函数。

我们可以把函数名赋值给一个变量,那么，这个变量就指向了该函数名所对应的函数。 注意，函数名`hi`不加() 只是表示这个`hi`是对应的是一个函数对象，而没有调用它，加上() 才会执行函数语句块里的内容，看下面代码：

```
>>> def hi():
...     print('hi')

>>> a = hi
>>> a
<function hi at 0x1031b0d08>
>>> hi
<function hi at 0x1031b0d08>      # a 和 hi 指向同一个对象
>>> a()
hi
>>> hi()
hi
```

如果定义一个函数，暂时还不知道这个函数要实现什么功能，或者还不知道怎么写功能，可以用pass来代替语句块

```
def func1():
    pass
```

那么调用这个函数什么也不敢。后续讲到的类也是如此。

2. 可以用Python 内置的`callable()`函数来判定是否可以被调用

```
>>> y = hello
>>> callable(y)
True
>>> x  = 1
>>> callable(x)
False
```

3. 记录函数，给函数写注释
   在函数体内用""" 或者 ''' 给函数写注释

```
>>> def funx(x):
...     '''this is add'''
...     return x + 1
...
>>> funx.__doc__
'this is add'
```

可以暂时先把 `__doc__` 理解为对象的帮助文档，我们后续会讲到

4. Python中所有函数都有返回值，如果没有指定返回值，则默认为None

```
>>> def a():
...     return 'a'
...
>>> def b():
...     print(1)

>>> x = a()
>>> print(x)
a
>>> y = b()
1               #这里的1是 执行b()函数体内 print(1) 打印出来的，并非返回值
>>> print(y)
None            # 这里才是b() 函数的返回值
```

参数详解

1. 字符串,数字,元组都是不可变的，所以用他们做参数，在函数内部参数有任何变化，函数外的同名变量不会受影响， 但如果可变元素则为参数，则....

```
# 参数不可变
>>> name = "derek"
>>> def change(name):
    name = 'lu'
>>> change(name)
>>> name
'derek'

# 参数可变
>>> arr = ['derek','lu']
>>> def chang_list(arr):
    arr[0] = 'wang'
>>> chang_list(arr)
>>> arr
['wang', 'lu']
```

2. 参数位置 在调用函数是，如果传入的参数值加上参数名称，则参数位置可调换，看下面代码示例

```
>>> def hello(greeting,name):
...     print('%s,%s' %(greeting,name) )
...
>>> hello('hi','derek')
hi,derek
>>> hello('derek','hi')
derek,hi
>>> hello(name='derek',greeting='hi')       # 关键字参数 位置调换
hi,derek
>>> hello(name='derek','hi')                # 非关键字参数在关键字参数后，报错
  File "<stdin>", line 1
SyntaxError: positional argument follows keyword argument
```

3. 默认参数 在定义函数时，可以给定传入参数的默认值，那么在调用函数时，可以不提供参数，或者部分提供，或者全部提供。
   定义函数的时候默认参数有值得注意的有两点：

- 默认参数必须是不可变的。
- 默认参数必须在正常参数之后

> 大家可以想想是为什么?

```
>>> def hello2(greeting='hello', name='derek'):
...     print('%s,%s' %(greeting,name) )
...
>>> hello2()
hello,derek
>>> hello2('byebye')
byebye,derek
>>> hello2('byebye','force')
byebye,force
>>> hello2(name='froce')
hello,froce
```

4. 可变参数 在Python 中，传入的参数个数可以是可变得，可以是1个，2个...可以是任意个。
   我们来举个例子，比如 打印出每次来访的人员名单，每次来的人员个数并不是一定的，

```
#  第一次 来了两个人
>>> visitors = ['小明','小红']
>>> def greeting(greet,members):
...     for m in members:
...         print('%s,%s' % (greet,m))
...
>>> greeting('hi',visitors)
hi,小明
hi,小红


#  第二次 来了三个人
>>> visitors2 = ['小明','小红','小刚']
>>> greeting('hello',visitors2)
hello,小明
hello,小红
hello,小刚
```

看示例虽然实现了功能，我们每次调用函数前都组装成一个列表作为参数传入，略显繁琐，python 为我们提供了可变参数来解决类似的问题.
解决办法:

```
def fun(*args):
    xxx
    xxx
```

参数前加 * 表示这个参数是可变参数，python 会自动把传进来的参数组装成一个元组。看示例

```
>>> def func(*args):
...     print(type(args))
...     for i in args:
...             print(i)
...
>>> func(1,2)
<class 'tuple'>       # args 是一个元组，系统自动将传入的参数组装成一个元组
1
2
>>> func(1,2,'a')
<class 'tuple'>
1
2
a
```

那么，我们之前的示例可以改为

```
def greeting(greet,*args):
    for m in args:
         print('%s,%s' % (greet,m))


调用该函数
>>> greeting('hi','小明','小红')
hi,小明
hi,小红

>>> greeting('hi','小明','小红','小刚')
hi,小明
hi,小红
hi,小刚
```

可以看到，我们不必讲来访的人员组成列表传入，而是将来访人员直接作为参数参数传入到函数内，但要注意的是，我们不能将可变参数放到前面，如果放到前面，会出现什么样的结果，大家可以试试。

5. 关键字参数 可变参数是在函数调用时自动将传入的0个或者N个参数组装成一个 `tuple`，而关键字参数允许你传入0个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个 `dict`.
   可变参数是用 `*` 来表示，而关键字参数是用 `**` 来表示，看代码示例：

```
>>> def print_params(**kwargs):
...     print(type(kwargs))
...     print(kwargs)
...
>>> print_params()
<class 'dict'>
{}

>>> print_params(x=1)
<class 'dict'>
{'x': 1}
>>> print_params(x=1,y=2)
<class 'dict'>
{'x': 1, 'y': 2}
```

从上面例子我们可以看到，当我们不传任参数时，系统会认为我们传了一个空的 `dict`,但我们传入1个或2个带参数名的参数时，系统会把所传入的参数组装成 `dict`。
举个实际的例子来说明关键字参数在实际中的应用：
比如 函数 `print_person()` 打印人员信息，收集过来的信息 除姓名 和 性别外必填，其他都是非必填，那么关键字参数是如何来实现这个功能的,我们看下面代码示例：

```
>>> def print_person(name,male,**kwargs):
...     print('name is %s,male is %s' %(name,male),'other:',kwargs)
...
# 只传入 name 和 male
>>> print_person('derek','M')
name is derek,male is M other: {}

# 除了 name 和 male 还传入了其他信息
>>> print_person('jack','M', age=10)
name is jack,male is M other: {'age': 10}
>>> print_person('lucy','F', age=10,add='shanghai')
name is lucy,male is F other: {'age': 10, 'add': 'shanghai'}
```

6. 参数组合 在python中定义函数，可以把正常参数，默认参数，可变参数， 关键字参数组合使用，但顺序必须是 正常参数，默认参数，可变参数， 关键字参数。
   定义一个包含所有参数的函数,并传入不同的参数调用：

```
def func(a,b,c='hello',*args,**kw):
    print('a=',a,'b=',b,'c=',c,'args =', args, 'kw =', kw)

>>> func(1,2)
a= 1 b= 2 c= hello args = () kw = {}
>>> func(1,2,'hi')
a= 1 b= 2 c= hi args = () kw = {}
>>> func(1,2,3,'a','b')
a= 1 b= 2 c= 3 args = ('a', 'b') kw = {}
>>> func(1,2,3,'a','b',x=1)
a= 1 b= 2 c= 3 args = ('a', 'b') kw = {'x': 1}
>>> func(1,2,3,'a','b',x=1,y=2)
a= 1 b= 2 c= 3 args = ('a', 'b') kw = {'x': 1, 'y': 2}
```

> 在实际工作中不建议将太多的参数组合，会影响函数的理解性。

神秘的作用域

函数内的变量 被称为局部变量（与全局变量对应，全局变量可以暂时理解为 函数外的变量），参数的工作原理类似于局部变量。

函数内读取全局变量正常，但如果全局变量名 和 函数内的 全局变量名相同，则将屏蔽全局变量。（本节开头演示）

- 函数内 读取 全局变量的方法 globals()['name']。

```
# 函数内调用全局变量
>>> name = 'derek'
>>> name2 = 'wang'
>>> def change2(value):
          print (value , globals()['name'])
>>> change2(name2)
wangderek
```

- 屏蔽全局变量

```
>>> x = 1
>>> def change(x):
       x = 2
>>> x
1
```

- 重新绑定全局变量 --在函数内声明一个变量，且这个变量是全局变量

```
>>> x = 1
>>> def change_global():
     global x
     x =  x +2
>>> change_global()
>>> x
3
```

- 嵌套 - -函数内定义函数

```
>>> def foo():
      def bar():
         print 'hello bar()'
      bar()
>>> foo()
hello bar()
```

没有名字的函数 -- 匿名函数

我们上面介绍的函数都是通过 关键字 `def` + 函数名() 来定义的，那么，函数能不能没有名字，这样就不必担心函数名冲突了-。
答案是肯定的，Python 中通过`lambda`关键字来定义匿名函数，匿名函数也是一个函数对象，也可以赋值给变量，在利用变量来调用函数。匿名函数示例如下：

```
lambda x,y: x-y
```

表示这个函数接受两个参数 x,y,并返回x-y

```
>>> f = lambda x,y:x-y
>>> f
<function <lambda> at 0x107ac3d08>
>>> f(7,2)
5
```

匿名函数对象赋值给了变量f，f指向了该函数，所以调用f(7，2)就是就是调用该匿名函数。

> 匿名函数是 函数式编程的基础，所以大家必须熟练掌握。函数式编程我们后续将会专门讲解。

练习

1. 定义一个函数，实现传入整数的累乘的和，比如出啊如5 ，实现1+2!+3!+...+5!的和。
2. 编写一个函数cacluate, 可以接收任意多个数,返回的是一个元组.
3. 模拟轮盘抽奖游戏 轮盘分为三部分: 一等奖, 二等奖和三等奖; 轮盘转的时候是随机的,

- 如果范围在[0,0.08)之间,代表一等奖,
- 如果范围在[0.08,0.3)之间,代表2等奖,
- 如果范围在[0, 1.0)之间,代表3等奖,

模拟本次活动1000人参加, 模拟游戏时需要准备各等级奖品的个数.

让我们来谈谈对象吧

在python中，一切皆对象。
python 是一门面向对象的语言。面向对象最重要的概念就是类（Class）和实例（Instance），我们可以把类当做模板，而实例就是根据模板创建出来的一个一个对象。我们那汽车来举例，类相当于我们建造汽车的图纸(模板)，而实际生活中，我开的车就是一个对象实例，你开的车也是一个对象实例。

面对对象除了类 和 实例，还有两个比较重要的概念：

- 属性： 表示这类东西具有的特征
- 方法： 表示这类东西可以做的事情。 还是拿上面的汽车来记录，汽车具有的属性有 品牌，车龄等等。汽车具有的方法有 加速，刹车等等。

如何定义类

在pythno 中，类是通过`class` 关键子来定义的：

```
class Car(object):
    pass
```

`class` 紧接类名`Car`，`(object)` 表示这个类继承`object`类，我们后续会讲到继承，如果有明确的继类，`()`中就明确指出。如果没有，则用`object`。

那在类中是如何定义属性和方法的呢，示例如下：

```
class Car(object):

    def __init__(self,brand,age):
        self.brand = brand
        self.age = age

    def broke(self):
        print(self.brand,'在刹车')

    def add(self):
        print(self.brand,'在加速')
```

- `__init__()`：是初始函数(与其他语言中的构造函数类似)，它在创建一个对象实例时默认被调用，不需要手懂调用。所以，我们用它来初始化一些属性。
- `self`：我们看到无论是 `__init__()`,还是 `broke()` 和 `add()`,它的第一个参数都是 `self`，它表示创建实例的本身，在 `__init__()` 函数内， `self.brand = brand` 就把属性绑定到了所创建的这个实例上。表示这个实例的有属性 `brand` 并且他的值就是外部传过来的 `brand`.
  而 `broke()` 中的 `self` 也表示对象实例本身。可以理解为这个实例的的broke方法。暂时不理解也没关系，先吧他认识是约定俗成的吧。在调用的时候 不用传该参数。

如何实例化类

python 中类的实例化也很简单，只要 实例名 = 类名() 就可以了，我们还是那上面的 `Car` 类来举例，因为在 `Car` 类中 `__init__()` 方法中初始化了属性(有除了self之外的参数)，随意我们就必须传入以之向匹配的参数。示例如下：

```
car1 = Car('BMW',2)
print(car1.brand)
brint(car1.age)
car1.broke

car2= Car('BENZ',3)
print(car1.brand)
brint(car1.age)
car1.broke


-------
BMW
2
BMW在刹车

BENZ
3
BENZ在刹车
```

面向对象编程三大特性之继承

> 面向对象编程三大特性有封装，继承，多态，封装 和多态过于抽象，对于初学者比较难以理解，所以暂时先不讲。

继承是一种创建类的方法，新建的类可以继承一个或多个类；
新建的类成为子类或者派生类，子类会遗传父类的属性和方法，被继承的类称为父类和超类。
如何用继承。我们还是拿上面的例子来说明，我们现在要穿个一个 mini_car 的类，他拥有Car 类所有的属性和方法，另外它还能有其他的方法或属性。或者重构父类的方法(有不同的实现方式)。

```
# 不想向类中添加任何其他的属性或者方法，可以使用关键字pass
def Mini_Car(Car):
    pass

# 如果有新的属性 和 新的方法，对原有方法有不同的实现方式
def Mini_Car(Car,brand,age,weight):
    Car.__init__(self,brand,age)
    self.weight = weight

    # 新方法
    def print_weight(self):
        print('i am %dkg' % self.weight)

    #重构原方法
    def brake(self):
        print('mini_car is brake')
```

我们来实例化一个对象

```
car3 = Mini_Car('cooper','3',1500)
car3.brake()
car3.print_weight()
car3.add()

--------
mini_car is brake
i am 1500kg
cooper 在加速
```

我们看到，我们在 `Mini_Car` 中并没有定义 `add()` 方法，但仍能调用。
要查看一个类是否是另一个类的子类 ， 可以用 内置的 issubclass 函数

```
>>>issubclass(Mini_Car,Car)
True

# 查看父类
>>> Mini_Car.__bases__
(<class 'Car'>,)
```

多个超类，多重继承，但劲量少用

```
>>> class A:
    def aa(self):
        print "this is a"


>>> class B:
    def bb(self):
        print "this is b"


>>> class c(A,B):
    pass

>>> cc = c()


>>> cc.aa()
this is a
>>> cc.bb()
this is b

>>> c.__bases__
(<class 'A'>, <class 'B'>)
```

练习

1，尝试定义个Person类，具有属性 姓名，年龄，方法有唱歌，跳舞

2. 设计一个表示动物的类：Animal，其中内部有一个color（颜色）属性和call（叫）方法。再设计一个Fish（鱼）类，该类中有tail（尾巴）和color属性，以及一个call（叫）方法。

遇到错误了咋办 -- 异常处理

异常处理是程序运行中遇到问题时的处理机制！

如何定义异常情况

python 用异常对象（exception object）来表示异常情况，遇到错误后，会引发异常，如果异常对象未被捕捉或处理，程序就会用所谓的回溯 （Traceback）执行终止 示例如下:

```
>>> def div():
...     1/0
...     print(111)
...
>>> div()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in div
ZeroDivisionError: division by zero
```

因为除数不能为0，所以程序就会终止执行，所以下面的`print(111)`就不会执行。

异常也能主动出发的哦

我们可以通过`raise`引发异常，程序会自动创建异常实例

```
>>> raise Exception

Traceback (most recent call last):
  File "<pyshell#100>", line 1, in <module>
    raise Exception
Exception
>>> raise Exception("hahahahahaha")

Traceback (most recent call last):
  File "<pyshell#101>", line 1, in <module>
    raise Exception("hahahahahaha")
Exception: hahahahahaha
```

一些重要的异常类

类　名	描　述
Exception	所有异常的基类
AttributeError	特性引用或赋值失败时引发
IOError	试图打开不存在文件（包括其他情况）时引发
IndexError	在使用序列中不存在的索引时引发
KeyError	在使用映射中不存在的键时引发
NameError	在找不到名字（变量）时引发
SyntaxError	在代码为错误形式时引发
TypeError	在内建操作或者函数应用于错误类型的对象时引发
ValueError	在内建操作或者函数应用于正确类型的对象，但是该对象使用不合适的值时引发
ZeroDivisionError	在除法或者模除操作的第二个参数为0时引发

让我们捕捉异常

1. 单个 `except` python 用 `try---except---` 来捕捉异常，示例如下： 就拿我们一开始的例子举例

```
>>> def div():
...     try:
...         1/0
...     except ZeroDivisionError:
...         print('222')
...
>>> div()
222
```

和上面没有加 `try---exception--` 比，系统没有抛出错误日志，上面这个例子用通俗的话来讲就是 尝试着执行 1/0这操作，如果系统遇到错误并且是ZeroDivisionError，那么就执行 `print('222')` 。
但如果我们我们写的 不是 `ZeroDivisionError` 而是其他错误类型，那么还会执行 `print('222')` 语句吗？我们来试下：

```
>>> def div():
...     try:
...         1/0
...     except IOError:
...         print('222')
...
>>> div()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in div
ZeroDivisionError: division by zero
```

系统抛出异常 `ZeroDivisionError`,而我们希望捕捉的是 `IOError` ,所以不会执行 `print('222')` 。

2. 多个 `except` 如果预期会出现多种错误，那么可以用多个 `except` 去捕捉,示例如下:

```
>>> def div():
...     try:
...         x = int(input("the first Num: "))
...         y = int(input("the second Num: "))
...         print (x/y)
...     except ZeroDivisionError:
...         print("the second number cat't be zero")
...     except ValueError:
...         print("That wasn't a number ,was it")
...
>>> div()
the first Num: 1
the second Num: 0
the second number cat't be zero
>>> div()
the first Num: 1
the second Num: a
That wasn't a number ,was it
```

输入两个数相除，出现的异常也许是除数为0，又或者为输入的不为数字，所以这边写了两个 `except` ,如果还会预期出现其他的错误，可以以此类推，在多增加 `except` .

3. 用一个`except`捕捉多个异常 上面如果嫌多个`except`麻烦，可以用一个`except`来代替，示例如下：

```
>>> def div():
...     try:
...         x = int(input("the first Num: "))
...         y = int(input("the second Num: "))
...         print(x/y)
...     except (ZeroDivisionError,ValueError):
...         print("the numbers was bug")
...
>>> div()
the first Num: 1
the second Num: 0
the numbers was bug
>>> div()
the first Num: 1
the second Num: a
the numbers was bug
```

`except (ZeroDivisionError,ValueError)`起到的效果是一样的。

4. 捕捉异常对象 如果我们想获取异常对象本身，可以用以下方法，示例如下：

```
>>> def div():
...     try:
...         x = int(input("the first Num: "))
...         y = int(input("the second Num: "))
...         print (x/y)
...     except (ZeroDivisionError,TypeError) as e:
...         print(type(e))
...
>>> div()
the first Num: 1
the second Num: 0
<class 'ZeroDivisionError'>
```

异常对象e包括了异常的一些基本信息，这个大家有兴趣的话可以看看他有哪些属性和方法。

5. 全捕捉 如果我们不能确定程序会抛出什么异常，那么我们可以用全捕捉，这样一个异常都不放过，示例如下：

```
def div():
    try:
        x = int(input("the first Num: "))
        y = int(input("the second Num: "))
        print(x/y)
    except :
        print("the num is error")
```

在`except`后不跟如何异常类型就是全捕捉

6. `else`在没有捕捉到异常情况下触发。 如果希望在程序一切正常，没有触发任何异常的情况下做些处理，可以用else，示例如下：

```
>>> def div():
...     try:
...         x = int(input("the first Num: "))
...         y = int(input("the second Num: "))
...         print (x/y)
...     except :
...         print("hahaha")
...     else:
...         print("it's ok")
...
>>> div()
the first Num: 2
the second Num: 1
2.0
it's ok
```

7. `finally`与`try`联合使用 不管异常有没有触发，都会执行`finally`语句块的内容。

```
>>> def div():
...     try:
...         x = int(input("the first Num: "))
...         y = int(input("the second Num: "))
...         print (x/y)
...     except :
...         print("hahaha")
...     else:
...         print("it's ok")
...     finally:
...             print('it is finally')
...
>>> div()
the first Num: 2
the second Num: 1
2.0
it's ok
it is finally

>>> div()
the first Num: 1
the second Num: 0
hahaha
it is finally
```

练习

1. 有一列表 urls = ['http://xxx.com','http://xxx.com','http://xxx.com','http://xxx.com','http://xxx.com',],自己定义一个函数get_response(index),index 为下标索引，为整数，函数调用：输入任意一个整数，返回下表对应的URL 的内容，用try except 分别捕获列表下标越界和url 404 not found 的情况。

> 这个练习要用到 `requests`,发送请求可以用 `requests.get(url)`,响应内容用返回结果的`text`属性

模块

在Python中,一个.py文件就称之为一个模块(Module)

自己也能建立模块并给别人使用

在pyhont 中任何程序都可以作为模块存在。 打开记事本，写入如下代码，保存为 `hello.py` ,并存放在 `D:\python` 这个目录中

```
#hello.py
print('hello Model')
```

在shell中调用

```
>>>import sys    #告诉解释器，除了 在默认的路径中寻找模块，还要在指定的目录中寻找模块
>>> sys.path.append('D://python')

>>> import sys as a  # a作为别名
>>> a.platform
'darwin'

>>> import hello
hello Model

>>> import hello
```

我们从示例看到，当第一次导入 `hello` 模块,系统会调用该模块里面的 `print('hello Model')` 并打印"hello Model",再次调用，并不打印"hello Model"，这是为啥，因为导入模块并不意味着在导入时执行某些操作（如打印）。他们主要用于定义，比如变量，函数等，因此，因为只需要定义这些东西一次，导入多次模块和导入一次的效果是一样的。
上面的例子中，我们改变了sys.path，他包含了一个目录列表，解释器在该列表中查找模块，如果你不想这么做，那么有2种方法：

1. 将模块放在正确的位置 列出 python解释器从哪里查找模块

```
>>> import sys,pprint
>>> pprint.pprint(sys.path)
['',
 'C:\\Python27\\Lib\\idlelib',
 'C:\\Python27\\lib\\site-packages\\setuptools-21.0.0-py2.7.egg',
 'C:\\Python27\\lib\\site-packages\\pip-8.1.1-py2.7.egg',
 'C:\\Windows\\system32\\python27.zip',
 'C:\\Python27\\DLLs',
 'C:\\Python27\\lib',
 'C:\\Python27\\lib\\plat-win',
 'C:\\Python27\\lib\\lib-tk',
 'C:\\Python27',
 'C:\\Python27\\lib\\site-packages',
 'c"/Python27']
```

- 个人建议，site-packages 目录是最佳选择！！！
- 如果数据结构过大，不能在一行打印完，可以商用pprint 中的pprint 函数代替print语句。

2. 告诉编译器去哪里寻找 把模块所在目录加入到 PYTHONPATH 环境变量中。

导入模块还有其他的方式

1. `from 模块名 import 对象名[as 别名]` 使用这种方式仅导入明确指定的对象，并且可以为导入的对象确定一个别名。这种导入方式可以减少查询次数，提高访问的速度，同时也可以减少程序员需要输入的代码量，不需要使用模块名作为前缀。

```
>>> from math import sin
>>> sin(3)
0.1411200080598672

>>> from math import sin as a
>>> a(3)
0.1411200080598672
```

2. `from 模块名 import *` 这种导入方式可以一次导入模块中通过__all__变量指定的所有对象。

> **all** 变量，当一个模块中你不想把所有的属性和方法暴露给调用方，那么可以通过 **all** 变量来指定想要暴露给调用方的属性 和 方法。

很重要的 __name__ 变量

当一个模块建立时，会自动加载一些内建变量，`__name__` 就是其中之一。它的作用就是可以显示一个模块的功能是被自己执行的还是被别的文件调用的。一个模块的功能被自己执行？最常见的就是一个模块中写了一个功功能，在模块中测试这个功能(调动这个功能)。示例如下

```
# Hello.py

def show_message():
    if __name__ == "__main__":
        print('是自己模块调用')
        print('%s' % __name__)
    else:
        print('是别的模块调用')
        print('%s' % __name__)


if __name__ =="__main__":
    show_message()
```

运行`Hello`模块文件，Python 解释器就会把`__name__`属性置为`__main__`,所以结果如下：

```
是自己模块调用
__mian__
```

那么，如果是被别的模块调用呢？ 会打印出什么信息呢？这个作为练习大家运行下。

多个模块 --> 包

在模块之上的概念，为了方便管理而将文件进行打包。包目录下第一个文件便是 **init**.py，然后是一些模块文件和子目录。 常见的包的结构：

```
package x
├─── __init__.py
├─── module_a.py
└─── module_b.py
```

库

具有相关功能模块(包)的集合，库是python的特色之一，python即具有功能强大的标准库(下载安装的python里那些自带的)、第三方库(由其他的第三方机构，发布的具有特定功能的模块)。

> 模块，包，库 实际都都是模块，只不过是个体和集体的区别。

介绍一些常用的标准库

1. sys sys 这个模块可让你能够访问与Pyhont解释器联系紧密的变量和函数.
   常见属性： 名称 | 描述 ---|--- argv | 命令行参数，包括脚本名称 exit([arg]) | 退出当前的程序，可选参数为给定的返回值或错误信息 modules | 映射模块名字到载入的字典 path | 查找模块所在的目录的目录名称列表 platform | 类似 win32 的平台标识符

- argv

```
# 新建 sys.py 保存在 C:\Python27
import sys
print sys.argv[0]
print sys.argv[1]

# 在shell 中运行该文件
C:\Python27>python sys.py xx
sys.py
xx
```

在示例中可以看住，第一个参数为运行文件本身，第二个是传入参数，依次类推。

- path 获取指定模块搜索路径的字符串集合，可以将写好的模块放在得到的某个路径下，就可以在程序中import时正确找到。

示例之前演示过。

- modules sys.modules是一个全局字典，该字典是python启动后就加载在内存中。每当程序员导入新的模块，sys.modules将自动记录该模块。当第二次再导入该模块时，python会直接到字典中查找，从而加快了程序运行的速度.

```
>>> print(sys.modules.keys())
['copy_reg', 'sre_compile', 'locale', '_sre', 'functools',
'encodings', 'site', '__builtin__', 'sysconfig', 'operator',
'__main__', 'types', 'encodings.encodings', 'encodings.gbk', 'abc',
'_weakrefset', 'encodings._codecs_cn', 'errno', 'encodings.codecs',
'sre_constants', 're', '_abcoll', 'ntpath', '_codecs',
'encodings._multibytecodec', 'nt', '_warnings', 'genericpath',
'stat', 'zipimport', 'encodings.__builtin__', 'warnings',
'UserDict', '_multibytecodec', 'sys', 'codecs', 'os.path',
'_functools', '_codecs_cn', '_locale', 'signal', 'traceback',
'linecache', 'encodings.aliases', 'exceptions', 'sre_parse', 'os',
'_weakref']
>>>
```

2. os os 库提供通用的、基本的操作系统交互功能。

名称	描述
getcwd()	获得当前工作目录，即当前 Python 脚本工作的目录路径。
mkdir(dirname)	在当前路径下，建立一个子文件夹。注意：文件已存在时会报错。
rmdir(dirname)	删除一个文件夹，不存在时会报错
listdir()	列出某目录下所有的目录和文件
rename()	修改路径下文件的名字
remove()	删除文件
chdir()	修改当前目录
os.path.abspath(path)	返回path在当前系统中的绝对路
os.path.join(path,*paths)	组合path和paths，返回一个字符串
os.path.exists(path)	判断path对应文件或目录是否存在，返回布尔类型
os.path.isfile(path)	判断path所对应的是否是已存在的文件，返回布尔类型
os.path.isdir(path)	判断path所对应的是否是已存在的目录，返回布尔类型

这里只演示os模块的部分方法，大家私下还要多熟悉os模块

```
>>> os.getcwd()
'/Users/xxxx'

>>> os.mkdir('aa')
>>> os.mkdir('aa')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileExistsError: [Errno 17] File exists: 'aa'

>>> os.rmdir('aa')
>>> os.rmdir('aa')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'aa'


>>> os.chdir('/Users/xx/Downloads')
>>> os.getcwd()
'/Users/xx/Downloads'


>>> os.listdir()
['day06', '.DS_Store', '91混血哥.mp4', '.localized', '测试开发体
系.xmind', 'markdown 图片', 'WP_20141021_12_57_18_Pro.jpg', 'day04',
'day03', '51reboot运维开发.txt', '归档.zip', 'day02', 'day05',
'Bear131[InApp].dmg']
>>> os.remove('91混血哥.mp4')
>>> os.listdir()
['day06', '.DS_Store', '.localized', '测试开发体系.xmind', 'markdown
图片', 'WP_20141021_12_57_18_Pro.jpg', 'day04', 'day03', '51reboot运
维开发.txt', '归档.zip', 'day02', 'day05',  'Bear131[InApp].dmg']
```

3. time 获得当前时间，操作时间和日期，从字符串读取时间以及格式化时间为字符串。

方法名	描述
asctime([tuple])	将时间元祖转化成字符串
localtime([secs])	将秒数转化成日期元祖，以本地时间为准
mktime([tuple])	将时间元祖转化成本地时间
sleep(secs)	休眠 secs 秒
strptime(string[,format])	将字符串转化成时间元祖
time()	当前时间

```
>>> import time
>>> time.asctime()
'Tue Mar 28 08:08:25 2017'    #获取时间字符串

>>> time.localtime()
time.struct_time(tm_year=2017, tm_mon=3, tm_mday=28, tm_hour=8,
tm_min=12, tm_sec=22, tm_wday=1, tm_yday=87, tm_isdst=0)
>>> a = time.localtime()
>>> time.asctime(a)
'Tue Mar 28 08:12:54 2017'

>>> time.mktime(a)
1490659974.0
>>> b = time.asctime(a)
>>> time.strptime(b)
time.struct_time(tm_year=2017, tm_mon=3, tm_mday=28, tm_hour=8,
tm_min=12, tm_sec=54, tm_wday=1, tm_yday=87, tm_isdst=-1)
>>> time.time()
1490660044.994
>>>
```

4. random 该模块报货返回随机数的函数，可以用于模拟或者用于任何产生随机输出的程序

header 1	header 2
random()	返回 0 < n <=1之间的随机实数
getranbits(n)	以长整型形式返回n个随机为
uniform(a,b)	返回随机实数n，其中 a<=n < b
randrange([start],stop,[step])	返回range(start,stop,setp) 中的随机数
choice(seq)	从序列seq 中返回任意元素
shuffle(seq[,random])	原地指定序列seq
sample(seq,n)	从序列seq 中选择N个随机且独立的元素

```
>>> import random as r
>>> r.sample([1,2,3,4,5,6],3)   # 从给定的徐磊中选择给定数目的元素，且确保元素互不相同
[3, 4, 2]
>>> r.choice([1,2,3]) #从给定的薛烈中返回任意元素
>>> r.choice([1,2,3])
>>> r.choice([1,2,3])
```

练习

1. 模块导入的几种方式?
2. `Hello.py` 模块被别的模块调用 `show_message` 方法 会打印出什么?

Open the File

Python 打开一个文件用`open()`方法，语法如下：
`open(name,[module[,buffering]])`

- `name` 为要打开文件的路径
- `module` 位打开模式 它和缓冲(`buffering`)参数都是非必填的。简单示例如下：

```
>>> f = open(r'C:\Python27\README.txt')
>>> print(type(f))
<class '_io.TextIOWrapper'>
>>> f
<_io.TextIOWrapper name='C:\\Python27\\README.txt' mode='r'
encoding='cp936'>


#如果文件不存在
>>>f = open(r'C:\TEXT\somefile.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory:
'C:\\TEXT\\somefile.txt't'
```

module 常用参数如下：

r	读模式
w	写模式
a	追加模式
b	二进制模式(可添加到其他模式中使用)
+	读/写模式(可添加到其他模式中使用)

open 函数的第3个参数(可选)控制着文件的缓冲：

- 0： 无缓冲，所有读写操作直接针对硬盘
- 1： 有缓冲，用内存替代硬盘、

file 的基本操作方法

1. 读和写 ： 文件(或流)最重要的方法就是提供或者接受数据。

```
# 写入
>>> f = open(r'D:\a.txt','w')
>>> f.write('hello')
>>> f.write('world')
>>> f.close()

#读
>>> f = open(r'D:\a.txt','r')  # 'r' 可以省略
# 带参数n 是按字节处理
>>> f.read(4)
'hell'
>>> f.read()
'oworld'
```

2. 行的读写

- `file.readline([n])` 从当前位置开始，直到一个换行符的出现（或者读取最多 N个字符 字符）
- `file.readlines()` 读取一个文件的所有行
- `writelines()` 给他一个字符串列表(可迭代的对象)，他会把所有字符串写入文件，果需要换行则要自己加入每行的换行符

```
# 准备文档 a.txt,里面内容如下:
hello world
this is the first line
this is the secod line

# 读整行
>>> f = open(r'D:\a.txt')
>>> f.readline()
'hello world\n'
>>> f.close()

# 带n 每行按字节进行处理
>>> f = open(r'D:\a.txt')
>>> f.readline(5)
'hello'
>>> f.close()

# readlines  将所有行组装成list
>>> f.readlines()
['hello world\n', 'this is the first line\n', 'this is the secod
line']
>>> f.close()

#writelines
>>> f.close()
>>> s = ['\nhello world\n', 'this is the first line\n', 'this is
the secod line']
>>> f = open(r'D:\a.txt','a')
>>> f.writelines(s)
>>> f.close()

查看 a.txt
hello world
this is the first line
this is the secod line
hello world
this is the first line
this is the secod line
```

3. 关闭文件 `close()` 关闭文件 使用 `open()` 方法一定要保证关闭文件对象，即调用 `close()` 方法。
   可以使用 `try---finally` 中使用，也可以使用 `with oepn(sdfsadfasdf) as f:` 语法示例如下：

```
try:
    f = open():
        xxx
except:
        xxx
finally:
        f.close()


等同于:
    with open(xxxx) as f:
        xxxxx
```

4. 对文件内容进行迭代(循环处理)

- 按字节进行处理

```
char = 'a'
f = open(filename)
while char:
    char = f.read(1)
    if not char:
        break
    process(char)  # 处理字符串函数
f.close()
```

* 按行进行处理

```
f = open(filename)
line = True
while line:
    line= f.readline()
    # 这里注意 bool 类型的使用
    if not line:
        break
    process(line)  # 处理行函数
f.close()
```

- 读取所有内容
  - 如果文件不是太大，可以使用 `read()` 和 `readlines()`
  - 如果文件很大，则使用fileinput 实现

```
import fileinput
for line in fileinput.input(filename):
    process(line)  #对行进行处理
```

在python近几个版本中，文件对象是可迭代(循环的)的，这就意味着可以直接在for循环中使用它们

```
f = open(filename)
for line in f:
    process(line)  # 处理函数
f.close()
```

练习

1. 有文件a.txt(内容不为空)，从第4个字节开始读取文件，并把文件名改为b.tx
2. 有文件 a.txt(内容不为空，且有多行)
    - 去掉文件前面的空格
    - 找出出现次数最多的字符

> 需要大家自己熟悉os 模块的方法