新梦想
NEW DREAM

第一章 python基础

新梦想
NEW DREAM

本章学习目标

- Python简介
- Python版本介绍
- Python安装与配置
- Python基础

新梦想
NEW DREAM

内容进度

- Python简介
- Python版本介绍
- Python安装与配置
- Python基础

Python起源

Python的作者，Guido von Rossum，确实是荷兰人。1982年，Guido从阿姆斯特丹大学(University of Amsterdam)获得了数学和计算机硕士学位。然而，尽管他算得上是一位数学家，但他更加享受计算机带来的乐趣。用他的话说，尽管拥有数学和计算机双料资质，他总趋向于做计算机相关的工作，并热衷于做任何和编程相关的活儿。

Python简介

- Python是一种简单易学，功能强大的编程语言，它有高效率的高层数据结构，简单而有效地实现面向对象编程。
- Python简洁的语法和对动态输入的支持，再加上解释性语言的本质，使得它在大多数平台上的许多领域都是一个理想的脚本语言，特别适用于快速的应用程序开发。
- Python的语法简洁，功能强大，具有丰富和强大的类库，非常适合初学者上手。
- Python不像java一样对内存要求非常高，适合做一些经常性的任务方面的编程。

Python起源

- Guido希望有一种语言，这种语言能够像C语言那样，能够全面调用计算机的功能接口，又可以像shell那样，可以轻松的编程。
- 1989年，为了打发圣诞节假期，Guido开始写Python语言的编译/解释器。Python来自Guido所挚爱的电视剧Monty Python's Flying Circus (BBC1960-1970年代播放的室内情景幽默剧，以当时的英国生活为素材)。他希望这个新的叫做Python的语言，能实现他的理念(一种C和shell之间，功能全面，易学易用，可拓展的语言)。Guido作为一个语言设计爱好者，已经有过设计语言的(不很成功)的尝试。
- 1991年，第一个Python编译器(同时也是解释器)诞生。它是用C语言实现的，并能够调用C库(.so文件)。从一出生，Python已经具有了：类(class)，函数(function)，异常处理(exception)，包括表(list)和词典(dictionary)在内的核心数据类型，以及模块(module)为基础的拓展系统。

新梦想
NEW DREAM

内容进度

- Python简介
- Python版本介绍
- Python安装与配置
- Python基础

新梦想
NEW DREAM

Python版本介绍

Python 2.x版本

Python 2.5是不兼容3.0功能的最后一个版本

Python 2.6是一个过渡版本，使用了Python 2.x的语法和库，同时允许使用部分Python 3.0的语法与函数

Python 2.7是2.x的最后一个版本，除了支持Python 2.x语法外，还支持部分Python 3.1语法

Python 3.x版本

Python的3.0版本，常被称为Python 3000，或简称Py3k，在设计的时候没有考虑向下兼容

Python应用


Google
YAHOO!
intel
cisco
IBM
PIXAR
NASA
Seagate
YouTube
3D
BitTorrent
hp
QUALCOMM
INDUSTRIAL
LIGHT & MAGIC

Python应用


Google
实现Web爬虫和搜索引擎中的很多组件
YAHOO!
Yahoo使用它（包括其他技术）管理讨论组
NASA
NASA在它的几个系统中既用了Python开发，又将其作为脚本语言
YouTube
视频分享服务大部分是由Python 编写的

新梦想
NEW DREAM

内容进度

- Python简介
- Python版本介绍
- Python安装与配置
- Python基础

Python下载与安装

Python下载：
Python最新源码，二进制代码，新闻资讯等可以在Python的官网查看到：
Python官网：https://www.python.org/

Python安装：
1、打开web浏览器在官网下载Python安装包；
2、下载2.7最新版本包后，然后双击安装，安装的时候注意勾选安装环境变量，然后全部点击"下一步"完成。

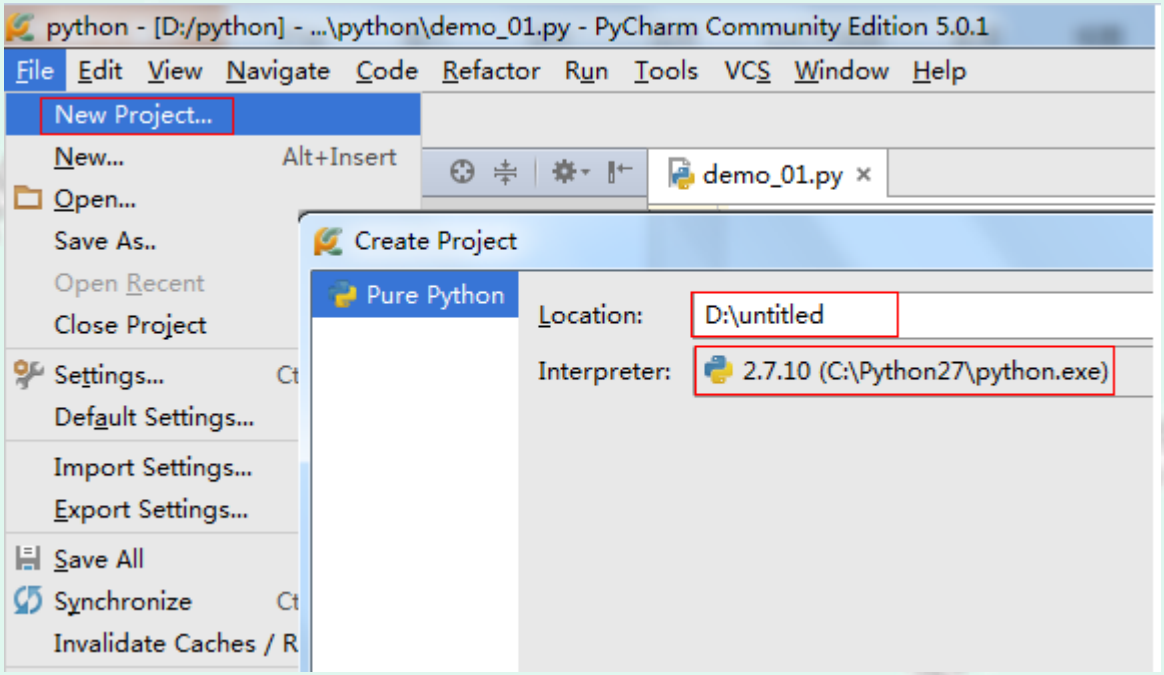
Pycharm安装：
PyCharm是一种Python IDE，带有一整套可以帮助用户在使用Python语言开发时提高其效率的工具，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。
Pycharm官网 http://www.jetbrains.com/pycharm/download
Pycharm 分为Professional版本和Community版本，Professional版本为收费版本，提供试用30天

新梦想
NEW DREAM

Python下载与安装

Pycharm使用简介：

第一步：点击File--New Project... 新建项目

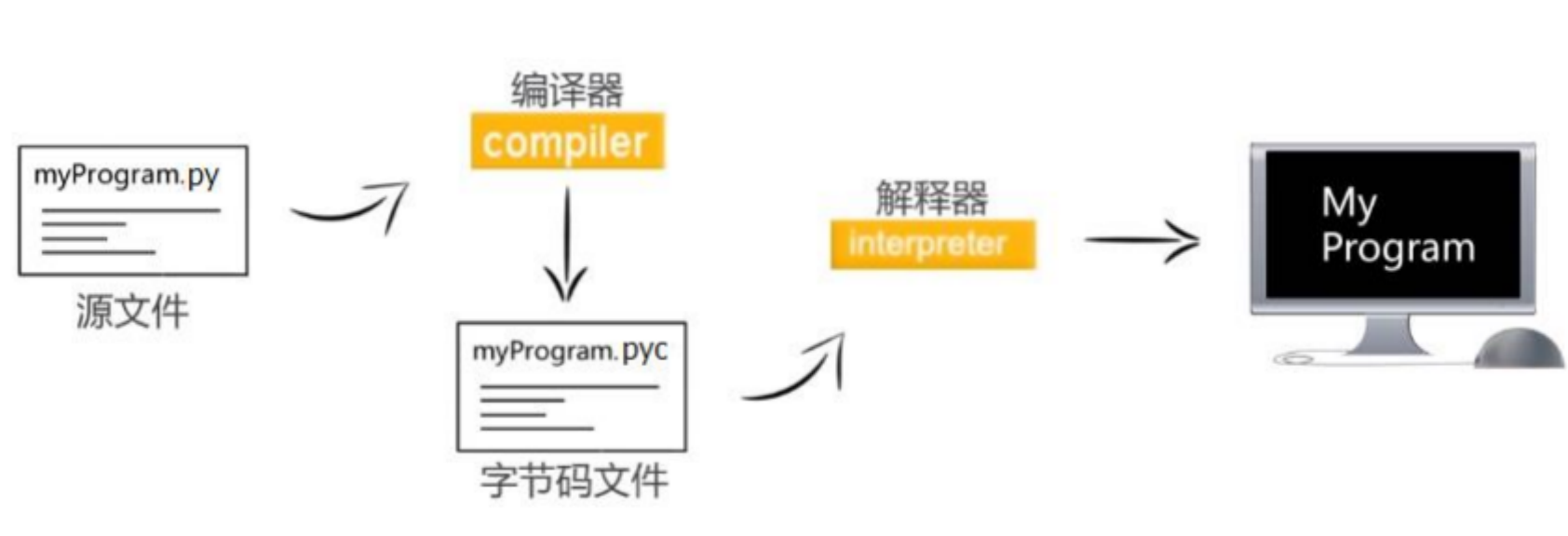


第二步：右击新建的项目--new--File--输入一个.py后缀的文件名即可

新梦想
NEW DREAM

Python文件类型

PVM(python virtual machine)
Python解释器执行Python代码时候，经历如下几个阶段：
1) 加载代码文件
2)翻译成AST（语法分析所获得的中间结果）
3)生成bytecode
4)在PVM（python virtual machine)上执行bytecode，PVM实际是一个基于栈的虚拟机。



Python文件类型

1、源代码文件：
Python 源代码文件以".py"为扩展名，由python语言解释，不需要编译。

2、字节代码文件：
Python源文件经编译后生成的扩展名为".pyc"的文件名，依然由python加载执行，不过速度会提高，也会隐藏源码
有两种编译方法：
方法一：python shell输入

```
import py_compile
py_compile.compile('demo_1.py')
```

方法二：python -m py_compile demo_1.py

3、优化代码：
优化编译后的程序，也是二进制文件，扩展名为".pyo"，适用于嵌入式系统。
编译方法：

```
python -O -m py_compile demo_1.py
```

新梦想
NEW DREAM

内容进度

- Python简介
- Python版本介绍
- Python安装与配置
- Python基础语法

Python基础语法

代码缩进：

学习Python与其它语言最大的区别就是，Python的代码块不使用大括号｛｝来控制类，函数以及其它逻辑判断

Python最具特色的就是用缩进来写模块。

空白在Python中是重要的。事实上行首的空白是重要的。它称为缩进。在逻辑行首的空白（空格和制表符）用来决定逻辑行的缩进层次，从而用来决定语句的分组。

这意味着同一层次的语句必须有相同的缩进。每一组这样的语句称为一个块。

备注：缩进必须严格执行，否则代码报错

多行语句：

Python中一般以新行作为语句的结束符。

但是我们可以使用斜杠（\）将一部分语句分为多行显示。

如果语句中包含[]、{}或()就不需要使用多行连接符

Python基础语法

举例：

```
total = testa +\
        testb
days=['Sunday','Monday','Tuesday',
        'wednesday']
```

引号：

Python 使用单引号（'）、双引号（"）、三引号（''' """）来表示字符串。其中三引号可以由多行组成，它是编写多行文本的快捷语法，常用于文档字符串，在文件的特定地点，被当作注释。

```
word = 'hello'
word = " hello,world! "
word =""" Every new day begins
with possibilities. """
```

Python基础语法

注释：

Python中单行注释采用 # 开头，Python中没有块注释，所以现在推荐的多行注释也用#

在PyCharm中，选中要注释的行，按住Ctrl + / 可以完成快捷注释

标识符：

标识符是用户编程时使用的名字，如：变量、常量、函数、语句块等的名字都叫做标识符。在Python里，标识符由字母、数字、下划线组成，但不能以数字开头。

举例:test_01 、 _test01 都是正确的 01_test错误的。

Python中的标识符是区分大小写的。

Python中以下划线开头的标识符有特殊意义。

以单下划线开头（_foo）的代表不能直接访问的类属性，需通过类提供的接口进行访问，不能用“ from *** import * ”而导入；

以双下划线开头的（__foo）代表类的私有成员；

以双下划线开头和结尾的（__foo__）代表Python里特殊方法专用的标识，如__init__代表类的构造函数。

Python基础语法

Python空行：

函数之间或类的方法之间用空行分隔，表示一段新的代码的开始。类和函数入口之间也用一行空行分隔，以突出函数入口的开始。

空行与代码缩进不同，空行并不是Python语法的一部分。书写时不插入空行，Python解释器运行也不会出错。但是空行的作用在于分隔两段不同功能或含义的代码，便于日后代码的维护或重构。

记住：空行也是程序代码的一部分。

同一行显示多条语句：

Python可以在同一行中使用多条语句，语句之间使用分号(;)分割，以下是一个简单的实例：

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

新梦想
NEW DREAM

Python基础语法

代码组：

多个语句构成代码组，缩进相同的一组语句构成一个代码块。比如if、while、def和class这样的复合语句，首行以关键字开始，以冒号（:）结束，该行之后的一行或者多行代码构成代码组。我们将首行及后面的代码组称为一个子句（clause）。

举例：

```
if  expression :
        suite
elif expression :
        suite
else :
        suite
```

新梦想 NEW DREAM

Python基础语法

用户输入：

raw_input()与input()均是python的内建函数，通过读取控制台的输入与用户实现交互。区别如下：

1、raw_input() 将所有输入作为字符串看待，返回字符串类型；
input() 在对待纯数字输入时具有自己的特性，它返回所输入的数字的类型（ int, float ）；

2、raw_input() 直接读取控制台的输入，没有类型的限制；input()输入内容必须是一个合法的python表达式，如果输入字符串，必须使用引号把它引起来，否则它会引起SyntaxError ；

3、input() 本质上还是使用 raw_input() 来实现的，只是调用完 raw_input() 之后再调用 eval() 函数，所以，你甚至可以将表达式作为 input() 的参数，并且它会计算表达式的值并返回它。

举例：

```
raw_input_A = raw_input("raw_input: ")
input_A = input("Input: ")  （报错）
type (input_A )
```

新梦想
NEW DREAM

第二章 Python变量、数据类型、运算符

湖南软测信息技术有限公司

新梦想
NEW DREAM

本章学习目标

- Python变量
- Python数据类型
- Python运算符

新梦想
NEW DREAM

本章学习目标

- Python变量
- Python数据类型
- Python运算符

新梦想
NEW DREAM

Python变量

Python变量：
变量是计算机内存中的一块区域，存储规定范围内的值，值可以改变，通俗的说变量就是给数据起个名字。

Python变量命名：
1、以数字、字母、下划线开头；
2、数字不能开头；
3、不可以使用关键字；

Python变量赋值：
Python中的变量不需要声明，变量的赋值操作既是变量声明和定义的过程。
每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。
等号（=）运算符左边是一个变量名,等号（=）运算符右边是存储在变量中的值。
举例：

```
name = "John"
```

Python变量

多个变量赋值：

Python允许你同时为多个变量赋值。例如：

```
a = b = c = 1
```

以上实例，创建一个整型对象，值为1，三个变量被分配到相同的内存空间上。

您也可以为多个对象指定多个变量。例如：

```
d, e, f = 2, 3, "john"
```

以上实例，两个整型对象2和3的分配给变量d和e，字符串对象"john"分配给变量f。

```
print a,b,c,d,e,f
1 1 1 2 3 "john"
```

Python变量分类：

1、不可变变量：数字、元组()、字符串（值变的时候会指向一个新地址）

2、可变变量：列表[]、字典｛｝（值变、id不变）

新梦想
NEW DREAM

本章学习目标

- Python变量
- Python数据类型
- Python运算符

新梦想
NEW DREAM

Python数据类型

Python数字类型：

数字数据类型用于存储数值。他们是不可改变的数据类型，这意味着改变数字数据类型会分配一个新的对象。
当你指定一个值时，Number对象就会被创建：num=15

Python支持四种不同的数值类型：
int（有符号整型）
long（长整型[也可以代表八进制和十六进制]）
float（浮点型）
complex（复数）
备注：
长整型也可以使用小写"L"，但是还是建议您使用大写"L"，避免与数字"1"混淆。Python使用"L"来显示长整型。

Python还支持复数，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

新梦想
NEW DREAM

Python数据类型

数字类型举例：

一些数值类型的实例：

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Python数据类型

Python字符串（String）类型：

字符串或串(String)是由数字、字母、下划线组成的一串字符。
一般记为 :
s=″a1a2・・・an″(n>=0)

它是编程语言中表示文本的数据类型。
python的字符串列表有2种取值顺序:
从左到右索引默认0开始的，最大范围是字符串长度少1
从右到左索引默认-1开始的，最大范围是字符串开头
如果你想要取得一段子串的话，可以用到变量[头下标:尾下标]，就可以截取相应的字符串，其中下标是从0开始算起，可以是正数或负数，下标可以为空表示取到头或尾。
比如: s = 'ilovepython'　　s[1:5]的结果是love。

加号（+）是字符串连接运算符，星号（*）是重复操作。

Python数据类型

字符串举例：

```
str = 'Hello World!'
print str # 输出完整字符串
print str[0] # 输出字符串中的第一个字符
print str[2:5] # 输出字符串中第三个至第五个之间的字符串
print str[2:] # 输出从第三个字符开始的字符串
print str * 2 # 输出字符串两次
print str + "TEST" # 输出连接的字符串
```

Python支持格式化字符串的输出。
最基本的用法是讲一个值插入到一个有字符串格式化%s的字符串中。
举例：

```
print ("His name is %s"%("Aviad"))
print ("He is %d years old"%(25))
print ("His height is %f m"%(1.83))
```

新梦想 NEW DREAM

Python数据类型

Python列表(list)：

List（列表） 是 Python 中使用最频繁的数据类型。
列表可以完成大多数集合类的数据结构实现。它支持字符，数字，字符串甚至可以包含列表（所谓嵌套）。
列表用[ ]标识，是python最通用的复合数据类型。
列表中值的分割也可以用到变量[头下标:尾下标]，就可以截取相应的列表，从左到右索引默认0开始的，从右到左索引默认-1开始，下标可以为空表示取到头或尾。
加号（+）是列表连接运算符，星号（*）是重复操作。

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
print list # 输出完整列表
print list[0] # 输出列表的第一个元素
print list[1:3] # 输出第二个至第三个的元素
print list[2:] # 输出从第三个开始至列表末尾的所有元素
print tinylist * 2 # 输出列表两次
print list + tinylist # 打印组合的列表
```

Python数据类型

列表常用操作方法：

```
L.append(var)     #追加元素
L.insert(index,var) #在指定位置插入元素
L.pop(var)        #返回最后一个元素，并从list中删除之
L.remove(var)     #删除第一次出现的该元素
L.count(var)      #该元素在列表中出现的个数
L.index(var)      #该元素的位置,无则抛异常
L.extend(list)    #追加list，即合并list到L上
L.sort()          #排序
L.reverse()       #反转
```

list的复制

```
L1 = L        #L1为L的别名，用C来说就是指针地址相同，对L1操作即对L操作。函数参数就是这样传递的
L1 = L[:]     #L1为L的克隆，即另一个拷贝。
```

Python数据类型

Python元组（Tuple）：
元组是另一个数据类型，类似于List（列表）。
元组用"()"标识。内部元素用逗号隔开。但是元素不能二次赋值，相当于只读列表。

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tuple[2] = 1000 # 元组中是非法应用
list[2] = 1000 # 列表中是合法应用

print tuple * 2 # 输出元组两次
```

新梦想
NEW DREAM

Python数据类型

Python字典（Dictionary）：

字典(dictionary)是除列表以外python之中最灵活的内置数据结构类型。列表是有序的对象结合，字典是无序的对象集合。

两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。

字典用"{ }"标识。字典由索引(key)和它对应的值value组成。

字典也被称作关联数组或哈希表。

字典两个注意事项：

1、字典中的键必须独一无二，但值则不必。创建时如果同一个键被赋值两次，后一个值会被记住；

2、键必须不可变，可以用数字，字符串或元组充当，但是不可以用列表

Python数据类型

创建字典：
简单地说字典就是用大括号包裹的键值对的集合。（键值对也被称作项）
一般形式：

```
adict = {}
adict = {key1：value1， key2：value2， …}
```

使用工厂方法dict()创建字典：

```
adict = dict() 或 adict = dict((['x',1],['y',2]))
```

关键字参数创建字典，如：adict= dict(name='allen',age='40')

使用内建方法 fromkeys(S [, v]) ，New dict with key from S and value equal to v ，即将S里的元素作为键，v作为所有键的值，即字典中的元素具有相同的值， v 的默认值为 None。

```
L1 = [1,2,3]  d.fromkeys(L1)
dict.fromkeys(L1,'over')  {1: 'over', 2: 'over', 3: 'over'}
b={}.fromkeys(('x','y'),1)
```

Python数据类型

更新字典操作：
adict[new_key] = value 形式添加一个项
adict[old_key] = new_value 更新一个数据项（元素）或键值对
del adict[key] 删除键key的项 / del adict 删除整个字典

字典常用方法：
1、adict.keys() 返回一个包含字典所有KEY的列表；
2、adict.values() 返回一个包含字典所有value的列表；
3、adict.clear() 删除字典中的所有项或元素；
4、adict.get(key, default = None) 返回字典中key对应的值，若key不存在字典中，则返回default的值（default默认为None）；
5、adict.pop(key[,default]) 和get方法相似。如果字典中存在key，删除并返回key对应的vuale；如果key不存在，且没有给出default的值，则引发keyerror异常；
6、adict.update(bdict) 将字典bdict的键值对添加到字典adict中，无则添加，有则覆盖

Python数据类型

遍历字典的方法：

遍历字典的key（键）

```
for key in adict.keys():
        print key
```

遍历字典的value（值）

```
for value in adict.values():
        print value
```

遍历字典的key-value

```
for item, value in adict.items():
        print  ‘key=%s, value=%s’ %(item, value)
```

新梦想 NEW DREAM

Python数据类型

Python 数据类型转换：

当我们需要对数据内置的类型进行转换，数据类型的转换，你只需要将数据类型作为函数名即可。

常用的数据类型转换：

int(x [,base])	将x转换为一个整数
long(x [,base])	将x转换为一个长整数
float(x)	将x转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效Python表达式,并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为Unicode字符
ord(x)	将一个字符转换为它的整数值
dict(d)	创建一个字典。d 必须是一个序列(key,value)元组。

Python数据类型

Python 序列操作：

序列包含：字符串、列表、元组

序列的两个特征是“索引”和“切片”，索引，根据Index获取特定元素，切片，获取序列片段。

常见序列操作：

len()：求序列的长度　　+：连接两个序列

*：重复序列元素　　in：判断元素是否在序列中

max()：返回序列最大值　　min()：返回序列最小值

cmp()：比较两个序列　　([ ] | [:])：切片操作符

举例：

```
names = ('Faye', 'Leanna', 'Daylen', 'Tom', 'Kime', 'Tim')
print(names[4])    print(names[-2])  print(names[:-2])
print(names[-4:-2])
```

list['23.34%','30.88%','15.99%'] 假设想取出数据部分比较大小，可以如下操作：

float(list[0][:-1]) 返回结果23.34

新梦想
NEW DREAM

本章学习目标

- Python变量
- Python数据类型
- Python运算符

新梦想 NEW DREAM

Python运算符、表达式

Python 运算符：

算术运算

运算符	说明	举例
+	加法	1 + 2
-	减法	4 - 3.4
*	乘法	7 * 1.5
/	除法	3.5 / 7
%	取余	7 % 2
**	幂	3**3
//	整除	20//3

运算符	说明	举例
==	等于	1==1
!=	不等	1!=0
<>	不等	1<>0
>	大于	1>0
<	小于	1<0
>=	大等	1>=0
<=	小等	1<=0
and	与	1and 2
or	或	1 or 0

关系运算符

运算符	说明	举例
&	与	1 & 4
|	或	2 | 5
^	异或	2 ^ 3
~	翻转	~5
<<	左移	5 << 3
>>	右移	6 >> 1

身份运算符：
is、is not
判断是否引用同一对象
成员运算符：
in、not in
指定序列中能否找到值

新梦想
NEW DREAM

Python运算符优先级

运算符	描述
**	指数 (最高优先级)
~ + -	按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@)
* / % //	乘，除，取模和取整除
+ -	加法减法
>> <<	右移，左移运算符
&	位 'AND'
^ |	位运算符
<= < > >=	比较运算符
<> == !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符
in not in	成员运算符
not or and	逻辑运算符

新梦想
NEW DREAM

第三章 Python流程控制

湖南软测信息技术有限公司

新梦想
NEW DREAM

本章学习目标

- 条件语句
- 循环语句

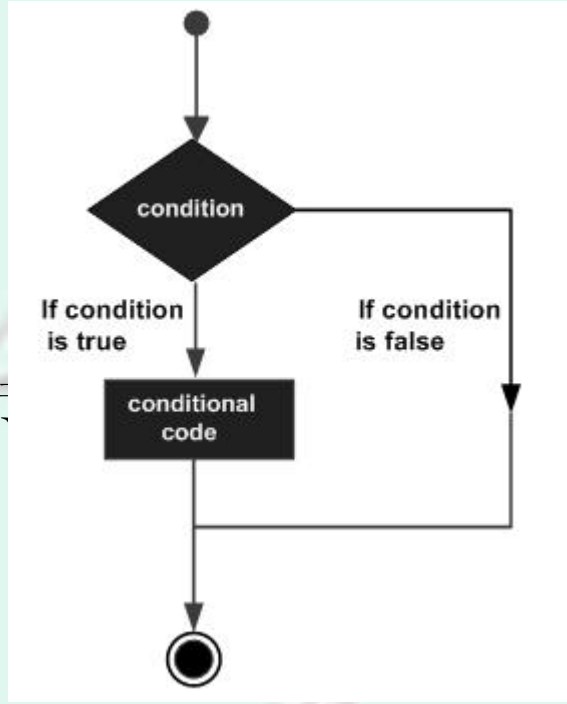
新梦想
NEW DREAM

本章学习目标

- 条件语句
- 循环语句

Python条件语句

Python条件语句是通过一条或多条语句的执行结果（True或者False）来决定执行的代码块。可以通过下图来简单了解条件语句的执行过程:



Python程序语言指定任何非0和非空（none）值为true，0 或者 none为false。

Python 编程中 if 语句用于控制程序的执行，基本形式为:

```
if 判断条件:
    执行语句……
else:
    执行语句……
```

其中"判断条件"成立时（非零），则执行后面的语句，而执行内容可以多行，以缩进来区分表示同一范围。

else 为可选语句，当需要在条件不成立时执行内容则可以执行相关语句

Python条件语句

if 语句的判断条件可以用>（大于）、<(小于)、==（等于）、>=（大于等于）、<=（小于等于）来表示其关系。
当判断条件为多个值时，可以使用以下形式：

```
if 判断条件1:
    执行语句1……
elif 判断条件2:
    执行语句2……
elif 判断条件3:
    执行语句3……
else:
    执行语句4……
```

由于 python 并不支持 switch 语句，所以多个条件判断，只能用 elif 来实现，如果判断需要多个条件需同时判断时，可以使用 or （或），表示两个条件有一个成立时判断条件成功；使用 and （与）时，表示只有两个条件同时成立的情况下，判断条件才成功。

新梦想
NEW DREAM

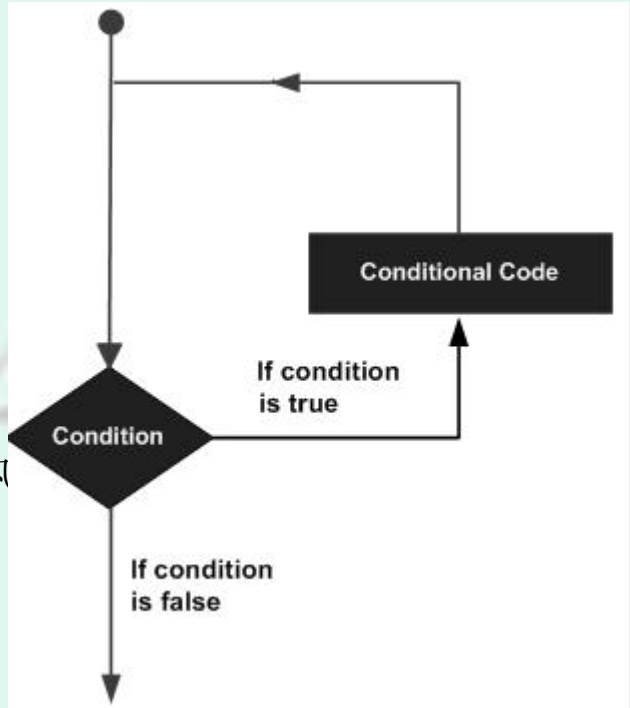
本章学习目标

- 条件语句
- 循环语句

Python循环语句

程序在一般情况下是按顺序执行的。编程语言提供了各种控制结构，允许更复杂的执行路径。

循环语句允许我们执行一个语句或语句组多次，右图是在大多数编程语言中的循环语句的一般形式。



Python提供了for循环和while循环（在Python中没有do..while循环）：

while 循环：给定的判断条件为 true 时执行循环体，否则退出循环体。

for 循环：重复执行语句

循环控制语句：

循环控制语句可以更改语句执行的顺序。Python支持以下循环控制语句：

break 语句：在语句块执行过程中终止循环，并且跳出整个循环

continue 语句：在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。

pass 语句：pass是空语句，是为了保持程序结构的完整性。

Python循环语句

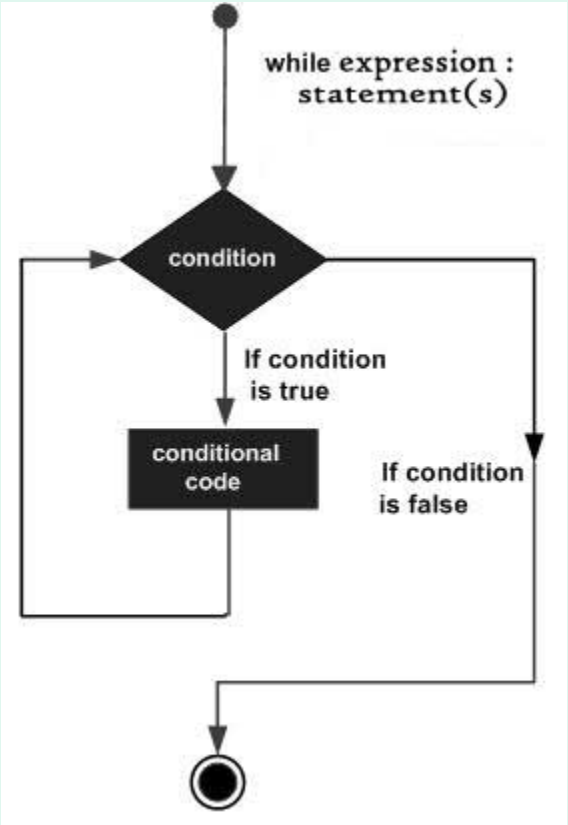
while循环：

```
while 判断条件：
    执行语句……
```

执行语句可以是单个语句或语句块。判断条件可以是任何表达式，任何非零、或非空（null）的值均为true。当判断条件假false时，循环结束。

举例：

```
i=10
count = 0
while i>0:
    count=count+i
    i=i-1
print count
```



while 语句时还有另外两个重要的命令 continue，break 来跳过循环，continue 用于跳过该次循环，break 则是用于退出循环，此外"判断条件"还可以是个常值，表示循环必定成立

Python循环语句

for循环：

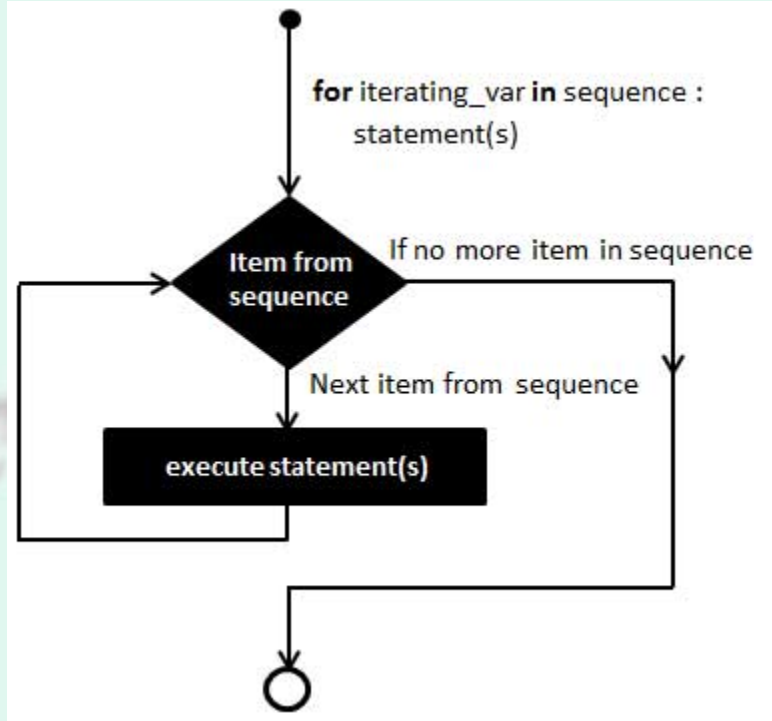
Python For循环可以遍历任何序列的项目，如一个列表、元组或者一个字符串。

```
for iterating_var in sequence:
        statements(s)
```

举例：

```
for letter in 'Python':
   print '当前字母 :', letter

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:
   print '当前字母 :', fruit
```



通过序列索引迭代：

```
fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
   print '当前水果 :', fruits[index]
```

Python循环语句

循环使用 else 语句

在 python 中，for … else 表示这样的意思，for 中的语句和普通的没有区别，else 中的语句会在循环正常执行完（即 for 不是通过 break 跳出而中断的）的情况下执行，while … else 也是一样。

举例：求质数

```
for num in range(10,20):  # 迭代 10 到 20 之间的数字
   for i in range(2,num): # 根据因子迭代
      if num%i == 0:      # 确定第一个因子
         j=num/i          # 计算第二个因子
         print '%d 等于 %d * %d' % (num,i,j)
         break            # 跳出当前循环
   else:                  # 循环的 else 部分
      print num, '是一个质数'
```

Python循环语句

Python 循环嵌套：
Python 语言允许在一个循环体里面嵌入另一个循环。

Python for 循环嵌套语法：

```
for iterating_var in sequence:
   for iterating_var in sequence:
      statements(s)
   statements(s)
```

Python while 循环嵌套语法：

```
while expression:
   while expression:
      statement(s)
   statement(s)
```

也可以在循环体内嵌入其他的循环体，如在while循环中可以嵌入for循环，反之，你可以在for循环中嵌入while循环。

Python循环语句

用for循环实现冒泡排序（升序）：

```
array = [3,2,1]
for i in range(len(array) - 1, 0, -1):
    print i
    for j in range(0, i):
        print j
        if array[j] > array[j + 1]:
            array[j], array[j + 1] = array[j + 1], array[j]
print array
```

第二种实现：

```
arr=[3,8,7,4,3,0,21,33,45,67]
for i in range(1,len(arr)):
            for j in range(0,len(arr)-i):
                if arr[j]>arr[j+1]:
                                arr[j],arr[j+1]=arr[j+1],arr[j]
print arr
```

新梦想
NEW DREAM

第四章 Python函数

新梦想
NEW DREAM

本章学习目标

- Python函数

Python函数

函数概念：

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。函数能提高应用的模块性和代码的重复利用率，可以被用户调用。

定义函数语法：

```
def function(params):
    block
return expression/value
```

（1）在Python中采用def关键字进行函数的定义，不用指定返回值的类型。

（2）函数参数params可以是零个、一个或者多个，同样的，函数参数也不用指定参数类型，因为在Python中变量都是弱类型的，Python会自动根据值来维护其类型。

（3）return语句是可选的，它可以在函数体内任何地方出现，表示函数调用执行到此结束；如果没有return语句，会自动返回NONE，如果有return语句，但是return后面没有接表达式或者值的话也是返回NONE。

Python函数

调用函数语法:

函数名（参数表）

函数举例:

```
def printHello():
    print 'hello'

def printNum():
    for i in range(0,10):
        print i
    return

print printHello()
print printNum()  print add(1,2)
```

Python函数

Python函数参数：

形参全称是形式参数，在用def关键字定义函数时函数名后面括号里的变量称作为形式参数。实参全称为实际参数，在调用函数时提供的值或者变量称作为实际参数。举例：

```
def add(a,b):      #a和b为形参
    return a+b
c=add(2,3)         #2和3为实参
print c
```

所有参数（自变量）在Python里都是按引用传递。如果你在函数里修改了参数，那么在调用这个函数的函数里，原始的参数也被改变了。

在Python中一切皆对象，变量中存放的是对象的引用。在Python中包括我们之前经常用到的字符串常量，整型常量都是对象。

Python函数

举例：

```
print id(5)
print id('python')
x=2
print id(x)
y='hello'
print id(y)

def changeme( mylist ):
   "修改传入的列表"
   mylist.append([1,2,3,4]);
   print "函数内取值: ", mylist
   return
mylist = [10,20,30];     changeme( mylist );     print "函数外取值: ", mylist
```

Python函数

Python函数参数类型：

必备参数、关键字参数、缺省参数、任意个数参数

必备参数须以正确的顺序传入函数，也叫做位置参数，即参数是通过位置进行匹配的，从左到右，依次进行匹配，这个对参数的位置和个数都有严格的要求。举例：

```
def printme( str ):
   print str;
   return;
printme();  #函数调用
```

关键字参数和函数调用关系紧密，调用方用参数的命名确定传入的参数值。你可以跳过不传的参数或者乱序传参，因为Python解释器能够用参数名匹配参数值。

```
def printme( str ):
   print str;
   return;                    printme( str = "My string")  #函数调用
```

新梦想 NEW DREAM

Python函数

缺省参数：调用函数时，缺省参数的值如果没有传入，则被认为是默认值。

```
def printinfo( name, age = 35 ):
   print "Name: ", name
   print "Age ", age
   return
printinfo( age=50, name="miki" )     printinfo( name="miki" )
```

使用默认参数有一点要非常注意，在重复调用函数时默认形参会继承之前一次调用结束之后该形参的值

```
def insert(a,L=[]):
    L.append(a)
    print L
insert('hello')
insert('world')
```

Python函数

收集参数（不定长参数）：当需要一个函数能处理比当初声明时更多的参数。这些参数叫做收集参数。

```
def printinfo( arg1, *vartuple ):
  print "输出: "
  print arg1
  for var in vartuple:
    print var
  return
printinfo( 10 )
printinfo( 70, 60, 50 )
```

'*'和'**'表示能够接受0到任意多个参数，'*'表示将没有匹配的值都放在同一个元组中，'**'表示将没有匹配的值都放在一个dictionary中。

Python中函数是可以返回多个值的，如果返回多个值，会将多个值放在一个元组或者其他类型的集合中来返回。

Python函数

```
def fun_var_args(farg, *args):
    print 'args:', farg
    for value in args:
        print 'another arg:',value
# *args可以当作可容纳多个变量组成的list或tuple
fun_var_args(1, 'two', 3, None)

def fun_var_kwargs(farg, **kwargs):
    print 'args:',farg
    for key in kwargs:
        print 'another keyword arg:%s:%s' % (key, kwargs[key])
# myarg1,myarg2和myarg3被视为key， 感觉**kwargs可以当作容纳多个key和value的dictionary
fun_var_kwargs(1, myarg1='two', myarg2=3, myarg3=None)
```

Python函数

匿名函数

用lambda关键词能创建小型匿名函数。这种函数得名于省略了用def声明函数的标准步骤。

Lambda函数能接收任何数量的参数但只能返回一个表达式的值，同时不能包含命令或多个表达式。

匿名函数不能直接调用print，因为lambda需要一个表达式。

lambda函数拥有自己的名字空间，且不能访问自有参数列表之外或全局名字空间里的参数。

虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率

语法

lambda函数的语法只包含一个语句，如下：

lambda [arg1 [,arg2,.....argn]]:expression

举例：

```
sum = lambda arg1, arg2: arg1 + arg2;
print "Value of total : ", sum( 10, 20 )
```

新梦想
NEW DREAM

Python函数

Python局部变量和全局变量

局部变量在函数内部定义，拥有一个局部作用域，局部变量只能在其被声明的函数内部访问。

全局变量在函数外部定义，拥有全局作用域，全局变量可以在整个程序范围内访问。函数内部可以通过global强制定义全局变量，当函数调用时，全局变量生效。

在Python中查找名字的规则是LGB规则：先局部(Local)，次之全局(Global)，再次之内置(Build-in)

举例：

```
name="Jims"
def set1():
    global name
    name="ringkee"
set1()
print name
```

新梦想
NEW DREAM

第五章 Python面向对象编程

新梦想
NEW DREAM

本章学习目标

- Python面向对象编程
- Python模块
- Python包

Python面向对象编程

Python是解释性语言，但是它是面向对象的，能够进行对象编程。

类(Class): 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。

类是对现实世界中一些事物的封装，定义一个类可以采用下面的方式来定义：

```
class className:
    block
```

在block块里面就可以定义属性和方法了。当一个类定义完之后，就产生了一个类对象。类对象支持两种操作：引用和实例化。引用操作是通过类对象去调用类中的属性或者方法，而实例化是产生出一个类对象的实例，称作实例对象。

Python面向对象编程

举例：

```
class people:
    name = 'jack'      #定义了一个属性
    def printName(self): #定义了一个方法
        print self.name
print people.name  #类对象people
p=people()        #实例化对象p
p.printName()
```

该段代码定义了一个people类，里面定义了name属性，定义了类之后，p = people( )实例化了一个对象p，然后就可以通过p来读取属性和使用方法了。这里的name和age都是公有的，可以直接在类外通过对象名访问，如果想定义成私有的，则需在前面加2个下划线 '__'。

Python面向对象编程

Python类中的方法：

在类中可以根据需要定义一些方法，定义方法采用def关键字，在类中定义的方法至少会有一个参数，一般以名为'self'的变量作为该参数（用其他名称也可以），而且需要作为第一个参数。

```
class people:
    __name = 'jack'  #私有变量
    __age = 12       #私有变量
    def getName(self):
        return self.__name
    def getAge(self):
        return self.__age
p = people()
print p.getName(),p.getAge()
```

self指的是类实例对象本身(注意：不是类本身)。

在上述例子中，self指向Person的实例p，可以把它当做C++中类里面的this指针一样理解，就是对象自身的意思

Python面向对象编程

Python类中内置的方法

在Python中有一些内置的方法，这些方法命名都有比较特殊的地方（其方法名以2个下划线开始然后以2个下划线结束）。

1、__init__(self,....)：构造方法，在生成对象时调用，可以用来进行一些初始化操作，不需要显示去调用，系统会默认去执行。构造方法支持重载，如果用户自己没有重新定义构造方法，系统就自动执行默认的构造方法。

2、__del__(self)：析构方法，在释放对象时调用，支持重载，可以在里面进行一些释放资源的操作，不需要显示调用。

3、__dict__：类的属性（包含一个字典，由类的数据属性组成）

4、__doc__:类的文档字符串

5、__name__: 类名

6、__module__: 类定义所在的模块（类的全名是'__main__.className'，如果类位于一个导入模块mymod中，那么className.__module__ 等于 mymod）

7、__bases__：类的所有父类构成元素（包含了以个由所有父类组成的元组）

Python面向对象编程

类方法：

是类对象所拥有的方法，需要用修饰器"@classmethod"来标识其为类方法。它能够通过实例对象和类对象去访问。类方法的用途就是可以对类属性进行修改。对于类方法，第一个参数必须是类对象，一般以"cls"作为第一个参数，举例：

```
class people:
    country = 'china'
    @classmethod
    def getCountry(cls):        #类方法
        return cls.country
    @classmethod
    def setCountry(cls,country): #类方法
        cls.country = country
p = people()
p.setCountry('japan')
```

Python面向对象编程

实例方法：

在类中最常定义的成员方法，它至少有一个参数并且必须以实例对象作为其第一个参数，一般以名为'self'的变量作为第一个参数。（注意：不能通过类对象引用实例方法）

静态方法：需要通过修饰器"@staticmethod"来进行修饰，静态方法不需要多定义参数。

```
class people:
    country = 'china'
    def getCountry(self):    #实例方法
        return self.country
    @staticmethod          #静态方法
    def getsCountry():
        return people.country
p = people()
print p.getCountry()        #正确，可以用过实例对象引用
print people.getCountry()   #错误，不能通过类对象引用实例方法
print people.getsCountry() #静态方法可以通过类对象，实例化对象都能调用
```

Python面向对象编程

Python继承和多继承:

在Python中，可以让一个类去继承一个类，被继承的类称为父类或者超类、也可以称作基类，继承的类称为子类。并且Python支持多继承，能够让一个子类有多个父类。类的继承定义基本形式如下:

```
#父类
class superClassName:
    block
#子类
class subClassName(superClassName):
    block
```

声明子类的时候在括号中写要继承的父类，如果有多个父类，多个父类名之间用逗号隔开。

Python面向对象编程

```
class UniversityMember:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def getName(self):
        return self.name


class Student(UniversityMember):
    def __init__(self,name,age,sno,mark):
        UniversityMember.__init__(self,name,age)    #注意要显示调用父类构造方法
                        self.sno = sno
        self.mark = mark
    def getMark(self):
        return self.mark
```

代码解释：大学中的每个成员都有姓名和年龄，而学生有学号和分数这2个属性，大学成员为父类，学生为子类

Python面向对象编程

1）在Python中，如果父类和子类都重新定义了构造方法__init__()，在进行子类实例化的时候，子类的构造方法不会自动调用父类的构造方法，必须在子类中显示调用。

2）如果需要在子类中调用父类的方法，需要以 父类名.方法 这种方式调用，以这种方式调用的时候，注意要传递self参数过去。

对于继承关系，子类继承了父类所有的公有属性和方法，可以在子类中通过父类名来调用，而对于私有的属性和方法，子类是不进行继承的，因此在子类中是无法通过父类名来访问的。

关于多重继承，比如

class SubClass(SuperClass1,SuperClass2)

如果SubClass没有重新定义构造方法，它会自动调用第一个父类的构造方法，以第一个父类为中心。

如果SubClass重新定义了构造方法，需要显示去调用父类的构造方法，此时调用哪个父类的构造方法由你自己决定；若SubClass没有重新定义构造方法，则只会执行第一个父类的构造方法。并且若SuperClass1和SuperClass2中有同名的方法，通过子类的实例化对象去调用该方法时调用的是第一个父类中的方法。

Python面向对象编程

Python多态：

多态（Polymorphism）按字面的意思就是“多种状态”。在面向对象语言中，接口的多种不同的实现方式即为多态。举例 JAVA中的重载和重写、一个接口对应多个实现、一个父类有多个子类。

Python可以说是一种多态语言。在Python中很多地方都可以体现多态的特性，比如 内置函数len(object)，len函数不仅可以计算字符串的长度，还可以计算列表、元组等对象中的数据个数，这里在运行时通过参数类型确定其具体的计算过程，正是多态的一种体现。举例：

```
class A:
    def prt(self):
        print "A"
class D:
    pass
def test(arg):
    arg.prt()
```

```
class B(A):
    def prt(self):
        print "B"
```

```
class C(A):
    pass
```

- Python面向对象编程
- Python模块
- Python包

Python模块

Python模块：

模块是Pyhon最高级别的程序组织单元，它将程序代码和数据封装起来以便重用。实际的角度，模块往往对应Python程序文件。

每个文件都是一个模块，并且模块导入其他模块之后就可以使用导入模块定义的变量名。模块可以由两个语句和一个重要的内置函数进行处理。

import: 使客户端（导入者）以一个整体获取一个模块。

from:容许客户端从一个模块文件中获取特定的变量名。

reload:在不中止Python程序的情况下，提供了一个重新载入模块文件代码的方法。

搜索路径和路径搜索

搜索路径：查找一组目录

路径搜索：查找某个文件的操作

ImportError: No module named myModule

该错误说明：模块不在搜索路径里，从而导致路径搜索失败。

Python搜索模块的路径：

程序的主目录--》PTYHONPATH目录--》标准连接库目录--》任何的.pth文件的内容.新功能

Python模块

模块导入:

1、import语句

import放在程序前面，且按照Python标准库模块、Python第三方模块、自定义模块的顺序从上到下排开。

导入后，使用 模块名.函数名 引用

2、from-import语句

可以导入模块中指定属性:from module import name1[,nam2[,...nameN]]

注意:

(1)、导入可使用\进行换行

如： from module import nam1,name2,\

name3,name4....

(2)、不推荐from module import *

(3)、使用as更换名称:

from module import name1 as name2

通过这种方式引入的时候，调用函数时只能给出函数名，不能给出模块名，但是当两个模块中含有相同名称函数的时候，后面一次引入会覆盖前一次引入。

Python模块

系统模块调用举例：

```
import sys
import time
time.sleep(3)

from time import sleep
sleep(3)
```

用户自定义模块举例：

新建test.py，在test.py中定义了函数add：

```
def add(a,b):
    return a+b
```

在其他文件中可以先import test，然后通过test.add(a,b)来调用，当然也可以通过from test import add来引入。

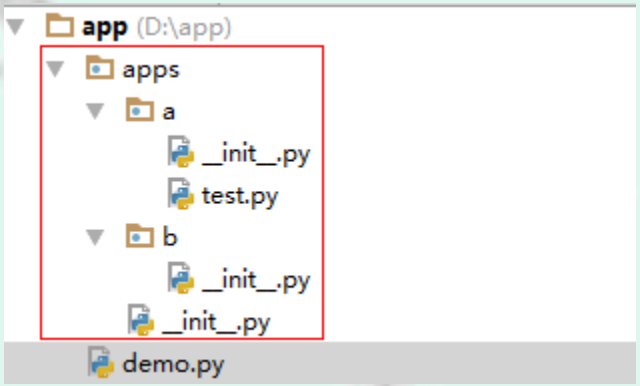
备注：在用import引入模块时，会将引入的模块文件中的代码执行一次。且只在第一次引入时才会执行模块文件中的代码

- Python面向对象编程
- Python模块
- Python包

Python包

包将有联系的模块组织在一起，有效避免模块名称冲突问题，让应用组织结构更加清晰。

一个普通的python应用程序目录结构：



apps是最顶层的包，a和b是它的子包，可以这样导入：

```
from apps.a import test
a=test.adds(3,4)
print a
```

上面代码表示：导入apps包的子包a的test模块时，然后调用adds方法。

每个目录下都有__init__.py文件，这个是初始化模块，from-import语句导入子包时需要它，可以在里面做一些初始化工作，也可以是空文件。

新梦想
NEW DREAM

第六章 Python文件操作、异常处理

新梦想
NEW DREAM

本章学习目标

- Python 文件操作
- Python异常处理

新梦想
NEW DREAM

本章学习目标

- Python 文件操作
- Python异常处理

Python文件操作

Python文件操作：

Python提供了必要的函数和方法进行默认情况下的文件基本操作。可以用file对象做大部分的文件操作。

open函数

你必须先用Python内置的open()函数打开一个文件，创建一个file对象，相关的辅助方法才可以调用它进行读写。

语法：

file object = open(file_name [, access_mode][, buffering])

各个参数的细节如下：

file_name：file_name变量是一个包含了你要访问的文件名称的字符串值。

access_mode：access_mode决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。

buffering:如果buffering的值被设为0，就不会有寄存。如果buffering的值取1，访问文件时会寄存行。如果将buffering的值设为大于1的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。

Python文件操作

不同模式打开文件的完全列表(上):

r 以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。

rb 以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。

r+ 打开一个文件用于读写。文件指针将会放在文件的开头。

rb+ 以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。

w 打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

wb 以二进制格式打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

w+ 打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

wb+ 以二进制格式打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

新梦想
NEW DREAM

Python文件操作

不同模式打开文件的完全列表(下):

a　　打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

ab　　以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

a+　　打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。

ab+　　以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

File对象的属性:

file.closed 返回true如果文件已被关闭，否则返回false。

file.mode　返回被打开文件的访问模式。

file.name　返回文件的名称。

file.encoding 返回文件编码

Python文件操作

Close()方法:

File对象的close（）方法刷新缓冲区里任何还没写入的信息，并关闭该文件，这之后便不能再进行写入。

语法:　fileObject.close()

read()方法:

read（）方法从一个打开的文件中读取一个字符串。需要重点注意的是，Python字符串可以是二进制数据，而不是仅仅是文字。

语法：fileObject.read([count])

在这里，被传递的参数是要从已打开文件中读取的字节计数。该方法从文件的开头开始读入，如果没有传入count，它会尝试尽可能多地读取更多的内容，很可能是直到文件的末尾。

举例:

```
a=open('c:/a.txt','r+')
str=a.read(3)
print str
```

新梦想
NEW DREAM

Python文件操作

Write()方法

Write()方法可将任何字符串写入一个打开的文件。需要重点注意的是，Python字符串可以是二进制数据，而不是仅仅是文字。Write()方法不在字符串的结尾不添加换行符('\n')：

语法：

fileObject.write(string);

在这里，被传递的参数是要写入到已打开文件的内容。

文件位置：

Tell()方法告诉你文件内的当前位置；换句话说，下一次的读写会发生在文件开头这么多字节之后：

seek（offset [,from]）方法改变当前文件的位置。Offset变量表示要移动的字节数。From变量指定开始移动字节的参考位置。

如果from被设为0，这意味着将文件的开头作为移动字节的参考位置。如果设为1，则使用当前的位置作为参考位置。如果它被设为2，那么该文件的末尾将作为参考位置。

Python文件操作

重命名和删除文件

Python的os模块提供了帮你执行文件处理操作的方法，比如重命名和删除文件。

要使用这个模块，你必须先导入它，然后可以调用相关的各种功能。

rename()方法：

rename()方法需要两个参数，当前的文件名和新文件名。

语法：

os.rename(current_file_name, new_file_name)

remove()方法

你可以用remove()方法删除文件，需要提供要删除的文件名作为参数。

语法：

os.remove(file_name)

Python文件操作

其它操作文件方法:

fp.readline([size])　　#读一行，如果定义了size，有可能返回的只是一行的一部分

fp.readlines([size])　#把文件每一行作为一个list的一个成员，并返回这个list。其实它的内部是通过循环调用readline()来实现的。如果提供size参数，size是表示读取内容的总长，也就是说可能只读到文件的一部分。

fp.writelines(seq)　　#把seq的内容全部写到文件中(多行一次性写入)。这个函数也只是真实地写入，不会在每行后面加上任何东西。

fp.flush()　　#把缓冲区的内容写入硬盘

fp.next()　　#返回下一行，并将文件操作标记位移到下一行。把一个file用于for … in file这样的语句时，就是调用next()函数来实现遍历的。

新梦想
NEW DREAM

Python文件操作

目录操作：

mkdir()方法：可以使用os模块的mkdir()方法在当前目录下创建新的目录

语法：os.mkdir("newdir")

chdir()方法：可以用chdir()方法来改变当前的目录。

语法：os.chdir("newdir")

rmdir()方法：删除目录，目录名称以参数传递。在删除这个目录之前，它的所有内容应该先被清除。

语法：os.rmdir('dirname')

makedirs()方法：创建多级目录：

语法：os.makedirs（r“c：\python\test”）

getcwd()方法：得到当前工作目录

语法： os.getcwd()

isdir()/isfile()方法：检验给出的路径是否是一个目录/文件

os.path.isdir('E:\\book\\temp')

新梦想
NEW DREAM

本章学习目标

- Python 文件操作
- Python异常处理

Python异常处理

异常处理:

异常处理，是编程语言或计算机硬件里的一种机制，用于处理软件或信息系统中出现的异常状况（即超出程序正常执行流程的某些特殊条件）如：文件找不到、网络连接失败、非法参数等。异常是一个事件，它发生在程序运行期间，干扰了正常的指令流程。

一般情况下，在Python无法正常处理程序时就会发生一个异常，异常是Python对象，表示一个错误，当Python脚本发生异常时我们需要捕获处理它，否则程序会终止执行。

在Python中，异常也是对象，可对它进行操作。所有异常都是基类Exception的成员。所有异常都从基类Exception继承，而且都在exceptions模块中定义。Python自动将所有异常名称放在内建命名空间中，所以程序不必导入exceptions模块即可使用异常。

捕捉异常可以使用try/except语句。

try/except语句用来检测try语句块中的错误，从而让except语句捕获异常信息并处理。如果你不想在异常发生时结束你的程序，只需在try里捕获它。

Python异常处理

语法:

```
try:
  block
except [exception,[data...]]:
  block
else:
  block
```

该种异常处理语法的规则是:

- 执行try下的语句，如果引发异常，则执行过程会跳到第一个except语句。
- 如果第一个except中定义的异常与引发的异常匹配，则执行该except中的语句。
- 如果引发的异常不匹配第一个except，则会搜索第二个except，允许编写的except数量没有限制。
- 如果所有的except都不匹配，则异常会传递到下一个调用本代码的最高层try代码中。
- 如果没有发生异常，则执行else块代码。

Python异常处理

举例：

```
try:
  f = open(“file.txt”,”r”)
except IOError, e:
  print e
```

解释：捕获到的IOError错误的详细原因会被放置在对象e中,然后运行该异常的except代码块

捕获所有的异常

```
try:
  a=b
  b=c
except Exception,ex:
  print Exception,":",ex
```

使用except子句需要注意的事情，就是多个except子句截获异常时，如果各个异常类之间具有继承关系，则子类应该写在前面，否则父类将会直接截获子类异常。放在后面的子类异常也就不会执行到了。

Python异常处理

try-finally 语句：

无论是否发生异常都将执行最后的代码。

```
try:
    block
finally:
    block   #退出try时总会执行
```

注意：else语句也不能与finally语句同时使用。举例：

```
try:
    fh = open("testfile", "r")
    try:
        fh.write("This is my test file for exception handling!!")
    finally:
        print "Going to close the file"
        fh.close()
except IOError:
    print "Error: can\'t find file or read data"
```

Python异常处理

触发异常：

Python中的raise 关键字用于引发一个异常，语法格式如下：

raise [Exception [, args [, traceback]]]

语句中Exception是异常的类型（例如，NameError）参数是一个异常参数值。该参数是可选的，如果不提供，异常的参数是"None"

举例：

```
def ThorwErr():
  raise Exception("抛出一个异常")
  print 'hello'    #抛出异常后语句不执行
try:
  ThorwErr()
except Exception,x:
  print x
```

备注：当函数中的异常触发后，退出该函数，print语句将不执行

一个异常可以带上参数，可作为输出的异常信息参数，如x，x接收raise抛出的错误信息。

变量接收的异常值通常包含在异常的语句中。在元组的表单中变量可以接收一个或者多个值。元组通常包含错误字符串，错误数字，错误位置。

Python异常处理

用户自定义异常：

通过创建一个新的异常类，程序可以命名它们自己的异常。异常应该是典型的继承自Exception类，通过直接或间接的方式。自定义异常使用raise语句引发，而且只能通过人工方式触发。举例：

```
class DivisionException(Exception):
      def __init__(self, x, y):
            Exception.__init__ (self, x, y)      #调用基类的__init__进行初始化
            self.x = x
            self.y = y
if __name__ == "__main__":
      try:
            x = 3
            y = 2
            if x % y > 0:                                  #如果大于0， 则不能被初始化，抛出异常
                  print x/y
                  raise DivisionException(x, y)
      except DivisionException,div:                        #div 表示DivisionException的实例对象
         print "DivisionExcetion: x/y = %.2f" % (div.x/div.y)
```

Python异常处理

断言：

在没完善一个程序之前，不知道程序在哪里会出错，与其让它在运行时崩溃，不如在出现错误条件时就崩溃，这时候就需要assert断言的帮助。在Python中使用assert做断言，语法格式：

```
assert expression
assert 表达式
```

assert断言是声明其布尔值必须为真的判定，assert语句失败的时候，会引发一个AssertionError。 可以理解assert断言语句为raise-if-not，用来测试一个条件，其返回值为假，就会触发异常。举例：

```
x = 23
assert x > 0, "x is not zero or negative"
assert x%2 == 0, "x is not an even number"
```

with...as语句的用法：

```
with EXPRESSION [ as VARIABLE] WITH-BLOCK
with open("/tmp/foo.txt") as file:
    data = file.read()
```

该语句等价于右边语句。

```
file = open("/tmp/
foo.txt")
try:
    data = file.read()
finally:
    file.close()
```

Python异常处理

附录：Python标准异常一

BaseException	所有异常的基类
SystemExit	解释器请求退出
KeyboardInterrupt	用户中断执行(通常是输入^C)
Exception	常规错误的基类
StopIteration	迭代器没有更多的值
GeneratorExit	生成器(generator)发生异常来通知退出
SystemExit	Python 解释器请求退出
StandardError	所有的内建标准异常的基类
ArithmeticError	所有数值计算错误的基类
FloatingPointError	浮点计算错误
OverflowError	数值运算超出最大限制
ZeroDivisionError	除(或取模)零 (所有数据类型)
AssertionError	断言语句失败
AttributeError	对象没有这个属性
EOFError	没有内建输入,到达EOF 标记
EnvironmentError	操作系统错误的基类

Python异常处理

附录：Python标准异常二

IOError 输入/输出操作失败
OSError 操作系统错误
WindowsError 系统调用失败
ImportError 导入模块/对象失败
KeyboardInterrupt 用户中断执行(通常是输入^C)
LookupError 无效数据查询的基类
IndexError 序列中没有没有此索引(index)
KeyError 映射中没有这个键
MemoryError 内存溢出错误(对于Python 解释器不是致命的)
NameError 未声明/初始化对象 (没有属性)
UnboundLocalError 访问未初始化的本地变量
ReferenceError 弱引用(Weak reference)试图访问已经垃圾回收了的对象
RuntimeError 一般的运行时错误
NotImplementedError 尚未实现的方法
SyntaxError Python 语法错误
IndentationError 缩进错误
TabError Tab 和空格混用
SystemError 一般的解释器系统错误

Python异常处理

附录：Python标准异常三

TypeError 对类型无效的操作
ValueError 传入无效的参数
UnicodeError Unicode 相关的错误
UnicodeDecodeError Unicode 解码时的错误
UnicodeEncodeError Unicode 编码时错误
UnicodeTranslateError Unicode 转换时错误
Warning 警告的基类
DeprecationWarning 关于被弃用的特征的警告
FutureWarning 关于构造将来语义会有改变的警告
OverflowWarning 旧的关于自动提升为长整型(long)的警告
PendingDeprecationWarning 关于特性将会被废弃的警告
RuntimeWarning 可疑的运行时行为(runtime behavior)的警告
SyntaxWarning 可疑的语法的警告
UserWarning 用户代码生成的警告