**# 中谷教育Python基础教程**

### Python概述和特点

Python的吉祥物是，蟒蛇

特点：

解释型的，面向对象做到了极致，简单的，集解释型和编译性与一体

Python是由ABC语言发展而来，ABC主要是为了解决数学、物理的科学计算

Guido发明的。

什么是web3.0时代？

Python的各个版本之间有很大的区别！！！1.0 2.0 3.0之间的差别都是很大的！！！

Java、C、C++、PHP、Python、C#

- Python是高阶动态语言，而不是简单的脚本语言！！！

- 在Python中对象的概念很广，比如，变量和类都是对象

- Python很容易扩展，可嵌入性，可移植性强，可以一直到任何平台上！！！

- Python有丰富的库，需要什么我们直接调用即可，达到快速开发的目的。

- Python其实就是一门胶水语言，可以轻易的粘合其他语言。

Python的合作伙伴Google、Yahoo、NASA美国宇航局、YouTube全球最大的视频分享网站

国内有：豆瓣的前台和后台

### 环境的搭建

问题1：

在win10安装msi安装包的时候常常会出现代码为2502、2503的错误。其实这种错误是由于安装权限不足造成的。

解决：以管理员身份打开命令行，执行这个程序即可。

问题2：

使python2.7和python3.6在windows共存

修改Python 3.6安装目录下的python.exe和pythonw.exe分别为python3.exe和pythonw3.exe。

这样使用命令python我们使用的是2.7版本的Python，而使用Python3则使用的是3.6版本的Python。

安装ipython

简介：IPython 是一个 python 的交互式 shell，比默认的python shell 好用得多，支持变量自动补全，自动缩进，支持 bash shell 命令，内置了许多很有用的功能和函数。

使用pip进行软件包安装的时候，对于Python2仍然使用pip install XXX，而对于Python3则需要使用python3 -m pip install XXX的形式进行安装。

比如我们使用python3 -m pip install ipython安装了ipython，我们还需要进入安装目录的Scripts子目录将ipython.exe删除或者修改为ipython.exe.bak备份文件。这样只剩下ipython3.exe，我们就可以在命令行使用ipython3进入python3的ipython环境了。

调用指定版本下的pip去下载偏python组件

C:\Users\Administrator>python3 -m pip install Django

问题：

安装python2.7的ipython时，提示 error: Microsoft Visual C++ 9.0 is required (Unable to find vcvarsall.bat). Get it from http://aka.ms/vcpython27

解决：于是到指定网站下载vc++，装好后重新运行pip install ipython 即可正常安装！！！

通过quit和exit都可以退出ipython界面！！！加个 () 也是可以的。

#### ipython的简单使用

注意：在win10中安装python一定要以管理员身份运行安装程序才能成功安装！！！

安装时勾选，添加到环境变量！！！

在dos命令行中可以通过 exit() 函数推出python的解释器

#### 在pythonidle中实现清屏：

>import os

>i = os.system('cls')

文件类型:

1.源文件： .py

2.编译后的文件 .pyc

3.优化的代码 .pyo

编译后的代码比源代码效率高

以上三种文件只要有可执行权限都可以直接执行！！！

如果要在Linux下直接运行Python程序，必须加上可执行权限：chmod +x 1.py

然后，通过 ./1.py 来运行。

Python的书写格式为：

这句话的作用是指明python解释器的位置

#!/usr/bin/Python

设置python的字符编码，注意，如果注释有中文也要设置字符编码为utf8，而且这句必须在开头，在import之前

#coding=utf-8

import xxx

打印 以一个字符串，python2.7中也可以使用print "hello world"

print("hello,world!")

注意：python中语句的结束没有分号，换行就表示语句的结束，靠缩进来判断逻辑关系，冒号+缩进 表示一个代码块

在命令行直接： Python 1.py

这种方法是无论有没有可执行权限都会调用Python来解释运行这个Python脚本。

\* Python的字节码文件

要引入py\_compile模块来编译

import py\_compile

调用这个模块的一个方法来编译另外一个py文件

py\_compile.compile('1.py')

\* 优化代码，就是经优化的源代码

生成方法：在命令行直接输入 -0就是optimize：优化的意思

python -o -m py\_compile('1.py')

### Python中的变量

Python中的变量不需要声明类型，他会根据所赋的值自动判断类型，可以通过id(var)查看变量的地址

Python中的变量刚好与C语言中的相反，在Python中变量名只是一个标签，是对数据的一个引用，给一个变量重新赋值就是把这个标签贴到另外一个数据身上（在C语言中是直接把原来的替换掉）

比如：

a = 123

b = 123

此时id(a) == id(b),也就是a、b变量同时指向内存中的123,内存中只有一个 123 ！！！

这就好像一个人有不同的称呼，但无论是哪个称呼都指的是你这个对象！！！

也就是说在python的内存中同样的数据只有一个，如果有其他的变量等于这个数据的值，只是把变量名指向这个数据而已。

再比如：

>>> a = 10

>>> id(a)

1458784032

>>> a=11

>>> id(a)

1458784048

两次id(a)的值不一样，说明a的指向发生了改变，给一个变量重新赋值的过程：

先再内存中生成‘数据对象’，然后将变量指向这个‘数据对象’。

Python的最大特点是：先在内存中存储好数据，然后通过一个标签去引用！！！不是事先开辟空间！！！

### Python中的运算

在python3.6中，整数除法的结果都是浮点数 3/2 = 1.5

在python2.7中，3/2 = 1 、

操作数中只要有一个浮点数，那么结果就为浮点数 3.0/2 = 1.5

> 浮点数：就是小数点是可以变化的数，因为在科学记数法中小数点是可以变的！！！

python中的帮助命令

>>> help

Type help() for interactive help, or help(object) for help about object.

整数除法 // 结果总为整数，小数部分为0

比如：3.0//2=1.0

整数除法可以理解成对结果取整，如果结果是浮点数，就是对浮点结果取整（也就是小数部分为0）

>>> 10//3

3

>>> 10.0//3

3.0

>>> 10/3

3.3333333333333335

求幂运算：2\*\*3=8

逻辑运算符： and or not

a = raw\_input()

接受输入，赋给a

但是这个函数返回的是一个字符串

所以，要 int(raw\_input("please input num:"))

在python3.6中只能使用 input() 函数！！！

### python中的数据类型

可以使用type(123)查看数据的类型

与C最大的不同是变量无需声明类型

比如一个数，如果很大，就是 long

如果较小，就是 int

python2.7 中 num = 1000L这样也是long类型

但是python3 中已经不支持这种语法！！！

浮点型：

float，double

复数类型：

实部+虚部

c = 3.14j

在Python中可以使用单引号、双引号、三引号(可以是单双引号)来定义一个字符串！！！这与C语言不一样！！！

Python的字符串中也可以使用转义字符

格式化的字符串：适合于需要多次使用换行的字符串中使用

>>> str2 = """tom:

... hello,

... nice to meet you

... """

>>> str2

'tom:\nhello,\nnice to meet you\n'

#这里三个单引号也是可以的。

使用print()输出多个字符串，python会将逗号转为空格

>>> print('hello','world','china')

hello world china

可以把字符串看成是单个字符的集合，数组，所以可以通过 str[0] 的方法取单个字符

取出并拼接 str[0] + str[3]

>>> str2

'tom'

>>> str2[0]+str2[2]

'tm'

字符串的切片： str[1:4] 这是一个区间[1,4)

str[:3] 从0取到3之前

str[3:] 从3取到最后

记住：右边是开区间即可！！！

还有第三个参数表示步长： str[::2]

如果第三个参数是负数，表示从后往前步进！！！

负数表示从后面确定下标，但是取的时候还是由前往后取！！！ str[-3:-1] 也就是取出 -3 -2

对于切片数字不是必须的，但是冒号是必须的！！！

str[::] : 表示取出所有的字符

str[:::2] : 表示隔一个取一个

### 序列

元组、列表、字符串 都属于序列

序列的基本操作：

len() 求长度

>>> len(l)

4

+ 拼接两个序列 临时的，在内存中的

>>> l = [1,2,3,4]

>>> ll = [5,6,7]

>>> l+ll

[1, 2, 3, 4, 5, 6, 7]

\* 重复序列几次 比如 abc\*5 就是重复abc五次

in 判断某个元素是否在序列中 返回布尔值 true false

>>> 1 in l

True

>>> 8 in l

False

max()

min(序列) 返回序列的最值

cmp(str1,str2) 比较两个序列

如果str1 > str2 返回 1

如果str1 < str2 返回 -1

相等时，返回 0

### 元组 tuple

特点：就是不可变的列表，用括号定义

尝试改变元组的值：

>>> t = (1,2,3)

>>> t[1]=3

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

定义：zmh = ('zmh',22,'male')

通过 元组名[下标] 取值

元组可以存储任意类型的数据类型

定义一个空元组 t = ()

注意：定义一个值的元组 t = (1,) 这个逗号不能省略！！！如果没有逗号，会理解为这个值的类型，而不是元组

不能修改元组：zmh[0] 不支持的！！！

将元组的值分别赋给多个变量：

zmh = ('zmh',22,'male')

name,age,gender = zmh

此时相当于：name = 'zmh'

age = 22

gender = 'male'

元组的每一项分别与前面的变量相对应

还可以通过 a,b,c = 1,2,3

来初始化一个变量，与上面元组类似

### 列表 very important

> 其实列表就是js中的索引数组

注意：数字，字符串和元组 都是\*\*不可改变的\*\*，如果是字符串的 '修改' 实际上是在内存中重新生成了一个新的对象，然后将这个变量指向新对象，不是本质上的修改

如果需要一个安全、比较固定的值是可以考虑使用元组

列表就是其他语言中的索引数组，是可以修改的！！！

定义：与定义元组类似，只是将 () 改为 []

zmh = ['zmh',22,'male']

取值和切片完全和元组一样

zmh[0]

zmh[0:2]

所谓 ‘列表可变’ 也就是可以可以给列表的单元重新赋值

修改：

zmh[0] = '朱明浩' 还是在原来的地址！！！所以是修改

添加：

zmh.append('Gaozhou')

添加 'Gaozhou' 到列表的最后

删除：

1.zmh.remove('Gaozhou')

如果有多个，则删除第一个出现的，删除的是指定的值

2.使用 \*系统函数\* del()删除：

del(zmh[3]) 删除下标为3的单元

del(zmh) 直接删除整个元组zmh

### 字典

Python中的字典其实就是js中的json对象，以键值对的形式书写

也就是php中的关联数组

zmh = {'name':'朱明浩','age':22,'gender':'male'}

字典是无序的！！！

取值：通过 字典名['key'] 的方式取值

添加/修改：

zmh['tel'] = '1234567'

如果没有这个键就是添加，有这个键就是修改他的值

列表只能通过list.append(xxx)到列表的最后！！！

删除：

del(zmh['tel'])

zmh.pop('tel') 删除/弹出 'tel' ,返回他的值

字典和列表都有这个pop方法，都是根据 键 来弹出！！！

清空整个字典:zmh.clear()

删除整个字典：del(zmh)

>>> d ={'a':1,'b':2,'c':3}

>>> d

{'a': 1, 'c': 3, 'b': 2}

# 添加或修改字典

>>> d['d'] = 4

>>> d

{'a': 1, 'c': 3, 'b': 2, 'd': 4}

# 弹出一个单元，返回这个单元的值

>>> d.pop('a')

1

>>> d

{'c': 3, 'b': 2, 'd': 4}

# 清空整个字典，返回 {}

>>> d.clear()

>>> d

{}

# 说明：取出dict字典中键为 a 的值，如果没有这个键，返回自定义的 'error'

>>> d.get('a','error')

1

>>> d.get('f','error')

'error'

get()方法为字典独有，列表没有

remove()方法为列表独有，字典没有

列表就是索引数组，字典就是关联数组

### 字典中的一些方法

dict.get(3,'error')

说明：取出dict字典中键为 3 的值，如果没有这个键，返回自定义的 'error'

### if语句

Python中同一缩进表示同一级代码

Python中的控制语句，比如if、while、for、def fun():这些语句最后都有一个 :

建议使用四个空格表示缩进，而不是一个tab键

if True:

if False:

True 和 False 的首字母要大写！！！

if语句的条件部分是多样的，可以是一个函数的调用，如果是函数的调用，则用函数的返回值作为判断

if fun():

print('ok')

else:

print('error')

-------华丽分割线-----------

if fun():

print('ok1')

elif express:

print('ok2')

else:

print('error')

#这里的elif没有次数限制

if语句也是可以嵌套的。

if x > 90:

if y > 90:

print('ok1')

elif express:

print('ok11')

elif express:

print('ok2')

else:

print('error')

单分支就是多个if语句并列的形式

if x > 80:

print('x>80')

if x > 90:

print('x>90')

这两个if语句是独立的！！！

也就是只有一个语句会被执行

-------华丽分割线-----------

#逻辑判断

if x>=90 and y>=90:

print('good')

> shift+右键点击文件 将会出现 复制路径

### for循环语句

for x in list:

print(x)

#这个x表示列表的每一项的值,这一点很重要

深入理解for，他的底层肯定是用：

for(i;i<len(list);i++){

return list[i]

}

这种方式实现的！！！

使用range()函数来生成一个范围的列表

range(top) 0-（top-1）

range(start,top)

range(start,top,step)

以上方法同样不包括结束top

遍历以一个列表

list = ['apple','orange','banana']

for index in range(len(list)):

print(list[index])

其实以上方法是多此一举的

for value in list:

print(value)

for循环同样可以用来遍历字典，但是迭代变量返回的是字典的各个 key

所以要通过 dict[key] 的方法来取每一项的值

{'a': 1, 'c': 3, 'b': 2, 'd': 4}

>>> for x in d:

... print(d[x])

...

1

3

2

4

>>> for x in d:

... print(x,'=',d[x])

...

('a', '=', 1)

('c', '=', 3)

('b', '=', 2)

('d', '=', 4)

dict.items() 方法返回一个元组列表，列表的每一项为元组，每个元组就是这个字典的每一个键值对单元！！！

>>> d = {'a':1,'b':2,'c':3,'d':4}

>>> d.items()

[('a', 1), ('c', 3), ('b', 2), ('d', 4)]

元组的拆分： k,v = tuple

所以字典还可以用如下方法遍历：

for key,value in dict.items():

print(key)

print(value)

>>> for key,value in d.items():

... print(key,value)

...

('a', 1)

('c', 3)

('b', 2)

('d', 4)

在for循环和while循环的最后可以加

else：

print('ending')

表示循环正常结束后执行的操作，如果循环异常退出，比如break，continue，将不会执行

break 和 continue 的区别和php中的一样

break是跳出整个循环，continue是跳出此次循环，继续执行下一次循环！

可以在代码中使用 pass 来占位，表示未知代码，只是起到占位的作用。

### while循环

与其他编程语言的while一样，也有 else: 代码块

### 函数

>特别注意：在设置参数的默认值时，一定是 写在参数列表的最有边，这样我们传实参时才不至于有冲突！！！

函数的默认参数：

def fun(x,y,z=3):

#有默认值的参数要放在最右边

fun(1,2) //这样就可以不写第三个参数

python中是可以指定参数传值的，比如

fun(x=1,y=2)

#### 局部变量和全局变量

函数里面的变量只属于这个函数，是一个局部变量，不能在外部调用。

可以在函数内部使用global将一个变量声明为全局变量

global a

这样就可以在函数外面使用这个变量。

需要注意的是:只有当这个函数被调用后，这个全局变量才存在！！！

如果全局有一个变量 a = 1

在函数中 global a = 2

当调用函数，就相当于给全局变量a重新赋值。

#### 函数的返回值

return xxx

注意：return表示函数的结束，也就是在return下面的语句是不会被执行的！！！

如果没有给一个函数指定返回值，则返回值为None！！！

#### 将元祖的元素当作参数传递

t = ('name','zmh')

def fun(x,y):

print("%s:%s" % (x,y))

#格式化字符串输出，与C不同的是，有 % 在中间隔开，如果只传递以一个参数，括号可以省略

fun(\*t)

>>> t = ('name','zmh')

>>> def fun(x,y):

... print(x,y)

...

>>> fun(\*t)

('name', 'zmh')

像上面这种 元组 和 函数的参数列表是对应关系的，可以这样： fun(\*t) 传递参数

#### 将字典传给函数参数列表

> 字典是无序的！！！

dict = {'name':'zmh','age':22}

def f(name,age):

print("name:%s,age:%d" % (name,age))

f(\*\*dict)

#通过两个 \* 来引用

执行结果：

>>> dict = {'name':'zmh','age':22}

>>> def f(name,age):

... print("name:%s,age:%d" % (name,age))

...

>>> f(\*\*dict)

name:zmh,age:22

注意：字典的键必须要与函数的形参一样！！！

为什么直接赋值py脚本到idle总是无法正常执行呢？

答：因为这个解释器是一条一条的执行命令的，以上代码一共有三条语句，所以直接执行是不可以的。

#### 接收多余的参数

1. 元组形式

def f(x,\*args):

print(x)

print(args)

f(1)

f(1,2,3)

2和3存到args元组里

注意：如果函数声明时没有 \*args

传多个参数是会直接报错的！！！

2. 字典形式

def f(x,\*t\_args,\*\*d\_args):

print(x)

print(t\_args) #接收传递过来的元组

print(d\_args) #接收传递过来的字典

f(1,2,3,4,a='zmh',b='cbh')

执行结果：

1

(2, 3, 4)

{'a': 'zmh', 'b': 'cbh'}

#### python中的匿名函数lambda

语法：fun=lambda 参数x,参数y : x\*\*y

调用：fun(3,2)

无需return语句

结果：9

他构造的是一个函数对象。

>>> ff = lambda x,y : x\*\*y #此部分为 返回值

>>> ff(2,2)

4

python中没有switch语句，但是可以通过字典来实现。

思路：

1. 定义加减乘除四个函数

2. 定义一个字典operator + - \* / 分别对应以上四个方法名

3. 定义一个处理逻辑的函数

def operate(x,o,y):

print operator.get(o,lambda \*args:)(x,y,\*args)

4. 调用 operate(3,'+',5)

对于字典的get(k,[d])方法，如果字典没有 k 键，则执行第二个参数部分！！！

#### 内建函数

1. callable(函数名) 判断某个函数是否可以调用，返回 True 或 False

>>> callable(list.pop)

True

>>> callable(dict.pop)

True

2. isinstance(l,list) 判断 l 是否为 list 类型，即是相当于php中的 instanof()

>>> l

[3, 4]

>>> isinstance(l,list)

True

3. cmp(str1,str2) 比较两个字符串，判断长度大小，str1>str2返回 1,否则 -1

4. type(object) 返回object的类型

5. xrange(10)

------------------------------------------

类型转换函数：

int()

>>> str

'123'

>>> int(str)

123

>>> str

'123'

float()

list()

>>> t

('name', 'zmh')

>>> list(t)

['name', 'zmh']

>>> t

('name', 'zmh')

tuple() 元组和字典的相互转换 ，只是在内存中临时转换，不改变原来的！！！

>>> l

[3, 4]

>>> tuple(l)

(3, 4)

>>> l

[3, 4]

str()

>>> a = 123

>>> str(a)

'123'

>>> a

123

由上面可以得出：

1.基本数据类型的对象的方法可以直接使用！！！

比如：元组，列表，字符串

2.所有的类型转换都是在内存中作临时的转换！！！

------------------------------------------

进制转换：

hex() 转十六进制

oct() 转八进制

chr(97) 结果 a

ord('a') 结果 97

#### 字符串处理函数：

1. 首字母大写：str.capitalize()

这是一个类当中的方法

2. 转大写：str.upper()

3. 转小写：str.lower()

4. 替换：str.replace('old','new')

第一个参数是字符串中的需要被替换的子字串，第二个参数是用来替换的字符，第三个参数是替换的次数，默认只替换第一个

5. 切片：str.splict(sep,maxsplit)

返回一个列表，可以指定分隔符，如果没有指定分隔符，将使用空格作为分隔符。

第二个参数是限定切割的次数

>>> str = 'hello world'

>>> str.split()

['hello', 'world']

>>> str1 = 'helloworld'

>>> str1.split()

['helloworld']

>>> str2 = '192.168.1.130'

>>> str2.split('.')

['192', '168', '1', '130']

>>> str2.split('.',2)

['192', '168', '1.130']

以上操作还可用以下方法实现：

import string

str = '192.168.1.1'

string.replace(str,'.','-')

这是类似php的写法，区别就是需要指定字符串对象

可以通过 help(str) 查看所有字符串的属性和方法！！！

#### 序列常用函数

1. 过滤 filter(fun,list)

作用：将序列的每个单元当作参数传递个函数fun，如果函数返回True，则取出这个序列单元，最终返回一个符合条件的序列。

fun 为自定义函数，起到过滤作用。

在python3.6中

>>> def fun(x):

if x>4:

return True

>>> l=[1,3,5,6,7,8]

>>> filter(fun,l)

<filter object at 0x034982F0>

>>> list(filter(fun,l))

[5, 6, 7, 8]

-------------------------

在python2.7中，无需转换

>>> l = [1,2,3,4,5,6]

>>> def fun(x):

... if x % 2 == 0:

... return True

...

>>> filter(fun,l)

[2, 4, 6]

在python3.6中返回的是一个过滤对象。这一点与python2不同，如果要显示序列，可以强转。list()

> 注意：

变量的命名一定不要使用关键字，否则会出现很多意想不到的错误！！！

2. zip(序列1，序列2，序列3)

作用：将每个序列按照竖直方向组成一个元组，返回一个包含多个 元组 的列表

在python3.6中

name = ['zmh','cbh','wys']

age = [20,21,22]

tel = [111,222,333]

zip(name,age,tel)

结果：

>>> zip(name,age,tel)

<zip object at 0x03475F30>

>>> r=zip(name,age,tel)

>>> list(r)

[('zmh', 20, 111), ('cbh', 21, 222), ('wys', 22, 333)]

注意：

1. 返回的结果是zip对象，同样需要强转为列表

2. 使用zip时如果列表的长度不一致，则按照最短的列表进行并行遍历组合

3. 如果是使用map，对于长度不够的列表会自动用None进行填充，所以，所有列表都会被保留

------------------------

在python2.7中

>>> l = [1,2,3,4,5,6]

>>> m = [2,4,6,8,10,12]

>>> n = [11,12,13]

>>> zip(l,m)

[(1, 2), (2, 4), (3, 6), (4, 8), (5, 10), (6, 12)]

>>> zip(l,m,n)

[(1, 2, 11), (2, 4, 12), (3, 6, 13)]

>>> map(None,l,m,n)

[(1, 2, 11), (2, 4, 12), (3, 6, 13), (4, 8, None), (5, 10, None), (6, 12, None)]

3. map(fun,name,age,tel)

如果指定了一个函数，则是将并行遍历的结果作为参数传递到这个函数，经函数处理返回一个列表（往往是一个值的列表，而不是元组的列表）

在python3.6中

>>> a = [1,2,3]

>>> b = [5,6,7]

>>> def test(x,y):

return x\*y

>>> map(test,a,b)

<map object at 0x03498FF0>

#返回的是一个map对象

>>> re = map(test,a,b)

>>> list(re)

[5, 12, 21]

-------------------------

在python2.7中

map(test,a,b)返回的直接是一个处理后的列表

>>> l = [1,2,3,4,5,6]

>>> m = [2,4,6,8,10,12]

>>> def test(x,y):

... return x+y

...

>>> map(test,l,m)

[3, 6, 9, 12, 15, 18]

4. reduce(fun,l)

作用：每次取出列表的两个单元作为参数传个fun，重复进行计算。

python2.7中

>>> l = [1,2,3,4,5,6]

>>> def jiecheng(x,y):

... return x+y

...

>>> reduce(jiecheng,l)

21

python3.6中没有reduce这个函数！！！

### 模块

在python中模块其实就是 .py 文件，只不过是这种模块文件里面定义了很多方法

在需要引用这些方法的文件开头通过import引入这个模块，然后再通过

module.fun()来调用模块里面的方法。

所有的python模块都有 \_\_name\_\_ 属性，这是一个变量，如果是直接调用，结果为 \_\_main\_\_ ,如果是通过其他文件来调用，则显示这个模块名。

通过这个特性，我们可以再模块文件内部作个判断，

if \_\_name\_\_ == '\_\_main\_\_':

retuan xxx

也就是只有直接解释这个模块文件按，才会执行if语句！！！

python中模块调用的查找顺序，当前目录->lib目录

> .py .pyc .pyo 都可以是模块文件，也就是我们自己写的文件一定不要和模块文件名同名，否则引入的是我们自己的，而不是lib目录下我们所需要的！！！

### 包

python中是用 包 来管理相同类型的模块的。

包 其实就是一个文件夹，只不过这个文件夹下面有 \_\_init\_\_.py 这个文件和很多模块文件。

存在即是合理！！！

一切都有原因，都有理由，只是有些是你所不知道的！！！

包的导入：

import pack.module.py

import 模块名 as 别名

直接引入模块中的一个方法：

from module import fun

### 正则表达式

> 案例：做 爬虫

python中正则的使用：

1. 引入 import re

2. 定义规则：pattern = r'abc'

3. 使用re模块的方法处理字符串

re.findall(s,'aaaaaaaaaa')

参数1：正则表达式

参数2：需要匹配的字符串

返回所有匹配成功的字符串，放在一个数组/列表

>>> import re

>>> pattern = r'\d{4}-?\d{8}'

>>> tel = '0668-12345678'

>>> re.findall(pattern,tel)

['0668-12345678']

-------------------------

贪婪模式与非贪婪模式

+ 就是表示贪婪模式 匹配一个或多个，如果有多个就一定是匹配多个

+？就是非贪婪模式，只匹配一个

-----------------------------

#### 正则高级部分

对于要经常使用的正则表达式，可以使用re 模块的一个方法 re\_obj = re.compile(pattern) ,将一个正则表达式pattern编译成一个正则对象，编译可以大大提高效率。

使用这个正则对象：

re\_obj.findall('0668-12345678')

说明：用re\_obj这个对象的规则来匹配后面的 字符串

>>> import re

>>> pattern = r'\d{4}-?\d{8}'

>>> tel = '0668-12345678'

>>> p\_tel = re.compile(pattern)

>>> p\_tel

<\_sre.SRE\_Pattern object at 0x02D9D700>

>>> p\_tel.findall(tel)

['0668-12345678']

>>> p\_tel.findall('0668-6132046')

[]

编译时，re.compile(pattern,re.I) 可以忽略大小写。

---------------------------------

#### 常用的re方法：

1. re\_obj.match('hello zmh')

2. re\_obj.search('hello zmh')

相同点：都是在字符串中查找符合规则的\*\*子串\*\*

这两个方法的区别：

match() 只匹配字符串的开头是否符合规则，如果匹配，返回一个match对象，如果不匹配，返回 空

search() 会对字符串进行扫描，也就是无论在哪里都可以匹配，匹配成功返回search对象。

match：

>>> z\_re = re.compile(r'hello zmh',re.I)

>>> z\_re

<\_sre.SRE\_Pattern object at 0x032B4DA0>

#匹配成功

>>> z\_re.match('hello zmh nice')

<\_sre.SRE\_Match object at 0x02D5FA68>

#匹配失败，什么也没返回

>>> z\_re.match('nice hello zmh')

>>>

search：

>>> z\_re.search('nice hello zmh')

<\_sre.SRE\_Match object at 0x02DCCD08>

>>> z\_re.search('nice hello zmh good')

<\_sre.SRE\_Match object at 0x02D5FA68>

>>> z\_re.search('hello zmh nice')

<\_sre.SRE\_Match object at 0x02DCCD08>

3. re\_obj.finditer('hello zmh')

返回一个迭代器 x

mat\_obj = x.next() 返回 match对象

mat\_obj.group(0) 查看匹配的到的字串

4. 正则替换函数

re.sub(pattern,repl,string)

说明：先用正则pattern匹配string，然后用 repl 替换匹配到的字串，返回替换后的字符串

>>> pt = r'\d{4}'

>>> re.sub(pt,'nice','1111goodhello2222')

结果：

'nicegoodhellonice'

5.正则切割函数

re.split(pattern,string)

方法1：

>>> s = '123+456\*789'

>>> re.split(r'[\+\-\\*]',s)

['123', '456', '789']

方法2：

>>> s = '123+456\*789'

>>> rs\_re = re.compile(r'[\+\-\\*]')

>>> rs\_re.split(s)

['123', '456', '789']

也就是说：我们可以使用re这个模块来调用正则对象的方法；

也可以使用编译生成的正则对象来调用正则方法！！！

>在引入一个模块后可以使用 dir(module) 显示这个模块的所有属性和方法！！！

然后通过查看帮助了解其用法。

#### 正则的其他修饰限制符

re.S 包括 \n

re.M 多行匹配，会影响 ^ $

re.X 可以忽略正则里面的 \n

正则里的 分组 (\.com|\.cn)

注意：当有分组的时候，re.findall()优先返回分组的数据。

### 面向对象

> 类 对象 方法 属性

类的定义：

global = 'GLOBAL'

class Ren：

#公有属性、类属性，在类外也可以使用

height = 165

weight = 65

#私有属性，只能在类内部使用

\_\_money = 100

def say(self):

lie = '我很欣赏你'

print('我有%d元' % self.name)

#访问方法的局部变量，不用self

print(lie)

#访问的是全局变量global

print(global)

类的实例化：

zmh = Ren()

#通过对象来访问或者修改类的公有属性

zmh.name = 'zmh'

zmh.say()

#通过类名直接访问类的公有属性

print Ren.height

#类的内置属性 \_\_dict\_\_ 返回类的属性和方法的键值对

print(\_\_dict\_\_)

在调试时，可以使用：

对象.\_类名\_\_私有属性

查看私有属性的值。

与私有属性一样，私有方法的命名也是前面 \_\_fun()

与其他编程语言一样，如果要在类外面访问或者设置私有属性，可以在类内部提供一个公有的 get 或者 set 方法

get方法有返回值

def get(self,age):

return self.age

第一个参数是类本事，不用传参！！！

set方法用于设置、修改属性值

def set(self,age,a):

self.age = a

if \_\_name\_\_ =='\_\_main\_\_'

表示直接用 python 解释器来解释这个 .py 文件时才执行 if 部分代码，如果是在其他的 .py 文件中通过 import 导入的话，if 代码块将不会被执行！！！

理解问题一定要深刻。

类里面的公共方法不能直接通过 类名.方法名 访问

1. 类方法 classmethod(公共方法)

作用：将一个公共方法转换成类方法，返回一个类方法，在类外部就可以通过 类名.类方法名 来直接调用

cm = clasmethod(fun)

2. 静态方法:staticmethod(公共方法)

sm = staticmethod(fun)

说明：以上两种方法实现的效果是一样的，但是，过程不一样

classmethod 定义的是动态类方法，也就是只有在调用时才加载进内存，占用资源较少

staticmethod 则是随着类的加载而加载进内存，所以在执行效率上，静态类方法更快，同时他占用的资源也就更多

除了以上两种定义方法，还可以使用修饰符，将一个公共方法转成类方法

@classmethod

def fun():

@staticmethod

def fun():

### 内部类

类的内部还有类，内部类的定义完全跟类的定义一样

如果想访问内部类的方法，可以实例化内部类

内部类的实例化：

方法1：

内部类 = 外部类对象.内部类()

print(内部类.属性)

方法2：

内部类 = 外部类.内部类()

### 魔术方法

def \_\_str\_\_():

return '我是类的魔术方法\_\_str\_\_'

作用：直接打印 类 时，返回的信息

初始化方法：def \_\_init\_\_(self,age=22):

在初始化实例时自动执行

析构方法：def \_\_del\_\_():

在对象销毁时自动调用

python 中也有强大的垃圾回收机制

使用：引入 gc 模块

调用 gc.collect() 方法

### python中的封装、继承、多态

直接 class 子类(父类)：

即可实现继承。

python中支持 多继承 ：一个子类继承多个父类

class 子类(父类1,父类2)

对于继承多个父类的，首先使用先继承的父类的相关属性！！！

### 连接数据库

使用：

#导入数据库模块

#注意：首先要安装mysql的驱动程序，这样才能导入这个模块！！！本机装的是32bit mysql软件

import MySQLdb

#连接数据库,在连接的同时选择好那个库！！！

conn = MySQLdb.connect(user='root',passwd='root',host='localhost',db='python\_test')

创建游标

cur = con.cursor()

#通过游标发送sql语句

cur.execute('insert into userinfo(name,age,gender) values ('zmh',22,'male') ')

--------------------------------

但实际开发中我们一般使用以下方法来发送sql语句

sql = "insert into userinfo(name,age,gender) values(%s,%s,%s)"

cur.execute(sql,('zmh',22,'male'))

#一次性插入多条数据,通过使用 cur.executemany(sql,[(),()])

sqlm = "insert into userinfo(name,age,gender) values(%s,%s,%s)"

cur.executemany(sqlm,[('cbh',22,'male'),('wys',23,'male')])

------------------------------

### 数据库全部操作代码示范

#coding=utf-8 这一句必须在最开头，第一行，在import之前！！！

import MySQLdb

conn =

# 更新数据，成功返回 1

# sql = 'update usrinfo set name="zmh" where id=1'

# cur.execute(sql)

# 删除记录，成功返回 1

# sql = 'delet MySQLdb.connect(user='root',passwd='root',host='localhost',db='python\_test')

cur = conn.cursor()

#插入数据,返回插入的记录数

# sql = "insert into usrinfo(name,age) values(%s,%s)"

# res = cur.execute(sql,('zmh',22))

e from usrinfo where id=2'

# cur.execute(sql)

#插入多条数据

# sql = "insert into usrinfo(name,age) values(%s,%s)"

# res = cur.executemany(sql,[('cbh',22),('bch',20),('wys',23)])

# 查询数据,注意，必须是先通过execute执行查询，得出结果集，才能从结果集里面取数据。

# 返回元组

# ((1L, 'zmh', 22), (3L, 'cbh', 22), (4L, 'bch', 20), (5L, 'wys', 23))

# sql = 'select \* from usrinfo'

# re = cur.execute(sql)

# res = cur.fetchall()

# print(res)

# 如果是使用fetchone() 是每次取一条，然后游标下移

# res = cur.execute('select \* from usrinfo')

# print(cur.fetchone())

# print(cur.fetchone())

# print(cur.fetchone())

# print('#'\*20)

# 重置指针

# cur.scroll(0,'absolute')#表示从头开始移动0位

# print(cur.fetchone())

# fetchmany(num) 取出指定条数

# res = cur.execute('select \* from usrinfo')

# print(cur.fetchmany(4))

# 合二为一,因为这个cur.execute('select \* from usrinfo')返回的就是查询结果集的总数。

# print(cur.fetchmany(cur.execute('select \* from usrinfo')))

# 关闭：

# 先关游标 cur.close()

# 再关连接 conn.close()

### 方法详解：

类方法和静态方法的区别

1. 类方法，必须要有self，通过 self.属性 来访问类属性

2. 静态方法，没有 self，通过 类名.属性名 来访问类属性

@classmethod

def classfun(self):

print(self.name)

@staticmethod

def staticfun():

print(Ren.name)

通过以上设置，就可以在外面直接用类名调用

除了使用修饰符，还可以使用

classmethod()

staticmethod()

进行转换！！！

------------------------------------

### python网络编程

使用socket模块

#导入socket模块

import socket

# 生成一个socket句柄，参数1：ip类型

#参数2：协议

s = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

# 绑定ip和端口，端口可以是1024-65535之间的任何，但不要使用一些已经被其他软件占用的端口，比如3306

s.bind(('127.0.0.1',8888))

#监听客户端

s.listen(10)

#s.accept()返回一个元组，也就是 socket对象 和 ip地址+端口的元组，

注意：这个函数只有在有客户端连接才会返回句柄，程序才会继续往下执行！！！

conn,addr = s.accept()

while True:

#使用这个连接向客户端发送数据

conn.send('hello，I am server')

#获取客户端信息,每次获取1024字节

strinfo = conn.recv(1024)

print(strinfo)

客户端：

import socket

c = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

# 作为客户端，只需要连接即可！！！

c.connect(('127.0.0.1',8888))

while True:

# 向服务器发送数据

c.send('hello,i am client')

#接收服务器数据，每次接收 1K

print(c.recv(1024))

#睡 1 秒

time.sleep(1)

说明：只要运行服务器端py脚本，就是运行了一个服务器，会一直在监听客户端连接，当运行客户端py脚本时，就是连接上了服务器，客户端向服务器发送数据，服务器监听到了，也向客户端发送数据！！！

在python2.7中 input() 和 raw\_input() 都可以使用！！！

### 浅拷贝和深拷贝

数字和字符串是属于不可变类型，列表和字典是属于可变类型

深拷贝：

import copy

a = [1,2,3['a','b','c'],'d']

b = a # 是对引用的拷贝，b a 指向相同的地址空间

c = copy.copy(a) #浅拷贝 是独立的一份，地址与 a 不一样

#注意这里只是外层空间的一个深拷贝，内层列表还是公用同一个地址空间！！！

#真正的深拷贝是将所有的元素都独立开

d = copy.deepcopy(a)

>>> import copy

>>> a = [1,2,3,['a','b','c']]

>>> a

[1, 2, 3, ['a', 'b', 'c']]

>>> b = a

>>> b

[1, 2, 3, ['a', 'b', 'c']]

>>> b.append('d')

>>> b

[1, 2, 3, ['a', 'b', 'c'], 'd']

>>> a

[1, 2, 3, ['a', 'b', 'c'], 'd']

>>> id(a)

61478112

>>> id(b)

61478112

#浅拷贝，外层地址已经不一样，但内层列表的地址还是不变

>>> c = copy.copy(a)

>>> c

[1, 2, 3, ['a', 'b', 'c'], 'd']

>>> id(c)

61535656

>>> id(a[3])

61479312

>>> id(c[3])

61479312

#深拷贝：

>>> d = copy.deepcopy(a)

>>> d

[1, 2, 3, ['a', 'b', 'c'], 'd']

#外层地址不一样

>>> id(d) == id(a)

False

>>> d.append('e')

>>> d

[1, 2, 3, ['a', 'b', 'c'], 'd', 'e']

>>> a

[1, 2, 3, ['a', 'b', 'c'], 'd']

#内层列表地址也不一样

>>> id(a[3]) == id(d[3])

False

>>> d[3].append('d')

>>> d

[1, 2, 3, ['a', 'b', 'c', 'd'], 'd', 'e']

>>> a

[1, 2, 3, ['a', 'b', 'c'], 'd']

### 文件与目录

方法1：

>>> fo = open('./文件操作.txt')

>>> fo.read()

'hello world'

>>> fo.close()

方法2：

>>> f1 = file('./文件操作.txt')

>>> f1.read()

'hello world'

>>> f1.close()

------------------------------------------

这是一个什么错误：

>>> fnew.write("new contents")

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IOError: [Errno 0] Error

使用 w 或者 w+ 方式打开文件都会先清空原来的数据

默认是以只读 r 方式打开

读写方式打开 r+ 指针在开头，写入时直接覆盖前面的!!!

>>> f = open('new.txt','r+')

>>> f.write('000')

>>> f.close()

#通过f.read()，文件指针已经移到最后，所以是在最后插入

>>> f = open('new.txt','r+')

>>> f.read()

>>> f.write('000')

>>> f.close()

#### 文件对象方法

open方法返回的是一个迭代对象，可以用for循环打印出每一行。

直接读取，读到的是一大串：

>>> f = open('new.txt','r+')

>>> f.write('hello\nnice to meet you\nhow old are you')

>>> f.close()

>>> f = open('new.txt')

>>> s = f.read()

>>> s

'hello\nnice to meet you\nhow old are you'

>>> f.close()

通过for循环迭代读取，读到的是每一行

>>> for i in open('new.txt'):

... print i

...

hello

nice to meet you

how old are you

f.readline() 每次读取一行

f.readlines() 返回i一个列表，列表的每个元素就是文件的每一行

>>> f.readlines()

['hello\n', 'nice to meet you\n', 'how old are you']

f.next() 和 readline() 类似，如果超出会提示错误

>>> f = open('new.txt')

>>> f.next()

'hello\n'

>>> f.next()

'nice to meet you\n'

>>> f.next()

'how old are you

>>> f.next()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

StopIteration

-----------------------

f.writelines() 写入多行数据

>>> l = ['one\n','two\n','three\n']

#以追加模式打开

>>> f = open('new.txt','a')

>>> f

<open file 'new.txt', mode 'a' at 0x030228B8>

#f.writelines 与 f.readlines 是一个相反的过程！！！

>>> f.writelines(l)

>>> f.close()

#### 控制文件指针

f.seek(0，0) 指针回到开头

f.seek(0，2) 指针回到开头

参数说明：偏移量,选项（指针的位置）

0：开头

1：当前位置 往后偏移

2：尾部 往前偏移

#### 提交更新 在不关闭文件的情况下更新文件内容

f.flush()

>>> f = open('new.txt','a')

#以追加的方式打开不能读

>>> f.read()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IOError: File not open for reading

>>> f.write('last')

>>> f.flush()

------------------------------------------

错误：为什么以读写方式打开，不能写呢？

读到最后就不能再写了？什么情况？

原来是指针作怪：只需 f.seek(0,2) 将指针移到最后即可！！！

------------------------------------------

>>> f = open('new.txt','r+')

>>> f.read()

'hello\nnice to meet you\nhow old are youone\ntwo\nthree\n'

>>> f.write('last')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IOError: [Errno 0] Error

问题：如何在windows安装telnet？

### os 模块

import os

#创建目录

os.mkdir('test')

#创建具有层次结构的mulu

>>> os.makedirs('a/b/c')

#列出目录

>>> os.listdir('./')

['db.py', 'new.txt', 'test', '\xce\xc4\xbc\xfe\xb2\xd9\xd7\xf7.txt']

#删除目录 只能删除空目录

>>> os.rmdir('./test')

>>> os.listdir('./')

['db.py', 'new.txt', '\xce\xc4\xbc\xfe\xb2\xd9\xd7\xf7.txt']

#删除多级目录 如果目录不存在是会报错的。

>>> os.removedirs('a/b')

#改变目录

>>> os.chdir('../')

#获取当前目录

>>> os.getcwd()

'E:\\Python\xbd\xcc\xb3\xcc\\pythontest'

### python 中的异常处理

使用方法：

在有可能出现异常的地方使用 try: 包起来，然后用 except xxx: 捕获处理 ，可以有多条 except 语句 分别捕获不同的异常

如果出现异常，程序停止往下运行

In [3]: try:

...: open('a.txt')

print var

# 这里写需要捕获的异常的名称，出现异常的信息

...: except IOError,msg:

# 打印异常信息

...: print msg

except NameError,msg:

pass

# 无论有没有出现异常都会执行finally语句

finally：

f.close()

...:

[Errno 2] No such file or directory: 'a.txt'

raise抛出异常，在任何情况下都可以使用raise抛出异常

raise 异常名('异常信息')

注意：这个异常名一定是python内置的异常名，不能是自定义的！！！

异常捕获也是可以嵌套的！！！

常见的异常都有：

AttributeError，NameError，ValueError，IOerror，

ImportError，KeyError， SyntaxError，TypeError，

IndexError，IndentationError

### Django框架开发初步 传智公开课-刑老师主讲

1. 安装Django： pip install Django

2. 创建项目：django-admin startproject myproject

3. 创建应用： 进入项目的根目录 运行python manage.py startapp myapp

4. 在项目根目录下创建静态资源目录 static 和 模板目录

5. 在应用的 models 文件中创建 数据库表，需要继承 models.Model

class Subject(modles.Model):

#设置表的字段和相关选项

name = CharField(length=128)

price = IntegerField(default=0)

6. 在 views 中引入 models中的 数据库表类 ，定义一个方法index，通过 Subject.objects.all() 方法取出所有的数据

7. 将取出的数据放在字典，然后通过

return render(req,'index.html',contents)

分配到模板中

from django.shortcuts import render

# 导入 应用中models下的 Subject 类

from myapp.models import Subject

#这个 view 相当于控制器，调用model完成业务逻辑

#这个参数req是必须要有的。

def index(req):

#返回一个列表，每个列表项是一条记录

subject\_list = Subject.objects.all()

contents = {'subject':subject\_list}

#将数据以 字典 的形式分配到模板中

return render(req,'index.html',contents)

8. 在setting.py 文件中包含 myapp

配置好 静态文件目录 模板文件目录

STATIC\_URL = '/static/'

#静态文件路径配置

#拼接路径 这个BASE\_DIR 就是项目的根目录。

STATIC\_PATH = os.path.join(BASE\_DIR,'static')

STATICFILES\_DIRS = (

STATIC\_PATH,

)

#模板文件配置

直接在TEMPLATES加上：

'DIRS': [os.path.join(BASE\_DIR,'templates')],

9. 编辑模板文件 index.html

首先在开头引入静态文件：

{%%}是django的模板标签

{% load staticfiles %}

<img src="{% static 'images/01.jpg'%}" />

<ul>

{% for itme in subject %}

<li>

{{item.name}}

{{item.price}}

</li>

{% endfor %}

</ul>

10. 生成数据库，cd 到项目根目录

执行：

1. python manage.py migrate

2. python manage.py createmigrations

3. python manage.py migrate

11. 设置路由

#引入 views 中的方法

from myapp.views import index

urlpatterns = [

url(r'^admin/', admin.site.urls),

#设置符合正则的url，指向 views中的index

url(r'^index/', index),

]

12. 生成一个超级管理员

项目根目录下运行：

python manage.py createsuperuser zmh

13. 配置 admin.py 文件，使得可以从后台操作数据库表

#引入 数据库表类并注册到 admin.site

from myapp.models import Subject

# Register your models here.

admin.site.register(Subject)

14. 运行程序

项目根目录下运行：

python manage.py runserver

15. 浏览器访问

localhost:8000/index

注意：首次运行要先进入后台，添加数据，然后才能成功访问 index