# PYTHON

## config

### 安装

环境变量path： python所在路径 及python路径下\Scripts

文本文件a.py

命令行python a.py执行

直接运行phython可以进入phython命令行，直接输入执行语句

Python命令行参数

|  |  |
| --- | --- |
| 选项 | 描述 |
| -i | python -i t.py  运行t.py文件后不立即退出python shell |
| -h | python -h  显示帮助信息 |
| -V | 输出Python版本号 |

Lib\idlelib\idle.bat可以启动python自带的gui编程环境

### 编译

python -O -m py\_compile a.py

编译成优化的字节码

### 包管理

Scripts文件夹下pip.exe可以进行包管理

可以将包安装到 \Lib\site-packages 下

pip install \*\* 安装\*\*\*

pip list 列出已经安装的包

包文件夹中包含\_\_init\_\_.py，可以为空

该文件在import模块时执行

lib下载地址:

<https://www.lfd.uci.edu/~gohlke/pythonlibs/>

（关键字edu whl）

1.下载whl文件后缀改为zip解压后直接拿出即可

2.pip install \*\*\*.whl

### 虚拟环境

在当前目录下生成venv文件夹，文件夹内包含虚拟环境，安装的模块都将放在该文件夹内

python -m venv venv

激活使用虚拟环境，此时命令提示符会显示(venv)，此时对python操作时会使用该虚拟环境下的安装模块（linux使用. venv/bin/activate）

venv\Scripts\activate

使用虚拟环境中的python.exe运行程序，可以调用到虚拟环境下安装的模块

venv\Scripts\python a.py

## grammar

### 导入包

from abc import \*

from aaa.def import \* 从aaa包中导入其中def.py内的所有内容

import math

import math as mt 导入包并重新命名

### 声明

#coding=utf-8则支持中文

str.decode('utf-8').encode('gbk') //字符串输出时使用

#!/usr/bin/python

告诉操作系统执行这个脚本的时候，调用/usr/bin下的python解释器  
这种用法是为了防止操作系统用户没有将python装在默认的/usr/bin路径里。当系统看到这一行的时候，首先会到env设置里查找python的安装路径，再调用对应路径下的解释器程序完成操作。

#!/usr/bin/env python 3.3

### 注释

#注释

### doc

**def** fun(x):  
 *"""this is doc"""* print(x)  
  
print(fun.\_\_doc\_\_)

### 语句格式

a = 1

b = 2.3

c = a+b

变量无类型

语句不需要分号结束

依靠排版区分语句

区分语句作用域用Tab

两个语句在同一行使用分号隔开

行连接保留第二行空格

str = "a\

def"

### 数据类型

int,float,bool,complex

bool对应True 和 False

4+2j complex

type(\*) 获取数据类型

type(\*) == type(1) 可以判断\*是否和1是相同的类型

### 类型转换

int(x )         将x转换为一个整数

long(x)         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](http://www.2cto.com/kf/web/Python/" \t "http://www.cnblogs.com/wuxiangli/p/_blank)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

hex(x )                 将一个整数转换为一个十六进制字符串

oct(x )                 将一个整数转换为一个八进制字符串

数字转字符串

s = str(12)

字符串转数字【s.isdigit()】

f = float(s)

i = int(s)

字符转数字

ord(c)

数字转字符

chr(i)

### 运算符

无自增运算

连续赋值

a,b = 3,3+1

a = 2\*\*3 #2的幂运算 2\*2\*2

a = 3//2.0 #取整运算

a = 3%2 #取余

逻辑运算符:

not/and/or(没有!,||,&&)

结果 True 和 False

### 语句结构

#### 选择

# coding=utf-8

score = 75

if score>90:

print("优秀")

elif score>60:

print("合格")

else:

print("不合格")

#### 循环

yyy在循环结束后输出，range最后一个参数可以定义step

for i in range(1,10):

print("item{0}:{1}".format(i,i + 0.3))

print("yyy")

a = 0

while a < 10:

print(a)

a+=1

### 函数

def fun(a,b=5):

if a > b:

return a

else:

return b

print(fun(2,4))

### Class

\_xxx 保护成员

\_\_xxx  私有成员

\_\_xxx\_\_ 系统定义名字

dir(类名) 获取类中的属性和方法

#### 定义

class CC:

val = 10

def \_\_init\_\_(self,\_name):

self.name = \_name

def show(self):

print(self.name)

print(self.val)

使用：

c = CC("abc")

c.show()

类内函数第一个参数都是类的对象

类的成员变量可以通过self.\*\* 直接创造

\_\_init\_\_ 构造函数

#### 继承

class DD(CC):

def \_\_init\_\_(self,nn):

CC.\_\_init\_\_(self,nn)

def Sw(self):

print(self.\_name)

d = DD("yy")

d.Sw()

### 异常

#### 异常处理

**try**:  
 val = dic[7]  
**except** (KeyError,TypeError) **as** erro: #或者**except** Exception **as** e:  
 print(**" : %s"**%erro)

**else:**

无任何异常时执行

#### 抛出异常

**raise** Exception(**"抛出一个异常"**)

### 字符串

#### 编码转换

默认utf-8

cstr=str.encode(“gbk”)

转成byte流

str.encode('utf-8')

byte流转成字符串

by.decode(encoding = "utf-8")

#### 引号使用

单引号中可以包含双引号，双引号中可以包含单引号

c = 'abc'

d = 'it is a "apple"'

e = "it's a dog"

三引号支持换行，保留编辑时的换行格式 （三个单引号或者三个双引号）

f = '''it

is

a

dog'''

#### 转义

字符串前加r，字符串中\n之类的转义字符可以当作普通字符

路径可以使用

f = r"abc\ndef"

#### 字符串重复

字符串重复10次

"abc\n"\*10

#### 截取字符串

c = "abcdefg"

f1 = c[1] #取1位置，结果b

f2 = c[1:] #取1位置及之后所有，结果bcdefg

f3 = c[:3] #从0位置取到3位置结束，结果abc

f4 = c[1:3] #从1位置取到3位置，结果bc

#### 格式化字符串

s = **"val:%d,%.2f,%s"**% (12,1.523,**"abc"**)

s = **"val:{0},{1},{2}"**.format(12,1.5,**"abc"**)

%5s 表示字符串至少5个字符，如果不够则前面空格补齐

%-5s 表示字符串至少5个字符，如果不够则后面空格补齐

#### 拆分

str.split(**","**) 返回列表[]

#### 判断是否是整数

str.isdigit()

#### 查找

str.find(**"de",0,6**) 后两个参数表示从查找范围，可以省略

str.index(**"de"**) 如果找不到会报错

返回查找到的下标，-1表示没有找到

#### 连接字符串

str.join(**"def"**)

#### 替换

str = str.replace(**".zip"**,**""**)

### yield

def fun(start,end):

while start < end:

yield start

start = start + 1

for n in fun(0,10):

print(n)

### main

if \_\_name\_\_ == "\_\_main\_\_":

print "xxx"

每个文件都由一个模块名\_\_name\_\_，如果是被import\_\_name\_\_为文件名。如果是运行开始的文件则为\_\_main\_\_

使用if可以让内容只在该文件被启动运行时执行，而该模块被调用时不执行

### pass

def fun():

pass

#to do

方法或类下不能为空，使用pass起占位作用

### 装饰器

**def** TestFun(fun):  
 fun(10,20)  
  
@TestFun  
**def** fun(a,b):  
 print(a+b)

通过@引用一个调用当前函数的函数

### 数据集

#### 元祖

arry = (123,"abc",12.4)

arry = tuple( (123,"abc",12.4) )

arry[1] #元素“abc”

内容不能修改，只能读取

#### 列表

arry = [123,"abc",12.4]

arry = list([123,"abc",12.4])

Arry = list("abc")

str = arry[1] #访问元素“abc”

del li[1] #删除元素“abc”

arry[-1] #访问倒数第一个元素12.4

arry[1]=22 #修改元素

arr.append(3) #添加元素

l.sort() #排序

arry = arry \* 4 #元素变四倍

1. in arry #元素是否在集合中

遍历：

for obj in arry:

#生成顺序序列

arry = range(len(xNames))

#### 集合

s1 = set("abccd")

s2 = {'b','c','x','c'}

#只保留序列中非重复内容,去重

构造函数只能加入序列

#添加元素

s.add(3)

#交集 结果：b，c

x = s1 & s2

#并集 合起来不重复的元素 结果：a b c d x

y = s1 | s2

#差集 s1中去掉和s2的交集 结果：a d

z = s1 - s2

#### 字典

dic = {"str":1.2, 3.1:4}

dic =dict( {"str":1.2, 3.1:4} )

#修改元素，如果字典内有，则进行访问

dic['str'] += 5

#插入元素，如果字典内没有，则插入新值

dic[33] = 111

print(dic[33])

=========查找

**if** 5 **in** dic:

=========遍历

**for** key,value **in** dic.items():  
 **print** key,value

**for** key **in** dic:  
 **print** key,dic[key]

=========获取set

s = set(dic.values());  
s2 = set(dic.keys());

=========字符串格式化

dic = {**"str1"**:1.2, **"str2"**:**"aaa"**}  
ss = **"%(str1)f %(str2)s"**%dic  
print(ss)

#### 运算方法

元素个数 len(arr)

最小值 min(arr)

最大值 max(arr)

求和 sum(arr)

### lambda

结果要用\*ret进行取值

#### 操作每个元素

为每个元素执行相同操作

li = [1,2,3,4,5,6]  
ret = map(**lambda** x: x\*100,li)

#### 排序

**class** student:  
 **def** \_\_init\_\_(self,age,name):  
 self.age = age  
 self.name = name  
arry = (student(11,**'a'**),student(4,**'b'**),student(10,**'x'**))

对类中的age进行排序

ret = sorted(arry,key=**lambda** x:x.age)  
**for** stu **in** ret:  
 print(stu.age)

#### 过滤数据

只保留满足条件（x%2==0）的数

li = [1,2,3,4,5,6]  
ret = filter(**lambda** x: x%2==0,li)

arry = (**'xxx'**,**'abcxxxdef'**,**'xyz'**,**'x.x'**,**'axxxxxxa'**)

完全匹配：  
ret = filter((**lambda** x: re.match(**r'xxx'**,x)),arry)

匹配包含字符串的内容：

ret = filter((**lambda** x: re.search(**r'xxx'**,x)),arry)

只匹配x.x

ret = filter((**lambda** x: re.search(**r'x\.x'**,x)),arry)

#### 列表解析

从列表中获取满足条件的元素

li = [1,2,3,4,5,6]  
ret = [x **for** x **in** li **if** x%2==0 ]

### Py文件交互

在b文件中存在类CC

class CC:

def fun(self):

print("cc")

其他文件使用时

====导入文件

import b

c = b.CC()

c.fun()

====导入文件中的类

from b import CC

c = CC()

c.fun()

======函数调用

在b文件中只存在fun函数时

import b

b.fun()

======执行代码

如果b文件中有执行代码，被其他文件导入，因为导入在文件上面，所以b文件中的代码首先执行

### 输入输出

a = input(**"输入a:"**)

i = int(raw\_input(**"input:"**))

print(a)

print (a,b,c)

## file

### 读写文件

#### 中文路径转换

path = "E:\\文件.txt"

uipath = unicode(path , "utf8")

#### 写文件

f = open(**"D:\\1.txt"**,**"w"**) *#"a"追加，打开不清空*  
f.write(**"abcd\n"**)  
f.close()

#### 读文件

**for** line **in** open(**"D:\\1.txt"**):  
 pos1,pos2 = line.split(**","**)

读取文本文件全部内容

f = open(**"D:\\1.txt"**)  
dat = f.read()

#### 文件指针位置移动

f.seek(移动量,基准位置) f.seek(0,0)移动到开始位置

#### 读写utf8

**import** codecs

ll = codecs.open(**r"D:\decoderZip.pyw"**, **'r'**, **'utf-8'**).readlines()  
of = codecs.open(**r"D:\decoderZip1.pyw"**, **'w'**, encoding=**'utf8'**)  
**for** line **in** ll:  
 of.write(line)  
of.close()

### 遍历文件

os.sep windows下是\\

os.path.join 连接字符串，文件夹路径和文件名

读取出的文件、路径名中包含中文时，字符串不能print但可以正常使用

#### 遍历文件夹

import os

dirPath= "d:" + os.sep

for name in os.listdir(dirPath):

if os.path.isfile(os.path.join(dirPath,name)):

print "file:" + name

if os.path.isdir(os.path.join(dirPath,name)):

print "dir:" + name

返回目录下所有pdf文件的全路径

ret = glob.glob(**"D:\\\*.pdf"**)

#### 遍历所有文件

遍历d:\\123下所有jpg文件

walk返回一个三元组：

dirpath当前处理路径

dirnames当前处理路径下包含的路径

filenames当前处理路径下包含的所有文件名

#encoding=utf-8

import os

s = os.sep

root = "d:" + s + "123" + s

for dirpath, dirnames, filenames in os.walk(root):

for fileName in filenames:#获取每一个文件名

fileName\_ext = os.path.splitext(fileName)#拆分文件名和类型

if(fileName\_ext[1] == ".jpg"):

print os.path.join(dirpath,fileName)#输出文件路径

### 文件及目录操作

#### 目录拼接

ret = os.path.join(**"D:"**+ os.path.sep + **"abc"**,**"def"**)

得到D:\abc\def

ret = os.path.split('D:\\abc.jpg')

得到('D:\\abc', 'def')

ret = os.path.splitext(**"D:\\abc.jpg"**)

得到('D:\\abc', '.jpg')

ret = os.path.abspath(**"..\\\\abc\\def"**)

将多余的分隔符去掉，将相对路径转换为绝对路径

#### 当前文件路径

curdir = os.path.abspath( os.curdir)

或者：curdir = os.path.split(os.path.abspath(\_\_file\_\_))[0]

#### 判断是否存在

判断目录或文件是否存在

os.path.exists(strPath)

#### 创建目录

makedirs只能创建不存在的目录

if(not os.path.exists(strPath)):

os.makedirs(strPath)

#### 删除目录

删除空目录：

os.rmdir(srcPath)

可以删除非空目录

shutil.rmtree(srcPath)

#### 移动/重命名

shutil.move(srcPath,dstPath)

#### 拷贝/重命名

shutil.copy(srcPath,dstPath)

#### 删除文件

os.remove(srcPath)

#### 目录/文件状态

os.stat(dir)

## Thread

### 注意问题

线程中不能修改全局变量

### 导入模块

**import** threading

### 开启线程

传入类的\_\_thrRun函数作为线程入口

\_thread = threading.Thread(target=self.\_\_thrRun)  
\_thread.setDaemon(**True**)  
\_thread.start()

setDaemon默认为False，如果使用默认值，则进程结束后，线程依然执行

如果设置为True，则进程结束时线程停止

### 线程等待

等待\_thr1线程执行完成，如果传递参数，则只等待传递的时间

self.\_thr1.join()

### 暂停

**import** time

单位为s

time.sleep(1)

### 线程同步

self.lock = threading.Lock()

self.lock.acquire()

执行代码。。。

self.lock.release()

### 结束线程

类中启动函数定义startRun状态为True

def Start(self,begin,end):

self.startRun = True

类中停止函数定义startRun状态为False

def Stop(self):  
 self.startRun = False

类中线程函数判断startRun进行停止

def \_\_thrRun(self):

while (True):

if self.startRun == False: return

....

if (..):

self.startRun = False

return

## tkinter

*#coding=utf-8***import** tkinter **as** tk  
**import** threading  
**import** tkinter.messagebox **as** tkMB  
**import** tkinter.filedialog **as** tkFD  
  
  
**class** Application(tk.Frame):  
 **def** \_\_init\_\_(self, master=**None**):  
 super().\_\_init\_\_(master)  
  
 self.pack() *#pack布局  
 #可以将pack全部换成grid grid(row=1) grid(row=0,column=1)* self.create\_widgets()  
 **def** create\_widgets(self):  
 self.L = tk.Label(self,text=**"txt"**, fg=**"black"**, bg=**"white"**)  
 self.L.pack()  
  
 self.E = tk.Entry(self)  
 self.E.pack()  
  
 self.Etxt = tk.StringVar()  
 self.Etxt.set(**"xxx"**)  
 self.E[**"textvariable"**] = self.Etxt  
  
  
 self.BtnRun = tk.Button(self,width=15, height=5,text = **"Run"**,command = self.\_\_Run)  
 self.BtnRun.pack(padx=20, side=**'left'**)  
 **def** \_\_Run(self):  
 filePathName = tkFD.askopenfilename(filetypes=[(**'all files'**, **'.\*'**), (**'text files'**, **'.txt'**)])  
 self.L[**'text'**] = filePathName  
  
 \_thread = threading.Thread(target=self.\_\_thrRun)  
 \_thread.setDaemon(**True**)  
 \_thread.start()  
 **def** \_\_thrRun(self):  
 ss = self.Etxt.get()  
 tkMB.showinfo(**"提示"**, ss)  
 self.BtnRun[**'state'**] = tk.NORMAL  
  
root = tk.Tk()  
root.title(**'title'**)  
root.geometry(**'400x200'**)  
root.maxsize(400, 200)  
root.minsize(400, 200)  
app = Application(master=root)  
app.mainloop()

### 设置窗口大小和位置

root.title('窗口名')

root.geometry('600x400') *#设置了主窗口的初始大小600x400*

root.maxsize(600, 400)

root.minsize(300, 240)

### 设置窗口居中

**def** center\_window(root, width, height):  
 screenwidth = root.winfo\_screenwidth()  
 screenheight = root.winfo\_screenheight()  
 size = **'%dx%d+%d+%d'** % (width, height, (screenwidth - width)/2, (screenheight - height)/2)  
 root.geometry(size)

center\_window(root, 300, 240)

### 控件

#### 按钮

属性名进行设置

tk.Button(text=**"运行"**, fg=**"red"**,command=self.fun,width = 20,height = 10)

#### 文本标签

self.l1 = tk.Label(text=**"txt"**, fg=**"black"**, bg=**"white"**)

self.l1["text"]="xxx"

#### 编辑框

self.editTxt = tk.Entry() *#创建文本控件*self.editTxt.pack()  
  
self.contents = tk.StringVar() *#和Entry控件绑定的变量*self.contents.set("this is a variable")  
self.editTxt[**"textvariable"**] = self.contents  
  
ss = self.contents.get() *#获取字符串*

self.editTxt.bind(**'<Enter>'**,self.fun1) *#绑定Enter事件到函数 def fun1(self, event):*

#### 进度条

scale=Scale(top,from\_=10,to=40,orient=HORIZONTAL,command=fun)  
scale.set(12) *#设置起始位置*scale.pack(fill=X,expand=1)

**def** fun(ev=**None**):

### 控件可用

self.BtnRun[**'state'**] = tk.DISABLED

self.BtnRun[**'state'**] = tk.NORMAL

### Ttk

**from** tkinter.ttk **import** \*

可以取代（from Tkinter import \*）tk中的button等控件

self.l1 = Label(text=**"Test"**, style=**"BW.TLabel"**)

文本可以让底色透明

### 弹出对话框

**from** tkinter **import** \*  
  
**def** dialog():  
 win = Toplevel()  
 Label(win,text=**"abc"**).pack()  
 win.focus\_set()  
 win.grab\_set()  
 win.wait\_window()  
  
root = Tk()  
Button(root,text=**"btn"**,command=dialog).pack()  
root.mainloop()

### 特殊对话框

Lib\tkinter\

filedialog.py

messagebox.py

#### 文件和文件夹

**import** tkinter.filedialog **as** tkFD

选择文件对话框

filePathName =tkFD.askopenfilename(filetypes=[(**"图像"**, **".jpg"**)])

[('all files', '.\*'), ('text files', '.txt')]

文件夹选择对话框

dir = tkFD.askdirectory()

#### 消息框

**import** tkinter.messagebox **as** tkMB

tkMB.showinfo(**"提示"**,**"处理完成"**)

## Lib

### sys

#### 获取python环境路径

**import** sys  
sys.path

#### 获取命令行参数

脚本名： sys.argv[0]  
参数1： sys.argv[1]  
参数2： sys.argv[2]

#### 获取已加载的模块名

sys.modules

可以通过该字典移除已经加载的模块

### 获取环境变量

export CaffeHome=home/cpms/caffe

import os

env\_dist = os.environ

print env\_dist.get('CaffeHome')

### 执行系统命令

仅仅在一个子终端运行系统命令，而不能获取命令执行后的返回信息

os.system("help")

### string

遍历小写字母

**for** word **in** string.lowercase:  
 **print** word

### re

import re

pic\_url = re.findall('img src="(.\*?)"',html,re.S)

无re.S时，遇到字符串中的\n则不往下一行匹配

|  |  |
| --- | --- |
| match(pattern, string, flags = 0) | 使用带有可选标记的正则表达式的模式来匹配字符串。如果匹配成功，返回匹配对象，否则返回None |
| search(pattern, string ,flags = 0) | 使用可选标记搜索字符串中第一次出现的正则表达式模式。如果匹配成功，则返回匹配对象，否则返回None |
| findall(pattern, string[,flags] ) | 查找字符串中所有(非重复)出现的正则表达式模式，并返回一个匹配列表 |
| finditer(pattern, string[,flags] ) | 与findall()相同，但返回的是一个迭代器。对于每一次匹配，迭代器都能返回一个匹配对象 |
| split(pattern, string, max = 0) | 根据正则表达式的模式分隔符，split函数将字符串分割为列表，返回匹配列表，分割最多操作max次 |

### math

math.sqrt(9)

### random

产生1到10之间的随机数

random.uniform(1,10)

### time

暂停1s

time.sleep(1)

获取当前时间

s = time.strftime(**'%Y-%m-%d %H:%M%S'**,time.localtime(time.time()))

### datetime

start = datetime.datetime.now()

timeSpan = datetime.datetime.now() - start

### pickle

=======序列化到内存

lsSave = pickle.dumps(obj) #将对象序列化，序列化后的数据lsSave可以用来传递

l = pickle.loads(lsSave) # 将对象恢复

=======序列化到文件

f1 = open('1.pk','wb')

pickle.dump(obj,f1) #将对象放入文件

f1.close()

f2 = open('1.pk','rb') # 将对象从文件读回

lf = pickle.load(f2)

f2.close()

### dbm

key和val必须是字符串

**import** dbm  
db = dbm.open(**"D:\\people"**,**"c"**)  
db[**"aa"**] = **"10"**db[**"bb"**] = **"20"**db.close()

C：打开dbm文件，如果不存在则创建

N：新建dbm文件，如果存在则覆盖

W:打开dbm文件，不存在则失败

删除key-val

**del** db[**"aa"**]

获取所有key（返回list）

li = db.keys()

### sqlite

**import** sqlite3  
conn = sqlite3.connect(**"D:\\abc.db"**)  
cursor = conn.cursor()  
cursor.execute(**"create table aa(id int , name varchar(30),age int)"**)  
cursor.execute(**"insert into aa values(1,'n1',11)"**)  
cursor.execute(**"insert into aa values(2,'n1',11)"**)  
conn.commit()  
cursor.close()  
conn.close()

获取数据：

cursor.execute(**"select \* from aa"**)  
**for** row **in** cursor.fetchall():  
 print(row)

### **socket**

#### udp

##### 服务器

**from** socket **import** \*  
HOST = **'192.168.1.3'**PORT = 9999

s = socket(AF\_INET, SOCK\_DGRAM)  
s.bind((HOST, PORT))  
**while True**:  
 data, address = s.recvfrom(1024)  
 print(data, address)

s.sendto('this is the UDP server',address)    
s.close()

##### 客户端

from socket import \*

HOST='192.168.1.3'

PORT=9999

s = socket(AF\_INET,SOCK\_DGRAM)

s.connect((HOST,PORT))

while True:

message = raw\_input('send message:>>')

s.sendall(message)

data = s.recv(1024)

s.close()

#### tcp

##### 服务器

from socket import \*

address='127.0.0.1' #127.0.0.1是监听本机 0.0.0.0是监听整个网络

port=12345

buffsize=1024

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

s.bind((address,port))

s.listen(1) #最大连接数1

while True:

clientsock,clientaddress=s.accept()

print('connect from:',clientaddress)

while True:

recvdata=clientsock.recv(buffsize).decode('utf-8')

if not recvdata:

break

senddata=recvdata+'from sever'

clientsock.send(senddata.encode())

clientsock.close()

s.close()

##### 客户端

from socket import \*

address='127.0.0.1'

port=12345

buffsize=1024

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

s.connect((address,port))

while True:

senddata=”xxxx”

s.send(senddata.encode())

recvdata=s.recv(buffsize).decode('utf-8')

print(recvdata)

s.close()

### 读写ini

import configparser

cfg = configparser.ConfigParser()

cfg.read("F:\\cfg.ini")

val = cfg.get("section","key")

cfg.add\_section("section")

cfg.set("section","a","b")

cfg.write(open("F:\\cfg.ini","w"))

### 测试

**class** TesC(unittest.TestCase):  
 **def** setUp(self):  
 print(**'start'**)  
 **def** tearDown(self):  
 print(**'end'**)  
 **def** runTest(self):  
 self.failIf(1+1 != 2,**'erro'**)  
 self.failUnless(1+1 == 2,**'erro'**)  
 self.failUnlessEqual(1+1,2,**'erro'**)

suite = unittest.TestSuite()  
suite.addTest(TesC())  
runner = unittest.TextTestRunner()  
runner.run(suite)

### Zipfile

**import** zipfile

#### 压缩

f = zipfile.ZipFile('E:/test.zip','w',zipfile.ZIP\_DEFLATED)

f.write('D:/1.jpg')

f.write('D:/2.jpg')

f.close()

#### 解压

f = zipfile.ZipFile('E:/test.zip')

f.extractall() 不传递参数则解压所有文件到当前目录

f.close()

第一个参数为解压后路径，路径必须存在

第二个参数为需要解压的文件名列表（zip.namelist()可以获取）

第三个参数为密码，函数必须传入ascii字符串  
zip.extractall(**r'D:\1'**,**None**,**'123'**.encode(**'ascii'**))

### smtp

**import** smtplib  
**from** email.mime.text **import** MIMEText  
**from** email.utils **import** formataddr  
  
my\_sender = **'\*\*\*\*\*\*@qq.com'** *# 发件人邮箱账号*my\_pass = **'...'**my\_user = **'\*\*\*\*@qq.com'** *# 收件人邮箱账号***def** mail():  
 **try**:  
 msg = MIMEText(**'填写邮件内容'**, **'plain'**, **'utf-8'**)  
 msg[**'From'**] = formataddr([**"FromRunoob"**, my\_sender]) *# 括号里的对应发件人邮箱昵称、发件人邮箱账号* msg[**'To'**] = formataddr([**"FK"**, my\_user]) *# 括号里的对应收件人邮箱昵称、收件人邮箱账号* msg[**'Subject'**] = **"发送邮件测试"** *# 邮件的主题，也可以说是标题* server = smtplib.SMTP\_SSL(**"smtp.qq.com"**, 465) *# 发件人邮箱中的SMTP服务器，端口是25* server.login(my\_sender, my\_pass) *# 括号中对应的是发件人邮箱账号、邮箱密码* server.sendmail(my\_sender, [my\_user, ], msg.as\_string()) *# 括号中对应的是发件人邮箱账号、收件人邮箱账号、发送邮件* server.quit() *# 关闭连接* **except** Exception **as** e: *# 如果 try 中的语句没有执行，则会执行下面的 ret=False* print(e)  
 **return False  
 return True**

### tool

http 服务器:

python -m http.server 80

默认端口8080

### http

**from** urllib.parse **import** urlsplit, parse\_qs  
**from** http.server **import** HTTPServer, BaseHTTPRequestHandler  
  
**class** ProHeader(BaseHTTPRequestHandler):  
 **def** do\_GET(self):  
 url = urlsplit(self.path)  
 qs = parse\_qs(url.query)  
  
 rsp = **"get"** rsp = rsp.encode(**"gb2312"**)

self.send\_response(200)

self.send\_header("Content-type", "text/html; charset=gb2312")

self.send\_header("Content-Length", str(len(rsp)))

self.end\_headers()

self.wfile.write(rsp)  
   
 **def** do\_POST(self):  
 datas = self.rfile.read(int(self.headers[**'content-length'**]))

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 httpd = HTTPServer((**''**, 1234), ProHeader)  
 httpd.serve\_forever()

## c++Mix

### C调用python

文件名一定不能是test，abc.....系统模块有。

不存在#include <inttypes.h>

屏蔽该行，包含#include <stdint.h>

没有python36\_d.lib

pyconfig.h中：

pragma comment(lib,"python36\_d.lib")

改为python36.lib

#### Py文件

文件名MyTestMul

def add(i,j):

return i+j

#### c调用

Py\_SetPythonHome(L"C:\\py363");

Py\_Initialize();

if ( !Py\_IsInitialized() )

{

cout<<"Py\_Initialize 错误"<<endl;

return;

}

PyObject\* model = PyImport\_ImportModule("MyTestMul");

if (NULL == model)

{

cout<<"PyImport\_ImportModule 错误"<<endl;

return;

}

PyObject\* pfun = PyObject\_GetAttrString(model,"add");

PyObject\* pParm = PyTuple\_New(2);

PyTuple\_SetItem(pParm, 0, Py\_BuildValue("i",3));

PyTuple\_SetItem(pParm, 1, Py\_BuildValue("i",4));

PyObject\* pRetVal = PyEval\_CallObject(pfun, pParm);

int retVal = -1;

int ret = PyArg\_Parse(pRetVal, "i", &retVal);

if (0 == ret)

{

cout<<"PyArg\_Parse错误"<<endl;

return;

}

//Py\_DECREF(pfun);

//Py\_DECREF(pParm);

//Py\_DECREF(pRetVal);

Py\_Finalize();

=====构造参数

PyObject \*param = Py\_BuildValue("(i,i,i)", 123, 456, 789);// 构造元祖数据

PyObject \* pList = PyList\_New(0);

for (int i = 0; i < 3; i++) PyList\_Append(pList, Py\_BuildValue("i", i));

|  |  |
| --- | --- |
| 字符 | 含义 |
| s或者z | (string) [char \*]  将C字符串转换成Python对象，如果C字符串为空，返回NONE。 |
| s#或者z# | (string) [char \*, int] :将C字符串和它的长度转换成Python对象，如果C字符串为空指针，长度忽略，返回NONE。 |
| z | (string or None) [char \*] :作用同s |
| i或b、l（long） | (integer) [int] :将一个C类型的int转换成Python int对象。 |
| c | (string of length 1) [char] ：将C类型的char转换成长度为1的Python字符串对象。 |
| d或f | (float) [double] :将C类型的double转换成python中的浮点型对象。 |
| O& | (object) [converter, anything] ：将任何数据类型通过转换函数转换成Python对象，这些数据作为转换函数的参数被调用并且返回一个新的Python对象，如果发生错误返回NULL。 |
| (items) | (tuple) [matching-items] ：将一系列的C值转换成Python元组。 |
| [items] | (list) [matching-items] ：将一系列的C值转换成Python列表。 |
| {items} | (dictionary) [matching-items] ：将一系类的C值转换成Python的字典，每一对连续的C值将转换成一个键值对。 |

### python使用dll

字符串传递需要转换格式

cstr=str.encode(“gbk”)

cdecl调用方式：

ctypes.cdll.LoadLibrary

Stdcall调用方式：

ctypes.windll.LoadLibrary

#### 函数调用

extern "C" \_\_declspec(dllexport) char\* fun(int \*a,char \*buf,int bufLen)

{

\*a = \*a \* 10;

strcpy\_s(buf, bufLen, "xxxx");

return "abcdef";

}

*#coding=utf-8***import** ctypes  
*#加载dll*dl = ctypes.cdll.LoadLibrary(**r'C:\DT.dll'**)  
  
*#创建字符串缓冲区*bufLen = 100  
strBuf = ctypes.create\_string\_buffer(**'\0'**,bufLen)  
  
*#创建int\* byref将数值转换为指针*a = ctypes.c\_int(5)  
aRef = ctypes.byref(a)  
  
*#执行dll中的函数*pchar = dl.fun(aRef,strBuf,bufLen)  
  
*#将dll返回的char\*进行转换*szbuffer = ctypes.c\_char\_p(pchar)  
  
  
**print** a.value  
**print** szbuffer.value  
**print** strBuf.value

**"""  
sBuf = '123456789'  
pStr = ctypes.c\_char\_p( )  
pStr.value = sBuf  
bufLen = len(pStr.value)  
"""**

#### 类型映射

ctypes数据类型          C数据类型   
c\_char                          char   
c\_short                         short   
c\_int                             int   
c\_long                          long   
c\_ulong                        unsign long   
c\_float                          float   
c\_double                      double   
c\_void\_p                       void   
对应的指针类型是在后面加上"\_p"，如int\*是c\_int\_p

#### 结构体交互

struct DatStu

{

int num;

char val[512];

};

extern "C" \_\_declspec(dllexport) void fun(DatStu \*stu)

{

stu->num = 10;

strcpy\_s(stu->val, 512, "xxx");

}

**class** DatStu(ctypes.Structure):  
 \_fields\_ = [ (**"num"**, ctypes.c\_int),  
 (**"val"**, ctypes.c\_char \* 512)]  
  
  
dl = ctypes.cdll.LoadLibrary(**r'C:\DT.dll'**)  
  
dat = DatStu()  
dl.fun(ctypes.byref(dat))  
  
**print** dat.num  
**print** dat.val

### c++导出pyd

========dll编写

#include "Python.h"

static PyObject \*ex\_foo(PyObject \*self, PyObject \*Argvs)

{

int Argv1(0), Argv2(0);

if (!PyArg\_ParseTuple(Argvs, "ii", &Argv1, &Argv2))

{//ii表示解析出来两个int型参数

cout << "parse param failed" << endl;

return NULL;

}

cout << Argv1 << "+ " << Argv2 << " = " << Argv1 + Argv2 << endl;

Py\_INCREF(Py\_None);//Py\_INCREF增加PyObject对象的引用计数

//Py\_DECREF减小PyObject对象的引用计数

//为了让Python回收废弃的内存、或者防止Python过早地自动回收内存

//必须用Py\_EDCREF和Py\_INCREF来控制PyObject的引用计数

return Py\_None;

}

static PyMethodDef ModulesMethods[] =

{

{ "fun", ex\_foo, METH\_VARARGS, "fun() doc string" },

{ NULL, NULL }

//fun是导出函数名

//METH\_VARARGS，则用Tuple来传递参数 METH\_KEYWORDS，则用Dictionary的Key（键值）来传递参数

//"fun() doc string" 函数的说明字符串。在Python中就是函数的DocString \_\_doc\_\_。

};

static struct PyModuleDef ModuleDesc =

{ PyModuleDef\_HEAD\_INIT,

"Module", //内置的模块名 模块名.\_\_name\_\_来获取这个字符串

"This module is created by C++. And it Add two Integer s!", //模块的DocString，也可以用模块名.\_\_doc\_\_获得

- 1, //-1 模块在全局范围

ModulesMethods

};

PyMODINIT\_FUNC PyInit\_DllName(void)//生成文件名必须 DllName.pyd

{

//Py\_InitModule("DllName", ModulesMethods); py2

return PyModule\_Create(&ModuleDesc);

}

========使用

import DllName

DllName.fun(1,3)

## extend

### requests

pip install requests

访问本机网址时

url = **'http://localhost:8080/abc'**

#### 发送请求

header = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.140 Safari/537.36 Edge/17.17134'}

url = '\*\*\*\*\*\*'

response = requests.Response()

response = requests.get(url,headers = header)

contentTxt = response.text

contentBin = response.content

contentJson = response.json()

#### 发送post请求

url = 'http://.......'

d = {'userName': 'abc', 'passWord': '123'}

r = requests.post(url, data=d)

### html解析

#### bs4

**from** bs4 **import** BeautifulSoup

|  |  |
| --- | --- |
| 方法 | 说明 |
| soup = BeautifulSoup(txt,"html.parser") | 读取html文件字符串生成对象，解析器可以用html.parser或者lxml |
| obj = soup.tagName  txt = obj .text | 选择第一个tagName对象  返回值为一个新的BeautifulSoup对象  text是标签包围的内容  soup.tagName.tagName2多级选择 |
| val=soup.tagName['attr'] | 选择第一个tagName对象的attr属性，  如果属性是class，则返回列表，其他则返回字符串 |
| .parent | 选择父节点 |
| obj = soup.find('tagName')  obj = obj.find\_next('tagName') | 选择节点，用结果选择下一个 |
| obj = soup.find(''tagName'',attr= 'aa') | 选择''tagName''节点中attr属性为aa的节点 |
| obj = soup.find(''tagName'',class\_= 'bb') | 选择''tagName''节点中class属性包含bb的节点 |
| obj = soup.find(class\_= 'bb') | 选择class属性包含bb的节点 |
| soup.find\_all(''tagName') | 选择所有tagName节点  返回BeautifulSoup对象列表 |

#### lxml

**from** lxml **import** etree

|  |  |
| --- | --- |
| 方法 | 说明 |
| xpathObj=etree.HTML(text) | 读取html文本字符串初始化生成一个XPath解析对象 |
| xpathObj=etree.parse(obj,etree.HTMLParser()) | obj可以是文件路径，解析html文件并自动纠正html内容错误 |
| result=etree.tostring(xpathObj,encoding='utf-8')  txt = result.decode('utf-8') | 转换出对应的字符串数据 |
| obj = xpathObj.xpath('//li[@class="jyl"]')  for val in obj:  print(val.text) | xpath返回list对象  text是标签包围的字符（不包括子标签内容） |

### numpy

#### 数据转换

pythno list转换成矩阵

X = [[-4,-2,0,2], [-4,-2,0,2],[-4,-2,0,2],[-4,-2,0,2]]

X=np.array(X)

#### 矩阵生成

|  |  |  |
| --- | --- | --- |
| 类型 | 代码 | 结果 |
| 自定义生成矩阵 | def fun(x,y):  return y  np.fromfunction(fun,(3,3)) | [[0 1 2]  [0 1 2]  [0 1 2]] |
| 空矩阵 | np.empty((3,3)) | [[0 0 0]  [0 0 0]  [0 0 0]] |
| 赋值生成矩阵 | np.array([[1,0],[0,1]]) | [[1 0]  [0 1]] |
| 线性矩阵 | np.arange(1,8,2) | [1 3 5 7] |
| 矩阵纬度变化 | V = np.arange(1,8,2)  V.shape = 2,2 | 将[1 3 5 7]变成  [[1 3]  [5 7]] |

### matplotlib

#### 曲线绘制

**import** matplotlib.pyplot **as** plt  
**from** pylab **import** \* *#中文支持*mpl.rcParams[**'font.sans-serif'**] = [**'SimHei'**]  
  
  
  
x = [5,20,25,30,50]  
y1 = [0.855, 0.84, 0.835, 0.815, 0.81]  
y2=[0.86,0.85,0.853,0.849,0.83]  
plt.plot(x, y1, marker=**'o'**, mec=**'r'**, mfc=**'w'**,label=**u'曲线1'**)  
plt.plot(x, y2, marker=**'\*'**, ms=10,label=**u'y=曲线2'**)  
plt.legend()  
*#生成X的别名（X实时显示失效）  
#xNames = ['5', '20', '25', '30', '50']  
#plt.xticks(x, xNames, rotation=45)*plt.margins(0)  
plt.subplots\_adjust(bottom=0.15)  
plt.xlabel(**u"X"**)  
plt.ylabel(**"Y"**)  
plt.title(**"title"**)  
plt.show()

#### 三维绘制

from matplotlib import pyplot as plt

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

fig = plt.figure()

ax = Axes3D(fig)

#X = np.arange(-4, 4, 2)

#Y = np.arange(-4, 4, 2)

#X, Y = np.meshgrid(X, Y)

#R = np.sqrt(X\*\*2 + Y\*\*2)

#Z = np.sin(R)

X = [[-4,-2,0,2], [-4,-2,0,2],[-4,-2,0,2],[-4,-2,0,2]]

X=np.array(X)

Y = [[-4,-4,-4,-4],[-2,-2,-2,-2],[ 0, 0, 0, 0],[ 2, 2, 2, 2]]

Y=np.array(Y)

Z=[[-0.58617619,-0.9712778,-0.7568025,-0.9712778 ],[-0.9712778,0.30807174,0.90929743,0.30807174],[-0.7568025,0.90929743,0. ,0.90929743],[-0.9712778,0.30807174,0.90929743,0.30807174]]

Z=np.array(Z)

#ax.scatter(X, Y, Z) 散点

#ax.plot\_wireframe(X, Y, Z) 连线

ax.plot\_surface(X, Y, Z, rstride=1, cstride=1, cmap='rainbow') 曲面

plt.show()

### pillow

pip install pillow

**from** PIL **import** Image  
**import** numpy **as** np  
  
a = np.array(Image.open(**"D:\\1.jpg"**));  
b = [255,255,255] - a  
im = Image.fromarray(b.astype(**"uint8"**))  
im.save(**"D:\\2.jpg"**)

im = Image.open(filePathName)  
out = im.resize((200, 100), Image.ANTIALIAS) *# 图像缩放为200\*100*out.save(outfile)

### function

#### 财经

pip install tushare

import tushare as ts

ts.get\_today\_all()

#### 调用opencv

需要安装numpy

使用cv2.pyd

import cv2

img = cv2.imread("D:\\1.jpg")

cv2.namedWindow("Image")

cv2.imshow("Image", img)

cv2.waitKey (0)

cv2.destroyAllWindows()

#### 线性回归

**from** sklearn.linear\_model **import** LinearRegression  
  
x = [[1], [2], [3]]  
y = [[2], [4], [6]]  
  
model = LinearRegression()  
model.fit(x, y)  
x2 = [[9]]  
y2 = model.predict(x2)  
print(x2,y2)

#### 生成二维码

pip install Pillow

pip install qrcode

import qrcode

import PIL

url = 'http://....'

code\_maker = qrcode.QRCode(error\_correction=qrcode.constants.ERROR\_CORRECT\_H)

code\_maker.add\_data(url)

code\_maker.make(fit=True)

code\_image = code\_maker.make\_image()

code\_image = code\_image.convert('RGBA')

face\_image = PIL.Image.open('xxxxxx.png')

face\_image = face\_image.resize((100,100), PIL.Image.ANTIALIAS)

code\_width, code\_height = code\_image.size

code\_image.paste(face\_image, ((code\_width-100)//2, (code\_height-100)//2))

code\_image.save(r'qrcode.png')

### flask

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

return 'Hello, World!'

app.run()

#### 启动程序

app.run('0.0.0.0',1234) 监听所有ip，端口1234

#### 路由

路由末尾如果包含/，则访问时不包含/则自动加上/

路由末尾如果不包含/，则行为表现与一个文件类似，如果访问加/则404错误

|  |  |
| --- | --- |
| from flask import url\_for  with app.test\_request\_context():  pathstr = url\_for('static', filename='style.css')  print(pathstr)  url\_for('index') | 获取路由地址 |
| @app.route('/jyl/<paramname>')  def fun(paramname):  return 'p:%s'%(paramname) | 路由中获取url末尾的变量  可以限制paramname类型(默认string)：  <int:paramname>  <float:paramname>  <path:paramname> 可以包含斜杠的字符串 |
| @app.route('/login', methods=['GET', 'POST'])  def login():  if request.method == 'POST':  return do\_the\_login()  else:  return show\_the\_login\_form() | 使路由可以接受post和get方法 |

### Django

#### 下载

git clone <https://github.com/django/django.git>

#### 安装

python setup.py install

使用python命令运行django下载目录中的setup.py

安装到python目录下Lib\site-packages

PYHOME C:\py363

Path：%PYHOME%;%PYHOME%\Scripts;

%PYHOME%\Lib\site-packages\Django-2.1-py3.6.egg

#### 创建项目

django-admin-script.py startproject HelloWorld

创建的目录：

HelloWorld: 项目的容器。

manage.py: 一个实用的命令行工具，可让你以各种方式与该 Django 项目进行交互。

HelloWorld/\_\_init\_\_.py: 一个空文件，告诉 Python 该目录是一个 Python 包。

HelloWorld/settings.py: 该 Django 项目的设置/配置。

HelloWorld/urls.py: 该 Django 项目的 URL 声明; 一份由 Django 驱动的网站"目录"。

HelloWorld/wsgi.py: 一个 WSGI 兼容的 Web 服务器的入口，以便运行你的项目。

#### 启动服务

python manage.py runserver 0.0.0.0:8000

<http://127.0.0.1:8000/>

#### MVC

=======HelloWorld目录下创建view.py

【创建运行文件】

**from** django.shortcuts **import** render  
  
  
**def** hello(request):  
 context = {}  
 context[**'var'**] = **'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'  
 return** render(request, **'view.html'**, context)

使用字典类型的参数改变页面值

========urls.py修改

【运行文件绑定到启动页面】

**from** django.conf.urls **import** url  
**from** . **import** view  
  
urlpatterns = [  
 path(**'admin/'**, admin.site.urls),  
 url(**r'^$'**,view.hello)  
]

或者url(**r'^hello$'**,view.hello)

每个url代表一个地址栏输入的地址

=======HelloWorld目录同级（manage.py所在目录）下创建viewDir目录

【创建py文件绑定的模板】

创建view.html

包含内容<p>{{ var }}</p>

var为py文件可修改的值

=======settings.py修改

【将模板加入设置】

TEMPLATES = [  
 {  
 **'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,  
 **'DIRS'**: [os.path.join(BASE\_DIR,**'viewDir'**).replace(**'\\'**, **'/'**)], *#修改*

.........

.......

url() 可以接收四个参数，分别是两个必选参数：regex、view 和两个可选参数：kwargs、name

regex: 正则表达式，与之匹配的 URL 会执行对应的第二个参数 view。

view: 用于执行与正则表达式匹配的 URL 请求。

kwargs: 视图使用的字典类型的参数。

name: 用来反向获取 URL。

直接返回文字页面

**from** django.http **import** HttpResponse  
**return** HttpResponse(**"hello world"**)

#### 静态文件使用

工程目录\static\img 包含1.jpg

settings.py末尾

STATIC\_URL = **'/static/'**STATICFILES\_DIRS = [  
 os.path.join(BASE\_DIR, **"static"**),  
 **'/var/www/static/'**,  
]

引用时

{% load static %}

<img src="{% static "img/1.jpg" %}"/>

#### 模板标签

##### if/else

基本语法格式如下：

{% if condition %}

... display

{% endif %}

或者：

{% if condition1 %}

... display 1

{% elif condition2 %}

... display 2

{% else %}

... display 3

{% endif %}

根据条件判断是否输出。if/else 支持嵌套。

{% if %} 标签接受 and ， or 或者 not 关键字来对多个变量做判断 ，或者对变量取反（ not )，例如：

{% if athlete\_list and coach\_list %}

athletes 和 coaches 变量都是可用的。

{% endif %}

##### for

{% for %} 允许我们在一个序列上迭代。

与Python的 for 语句的情形类似，循环语法是 for X in Y ，Y是要迭代的序列而X是在每一个特定的循环中使用的变量名称。

每一次循环中，模板系统会渲染在 {% for %} 和 {% endfor %} 之间的所有内容。

例如，给定一个运动员列表 athlete\_list 变量，我们可以使用下面的代码来显示这个列表：

<ul>

{% for athlete in athlete\_list %}

<li>{{ athlete.name }}</li>

{% endfor %}

</ul>

给标签增加一个 reversed 使得该列表被反向迭代：

{% for athlete in athlete\_list reversed %}

...

{% endfor %}

可以嵌套使用 {% for %} 标签：

{% for athlete in athlete\_list %}

<h1>{{ athlete.name }}</h1>

<ul>

{% for sport in athlete.sports\_played %}

<li>{{ sport }}</li>

{% endfor %}

</ul>

{% endfor %}

##### ifequal/ifnotequal

{% ifequal %} 标签比较两个值，当他们相等时，显示在 {% ifequal %} 和 {% endifequal %} 之中所有的值。

下面的例子比较两个模板变量 user 和 currentuser :

{% ifequal user currentuser %}

<h1>Welcome!</h1>

{% endifequal %}

和 {% if %} 类似， {% ifequal %} 支持可选的 {% else%} 标签：8

{% ifequal section 'sitenews' %}

<h1>Site News</h1>

{% else %}

<h1>No News Here</h1>

{% endifequal %}

##### 注释

Django 注释使用 {# #}。

{# 这是一个注释 #}

##### 过滤器

模板过滤器可以在变量被显示前修改它，过滤器使用管道字符，如下所示：

{{ name|lower }}

{{ name }} 变量被过滤器 lower 处理后，文档大写转换文本为小写。

过滤管道可以被\* 套接\* ，既是说，一个过滤器管道的输出又可以作为下一个管道的输入：

{{ my\_list|first|upper }}

以上实例将第一个元素并将其转化为大写。

有些过滤器有参数。 过滤器的参数跟随冒号之后并且总是以双引号包含。 例如：

{{ bio|truncatewords:"30" }}

这个将显示变量 bio 的前30个词。

其他过滤器：

addslashes : 添加反斜杠到任何反斜杠、单引号或者双引号前面。

date : 按指定的格式字符串参数格式化 date 或者 datetime 对象，实例：

{{ pub\_date|date:"F j, Y" }}

length : 返回变量的长度。

##### include

{% include %} 标签允许在模板中包含其它的模板的内容。

下面这个例子都包含了 nav.html 模板：

{% include "nav.html" %}

##### 模板继承

模板可以用继承的方式来实现复用。

templates 目录中添加 base.html 文件，代码如下：

HelloWorld/templates/base.html 文件代码：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>tttt</title>

</head>

<body>

<h1>Hello World!</h1>

<p>aaaaaaa</p>

{% block mainbody %}

<p>original</p>

{% endblock %}

</body>

</html>

以上代码中，名为 mainbody 的 block 标签是可以被继承者们替换掉的部分。

所有的 {% block %} 标签告诉模板引擎，子模板可以重载这些部分。

hello.html 中继承 base.html，并替换特定 block，hello.html 修改后的代码如下：

HelloWorld/templates/hello.html 文件代码：

{% extends "base.html" %}

{% block mainbody %}<p>继承了 base.html 文件</p>

{% endblock %}

### mysql

pip install  PyMySQL

**import** pymysql.cursors  
connection = pymysql.connect(host=**'127.0.0.1'**, port=3306, user=**'root'**, password=**'root'**, db=**'testdb'**,  
 charset=**'utf8mb4'**, cursorclass=pymysql.cursors.DictCursor)  
cursor = connection.cursor()  
sql = **"SELECT \* FROM user"**cursor.execute(sql)  
result = cursor.fetchall()  
**for** data **in** result:  
 print(data)  
  
connection.commit()

### pymongo

pip install pymongo

import pymongo

|  |  |
| --- | --- |
| myclient = pymongo.MongoClient("mongodb://localhost:27017/") | 打开mongodb |
| dblist = myclient.list\_database\_names() | 获取数据库列表 |
| 选择数据操作集合 | |
| mydb = myclient['jyl'] | 选择数据库 |
| mycl = mydb["jylcl"] | 选择数据库集合 |
| 增加 | |
| mydict = { 'name':'aaa','age':11}  x = mycl.insert\_one(mydict)  id = x.inserted\_id | 向集合中插入文档  插入时"\_id"可以指定插入数据的id，不指定则自动分配  insert\_many()插入多个数据，输入参数为列表中包含多个集合 |
| 修改 | |
| myquery = { "name": "aaa" }  newvalues = { "$set": { "age": "100" } }  mycl.update\_one(myquery, newvalues) | 设置集合中name为aaa的文档中age为100 |
| 删除 | |
| myquery = { 'name':'aaa' }  mycl.delete\_one(myquery) | 从集合中删除指定文档 |
| myquery = { "name": {"$regex": "^j"} }  x = mycl.delete\_many(myquery)  count = x.deleted\_count | 从集合中删除所有name中首字母j开头的文档 |
| x = mycl.delete\_many({})  count = x.deleted\_count | 删除集合中所有文档 |
| 查询  取出的每个文档数据为字典类型 | |
| x = mycl.find\_one() | 查询集合中第一个文档 |
| for x in mycl.find():  print(x) | 查询集合中所有文档 |
| myquery = { "name": "aaa" }  for x in mycl.find(myquery):  print(x) | 查询集合中所有name为aaa的文档 |
| myquery = { "name": { "$regex": "^a"} }  for x in mycl.find(myquery):  print(x) | 查询集合中所有name为a开头的文档 |
| for x in mycl.find().sort("age", -1):  print(x) | 将查询的结果降序排列，参数2不写或为1则为升序 |

### pyqt

#### 安装

##### 模块安装

pip install pyqt5

pip install pyqt5-tools

##### pycharm配置

settings->tools->external tools进行添加

通过配置可以在pycharm中tools->external tools直接调用对应的pyqt程序

=======设置qtdesigner程序打开方式

Name:qtdesigner

Program：pyqt5-tools安装目录中designer.exe的路径

Work directory : $FileDir$

=======设置pyuic

Name: pyuic

Program： pythonEXE路径

Parameters：

-m PyQt5.uic.pyuic $FileName$ -o $FileNameWithoutExtension$.py

Work directory: $FileDir$

=======使用

使用qtdesigner生成\*\*\*.ui

选择该ui文件，执行pyuic生成对应的.py文件

【运行时执行的是：python.exe -m PyQt5.uic.pyuic \*\*.ui -o \*\*.py】

#### ui调用

app = QtWidgets.QApplication(sys.argv)

widget = QtWidgets.QWidget(None)

ui = Ui\_Form()

ui.setupUi(widget)

widget.show()

sys.exit(app.exec\_())

Ui\_Form为ui自动生成py文件中的类

#### 消息响应

**class** MyApp:  
 **def** \_\_init\_\_(self):  
 self.widget = QtWidgets.QWidget(**None**)  
 self.ui = Ui\_Form()  
 self.ui.setupUi(self.widget) *#产生btnObj对象* self.ui.btnObj.clicked.connect(self.fun) *# ui中的btnObj对象* self.widget.show()  
 **def** fun(self):  
 print(**"xxxx"**)  
  
app = QtWidgets.QApplication(sys.argv)  
obj = MyApp()  
sys.exit(app.exec\_())

### Scrapy

#### 安装

Twisted需要vc编译工具，可以使用Twisted的whl进行安装。

pip install scrapy

pip install pypiwin32

#### 创建项目

scrapy startproject 项目名

(Scripts加入path环境变量)

#### 建立爬虫

spiders目录下建立文件

jyl\_spider.py

import scrapy

class BlogSpider(scrapy.Spider):

name = 'jyl' #爬虫类的name属性，用来标识爬虫，该名字在一个项目必须是唯一的。

#allowed\_domains = [] #限制只爬取这个域名下的网页

start\_urls = ['https://blog.scrapinghub.com']

def parse(self, response):

for title in response.css('h2.entry-title'):

yield {'title': title.css('a ::text').extract\_first()}

#### 运行

项目目录（scrapy.cfg同级）下运行

scrapy crawl jyl

文件目录下运行

scrapy runspider jyl\_spider.py -o ret.json

将（yield ）输出ret.json文件[可以是csv/XML]

#### 调试

项目中建立新文件作为启动运行文件，文件内容：

from scrapy.cmdline import execute

#import os

#import sys

#添加当前项目的绝对地址 sys.path.append(os.path.dirname(os.path.abspath(\_\_file\_\_)))

#执行 scrapy 内置的函数方法execute，使用 crawl 爬取并调试，最后一个参数是爬虫名

execute(['scrapy', 'crawl', 'jyl'])

#### 设置

settings.py设置

ROBOTSTXT\_OBEY = False

DEFAULT\_REQUEST\_HEADERS = {

'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36',

'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8',

'Accept-Language': 'en',

}

#### pipelines

======创建piplelines

class JylPipeline(object):

def process\_item(self, item, spider):

name = item['name']

return item #继续传递item，下一个Pipeline可以接收

#raise DropItem("drop %s" % item) #扔掉item，下一个Pipeline不再接收

======settings.py设置加入piplelines

ITEM\_PIPELINES = {

'jyl.pipelines.JylPipeline': 0,

}

0表示优先级数字，数字越小优先级越高

======调用方式1：

item定义：

class JylItem(scrapy.Item):

name = scrapy.Field()

from jyl.items import JylItem

ii = JylItem()

ii['name'] = imgSrc

yield ii

======调用方式2：

yield {"name": imgSrc}

#### 创建请求

爬虫类中，创建新请求，并使用当前类的回调函数

yield scrapy.Request(new\_url,callback=self.parse)

#### response

|  |  |
| --- | --- |
| 属性 | 说明 |
| url (string) | response的 URL |
| status (integer) |  |
| headers (dict) | response头 |
| body (bytes) | 如果是字符串可以用response.text |
| flags (list) |  |
| request (Request object) | Response.request初始属性 |
| xpath | for imgSrc in response.xpath('//img/@src').extract()  遍历所有img标签的src属性 |
| .extract()  获取xpath对象内的所有数据，返回列表对象  .extract\_first()返回列表第一个数据 |

#### shell

scrapy shell "http://www.budejie.com/text/"

打开http://www.budejie.com/text/并进入shell命令

view(response)，打开默认浏览器并进入相应页面

response.css('title')，查看title的css样式

response.css('title::text').extract\_first()，获取标题中内容