目录

[1. 基础 3](#_Toc71483362)

[1.1. 安装 3](#_Toc71483363)

[1.2. 基本 3](#_Toc71483364)

[1.3. 输入输出函数 3](#_Toc71483365)

[1.4. 变量 3](#_Toc71483366)

[1.5. 运算符 7](#_Toc71483367)

[1.6. 流程控制 7](#_Toc71483368)

[1.7. 函数 8](#_Toc71483369)

[1.7.1. 日期函数 8](#_Toc71483370)

[1.7.2. 随机函数 9](#_Toc71483371)

[1.8. 文件操作 10](#_Toc71483372)

[1.9. 异常 10](#_Toc71483373)

[1.10. 包、库、模块 11](#_Toc71483374)

[1.11. 面向对象 12](#_Toc71483375)

[1.11.1. 定义 12](#_Toc71483376)

[1.11.2. 特性 12](#_Toc71483377)

[1.11.3. 类的方法调用 13](#_Toc71483378)

[2. 高级应用 14](#_Toc71483379)

[2.1. 浅拷贝与深拷贝 14](#_Toc71483380)

[2.2. 装饰器 16](#_Toc71483381)

[3. 数据处理 18](#_Toc71483382)

[3.1. Pandas 18](#_Toc71483383)

[3.1.1. 基础 18](#_Toc71483384)

[3.1.2. pandas操作Excel 18](#_Toc71483385)

[3.2. 绘图matplot 23](#_Toc71483386)

[3.2.1. 基础 23](#_Toc71483387)

[3.2.2. 23](#_Toc71483388)

[4. 爬虫 25](#_Toc71483389)

[4.1. 爬虫过程 25](#_Toc71483390)

[4.2. 开发环境 25](#_Toc71483391)

[4.3. 数据解析 26](#_Toc71483392)

[4.3.1. 正则表达式 26](#_Toc71483393)

[5. Vscode 27](#_Toc71483394)

[5.1. 基础 27](#_Toc71483395)

[5.1.1. 安装 27](#_Toc71483396)

[5.2. Git环境搭建 27](#_Toc71483397)

[5.2.1. 安装与申请 27](#_Toc71483398)

[5.2.2. 操作流程 27](#_Toc71483399)

[5.2.3. 常用命令 30](#_Toc71483400)

[5.3. 调试 33](#_Toc71483401)

[5.3.1. 环境搭建 33](#_Toc71483402)

[5.4. 插件 33](#_Toc71483403)

[6. 附录 34](#_Toc71483404)

[6.1. 换jupyter的背景 34](#_Toc71483405)

[6.2. 内置异常的层次结构 34](#_Toc71483406)

[6.3. 库的安装 35](#_Toc71483407)

[6.3.1. 基本操作 35](#_Toc71483408)

[6.3.2. 爬虫库 35](#_Toc71483409)

[7. 代码 37](#_Toc71483410)

[7.1. 37](#_Toc71483411)

# 基础

## 安装

* Python网站，anaconda，pycharm

## 基本

|  |  |  |
| --- | --- | --- |
| 注释 | ‘’，”” |  |
| 代码多行 | \， | 但是{}、[]、()中的换行不需加\ |

## 输入输出函数

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 语义 | 例程 | 备注 |
| input | 变量=input(“str”) | num=input(“请输入一个数”) | num是一个str类型的变量,类型转换 |
| print | print(value,,sep=’’,end=’\n’,file=,flush=False) | print(‘▊’,flush=True) | Flush每次直接刷新 |
| %d 十进制整数 %f 浮点数 %s 字符串 | print(“%d,%.2f” %(v1,v2) | 数字输出 |
| format方法 | print(“xxx{:.2f}” .format(v)) | 字符串输出 |
| f”字符串”方法 | print(f“{v1} xxx{v2}”) |
|  |  |  |  |

## 变量

|  |  |  |
| --- | --- | --- |
| 标识符 | 一般小写，类名首字母大写，下划线开头的类私有函数 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number  （数字）（不可变） | int  float | 0b二进制\0o八进制0x十六进制 | | 进制转换bin()；oct()；hex()；int(v,n) |
| bool | None,0,0.0,0.0+0j,””,[],(),{}都是假 | | Python3 中，把 True 和 False 定义成关键字了，但它们的值还是 1 和 0，它们可以和数字相加 |
| complex（j） | |  |  |
| String  （字符串） |  | |  |  |
| List  （列表） |  | |  |  |
| Tuple  （元组） |  | |  |  |
| Dictionary（字典） |  | |  |  |
| Set  （集合） |  | |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **字符串（不可变）** | | | | | | | | | | | | |
| 表示 | ‘’,””, ’”多行’’’ | | | | |  | | | | | 与注释类似。属于不变数据类型 | |
| 转换 | chr(num)、ord(ch) | | | | |  | | | | | 将整型转换为字符\字符转换为整型 | |
| 转义 | 避免英文中’，采用双引号 | | | | | 或\转移字符 | | | | |  | |
| \n | 换行符，将光标位置移到下一行开头。 | | | | | | | | | | |
| \r | 回车符，将光标位置移到本行开头。 | | | | | | | | | | |
| \t | 水平制表符，也即 Tab 键，一般相当于四个空格。 | | | | | | | | | | |
| \a | 蜂鸣器响铃。注意不是喇叭发声，现在的计算机很多都不带蜂鸣器了，所以响铃不一定有效。 | | | | | | | | | | |
| \b | 退格（Backspace），将光标位置移到前一列。 | | | | | | | | | | |
| \\ | 反斜线，在文件路径表示为D:\\syd\\a\\... | | | | | | | | | | |
| \' | 单引号 | | | | | | | | | | |
| \" | 双引号 | | | | | | | | | | |
| \ | 在字符串行尾的续行符，即一行未完，转到下一行继续写。 | | | | | | | | | | |
| 编码  解码 | str.encode(‘unicode-escape’)  res.decode(‘unicode-escape’)  str.encode(‘utf-8’)  res.decode(‘utf-8’) | | | | | 将字符串str编码为unicode/utf-8  将二进制res解码为字符 | | | | | unicode汉字采用两个字节  utf-8中英文一个字节，中文3字节 | |
| 运算 | 索引、切片 | | | | [i]\[star:end:step] | | | | | | 切片是左闭右开，setp=-1为逆序 | |
| 连接、重复 | | | | +、字符串\*次数 | | | | | |  | |
| 成员 | | | | in、not in | | | | | |  | |
| 串前字母 | | | | u’’表示unicode字符；r"\n\n\n\n”表示一个普通生字符串； b' ' 表示这是一个 bytes 对象；以 f开头表示在字符串内支持大括号内的python 表达式 | | | | | | | |
| 函数 | 查找替换 | | str1.find(str2,beg=0,end=len)  str1.index(str2,beg=0,end=len)  str.replace(old,new[,max]) | | | | | | | | 在str1中找str2，返回位置或-1  但是找不到，报异常  max指定替换次数 | |
| 分割拼接 | | str1.split(sep=””,num)  str.join(seq) | | | | | | 按照sep分割str1,num+1次，返回分割后的字符串  用str连接seq字符串，返回连接后的字符串 | | | |
| 对齐  消除字符 | | str.ljust(width[,fillchar])  str.rjust(width[,fillchar])  str.center(width[,fillchar]) | | | | str.lstrip([chars])  str.rstrip([chars])  str.strip([chars]) | | | 用(空格)左对齐  用(空格)右对齐  用(空格)中间齐 | | 去除开头指定字符  去除结尾指定字符  去除头尾指定字符 |
| 其他 | | str.count(str2,start=0,end=len)  str.capitalize()  str.title()  str.upper()/lower()  str.startswith(prefix,beg=0,end=)  str.endwith(suffix,beg=0,end=) | | | | | 统计str中str2的个数，返回个数  字符串首字母大写  将单词的首字母大写  所有字母大写、小写  检查字符串开头或结尾的特点,返回bool | | | | |
| 其他2 | | | str.isalnum()  str.isalpha()  str.isdigit() | | | | 判断是否为数字、字母，返回bool  判断是否为字母  是否为数字 | | | | |
|  |  | | | | |  | | | | |  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **列表（可变）** | | | | | |
| 定义  产生 | []  list(range(0:3:1))  len(listname) | | 可以用+，合并列表  [0,1,2]  列表的长度 | | 类似于数组，列表的元素可能为不同类型 |
| 操作 | 切片 | list[num],-1时为最后一个元素  list[1:3]取list[1]、list[2] | | | 左闭右开 |
| 追加 | list1.append(num) | | | 在list1后追加num |
| 插入 | list1.insert(station,v) | | | 在list1中的station位置插入v |
| 删除 | del listname[num]  el=list1.pop(num)  list1.remove(“listname”) | | 删除listname中的num位置的元素，这是py的函数  弹出list1的num位置的元素，返回弹出的元素  删除list1中的名字为检索内容的元素 | |
| 排序 | list1.sort(reverse=False)  sorted(listname) | | list1被改变，默认升序  py中的排序函数，list1不变，返回拍好的 | |
|  |  | | |  |
| 推导式 | [expr for expr1 in sq1 if condition1  for expr2 in sq2 if condition2  …] | | | | 等价于循环语句，更为简洁  list1=[i/10 for i in range(11)] |
| 其他 | 统计 | list1.count(elname) | | | 统计list1中elname的元素个数 |
| 极值 | max(list1)/min | | | 返回最大小值 |
| 成员 | el in/not in list1 | | | 返回bool |
|  |  | | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **元组（不可变）** | | | | | |
| 定义  产生 | ()  t1=(元素,) | | 索引：t1[0]访问元素 | | 类似于数组，列表的元素可能为不同类型 |
| 操作 | 切片 | t1[1:3],-1时为最后一个元素  t2=t1[::-1],逆序 | | | 不会改变原来的，形成新的元组 |
| 拼接 | t3=t1+t1 | | |  |
| 元组与列表转换 | l1=list(t1)  t1=tuple(l1) | | 元组转换为列表，可以修改  列表转换为元组，不可修改 | |
|  |  | | |  |
| 推导式 | [expr for expr1 in sq1 if condition1  for expr2 in sq2 if condition2  …] | | | | 等价于循环语句，更为简洁  list1=[i/10 for i in range(11)] |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **集合（可变）** | | | | |
| 定义  产生 | {}  s1={元素,} | | 对重复元素进行筛选 | 无序的、不重复  无法用下表访问 |
| 操作 | 访问 | in/not in | | 判断元素在集合内 |
| 追加 | s1.add(元素) | |  |
| 删除 | s1.remove(元素) | |  |
| 交集  并集 | s3=s1 & s2  s3=s1 | s2 | |  |
| 字符转集合 | s1=set(str1) | |  |
| 推导式 | s1={expr for expr1 in sq1 if condition1  for expr2 in sq2 if condition2  …} | | | 等价于循环语句，更为简洁  s1={i/10 for i in range(11)} |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **字典（可变）** | | | | | |
| 定义  产生 | {键key:值value}  d1={k1:v1,k2:v2,…} | |  | | 可以通过key找到value,查询速度较快 |
| 操作 | 获取值 | d1[k1]  d1.get(k1)、d1.get(k1,默认值) | | 若key不存在，返回异常，需要错误处理  这种方式，不会返回异常，返回none或返回默认值 | |
| 修改 | d1[k1]=new\_v1 | | | 修改值 |
| 增加 | d1[k3]=v3 | | | 增加键值对 |
| 删除  （内存中消失） | del d1[k1]  del d1 | | | py自带的del方法，删除k1-v1  删除整个字典 |
| 清空 | d1.clear()  tem=d1.pop(k1) | | | 删除字典中的元素，d1={}  删除字典中的键值对，返回删除的值 |
| 合并 | d1.update(d2)  d3={\*\*d1,\*\*d2} | | | 把d2追加到d1（相同键会覆盖）  d1和d2合并到d3中 |
| 获取全部键 | list(d1.keys()) | | | 返回d1中键所形成的列表 |
| 获取全部值 | list(d1.values()) | | | 返回d1中值所形成的列表 |
| 获取全部键值 | d1.items() | | | 返回d1中键值所形成的列表，一个键值对是一个元组 |
| 推导式 | {k:v for k in sq1 if condition1  …} | | | | 等价于循环语句，更为简洁  list1=[i/10 for i in range(11)] |

## 运算符

|  |  |  |  |
| --- | --- | --- | --- |
| \*\* | 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，求余数和取整除 |  |  |
| + - | 加法减法 |  |  |
| >> << | 右移，左移运算符 |  |  |
| & | 位 'AND' |  |  |
| ^ | | 位运算符 |  |  |
| <=、< > !=、>=、== | 比较运算符 | 比较的是变量的值 |  |
| = %= /= //= -= += \*= \*\*= | 赋值运算符 | 交换变量的值：x,y=y,x |  |
| is is not | 身份运算符 | 比较变量数值和地址是否同时相同，id取地址 |  |
| In、not in | 成员运算符 | 返回True和False |  |
| not、and、or | 逻辑运算符 | and判定的是前真，后边就是实际数值（不一定是真假）；若前假，直接输出假（真取后假取前）  or是真取前，假取后 |  |

## 流程控制

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 选择 | if condition：  states  else：  states | if condition：  states  elif condition：  states  else：  states | | if condition：  ifcondition：  states |  |
| 循环 | while condition:  states | | |  | 未知次数循环 |
| for v in list：  states | | for i in range(star,end,step):  states | | for可以遍历可迭代对象，用以下方法判定  from collection.abc import Iterable中的isinstance(v,Iterable)  step=-1为降序 |
| break\continue | | ontinue终止当前循环  break 终止整个循环 | |  |
| pass | | 空语句 | |  |
| while/for contion:  states  else: | | | |  |
|  |  | |  | |  |

## 函数

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 定义 | def 函数名(形参列表para):  函数体 | | | | | | 函数名不大写，类首字符大写 | |
| 调用 | 函数名(实参列表arg) | | | | | |  | |
| 函数参数 | 位置参数 | | f(1,3,5) | | | | 1,3,5对应的位置参数 | |
| 关键字参数 | | f(para3=5,para2=3,para1=1) | | | | 灵活的调整参数的位置 | |
| 默认值参数 | | 定义函数时，可以使用默认值参数 | | | | 带默认值参数，关键字参数放最后 | |
| 不定长参数 | | def 函数名(形参列表,\*args,\*\*kwargs): | | | | 未命名的用\*args，以元组  命名了的如key=value形式的用\*\*kwargs以字典 | |
| 返回值 | 无返回值 | | 返回none | | | |  | |
| 有返回值 | | return  yield(re.\_next\_()) | | | 返回数据较少  构造生成器，每次返回一个，处理完成后，再返回一个，结束返回异常 | | |
| 类型 | 无参，无返回 | | | | 无参，有返回 | 有参，无返回 | | 有参，有返回 |
| 变量作用域LEGB原则  (L->E->G->B) |  | | | 本地(Local):函数体内,局部变量 | | | 改变变量的作用域：  global 变量名  (修改局部变量为全局变量)  nonlocal 变量名  (修改嵌套作用的变量) | |
| 函数嵌套(Enclosing) | | |
| 文件(Global):全局变量 | | |
| 项目(Built-in):内置作用域 | | |
| 函数调用 | 嵌套 | 一个函数调用其他函数 | | | | |  | |
| 递归 | 一个函数调用自身 | | | | | 用压栈的方式 | |
| 匿名函数 | lambda | 定义：fa=lambda n:n\*\*2 定义：fa(10) | | | | | n为参数，n\*\*2为返回值 | |
|  |  |  | | | | |  | |

### 日期函数

|  |  |  |  |
| --- | --- | --- | --- |
| 函数类型 | 库 | 函数 | 说明 |
| 时间函数 | import time | time.time() 1970年到现在毫秒数 | * 时间戳—12323 |
| t=time.gmtime()格林尼治时间  t=time.localtime()当前时区时间  t=time.strptime(“字符串”,格式) | * 时间元组(struct\_time)—(tm\_year=2020)   从字符串中提取元组时间信息 |
| strt=strftime(format[,元组])1 | * 格式化字符串—“2020年-8月”，元组转换为字符串 |
| 日历函数 | import calendar | cal=calendar.month(2020.8)  cal=calendar.monthcalendar(2020.8)  cal=calendar.calendar(2020)  calendar.isleap(2020) | 返回字符串，打印月历  返回列表，打印月历  返回字符串，打印全年月历  返回bool，判定润平年 |

注1：format格式：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| %y 两位数的年份表示（00-99）  %Y 四位数的年份表示（000-9999） | %m 月份（01-12） | %d 月内中的一天（0-31）  %j 年内的一天（001-366） | %H 24小时制小时数（0-23）  %I 12小时制小时数（01-12） | %M 分钟数（00-59）  %S 秒（00-59） |
| %a 本地简化星期名称  %A 本地完整星期名称 | %c 本地相应的日期表示和时间表示 | %b 本地简化的月份名称  %B 本地完整的月份名称 | %p 本地A.M.或P.M.的等价符 | %% %号本身 |
| %U 一年中的星期数（00-53）星期天为星期的开始  %W 一年中的星期数（00-53）星期一为星期的开始 | %w 星期（0-6），星期天为星期的开始 | %x 本地相应的日期表示 | %Z 当前时区的名称  %X 本地相应的时间表示 |  |

### 随机函数

|  |  |  |  |
| --- | --- | --- | --- |
| 函数类型 | 库 | 函数 | 说明 |
| 均匀分布 | import random | random.seed(int) | 随机数种子，种子相同，随机数相同，一般用时间戳 |
| re=random.random() | 产生[0-1]之间的伪随机数 |
| re=random.uniform(a,b)  re=random.randint(a,b)  re=random.randrange(a,b,step) | 产生[a b]之间的伪随机浮点数  产生[a b]之间的伪随机整数  产生[a b]之间步长为step的数 |
| 正态分布 | re=random.normalvariate(mu,std)  re=random.gauss(mu,std) | 产生均值为mu，方差为std的正态分布 |
| 随机选择 | random.choice(seq) | 随机选择seq中的一个元素，seq可以为list、tuple、字符串等可迭代序列 |
| 随机打乱 | random.shuffle(x) | 用于将列表中的元素打乱，直接改变列表x |
| 随机抽样 | random.sample(seq,k) | 从seq中选出k个元素，形成新的list |
| 其他 | numpy | 高级随机块中 |  |

## 文件操作

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 函数类型 | 库 | 函数 | | | | 说明 |
| 打开  关闭 | 内置 | f=open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None) | | | | |
| file: 必需，文件路径（相对或者绝对路径）。 | | | | |
| mode: 可选，文件打开模式 | ’t’:文本模式，’b’二进制；  ’r’读，’w’写，原有文件存在会覆盖；’x’写，文件存在会报错‘，a’追加  ’+’打开一个文件进行更新（可读可写）  ‘r+’可读可写，打开时不清空，可写到文件任何位置，默认是开头位置  ‘a+’可读可写，打开时不清空，只写到文件尾 | | | |
| buffering: 设置缓冲 | | encoding: 一般使用utf8 | errors: 报错级别 | newline: 区分换行符 |
| closefd: 传入的file参数类型 | | | | |
| close() | | | | 关闭文件 |
| 读写 | res=f.read() 整个文件读取  res=f.read(num) 读取指定个数字符  res=f.readline(num) 读取单行；res=f.readlines(num) 返回列表  f.write(res) | | | | 打开后，进行读写后，关闭 |
| with open(\*\*\*) as f:  f.read() | | | | with创建上下文环境，不要close() |
|  |  |  | | | |  |

## 异常

|  |  |  |  |
| --- | --- | --- | --- |
| 异常 | Exception | 程序异常概念广泛， |  |
| 错误 | Error | 程序错误会终止程序，引发异常traceback |  |
| 异常类 | BaseException |  | 类名.\_\_base\_\_()  查看基类 |
| 异常  处理 | 捕获和处理 | try:  执行代码  except 异常名称A:  发生异常的执行代码  except 异常名称B as e:  发生异常的执行代码  else:  未捕获异常代码  finally:  不管有没有异常都会执行的代码 | 捕获  处理  print(e.\_\_class\_\_.\_\_name\_\_,e)  不发生异常处理  不管异常与否都会执行 |
| 主动  异常 | 抛出异常 | raise 异常类名字  raise 异常类的实例化对象()  raise 异常类的实例化对象(“提示信息”)  raise | 抛出异常，没有信息  抛出异常，有信息  抛出最近异常  抛出异常的异常 |
| 有条件抛出 | assert 表达式,[异常提示信息参数] | 表达式为True,不会触发异常;  为False，会触发断言异常 |

## 包、库、模块

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 包 | package |  | | 都是由模块构成  没有本质区别 |
| 库 | library |
| 模块 | module | 每一个文件都是模块， |
| 分类 | 自带 | 直接使用 | |  |
| 标准库 | import 库 | |  |
| 第三方库 | 需要安装和import | |  |
| 自己开发 |  | |  |
| 安装 | pip install 库名 -U 更新 pip install 库名==1.2 指定版本安装 pip uninstall 卸载  pip list 查看安装的库 pip show 库名  配置安装镜像：  pip install -i <https://pypi.tuna.tsinghua.edu.cn/simple/> 库名  pip config set global.index-url <https://pypi.tuna.tsinghua.edu.cn/simple/> | | | |
| 模块 | 导入  使用 | import 模块1，模块2，  import 模块名.函数 as 简写名称  from 模块名 import 函数名  注：搜索顺序：当前目录->默认路径 | | 导入模块  导入模块中函数 |
| 包 | 导入 |  | 含有\_\_init\_\_.py文件就是包 | |
| import package\_a.module\_a1 as aa | |
| from package\_a.module\_a1 import 函数名 | |
|  | |
|  |  |  | |  |

## 面向对象

### 定义

|  |  |  |  |
| --- | --- | --- | --- |
| 类 | 抽象概念 | class Name:  类属性  def 方法名(self ,变量名)  self.变量名=..  return … | 类名称，首字符大写  类属性  类方法类似于函数，但是形参第一为self为类对象本身  方法内的属性叫做实例属性 |
| 对象 | 是实体，类的实例 | 实例化对象=类名() | 创建对象，实例化对象 |
| 类的方法 | 构造与析构 | \_\_init\_\_()  \_\_del\_\_() | 初始化方法，类的对象实例化调用的函数，实例化属性  调用del删除对象的时候，默认调用函数 |
| 返回类的相关信息 | \_\_str\_\_() | 调用类的说明，调用print和str函数时会调用此函数 |

### 特性

|  |  |  |  |
| --- | --- | --- | --- |
|  | 定义和作用 | 分类 | 说明 |
| 封装 | 隐藏属性、方法及方法的实现称为封装  通过接口访问公有属性和方法，不能访问私有方法和属性 | 私有属性：\_\_属性名  可以通过公有方法来修改私有属性 | 公有属性可以通过“对象名.属性名”来修改，但是私有不行  但是可以通过如下方法访问：  对象名称.\_类名\_\_私有属性(方法) 来访问 |
| 私有方法：\_\_方法 |
| 继承 | 继承是创建类的一种方法，实现代码复用。分为单继承和多继承  所有类都是object的子类  父类：基类/超类  子类：派生类  子类继承父类：完全继承和部分继承 | 单继承：子类继承父类的公有属性和方法，私有属性和方法无法继承  class 父类名(object):  父类公有属性/方法  父类私有属性/方法  class 子类名(父类名):  子类的属性/方法 | 部分继承方法：   * supper().父类方法(参数) * 父类名称.父类方法(self,参数) |
| 多继承：  class 父类1:  class 父类2:  class 子类(父类1,父类2): | 子类继承过程中，父类多个情况下，当父类中存在相同函数时，从左到右依次调用先出现的父类。 |
| 多态 | 父类的同一个方法在不同子类中的不同的表现和行为  同一个函数名实现不同的函数功能  多态需要条件：   * 继承，多态一定发生在父类和子类之间 * 重写，子类需要重写父类的方法 | 多态：  class 父类(object):  def f1(self):  class 子类1(父类):  def f1(self):  class 子类2(父类):  def f1(self):  定义函数：  def fun(obj):  obj.f1()  使用：o1=子类1()  o1.f1() | 根据不同对象，同一函数名将调用不同的类中的实现函数 |

### 类的方法调用

|  |  |  |
| --- | --- | --- |
| 分类 | 说明 | 用法 |
| 实例化方法  instancemethod | 必须对类进行实例化 | aa=类名(参数)  aa.方法(self,参数) |
| 类方法  classmethod | 类中用@classmethod装饰器装饰的方法   * 不用实例化，直接调用类属性和方法 * 参数不是self，是cls * 可以被子类继承   调用方法：  类名.类方法 类名.类属性  返回的是类属性，不是实例化属性 | class 类名(obj):  类属性  def f1(self):  @classmethod  def f2(cls):  类名.f2(cls,) 直接调用 |
| 静态方法  staticmethod | 类中用@staticmethod装饰器装饰的方法   * 逻辑上实现了类的某种功能，与类有一定关系，但不使用类的其他属性和方法 * 为了便于代码的管理和维护 * 直接用类名进行调用，也可以实例化再调用 * 可以被子类继承   调用方法：  类名.类方法 类名.类属性  返回的是类属性，不是实例化属性 | class 类名(obj):  类属性  def f1(self):  @staticmethod  def f2(cls):   * 实例化调用：   aa=类名(参数)  aa.f2()   * 直接调用   类名.f2() |
|  |  |  |

实例：

class Dog(object):

def \_\_init\_\_(self,color,name):

self.color=color

self.name=name

def bark(self,adv):

self.adv=adv

return f"{self.color}{self.name}{self.adv}"

def \_\_del\_\_(self):

print(f"{self.color}{self.name}删除了")

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

return f"{self.color}{self.name}的信息"

aa=Dog("黄色的","小狗")

aa.bark("叫")

del aa

# 高级应用

## 浅拷贝与深拷贝

|  |  |  |
| --- | --- | --- |
| 赋值 | 就是对象引用的传递（拷贝对象的引用），并没有新开辟内存 | a=b,b变量仅是引用了a变量的地址 |
| 浅拷贝 | 变量外层的拷贝，而变量内部均为对象的引用 | copy.copy浅拷贝  copy.deepcopy深拷贝 |
| 深拷贝 | 对内外所有层的拷贝 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 定义 | 外层 | 内层 | 说明 |
| 浅拷贝 | 变量外层的拷贝，而变量内部均为对象的引用 | 可变数据（列表、字典） | 对象的引用（赋值） | 开辟新内存 |
| 不可变数据（元组、字符串） | 对象的引用 |
| 变量的嵌套与非嵌套：非嵌套[1,3]；嵌套[[1,2],(3,4)]   1. 外层数据类型为可变的（列表）：新变量为新开辟地址Ex1 2. 外层数据类型为不可变的（元组、字符串）：此时为对象引用（赋值）Ex2 3. 嵌套结构外层为可变的，内层也为可变：外层为新开辟的内存地址，内层为对原有数据的引用Ex3 4. 嵌套结构外层为不可变的，内层也为不可变：外层、内层均为为引用（赋值）Ex4 | | | |
| 深拷贝 | 对内外所有层的拷贝 | 可变 | 可变 | 内外层均开辟内存 |
| 可变 | 不可变 | 外层开辟地址，内层赋值 |
| 不可变 | 可变 | 内外层均开辟内存 |
| 不可变 | 不可变 | 内外层转换为赋值（引用） |
| 1. 外层为可变数据，内层也为可变数据：则内外均开辟新地址Ex4 2. 外层为可变数据，内层也为不可变数据：则外均开辟新地址，内层赋值（退化为浅拷贝）Ex5 3. 外层为不可变数据，内层也为可变数据：则内外均开辟新地址Ex6 4. 外层为不可变数据，内层也为不可变数据：则内外均赋值Ex7 | | | |
| 应用场景 | * 默认为浅拷贝 * 浅拷贝效率高， * 深拷贝备份数据，安全可靠 | | | |

Ex1：

import copy

a=[1,3]

b=copy.copy(a)#b为新开辟的地址空间

b.append(“OK”)#b中新追加元素，而a不变

Ex2：

import copy

a=(1,3)

b=copy.copy(a)#b是对a的引用（赋值），地址相同

Ex3：

import copy

a=[1,2]

b=[3,4]

c=[a,b]

d=copy.copy(c)

print(f"{id(c)}")

print(f"{id(d)}")#d为新开辟的地址

print(f"{id(a)}")#但是d中的两个元素的地址均为对a、b的引用

print(f"{id(d[0])}")

Ex4：

import copy

a=(1,2)

b=(3,4)

c=(a,b)

d=copy.copy(c)#d是对c的引用，地址相同，且c、d中的两个原始的地址与a、b相同

print(c)

print(d)

print(f"{id(c)}")# 1995709567880

print(f"{id(d)}") #1995709567880

print(f"{id(a)}") #1995709570312

print(f"{id(d[0])}") #1995709570312

Ex4：

import copy

a=[1,2]

b=[3,4]

c=[a,b]

d=copy.deepcopy(c)#d为新开辟内存地址，同时d中的元素也是新开辟的内存地址

print(c)

print(d)

print(f"{id(c)}")# 1995719323144

print(f"{id(d)}")# 1995718535816

print(f"{id(a)}")# 1995719320648

print(f"{id(d[0])}")# 1995718533448

Ex5：

import copy

a=(1,2)

b=(3,4)

c=[a,b]

d=copy.copy(c)#d为开辟新的地址，但是d中元素为a、b的引用

print(c)

print(d)

print(f"{id(c)}")# 1995719323144

print(f"{id(d)}")# 1995718535816

print(f"{id(a)}")# 1995709571848

print(f"{id(d[0])}")# 1995709571848

Ex6：

import copy

a=[1,2]

b=[3,4]

c=(a,b)

d=copy.deepcopy(c)#d的内外均开辟新的地址

print(c)

print(d)

print(f"{id(c)}")# 1995719323144

print(f"{id(d)}")# 1995718535816

print(f"{id(a)}")# 1995718536072

print(f"{id(d[0])}")# 1995719320136

Ex7：

import copy

a=(1,2)

b=(3,4)

c=(a,b)

d=copy.deepcopy(c)#d的内外层均为引用

print(c)

print(d)

print(f"{id(c)}")# 1995709568520

print(f"{id(d)}")# 1995709568520

print(f"{id(a)}")# 1995709570312

print(f"{id(d[0])}")# 1995709570312

## 装饰器

通过 @property 装饰器，将类实例化方法作为一种特殊的实例化属性使用。可以直接通过方法名来访问方法，不需要在方法名后添加一对“（）”小括号

* 对于类的某个实例化属性number，若需要保护这个实例化属性只能查询不能随意修改和删除，需要定义一个和实例化属性同名的实例化方法，并用@property来修饰
* 若修改或删除该实例化属性，单独编写@ [number.setter和@number.deleter装饰器装饰的number](mailto:number.setter和@number.deleter装饰器装饰的number)方法使用，通过实例化方法进行@property 装饰，就可以像普通实例化属性一样使用，即对象名.属性名

class People(object):

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

# 私有属性

self.\_\_number = 0

# 获取私有属性值 number = p1.number 会执行这个函数

@property

def number(self):

# 返回私有属性值

return self.\_\_number

# 设置私有属性值 p1.number = 666

@number.setter

def number(self, value):

# 设置\_\_number的值

self.\_\_number = value

# 删除私有属性 del p1.number 会执行这个函数

@number.deleter

def number(self):

# 删除属性

del self.\_\_number

p1 = People('张三', 22)

# 正常的对象属性赋值

# 对象.属性名 = 属性值

p1.name = '李四'

# 获取对象的属性值

name = p1.name

# 删除对象的属性

del p1.name

# 私有属性升级版

# 会去执行@property装饰number函数，函数执行完成后返回一个结果

num = p1.number

print(num)

# 会去执行@number.setter装饰的number函数，在函数中设置\_\_number属性的值

p1.number = 666

# 会去执行@property装饰number函数，函数执行完成后返回一个结果

print(p1.number)

# 会去执行@number.deleter装饰的number函数，在函数中会将\_\_number属性删除

del p1.number

# 会去执行@property装饰number函数，函数执行完成后返回一个结果

print(p1.number)

# 数据处理

## Pandas

### 基础

1. 配置环境

* pandas依赖处理Excel的xlrd模块，安装命令是pip install xlrd
* 安装pandas模块还需要一定的编码环境，确保你的电脑有这些环境：Net.4 、VC-Compiler以及winsdk\_web。
* 开始安装pandas，安装命令是pip install pandas
* 导入库import pandas as pd

### pandas操作Excel

|  |  |  |
| --- | --- | --- |
| 创建文件 | df=pd.DataFramen({‘ID’:[1,2,3…],’Name’:[n1,n2,n3,…]})  df=df.set\_index[‘ID’]  df.toexcel(‘文件路径’) | 创建帧（相当于sheet）  设置ID列为索引列，不设置会默认增加一列  创建excel文件 |
| 读取文件 | df = pd.read\_excel("C:\\文件路径[\\文件名.xlsx](file:///\\文件名.xlsx)")  df = pd.read\_excel("C:\\文件路径[\\文件名.xlsx](file:///\\文件名.xlsx)",header=0)  df = pd.read\_excel("[文件名.xlsx](file:///\\文件名.xlsx)",header=None)  df.columns=[‘ID’,’type’,….]  df = pd.read\_excel("文件名.xlsx",skiprows=3,usecols=’C:F’)  data = df.head(行号) df.tail(行号) | #直接默认读取到Excel的第一个表单  指定表头从何处开始读取数据（例如有些表头有乱数据）  没有标题行的数据表的读取  给没有标题行的数据增加标题  指定区域读取数据（跳过3个空行，读取C到F列的数据）  # 默认读取前5行的数据（相当于预览下数据），末尾数据 |
| df = pd.read\_excel("文件名.xlsx ", sheet\_name=' ') | 表单名的方式读取 |
| df = pd.read\_excel('文件名.xlsx ', sheet\_name=[1, 2]) | 表单索引来指定要访问的表单，0表示第一个表单 |
| df = pd.read\_csv('文件名.csv ',index\_col=’’) | 读取csv和txt文件 |
|  |  |
| 操作行列 | s1=pd.Series(dict)  l1=[1,2,3] l2=[‘k1’,’k2’,’k3’] s1=pd.Series(l1,index=l2)  s1=pd.Series([1,2,3],index=[1,2,3],name=’A’)  s2=pd.Series([10,20,30],index=[1,2,3],name=’B’)  df=pd.DataFrame({s1.name:s1,s2.name:s2})  df=pd.DataFrame(s1,s2) | 生成序列，利用字典生产序列  单独利用键值和数据生产序列  生成DataFrame  把s1、s2中的数据以列的形式并且列名分别为A\B做为sheet  以list的形式时，将每个元素作为行数据加如sheet，A\B作为行号 |
| df.shape  data = df.ix[0].values  data = df.ix[[1, 2]].values  data = df.ix[[1, 2], ['序号', '功能划分']].values  data = df.ix[:, ['序号', '功能划分']].values | 整个表的数据大小，行数，列数  **读取指定的单行**，数据会存在列表里面（0表示第一行 这里读取数据并不包含表头）  **读取指定多行**，在ix[]里面嵌套列表指定行数  **读取指定的多行多列值**：读取第一行第二行的序号以及功能  划分列的值  **获取所有行的指定列数据** |
| df.columns  df.index.values  df.sample(3).values  df['功能划分'].values | **获取列名**  **获取行号**  **获取指定行数的值**  **获取指定列的值** |
| **列的操作**例11 | |
| * df=pd.concat([df1,df2],axis=0).reset\_index(drop=True) * df['列名']=列数据 | * **增加列:**两个表合并，和行合并类似，但是可以同axis指定合并方向 |
| df.drop(columns=['列名'],inplace=True) | **删除列：** |
| df.insert(0,column='ID',value=range(1,len(df)+1)) | **插入列：** |
| df.rename(columns={'listprice':'ListPrice'},inplace=True) | **修改列名：** |
| df[‘列名’]= df[‘列名’].astype(int) | **列名属性修改** |
| df.dropna(inplace=True) | **删除无效数据:**扫描每一行中所有列的数据是否未na，若为哪，直接删除na行的数据 |
| **行的操作**例10 | |
| * df=df.append(col\_mean,ignore\_index=True)   col\_mean=df[['T1','T2','T3','sum','ave']].mean()  ignore\_index=True，表示忽略index   * df=df1.append(df2)   df=df1.append(df2).reset\_index(drop=True)   * newline=pd.Series[{‘列名1’,’列名2’,’’…}]   df=df.append(newline,ignore\_index=True) | * **增加行**，计算行数据可以表示为 * *把df2添加到df1*后，但是两个df都有index，而且还用重新生成新的index，因此为了消除原有的index，需要增加函数reset\_index * *手动增加新行数据*   Index不会自动生成，所有要忽略，否则报错。 |
| df.drop\_duplicates(subset=列名,inplace=True,keep='first')  dup=df.duplicated(subset='列名')#返回的是bool序列重复的为True  dup=df.duplicated(subset='ID')#返回的是bool的序列，  dup=dup[dup==True]#dup存贮重复的项  dupID=df.iloc[dup.index]#存贮重复项 | **重复数据处理**(对列名中的重复项就行筛选，inplace是否对当前df进行覆盖，keep表示保留的重复位置)  查找重复项 |
| * df.at[index,'列号']=数据 * df.iloc[35]=pd.Series({'ID':25,'discount':0.5,'listprice':1}) | **修改某行数据：***直接修改单元格数据*  *行替换*： |
| part1=df[:10]  part2=df[10:]  df=part1.append(newline1,ignore\_index=True).append(part2)/  .reset\_index(drop=True) | **插入行：**先把要插入的位置分成两部分，完成后再append，不要忘记reset\_index |
| * df.drop(index=df[0:3].index,inplace=True) * noname=df.loc[df['name']=='']   df.drop(index=noname.index,inplace=True)  df=df.reset\_index(drop=True) | **删除行数据：**用index删除  根据条件删除，删除name列的空行，然后用drop找到对应的index删除，同时reset\_index |
| **行列互换：**  Df2=df.transpose() |  |
| df.at[index]  df.at[行号,列号]=11 | 利用at函数实现自动填充  直接对单元格操作 |
| 计算 | df[列名]=df[列名]运算符df[列名]  df[列名].apply(lambda 表达式) | 类似与excel中的函数 例子1，**是以列(Series)为单位进行操作** |
| 排序 | df.sort\_values(by='列名',inplace=True,ascending=Ture)  df.sort\_values(by=['列名1',’列名2’],inplace=True,ascending=  [Ture,False]) | 对指定列名排序，例子2  inplace为‘真’覆盖原表，否则生成新表  ascending表示排序方式 |
| 筛选 | data=df.loc[df['listprice'].apply(lambda a:90>=a>20)]  data=df.loc[df.列名.apply(lambda a:90>=a>20)]  data=df.loc[df.列名.apply(lambda a:90>=a>20)].loc[…] | Loc[df[‘列名’]]定位到数据某列，然后用apply(函数)，对这列进行筛选。例3  进行多重筛选 |
| 统计分析 | 相关性：df.corr() | df中任意两列的相关性 |
| Excel高级 | **多sheet合并**：  df=df1.merge(df2,how='outer',on='ID').fillna(0)  df=df1.merge(df2,how='outer',left\_on=df1.ID,right\_on=df2.ID).fillna(0)  **单元数据校验：**  df.apply(listprice\_validation,axis=1)  **单元格内拆分数据：**  data=df.列名.str.split(n=2,expand=True)  **统计分析：**  sum(axis=1)/mean() | **合并两个sheet**，用“ID”列为索引进行合并，how是指定左右表格的合并的方式，以df1为准（outer表示最大）例4  apply函数，一个是函数名作为参数，axis=1表示按照行进行搜索(axis=0按照列)  对‘列名’中的指定列中的数据进行字符串拆分，拆分为n=2个，同时扩展成两列  求和（默认是按照列；axis=1表示按照行）/求平均 |
| **透视表：**  df.pivot\_table(index=’’,columns=’’,values=’’,aggfunc=np.sum) |  |
| **线性回归**：  from scipy.stats import linregress  df=pd.read\_excel('文件名')  slop,inter,r,p,std\_err=linregress(df.index,df….)  plt.scatter(df.index,df…)  plt.plot(df.index,df.index\*slop+inter) |  |
| 条件格式化： | 例9 |
|  |  |
|  |  |  |



例1：

import pandas as pd

df=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx',index\_col='ID')

print(df)

#对列操作

df['price']=df['listprice']\*df['discount']

#和上述语句一样功能，但是是对单元格操作（但是可以局部运算）

for i in df.index:

df['price'].at[i]=df['listprice'].at[i]\*df['discount'].at[i]

print(df)

#用apply函数完成更多的函数操作，可以用lambda表达式

df['price']= df['listprice'].apply(lambda x:x+2)

例2：

df=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx',index\_col='ID')

print(df)

df.sort\_values(by='listprice',inplace=True)

例3：

df=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx',index\_col='ID')

print(df)

data=df.loc[df['listprice'].apply(lambda a:90>=a>20)]

print(data)

例4：

import pandas as pd

df1=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx',sheet\_name='Sheet1')

df2=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx',sheet\_name='Sheet2')

print(df1)

print(df2)

df=df1.merge(df2,how=’left’,on='ID')# df=df1.merge(df2,how='outer',on='ID').fillna(0)

print(df)

例5：

import pandas as pd

def listprice\_validation(row):

if not 0<=row.listprice<100:

print(f'{ row.ID } has an invalid data {row.listprice}')

df=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx')

df.apply(listprice\_validation,axis=1)

例6：

import pandas as pd

df=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx',sheet\_name='Sheet2')

data=df.name.str.split(n=2,expand=True)

df['name1']=data[0]

df['name2']=data[1]

print(df)

例7：

import pandas as pd

df=pd.read\_excel('D:/Source Code/py/数据处理/temp1.xlsx')

score=df[['T1','T2','T3']]

row\_sum=score.sum(axis=1)

row\_mean=score.mean(axis=1)

df['sum']=row\_sum#增加新列

df['ave']=row\_mean#增加新列

col\_mean=df[['T1','T2','T3','sum','ave']].mean()

col\_mean['ID']='Summary'

df=df.append(col\_mean,ignore\_index=True)#追加行

print(df)

例8：

import pandas as pd

df=pd.read\_excel('D:/Source Code/py/数据处理/temp1.xlsx')

print(df)

df.drop\_duplicates(subset='ID',inplace=True,keep='first')

print(df)

dup=df.duplicated(subset='ID')#返回的是bool的序列，重复项为True，不重复为False

dup=dup[dup==True]#dup存贮重复的项

dupID=df.iloc[dup.index]

例9：（未调通）

import pandas as pd

def low\_red(s):

color='red' if s<100 else 'black'

return f'color:{color}'

def high\_green(col):

return{'background-color:lime'if s==col.max()else 'background-color:white'for s in col}

df=pd.read\_excel('D:/Source Code/py/数据处理/temp1.xlsx')

df.style.applymap(low\_red,subset=['T1','T2','T3']).apply(high\_green,subset=['T1','T2','T3'])

例10：

import pandas as pd

df1=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx',sheet\_name='Sheet1')

df2=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx',sheet\_name='Sheet2')

df=df1.append(df2).reset\_index(drop=True)

newline=pd.Series({'ID':36,'discount':0.444,'listprice':13334})

df=df.append(newline,ignore\_index=True)

#修改单元格

df.at[35,'ID']=25

#替换行

modline=pd.Series({'ID':25,'discount':0.5,'listprice':1})

df.iloc[35]=modline

#插入id=10的位置（index=9位置插入）

newline1=pd.Series({'ID':101,'discount':0.1,'listprice':13})

part1=df[:10]

part2=df[10:]

df=part1.append(newline1,ignore\_index=True).append(part2).reset\_index(drop=True)

# print(df)

#删除行数据

#df.drop(index=df[0:3].index,inplace=True)

for i in range(2,10):

df['name'].at[i]=''

print(df)

noname=df.loc[df['name']=='']

print(noname)

df.drop(index=noname.index,inplace=True)

print(df)

df=df.reset\_index(drop=True)

print(df)

例11：

## 绘图matplot

### 基础

导入import matplotlib.pyplot as plt

### 

|  |  |  |  |
| --- | --- | --- | --- |
| 柱状图 | df.plot.bar(x='listprice',y='discount',color='red',title=’名字’)  plt.tight\_layout()#  plt.bar(df.listprice,df.discount)  plt.xticks(df.listprice,rotation=90)  df.plot.bar(x='ID',y=['discount',‘price’],color=[‘orange’,'red'])  ax=plt.gca()  ax.set\_xticklabels(df[‘列名’],rotation=45,ha=’right’)  f=plt.gcf()  f.subplots\_adjust(left=,bottom=) | **基本柱状图**：用pandas调用matplot指定x，y和颜色  紧凑显示，显示完整坐标  直接用matplot，下xlabel,ylabel,titile  **分组柱状图**  对新坐标调整  用列名填充x坐标，并旋转45°，右对齐  调整图形位置 例1 |  |
| df.plot.bar(x='ID',y=['price','listprice'],color=['red','orange'],stacked=True) df.plot.barh | **叠加柱状图**：纵向  横向 |  |
| 饼图 | df['列名'].plot.pie(fontsize=8)  df['列名'].sort\_values(ascending=True).plot.pie(fontsize=8,startangle=-100) | 以某一列为比例，每块的标识为index的列  设置饼图的旋转方向，即利用排序 例2 |  |
| 折线  叠加图 | df.plot(y='listprice')  df.plot(y=['listprice','price'])  df.plot.area(y=['listprice','price']) | 折线图，x轴为index序列，y为指定列  多条折线  叠加区域图 例3 |  |
| 散点图  直方图  密度图 | df.plot.scatter(x='ID',y='listprice')  df.列名.plot.hist(bins=100)  df.price.plot.kde() | 以x，y为轴的散点图  直方图，bins指定每个直方的宽度  概率密度图 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

例1：

import pandas as pd

import matplotlib.pyplot as plt

df=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx’)

print(df)

# df.sort\_values(by='listprice',inplace=True,ascending=False)

df.plot.bar(x='listprice',y='discount',color='red')

plt.tight\_layout()

plt.show()

例2：

import pandas as pd

import matplotlib.pyplot as plt

df=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx',index\_col='price')

print(df)

df['listprice'].sort\_values(ascending=True).plot.pie(fontsize=8)

# df['listprice'].plot.pie(fontsize=8,counterclock=False)

plt.show

例3：

import pandas as pd

import matplotlib.pyplot as plt

df=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx')

print(df)

df.plot(y='listprice')

例4

import pandas as pd

import matplotlib.pyplot as plt

df=pd.read\_excel('D:/Source Code/py/数据处理/temp.xlsx')

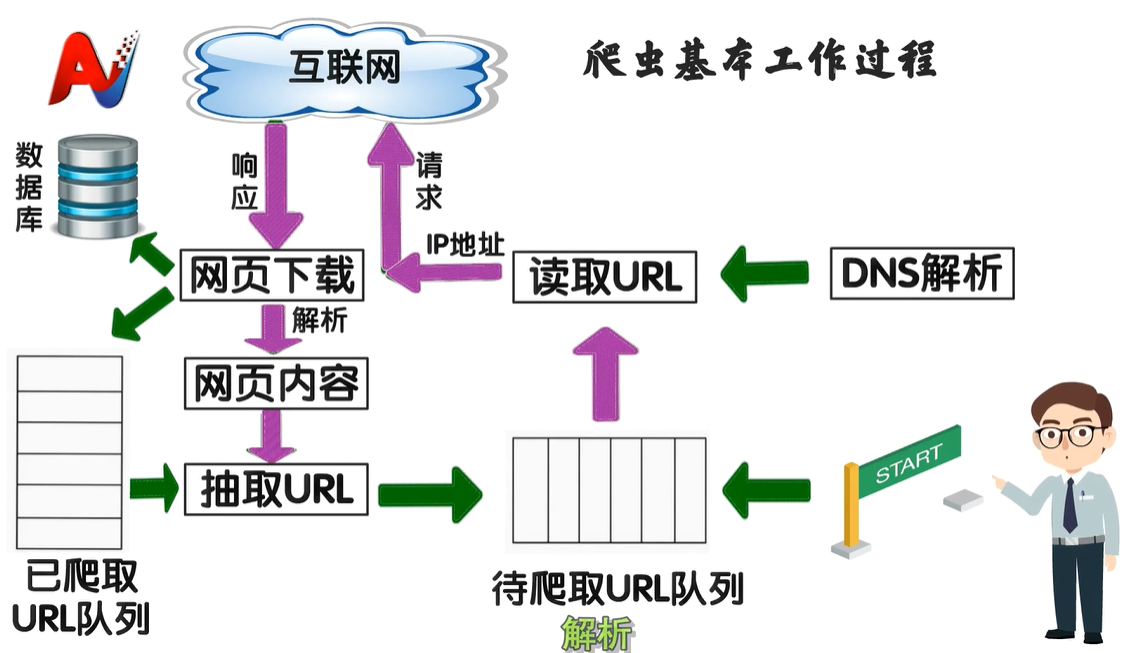
# print(df)

df.plot.scatter(x='ID',y='listprice')

plt.show()

# 爬虫

## 爬虫过程



* 向目标服务器发出请求
* 获取服务器的响应（源代码，json个数数据）
* 解析网页（json数据），提取信息

Ex：

Import requests

import requests

import os

#请求网站主页

# url="http://www.baidu.com"

# response=requests.get(url=url)

# response.encoding="utf-8"

# print(response.encoding)

# print(response.text)

url="http://www.qq.com"

response=requests.get(url=url)

response.encoding="GB2312"

print(response.status\_code)#返回的状态代码

print(response.encoding)#编码格式

os.makedirs("./data",exist\_ok=True)

with open("./data/腾讯主页.txt","w",encoding='GB2312') as f:

f.write(response.text)

## 开发环境

VScode+Ana+爬虫库+chrome（对应版本）+chrome的驱动程序（<http://npm.taobao.org/mirrors/chromedriver/> 找到对用chrome版本的驱动下载）

## 数据解析

### 正则表达式

import re

res=re.findall(pattern=pattern,string=str1) #pattern匹配的模式、待查找字符串特征，string待处理的字符串，res返回的结果

|  |  |  |
| --- | --- | --- |
| **元字符** | | |
| ^ | 匹配字符串开头 | ^abc表示从要查找的字符串开头查找，没有这个符号从任意位置查找 |
| $ | 匹配字符串结尾 | abc$表示从要查找的字符串结尾查找，没有这个符号从任意位置查找 |
| . | 匹配除换行符以外的任意字符 | a.c表示匹配abc、a\*c、a%c…. |
| \* | 匹配前一个字符或表达式0次或多次 | abc\*表示ab、abc、abcc、abccc….. |
| + | 匹配前一个字符或表达式1次或多次 | abc+表示abc、abcc、abccc….. |
| ? | 匹配前一个字符或表达式0次或1次 | abc?表示ab、abc |
| {n} | 匹配前一个字符或表达式n次 | ab{2}c表示abbc |
| {m,n} | 匹配前一个字符或表达式m-n次 | ab{1,3}c表示abc、abbc、abbbc |
| \ | 转义字符，后面的字符转义成为普通字符 | \.中的.就是不是匹配的字符了，就是本身的点的含义。如a\.c就是匹配a.c本身 |
| [] | 字符集合，匹配[]中的任意一个字符 | a[bcd]e表示abe、ace、ade |
| - | 表示范围 | [a-z]表示所有小写字母；[^A-Z]表示除大写字母外的所有字符 |
| () | 精确分组匹配 | (abc){2}表示abcabc |
| | | 逻辑或 | a(b|c)d表示abd、acd |
| \d  \D | 匹配一个数字相当于[0-9]  匹配非数字相当于[^0-9] | a\dc表示a0c、a1c….  a\Dc表示aac、abc… |
| \w  \W | 匹配数字、字母、下划线中任意一个字符  [a-zA-Z0-9\_]  匹配非上述的任意字符[^a-zA-Z0-9\_] | 提取邮箱地址比较有利  \w+@\w+\.\w+ |
| (.\*) | 贪婪模式 | 尽量多的选择字符 |
| (.\*?) | 非贪婪模式 | 严格按照匹配字符进行选择，尽量少的选取 |

[例子](#_代码)

import re

str1="密码是：1212323qq，2密码是：sdlfjslf，3密码是：23ho，"

#贪婪模式

res=re.findall("密码是：(.\*)，",str1)

print(res)# ['1212323qq，2密码是：sdlfjslf，3密码是：23ho']

#非贪婪模式

res1=re.findall("密码是：(.\*?)，",str1)

print(res1)# ['1212323qq', 'sdlfjslf', '23ho']

# Vscode

## 基础

### 安装

下载地址<https://code.visualstudio.com/>

## 调试

### 环境搭建

* 语言的开发环境
* 安装相应语言扩展包
* 配置launch.Json

1. 配置launch.Json

* 指定语言环境
* 指定调试类型
* 其他

注：大部分情况下，VS会自动配置，少部分需要手动配置

手动配置举例：

|  |  |  |  |
| --- | --- | --- | --- |
| 在项目下点调试中有 | 会在项目中增加lanch.json文件就是配置文件 | "version": "0.2.0",  "configurations": [  {  "name": "Attach to Chrome",  "port": 9222,  "request": "attach",  "type": "pwa-chrome",  "webRoot": "${workspaceFolder}"  },  {  "name": "Python: 当前文件",  "type": "python",  "request": "launch",  "program": "${file}",  "console": "integratedTerminal"}]} | |
|  |  |
| Type:指定编程环境  Name:配置的名字  Request：指定调试模式（launch,attach） | | launch | attach |
| Vs负责启动程序并给出现搭载一个调试器 | 为一个已经运行的暂不支持的调试的程序加一个调试器（一般是web程序）  需要两个：一个是已经启动的程序，另一个是attach程序 |

1. 调试技巧

|  |  |  |
| --- | --- | --- |
| 变量搜索 | 点击local，输入要搜索的变量名 |  |
| logpoit | 与breakpoit不同，不会阻塞程序，执行到此处可以设置输出程序类似与print的形式 |  |
| conditional breakpoint | 有条件的断点，当条件满足的时候，才会触发断点阻断程序 |  |

### 多语言调试

可以实现多种语言的混合调试，例如前端和后端的同时调试。过程类似上述方式，需要首先安装语言各自的调试环境，其次是VScode支持包，最后配置launchjson加入两种语言。

多语言调试包含两种方式：**一个项目中实现多个调试进程**

* 依次启动调试项(在调试启动项中依次启动)
* 使用compounds

Launchjson中加入compouds项，并修改属性name和configuration中加入[“语言A name”,”语言B name”]

## VScode的扩展extend

包括三类：

* 写代码的格式
* 工具
* 自定义

## Git环境搭建

### 安装与申请

* 安装Git：<https://git-scm.com/downloads>，点击下载，下载后安装
* 基础配置：
* 在git-bash 中 git config --global user.name "hotplane"

git config --global user.email "hotplanell@sina.com.cn"

(h798262Lg hotplane github.com)

* 生成ssh公钥和私钥

检查当前目录下是否存在：ls -ssh798262

获得公钥：ssh-keygen -t rsa -C [hotplanell@sina.com.cn](mailto:hotplanell@sina.com.cn)

ssh-keygen -t rsa -b 4096 -C "hotplanell@sina.com.cn"

添加私钥：eval $(ssh-agent -s)

ssh-add ~/.ssh/id\_rsa

在目录C:\Users\hot\.ssh下有公钥（id\_rsa.pub）和私钥（id\_rsa）

* 将公钥添加到github

登录github，新建ssh，<https://github.com/settings/ssh/new>

设置title，拷贝公钥文本到key

* 测试连接：ssh -T [git@github.com](mailto:git@github.com)

提示：Hi hotplane! You've successfully authenticated, but GitHub does not provide shell access.表示连接成功。

### 操作流程

常用两个方式建立项目，一种是git命令，一种是Vscode方式

|  |  |  |
| --- | --- | --- |
| git命令 | | |
| 建立项目文件夹 | 并进入此文件下 |  |
| git初始化 | git init | 提示：Initialized empty Git repository in D:/Source Code/VS/git/.git/ |
| 建立文件 | touch 文件名  git statues | 显示当前文件状态（untracked ，未跟踪） |
| 加入暂存区 | git add 文件名  git statues -s | 显示A readme.MD （A表示已经到暂存区，但是还未提交） |
| 提交文件 | git commit -m ‘init’  git status | 显示create mode 100644 readme.MD  显示On branch master |
| 撤销修改文件 | git restore 文件名 | 撤销对文件的修改 |
| 快速提交 | git commit -am '文件名' | 同时完成暂存区和提交两步操作 |
| 查看分支 | git branch <-l> | 显示\*master，\*表示当前分支 |
| 创建并切换分支 | git checkout -b 分支名  git checkout 分支名 | 显示Switched to a new branch 'feature/installation  切换分支 |
| 在不同分支下修改文件  合并两个分支  处理冲突 | git commit -am '文件名'  在一个分支下，  用git merge 另一个分支  手动修改文件，以使不冲突：  在master下修改完，  git add 文件名  git commit -m 'merge form feature/installation' | 在不同分支下修改文件，得到的内容不同。例如在master分支下修改，和在installation修改的同一个文件，在不同分支下显示的内容不同。分别在两个分支下提交。  可能会出现冲突  假设修改完master下的文件后，同步到installation下， |
| 临时保存工作分支 | git stash  git stash pop | 当在某个分支下工作，但没有完成时，临时要保存改分支，且并没有完成该分支工作（commit不完整）。不会显示modified状态了  恢复临时stash分支 |
| 修改记录 | git diff 文件名 |  |
| 从远方下载到本地 | git clone 远方ssh地址 | |
| 上传 | git add .  git commit -m 'init'  git push |  |
| 远程修改更新本地 | git pull |  |

|  |  |  |
| --- | --- | --- |
| VsCode | | |
| 建立项目文件夹 |  |  |
| 加入项目 | 在VsCode中Source Control |  |
| 新建文件 | 文件管理“+” | 新建文件提示，表示（untracked ，未跟踪） |
| 加入暂存区 | 点击要加入的文件，在Source Control中点 |  |
| 提交文件 | 在Source Control中消息框中输入“init”  点击上方的或ctrl+enter | 提交后，文件消失 |
| 撤销修改文件 | 在Source Control中选择 | |
| 快速提交 | 在Source Control中消息框中输入“update”，点击上方的或ctrl+ente | |
| 查看分支 | 点击即可查看 |  |
| 创建并切换分支 | 点击后， 填分支名 |  |
| 在不同分支下修改文件  合并两个分支 | 在管理中选“命令面板”，输入git merge  当有冲突时后，会选择以哪个为准 | |
|  | 选择，恢复时选择弹出存储 |  |
| 修改记录 | 自动标记 |  |
| 从远方下载到本地  git clone 克隆 | 命令面板输入git clone 粘贴ssh地址 |  |
| 上传-推送  git push | 暂存->提交init->push”推送” |  |
| 远程修改更新本地  git pull 拉取 | 拉取pull， | Vscode中还有“同步” |

### 常用命令

|  |  |  |
| --- | --- | --- |
| **配置** | | |
| 1. 仓库级的配置文件：在仓库的 .git/.gitconfig，该配置文件只对所在的仓库有效。  2. 全局配置文件：Mac 系统在 ~/.gitconfig，Windows 系统在 C:\Users\<用户名>\.gitconfig。  3. 系统级的配置文件：在 Git 的安装目录下（Mac 系统下安装目录在 /usr/local/git）的 etc 文件夹中的 gitconfig。 | | |
| # 查看配置信息  # --local：仓库级，--global：全局级，--system：系统级  $ git config <--local | --global | --system> -l  # 查看当前生效的配置信息  $ git config -l  # 编辑配置文件  # --local：仓库级，--global：全局级，--system：系统级  $ git config <--local | --global | --system> -e  # 添加配置项  # --local：仓库级，--global：全局级，--system：系统级  $ git config <--local | --global | --system> --add <name> <value>  # 获取配置项  $ git config <--local | --global | --system> --get <name>  # 删除配置项  $ git config <--local | --global | --system> --unset <name>  # 配置提交记录中的用户信息  $ git config --global user.name <用户名>  $ git config --global user.email <邮箱地址>  # 更改Git缓存区的大小  # 如果提交的内容较大，默认缓存较小，提交会失败  # 缓存大小单位：B，例如：524288000（500MB）  $ git config --global http.postBuffer <缓存大小>  # 调用 git status/git diff 命令时以高亮或彩色方式显示改动状态  $ git config --global color.ui true  # 配置可以缓存密码，默认缓存时间15分钟  $ git config --global credential.helper cache  # 配置密码的缓存时间  # 缓存时间单位：秒  $ git config --global credential.helper 'cache --timeout=<缓存时间>'  # 配置长期存储密码  $ git config --global credential.helper store | | |
| **本地** | | |
| git init | 初始化项目所在目录，初始化后会在当前目录下出现一个名为 .git 的目录 | |
| git status | 查看本地仓库的状态 | |
| git add | # 把指定的文件添加到暂存区中  $ git add <文件路径>  # 添加所有修改、已删除的文件到暂存区中  $ git add -u [<文件路径>]  $ git add --update [<文件路径>]  # 添加所有修改、已删除、新增的文件到暂存区中，省略 <文件路径> 即为当前目录  $ git add -A [<文件路径>]  $ git add --all [<文件路径>]  # 查看所有修改、已删除但没有提交的文件，进入一个子命令系统  $ git add -i [<文件路径>]  $ git add --interactive [<文件路径>] | 把要提交的文件的信息添加到暂存区中。当使用 git commit 时，将依据暂存区中的内容来进行文件的提交。 |
| git commit | # 把暂存区中的文件提交到本地仓库，调用文本编辑器输入该次提交的描述信息  $ git commit  # 把暂存区中的文件提交到本地仓库中并添加描述信息  $ git commit -m "<提交的描述信息>"  # 把所有修改、已删除的文件提交到本地仓库中  # 不包括未被版本库跟踪的文件，等同于先调用了 "git add -u"  $ git commit -a -m "<提交的描述信息>"  # 修改上次提交的描述信息  $ git commit --amend | 将暂存区中的文件提交到本地仓库中 |
| git restore | 取消对文件的修改 | |
| git diff | # 比较当前文件和暂存区中文件的差异，显示没有暂存起来的更改  $ git diff  # 比较暂存区中的文件和上次提交时的差异  $ git diff --cached  $ git diff --staged  # 比较当前文件和上次提交时的差异  $ git diff HEAD  # 查看从指定的版本之后改动的内容  $ git diff <commit ID>  # 比较两个分支之间的差异  $ git diff <分支名称> <分支名称>  # 查看两个分支分开后各自的改动内容  $ git diff <分支名称>...<分支名称> | 比较版本之间的差异 |
| git checkout | # 切换到已存在的指定分支  $ git checkout <分支名称>  # 创建并切换到指定的分支，保留所有的提交记录  # 等同于 "git branch" 和 "git checkout" 两个命令合并  $ git checkout -b <分支名称>  # 创建并切换到指定的分支，删除所有的提交记录  $ git checkout --orphan <分支名称>  # 替换掉本地的改动，新增的文件和已经添加到暂存区的内容不受影响  $ git checkout <文件路径> | 检出命令，用于创建、切换分支等 |
| git branch | # 列出本地的所有分支，当前所在分支以 "\*" 标出  $ git branch  # 列出本地的所有分支并显示最后一次提交，当前所在分支以 "\*" 标出  $ git branch -v  # 创建新分支，新的分支基于上一次提交建立  $ git branch <分支名>  # 修改分支名称  # 如果不指定原分支名称则为当前所在分支  $ git branch -m [<原分支名称>] <新的分支名称>  # 强制修改分支名称  $ git branch -M [<原分支名称>] <新的分支名称>  # 删除指定的本地分支  $ git branch -d <分支名称>  # 强制删除指定的本地分支  $ git branch -D <分支名称> | 操作 Git 的分支命令 |
| git merge | # 把指定的分支合并到当前所在的分支下  $ git merge <分支名称> | 合并分支 |
| git stash | 临时保存当前的工作分支 |  |
| **远程** | | |
| git clone | # 默认在当前目录下创建和版本库名相同的文件夹并下载版本到该文件夹下  $ git clone <远程仓库的网址>  # 指定本地仓库的目录  $ git clone <远程仓库的网址> <本地目录>  # -b 指定要克隆的分支，默认是master分支  $ git clone <远程仓库的网址> -b <分支名称> <本地目录> | 从远程仓库克隆一个版本库到本地 |
| git remote | # 列出已经存在的远程仓库  $ git remote  # 列出远程仓库的详细信息，在别名后面列出URL地址  $ git remote -v  $ git remote --verbose  # 添加远程仓库  $ git remote add <远程仓库的别名> <远程仓库的URL地址>  # 修改远程仓库的别名  $ git remote rename <原远程仓库的别名> <新的别名>  # 删除指定名称的远程仓库  $ git remote remove <远程仓库的别名>  # 修改远程仓库的 URL 地址  $ git remote set-url <远程仓库的别名> <新的远程仓库URL地址> | 操作远程库 |
| git pull | 从远程仓库获取最新版本并合并到本地。  首先会执行 git fetch，然后执行 git merge，把获取的分支的 HEAD 合并到当前分支。 | |
| git push | # 把本地仓库的分支推送到远程仓库的指定分支  $ git push <远程仓库的别名> <本地分支名>:<远程分支名>  # 删除指定的远程仓库的分支  $ git push <远程仓库的别名> :<远程分支名>  $ git push <远程仓库的别名> --delete <远程分支名> | 把本地仓库的提交推送到远程仓库 |

## 插件

1. 汉化插件

“扩展”按钮（）搜索Chinese (Simplified) Language Pack

1. Python插件（Py支持插件）

搜索Python for VSCode安装

# 附录

## 换jupyter的背景

pip install jupyterthemes -i [http://pypi.douban.com/simple --trusted-host pypi.douban.com](http://pypi.douban.com/simple%20--trusted-host%20pypi.douban.com)

jt -l

jt -t 主题名称

## 内置异常的层次结构

**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)  
 +-- **GeneratorExit**            生成器(generator)发生异常来通知退出  
 +-- **Exception**             常规错误的基类  
     +-- StopIteration              迭代器没有更多值   
     +-- StopAsyncIteration             必须通过异步迭代器对象的\_\_anext\_\_()方法引发以停止迭代  
     +-- 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  
     +-- BufferError                   与缓冲区相关的操作时引发  
     +-- EOFError                       没有内建输入,到达EOF 标记  
     +-- ImportError                     导入失败  
     |   +-- ModuleNotFoundError       找不到模块  
     +-- LookupError                     无效数据查询的基类  
     |   +-- IndexError                     序列中没有此索引(index)  
     |   +-- KeyError                       映射中没有这个键  
     +-- MemoryError                     内存溢出错误  
     +-- NameError                       未声明、初始化对象  
     |   +-- UnboundLocalError             访问未初始化的本地变量  
     +-- OSError                         操作系统错误，  
     |   +-- BlockingIOError               操作将阻塞对象设置为非阻塞操作  
     |   +-- ChildProcessError             子进程上的操作失败  
     |   +-- ConnectionError               与连接相关的异常的基类  
     |   |   +-- BrokenPipeError             在已关闭写入的套接字上写入  
     |   |   +-- ConnectionAbortedError     连接尝试被对等方中止  
     |   |   +-- ConnectionRefusedError     连接尝试被对等方拒绝  
     |   |   +-- ConnectionResetError       连接由对等方重置  
     |   +-- FileExistsError               创建已存在的文件或目录  
     |   +-- FileNotFoundError             请求不存在的文件或目录  
     |   +-- InterruptedError             系统调用被输入信号中断  
     |   +-- IsADirectoryError             在目录上请求文件操作  
     |   +-- NotADirectoryError           在不是目录的事物上请求目录操作  
     |   +-- PermissionError              在没有访问权限的情况下运行操作  
     |   +-- ProcessLookupError           进程不存在  
     |   +-- TimeoutError                 系统函数在系统级别超时  
     +-- ReferenceError               弱引用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  
     |   +-- RecursionError           解释器检测到超出最大递归深度  
     +-- 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  
     +-- 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  
           +-- PendingDeprecationWarning   关于构造将来语义会有改变的警告  
           +-- RuntimeWarning           可疑的运行行为的警告  
           +-- SyntaxWarning            可疑的语法的警告  
           +-- UserWarning             用户代码生成的警告  
           +-- FutureWarning           有关已弃用功能的警告的基类  
           +-- ImportWarning            模块导入时可能出错的警告的基类  
           +-- UnicodeWarning           与Unicode相关的警告的基类  
           +-- BytesWarning             bytes和bytearray相关的警告的基类  
           +-- ResourceWarning           与资源使用相关的警告的基类。

## 库的安装

### 基本操作

|  |  |  |  |
| --- | --- | --- | --- |
| 查看存在的库 | 在Anaconda Prompt中pip list |  |  |
|  |  |  |  |
|  |  |  |  |

### 爬虫库

|  |  |  |
| --- | --- | --- |
| 库名 | 操作 |  |
| request | 安装：  pip install request  pip install 包名 -i http://pypi.douban.com/simple/ --trusted-host pypi.douban.com  查看信息：pip show requests  导入：import requests\import os | 在Anaconda Prompt |
| 正则表达式 | Import re |  |

# 代码

## 

爬虫1：

import requests

import csv

import re

url='https://search.51job.com/list/010000,000000,0000,00,9,99,%25E5%25A4%25A7%25E6%2595%25B0%25E6%258D%25AE,2,1.html'

headers={'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.114 Safari/537.36'}

res=requests.get(url=url,headers=headers)

# print(res.status\_code)

# with open('./data/wuyou.txt','w',encoding='utf-8') as f :

# f.write(res.text)

job\_name\_list=re.findall('"job\_name":"(.\*?)","job\_title"',res.text)

# print(job\_name\_list)

job\_company\_list=re.findall('"company\_name":"(.\*?)","providesalary\_text"',res.text)

#print(job\_company\_list)

with open('./data/zwei.csv','w',encoding='GB2312',newline='') as f:

writer=csv.writer(f)

writer.writerow(['职位','公司'])

for i in range(len(job\_name\_list)):

data=[job\_name\_list[i],job\_company\_list[i]]

with open('./data/zwei.csv','a',encoding='GB2312',newline='') as f:

writer=csv.writer(f)

writer.writerow(data)