# 公用

## 快捷键

|  |  |
| --- | --- |
| ctrl + shift + - #折叠全部  ctrl + - #折叠单层  ctrl + g #跳行  函数名下输入”””回车，自动添加注释格式  脚本文件注释自动化添加:File | Settings | Editor | File and Code Templates | Python Script   |  | | --- | | #!/usr/bin/env python  # -\*- coding: utf-8 -\*-  # @Time : ${DATE} ${TIME}  # @Author : 刘双喜  # @File : ${NAME}.py  # @Description : 添加描述 | |

# 一．数据类型

## 1.字符串

### 1.1属性

* 有序可变

### 1.2功能

#### 特殊

|  |
| --- |
| float('inf') # 无穷大  float('-inf') # 无穷小  float(‘nan’) == float(‘nan’) # False |

#### 居中

|  |
| --- |
| a = "Alex li 金角大王"  print(a.center(50,"-")) # 居中字符串并填充左右  output : --------------------Alex 金角大王--------------------- |

#### 判断

|  |
| --- |
| print(a.endswith("王")) # False 判断结尾 print(a.startswith("Ale")) # Ture 判断开头  print(a.isdigit()) # 判断是否是整数 False print("22".isdigit()) # 判断是否是整数 Ture  ‘z’ in a //判断字符z在不在字符串里  a.isalnum() //是不是字母或数字  a.isalpha() //是不是字母  a.isdigit() //是不是数字  a.isspace() //是不是包含空白字符\ t，\ n，\ r或者空格  a.istitle() //是不是首字母大写  a.issupper() //是不是都是大写字母  a.islower() //是不是都是大写字母  a.swapcase() //将大写字母转换成小写，小写字母转换成大写  a.title() //每个单词的首字母大写，以空格判断 |

#### 查找

|  |
| --- |
| print(a.count("l",0,4)) # 从索引（包含）0到3查找l出现的次数 print(a.find("i")) # 字符查找，返回-1代表没找到，如果找到了，就返回所查字符的索引  a.split(‘<msg>’)[-1] # 查找最后一个<msg>后面的字符，无则返回列表 |

#### 拼接

|  |
| --- |
| l = ["alex","black girl","peiqi"]  "-".join(l) # 拼接字符串， alex-black girl-peiqi |

#### 分割

|  |
| --- |
| print(a.split("l",1)) # 字符串以字符l为分隔符判断，分割次数为1，分割成列表 output: ['A', 'ex li 金角大王'] |

#### 替换

|  |
| --- |
| print(a.replace("l","M" ,1)) # 字符串替换，l替换成M，只替换一次，返回  output: AMex li 金角大王 |

#### 剥离

|  |
| --- |
| print(a.strip()) # 只去掉字符串a中开头和结尾的空格和转义字符(默认)  # a.lstrip() a.rstrip() 去掉从头、从尾开始的。。。 |

#### 转换

|  |
| --- |
| a.upper() //返回a字符串的大写  a.lower() //返回a字符串的小写  list(str) //把字符串每个字符变成列表元素  a[::-1] //字符串反转  a.title()//将首字母大写 |

#### 进制转换和逻辑运算

|  |
| --- |
| int(‘0xf’,base=16) //字符转换：16进制数转换成整型10进制  int(‘0xf’,base=8) # 8转10  bin(x)\oct(x)\hex(x) //十进制转换为2进制、8进制、16进制  逻辑与：and  逻辑或：or  1 << n # 2的n次方，效率比2\*\*n更高  i >> j #i除以2的j次方，取整的结果。 |

#### 取值

|  |
| --- |
| a[3:6] //字符串切片，取索引3开始，切到索引5，**顾头不顾尾**  a[3] //字符串取值索引3 |

#### 编码

|  |
| --- |
| 1、字符串前加 u  例：u"我是含有中文字符组成的字符串。"  作用：后面字符串以 Unicode 格式 进行编码，一般用在中文字符串前面，防止因为源码储存格式问题，导致再次使用时出现乱码。  2、字符串前加 r  例：r"\n\n\n\n”　　# 表示一个普通生字符串 \n\n\n\n，而不表示换行了。  作用：去掉反斜杠的转义机制。  （特殊字符：即那些，反斜杠加上对应字母，表示对应的特殊含义的，比如最常见的”\n”表示换行，”\t”表示Tab等。 ）  应用：常用于正则表达式，对应着re模块。  3、字符串前加 b  例:  response = b'<h1>Hello World!</h1>'  #b’ ’ 表示这是一个 bytes 对象  作用：b" "前缀表示：后面字符串是bytes 类型。  用处：网络编程中，服务器和浏览器只认bytes 类型数据。  如：send 函数的参数和 recv 函数的返回值都是 bytes 类型  附：  在 Python3 中，bytes 和 str 的互相转换方式是  str.encode(‘utf-8’)  bytes.decode(‘utf-8’) |

#### 填充

|  |
| --- |
| # 往字符串左边填充2,直到长度为5  '1'.rjust(5, '2')  # 往字符串右边填充  '1'.ljust(5, '2') |

## 2.列表

### 2.1属性

* 有序可变序列

### 2.2功能

#### 切片

|  |
| --- |
| ls[index % len(ls)] # 列表取值不会超索引的巧妙方法  list[0:4] //取值索引0到索引3，顾头不顾尾  list[:4] //取值索引0到索引3简写，顾头不顾尾  list[3:] //取值索引3到结尾  list[-3:-1] //取值倒数第三个到倒数第二个，顾头不顾尾  list[-3:] //取值倒数第三个到最后一个  list[1:-1] //取值索引1到倒数第二个，顾头不顾尾  list[1:-1:2] //以步长为2取值索引1到倒数第二个  list[::5] //以步长5从头开始取值  list[::-5] //以步长5从尾开始取值  高级切片:  url = "http://localhost/browser/Folder/Folder2/Folder3/the%20end%20xd.txt"  url = url[:url.rfind('/')]  # 切片赋值-增加和修改元素  l = list("0123456789")  l[6:] = list("abcdef")  print(l)  # 输出 ['0', '1', '2', '3', '4', '5', 'a', 'b', 'c', 'd', 'e', 'f']  # 切片赋值-插入元素  l = list("abcd")  l[2:2] = list("12")  print(l)  # 输出 ['a', 'b', '1', '2', 'c', 'd']  # 切片赋值-删除元素  l = list("helloworld")  l[2:-3] = []  print(l)  # 输出 ['h', 'e', 'r', 'l', 'd']  复制的3种方式：  浅复制,只复制内存地址:  new=old[:]  new=list(old)  new=old.copy()  深复制,把值拿过来:  new=copy.deepcopy(old) |

#### 合并

|  |
| --- |
| n1.extend(n2) //n2列表加到n1列表尾部 |

#### 判断

|  |
| --- |
| "X" in S //判断X在不在S列表  [] is [] # 结果为False，因为创建的两个列表的地址不一样，is判断的是地址，==判断的是值 |

#### 创建

|  |
| --- |
| list=[0]\*5 //创建一个5个元素为0的列表  list\_1 = list(“01234”) //生成列表['0', '1', '2', '3', '4'] |

#### 嵌套

|  |
| --- |
| names.insert(2,[1,2,3])  names ['alex', 'jack', [1, 2, 3], '黑姑娘']  names[2][1] -> 2 |

#### pop删除

|  |
| --- |
| list.pop() //默认删除最后一位元素并返回删除的值  list.pop(1) //指定删除索引为1的元素并返回删除的值 |

#### remove删除

|  |
| --- |
| list1.remove(“a”) //删除从左至右第一个找到的字符‘a’，删除后元素会往前移 |

#### 清空

|  |
| --- |
| list1.clear() |

#### 插入和修改

|  |
| --- |
| list1 [0] = ‘a’ //将第一个列表元素改成’a’  list1[-1] = ‘a’ //将倒数第一个列表元素改成‘a’  list1.insert(3,"X") //在S列表的第3元素插入X字符串  list1.append("X") //在列表末尾追加  # 使用append实际是修改一个列表,使用+实际是创建一个新的列表。  list1.extend([1,2,3])//在列表末尾追加多个元素r  "-".join(list) //以-的方法拼接列表list里的元素，只能是字符串 |

#### 查找和计数

|  |
| --- |
| list1.index(“item”) //返回从左匹配到的第一个item的索引号  list1.count(“item”) //返回item的个数  len(“list1”) //返回列表的元素个数 |

#### 排序

|  |
| --- |
| list1.sort(reverse=True) //倒序排序列表，以从前往后大写、小写、中文排序  list1.sort(key=lambda x:x[0]) //以列表list中元素的一维数据进行排序  list1.sort(key=lambda x:x[1]) //以列表list中元素的二维数据进行排序  list1.sort(key=lambda x:int(x[1])) //以列表二维数据的整形进行排序  # 首先以列表set\_s元素在s中出现的次数的**降序**进行排序，次数相同则以ascll码**升序**进行排序  sorted(set\_s, key=lambda x:(-s.count(x),x)) |

#### 反转

|  |
| --- |
| list1.reverse() //反转列表  list\_new = list1[::-1] //反转  list(map(list, zip(\*a))) //二维矩阵列表的转置 |

#### 索引

|  |
| --- |
| 凭借索引可以改循环体的值  >>> b=['a','b']  >>> a=list(enumerate(b,start=1))  >>> a  [(1, 'a'), (2, 'b')]  >>> for i in a:  ... print(i)  ...  (1, 'a')  (2, 'b')  >>> for i in a:  ... print(i[0],i[1])  ...  1 a  2 b |

#### 对比

|  |
| --- |
| list(set(a)-set(b)) 快捷对比两个列表的不同元素 |

## 3.元组

### 3.1属性

* 不可变有序序列
* 元组内嵌套的列表等可以改变

### 3.2功能

### 转换

|  |
| --- |
| tuple(ls) //ls转换成元组 |

## 4.字节

### 4.1属性

* 使用16或32位的十六进制字面量（分别加上前缀\u或\U）
* 不可变的序列对象

### 4.2功能

#### 定义

|  |
| --- |
| 定义 str1 = b’hello’ |

#### 转换

|  |
| --- |
| str.encode() 字符串转字节  str1.decode() 字节转字符串 |

## 5.集合

### 5.1属性

* 无序不重复
* 自动去重，但是一个字符串算一个元素不会去重
* 集合是不可哈希类型（字典、列表、集合），集合只能存放可哈希类型（元组、字符串）

### 5.2功能

#### 定义

|  |
| --- |
| s = set({ }) ；s = {1，2，3} |

#### 修改

|  |
| --- |
| s.add(x) 增加  s.update(“xx”) xx去重后添加到s  s.remove（val）删除 |

#### 交并补

|  |
| --- |
| x & y ； &交集 |并集 -差集 |

## 6.字典

### 6.1属性

* 无序值可变

### 6.2功能

#### 定义

|  |
| --- |
| info ={“lsx”:[23,’nan’,’hanzu”], ”ljb”:[24,’nan’,’hanzu’] } |

#### 序列创建

|  |
| --- |
| dict.fromkeys(seq, value)  以序列 seq 中元素做字典的键，value 为字典所有键对应的初始值 |

#### 修改

|  |
| --- |
| info[“mc”] = [22,’nan’,’hanzu’] //增加或者修改原有”mc“  info.pop(“lsx”) //指定删除lsx，返回lsx的值[23,’nan’,’hanzu”]  del info[“lsx”] //指定删除lsx，不返回  不确定键是否存在时赋值：  my\_dict.setdefault('key1', {}).setdefault('key2', {}).setdefault('key3', {})['key4'] = 'value' |

#### 查询

|  |
| --- |
| info.keys() //返回所有key  info.values() //返回所有value  len(info) //返回字典长度，即key个数  dict.get(“key”) //返回对应值，如没关键字则返回空  dict.setdefult(‘key’, ‘1’) //查询key，没有则添加key返回1，有则返回值不操作字典  dict.get(0) or dict.get(1) //获取0或者1  dict.get(0,1) //获取0，没有则返回默认值1 |

#### 转换

|  |
| --- |
| info.items() //转换一个嵌套列表 dict\_items([('ljb', [23, 'nv', 20000]), ('lsx', [23, 'nan', 2000])]  a = dict([['a', 'b'], [1,2]]) print(a) # {'a': 'b', 1: 2} |

|  |
| --- |
| s = [('a',1),('b',2),('c',3),('a',4),('b',5),('c',6)]  d = defaultdict(list)  for k,v in s:  print(k,v)  d[k].append(v)  print(d) #嵌套列表转字典 |

#### 遍历

|  |
| --- |
| for k,v in info.items():  print(k,v) //字典遍历结果为： lsx [23, 'nan', 20000] ljb [23, 'nv', 20000]  for k in info:  print(k,info[k]) //结果同上一样 运行效率更高 |

## 7.汇总

|  |
| --- |
| dict2.update(dict1) # 字典1合并到字典2 |

|  | **列表** | **元组** | **集合** | **字典** |
| --- | --- | --- | --- | --- |
| 英文 | list | tuple | set | dict |
| 可变与不可变类型 | 可变类型 | 不可靠类型 | 可变类型 | 可变类型 |
| 可否重复 | 是 | 是 | 否 | 是 |
| 存储方式 | 值 | 值 | 键(不能重复) | 键值对(键不能重复) |
| 是否有序 | 有序 | 有序 | 无序 | 无序，自动正序 |
| 初始化 | [1,'a'] | ('a', 1) | set([1,2]) 或 {1,2} | {'a':1,'b':2} |
| 添加 | append | 只读 | add | d['key'] = 'value' |
| 读元素 | l[2:] | t[0] | 无 | d['a'] |

|  |
| --- |
| 可变类型与不可变类型的区别:  margin = [3] a = margin margin[0] = 1 print(a) # a=[1] ,a的id和margin的id一致  margin = 3 a = margin margin = 1 print(a) # a=3,a的id和margin的不一致 |

|  |
| --- |
| 局部变量：  定义在函数内的变量  若定义一个变量在函数内以及其内部函数使用，又不希望定义为全局变量，可使用nonlocal声明  全局变量：  定义在函数外的变量  若全局变量在函数内进行赋值，则会变成局部变量。若要在函数内改变全局变量，需要用global声明变量为全局变量 |

# 二．基本语句

## 1.if

|  |
| --- |
| a = ‘0’  if a is ‘1’:  c = 1  elif a is ‘2’:  c = 2  else:  c = 3  简写：  c = {‘1’:1, ‘2’:2}.setdefult(a,3) |

## 2.for循环

### 1.1属性

* 可循环列表、字符串等

### 1.2功能

|  |
| --- |
| for i in range(10): # **range(10)这个函数只会执行一次!**  continue //跳过本次循环  break //结束这个循环  print(i)  for a,\*b in [('foo', 1, 2),('bar', 'hello'),('foo', 3, 4)]：  if a == ‘foo’:  print(b) |

## 3.while循环

### 2.1属性

* 为Ture时执行

### 2.2功能

|  |
| --- |
| while 条件：  continue //跳过本次循环  break //结束这个循环,不会进入else  print(‘’)  else: # 如果条件不满足进入else  print(‘else’) |

## 4.异常处理语句

### 3.1属性

* 用于处理当程序运行异常时给予的操作

### 3.2功能

|  |
| --- |
| 用法一：  try:  <语句> #运行代码  except ：  <语句> #如果在try部份引发了异常执行的语句  else:  <语句> #如果没有异常发生  用法二：  try:  <语句> #运行代码  except Exception as e ：  print(e) #如果在try部份引发了异常,将错误赋值给e，并输出  else:  <语句> #如果没有异常发生  用法三：（手动抛出异常）  try:  a = input("输入一个数：")  #判断用户输入的是否为数字  if(not a.isdigit()):  raise ValueError("a 必须是数字")  except ValueError as e:  print("引发异常：",repr(e))  示例四：  try:  print('11')  a=[]  a[0]  print('22') except ZeroDivisionError as E: # 捕获对异常才会进入  print('33') else: # 无异常才会进入  print('44') finally: # 捕获对异常最终会执行  print('55') print('66') # 没捕获对异常则程序报错，不会执行到这 |

### 3.3常见异常

* BaseException 所有异常的基类
* Exception 常规错误的基类
* EnvironmentError 操作系统错误的基类
* ImportError 导入模块/对象失败
* Warning 警告的基类

## 5.with语句

|  |
| --- |
| with语句常用来打开文件，因为with会自动关闭文件句柄，例如：  with open("/tmp/foo.txt") as file:  data = file.read()  其原理如下：  基本思想是:with所求值的对象必须有一个enter()方法，一个exit()方法。  紧跟with\*\*后面的语句被求值后，返回对象的\*\*\_\_enter\_\_()方法被调用，这个方法的返回值将被赋值给as后面的变量。当with后面的代码块全部被执行完之后，将调用前面返回对象的exit()方法。 |

## 6.表达式

### 1.逻辑表达式

|  |
| --- |
| int(‘0xf’,base=16) //字符转换：16进制数转换成整型10进制  bin(x)\oct(x)\hex(x) //十进制转换为2进制、8进制、16进制  逻辑与：and  print(0 and 1) -- 0  print(1 and 0) -- 0  print(0b00111100&0b00001101) – 多于1位要加0b  print('0' and '1') -- ‘1’  print('0' and 'a') -- ‘a’  print(1011 & 1100) – 此时为十进制整数之间的运算  逻辑或：or  print('c' and '0') -- ‘c’  位运算&：  3 & 5 -- 1  3 and 5 – 5  //运算，整除，并且不已科学计数法表示（科学计数法会降低大数的精度） |

### 2.\*和\*\*表达式

|  |
| --- |
| \* 解包  \*[1,2,3] 为1 2 3  \*\* 将函数入参变成字典 |

### 3.海象运算符(var:= obj)

|  |
| --- |
| 用于给变量赋值并返回赋的值  1、双重赋值  a = (c := 1)  2、while  while (p := input()) != ‘1’:  continue  w=0  while (w := w+1) < 3:  print(w)  3、用于推导式  def f(i):  return i\*\*2-1  nums2 = [j for i in nums1 if (j := f(i)) > 50] |

# 三．模块

## 1.属性

可下载第三方模块：pip install xx -i <https://pypi.douban.com/simple>

## 2.功能

### 导入

|  |
| --- |
| import module\_a #导⼊  from module import xx # 导⼊入某个模块下的某个方法 or 子模块  from module.xx.xx import xx as rename #导入后⼀个方法后重命名  from module.xx.xx import \* #导入⼀个模块下的所有方法，【如果其他目录写了\_\_all\_\_=[]，则只能调用all列表里的函数和数据和类等】 |

### 调用

|  |
| --- |
| module\_a.xxx # 调⽤  A程序调用其他目录模块：  1. 先查找A程序默认的模块查找路径sys.path  2. 把其他模块的路径添加到A程序的查找路径sys.path.append(“其他模块的路径”)  3. 对于其他电脑上，”其他模块”的路径不一样，但是A程序和其他模块的相对路径一样，只需把A程序的绝对路径改变os.path.dirname(\_file\_) 去掉A程序路径最后一项  import sys,os sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))))  调用时导入本路径相对于包的路径  4. \_\_name\_\_在程序中默认等于\_\_main\_\_,作为库时\_\_name\_\_为库名，所以库中加上下列语句，库在被调用时不会执行。  if \_\_name\_\_ == '\_\_main\_\_':  执行程序的内容，但是不会被调用执行的部分 |

### 包

|  |
| --- |
| 1. 在文件夹里创建一个\_init\_.py ，文件夹就变成了包  2. 在其他文件调用包内的模块时，会自动执行该包路径里的所有\_init\_.py ，相当于初始化 |

## 3.方法

### random（产生随机数）

|  |
| --- |
| import random//导入  random.choice("abc") #字符串中随机返回一位，也可以是列表  random.sample(s,3)//从数据源s中随机取出3个值  random.randint(1,100,2)//1到100随机抽取步长2偶数打印一个  random.uniform(a,b) //a到b的范围随机生成一个浮点数  random.random() //返回一个随机浮点数  random.shuffle(数据源) //数据源随机打乱  x = 4 if random.randrange(100) > 89 else 2 //以十分之一的概率生成4 |

### string（产生数字和字母）

|  |
| --- |
| import string  string.ascii\_letters#返回小写和大写字母  string.ascii\_uppercase#返回大写字母  string.ascii\_lowercase#返回小写字母  string.punctuation#返回特殊字符  string.digits#返回数字 |

### os（操作系统命令）

|  |
| --- |
| os.chdir(‘’) //切换目录  os.rename(old-name,new-name) //更改文件名  os.getcwd() //得到当前工作目录，即当前Python脚本工作的目录路径  os.listdir() 返回指定目录下的所有文件和目录名:  os.remove() 函数用来删除一个文件:  os.removedirs (r"c: \python") 删除多个目录:  os.path.isfile() 检验给出的路径是否是一一个文件:  os.path.isdir() 检验给出的路径是否是一个目录:  os.path.isabs() 判断是否是绝对路径:  os.path.exists() 检验给出的路径是否真地存:  os.path.exists(‘~$1.xlsx’) # 判断excel是否打开  os.path.split() 返回一个路径的目录名和文件名:  e.g  os.path.split(' /home/ swaroop/ byte/ code/poem. txt')  结果:( ' /home/ swaroop/byte/code'，' poem. txt' )  os.path.splitext() 分离扩展名:  e.g  os.path.splitext(' /usr/1ocal/test.py')  结果: (' /usr/1oca1/test', ' .py')  os.path.dirname() 获取路径名:  os.path.abspath() 获得绝对路径:  os.path.basename( ) 获取文件名:  os.system() 运行she11命令:  os.getenv( "HOME" ) 读取操作系统环境变量HOME的值:  os.environ 返回操作系统所有的环境变量:  os.environ. setdefault( ' HOME', ' /home/alex') 设置系统环境变量，仅程序运行时有效:  os.linesep 给出当前平台使用的行终止符,Windows使用'\r\n', Linux and MAc使用'\n '  os.name 指示你正在使用的平台:,对于windows，它是'nt',而对于linux/Unix用户，它是'posix '  os.rename(old, new) 重命名:  os.makedirs(r"c: \python\test") 创建多级目录:  os.mkdir("test") 创建单个目录:  os.stat(file) 获取文件属性:  os.chmod(file) 修改文件权限与时间戳:  os.path.getsize(filename) 获取文件大小:  os.path.join(dir, filename ) 结合目录名与文件名:  os.chdir(dirname) 改变工作目录到dirname:  os.get\_terminal\_size() 获取当前终端的大小:  os.kill( 10884, signal. SIGKILL) 杀死进程;  os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]]):   * 其中 top 是你所要遍历的目录的地址，返回的是一个三元组 (root, dirs, files)。root 所指的是当前正在遍历的这个文件夹的本身的地址；dirs 是一个 list ，内容是该文件夹中所有的目录的名字 (不包括子目录)；files 同样是 list , 内容是该文件夹中所有的文件 (不包括子目录)   os.path.expanduser("~/Documents") # 获取当前系统主目录路径 |

### pickle（序列化python专用）

|  |
| --- |
| 模块提供了四个功能: dumps、 dump、loads、load  import pickle  data = {'k1' :123, 'k2': 'Hello'}  # pickle . dumps将数据通过特殊的形式转换位只有python语言认识的字符串  P\_ str = pickle . dumps(data) #注意dumps会把数据变成bytes格式  print(P\_ str )  # pickle . dump将数据通过特殊形式转换位只有python语言认识的字符串，并写入文件  with open( ' result.pk', "wb") as fp:  pickle . dump(data, fp)  # pickle. load从文件 里加载  f = open("result.pk","rb")  d = pickle.load(f)  print(d) |

### json（序列化）

|  |
| --- |
| import j son  # json.dumps 将数据通过特殊的形式转换位所有程序语言都认识的字符串  j\_ str = json. dumps(data) #注意json dumps生成的是字符串，不是bytes  print(j\_ str)  #dump入文件  with open( ' result. json','w' ) as fp:  json . dump(data, fp)  #从文件里load  with open( "result. json") as f:  d = json.load(f)  print(d) |

### sys（读取外部参数）

|  |
| --- |
| import sys  print(sys.argv) # sys.argv即生成一个列表，其第一个元素是本程序路径，第二个元素和后面的元素依次存入运行程序时输入的外部变量  eg.  如运行 python 读取外部参数.py “1” “2”  则程序中sys.argv列表为[‘读取外部参数’,’1’,’2’]  # 每次读取键盘输入的一行参数，可以读取任意行。input只能获取已知行  n\_list = []  for line in sys.stdin:  a = line.strip()  if 'q' == line.rstrip():  break      n\_list.append(int(a)) |

### time（时间）

|  |
| --- |
| print(time.localtime()) # 将时间戳转为当前时区（UTC+8）的元组，不填默认当前时间戳转换为时间元组  print(time.gmtime()) # 将时间戳转为0时区（UTC）的元组，不填默认当前时间戳  print(time.time()) # 返回当前时间戳  t = (2021, 11, 5, 13, 34, 22, 4, 309, 0) # 将时间元组转换为时间戳  print(time.mktime(t))  time.sleep(1) # 延时1s  print(time.strftime("%Y-%m-%d %X------%Z", time.localtime())) # 把时间元组转换为格式时间  print(time.strptime("2021-11-05 13:54:58", "%Y-%m-%d %X")) # 将格式时间 逆转成时间元组  时间格式化：Y包含世纪的年 ;y不包含世纪的年; m月 ;d日 ;H24制时 ;I12制时 ;M分 ;S秒 ; w周数字 ; a周英文缩写 ; A周全拼 ; b月缩写 ; B月全拼  j一年的第多少天 ; p上下午 ; U一年的第多少周（周日为开始）; W一年的第多少周（周一为开始） ; c、X格式化的整体时间 ; x格式化的日期整体显示  z时区偏移 ;Z时区 |

### datetime（时间）

|  |
| --- |
| datetime模块下不同的类  (1) datetime.date：日期表示类，可以表示年、月、日等。  (2) datetime.time：时间表示类，可以表示小时、分钟、秒、毫秒等。  (3) datetime.dateime：日期和时间表示的类，功能覆盖 date 和 time 类。  (4) datetime.timedelta： 与时间间隔有关的类。  (5) datetime.tzinfo：与时区有关的信息表示类。  ————————————————  import datetime as dt  from datetime import datetime  from datetime import timedelta  # 时间戳转时间  dt.date.fromtimestamp(1636115536.693809) # 返回date类型  dt.datetime.fromtimestamp(1636115536.693809) # 返回datetime类型  # 返回一个 datetime 类型，表示当前的日期时间，精确到微秒。  now = datetime.now()  # 返回一个 datetime 类型，表示指定的日期和时间，可以精确到微秒。  datetime (year, month, day, hour=0, minute=0，second=0, microsecond=0)  # 返回datetime对象的年、月、日…  now.year  now.month  now.day  now.hour  now.minute  now.second  now.microsecond  now.timsteamp() # 返回时间戳  now.timetuple() # 返回时间元组  now. isoweekday() # 返回星期1~7  # datetime对象加减  now + timedelta(hours=2)  datetime.now() – now  # datetime对象格式化输出  now.strftime("%A %d")  print("今天是 {0:%Y} 年 {0:%m} 月 {0:%d} 日".format(now))  # 格式化字符串    # 逆转时间格式  from datetime import datetime  # 创建一个datetime对象 dt = datetime.now()  # 使用strftime方法将datetime对象转换为字符串 dt\_str = dt.strftime('%Y-%m-%d %H:%M:%S') print('原始日期时间字符串：', dt\_str, type(dt\_str))  # 使用strptime方法将字符串转换回datetime对象 dt\_reversed = datetime.strptime(dt\_str, '%Y-%m-%d %H:%M:%S') print('逆转后的日期时间：', dt\_reversed, type(dt\_reversed))  打印：  原始日期时间字符串： 2024-01-07 12:14:58 <class 'str'>  逆转后的日期时间： 2024-01-07 12:14:58 <class 'datetime.datetime'> |

### openpyxl（表格操作）

#### 创建

|  |
| --- |
| from openpyxl import Workbook  date = Workbook() # 创建  # 创建工作表  data.create\_sheet(‘Sheet2’)  sheet = date.active # 获取当前活跃工作簿  print(sheet.title)  sheet.title = "工作簿1" # 工作簿名称修改  date.save("我的第一个工作簿.xlsx") # 保存 |

#### 加载修改

|  |
| --- |
| from openpyxl import load\_workbook  date = load\_workbook("我的第一个工作簿.xlsx") # 加载  print(date.sheetnames) # 获取所有的工作簿名  end\_row = sheet.max\_row # 最后一行  end\_col = sheet.max\_colum # 最后一列  sheet = date["2"] # 选择编辑的工作簿  sheet["A3"] = 'A3' # 修改  date.save('我的第一个工作簿.xlsx')  date = load\_workbook("x.xlsx"，data\_only=True)#计算单元格公式再加载 |

#### 遍历

|  |
| --- |
| from openpyxl import load\_workbook  date = load\_workbook("我的第一个工作簿.xlsx")  sheet = date["3"]  print("------按行遍历--------")  for row in sheet: # 循环获取表数据  row\_value\_list = [cee.value for cell in roww]  for cell in row: # 循环获取每个单元格数据  print(cell.value, end=",")  cell.value = 5  print()  print("------按指定行遍历--------")  for row in sheet.iter\_rows(min\_row=2,max\_row=5,min\_col=3,max\_col=5):  for cell in row:  print(cell.value,end=",")  print()  print("------按列遍历--------")  for column in sheet.columns:  for cell in column:  print(cell.value,end=",")  print()  print("------按指定列遍历--------")  for col in sheet.iter\_cols(min\_row=6,max\_row=8,min\_col=2,max\_col=5,):  for i in col:  print(i.value,end=",")  print()  遍历时的row[3]就是本行第三个单元格 |

#### 修改字体格式

|  |
| --- |
| from openpyxl.styles import Font,colors,Alignment  from openpyxl import load\_workbook  # 加载和修改内容  date = load\_workbook("我的第一个工作簿.xlsx")  sheet = date['3']  sheet['A3'] = '我'  # 修改字体  myfont = Font(name="宋体",size=20,italic=True,color=colors.BLUE)  sheet['A3'].font = myfont  # 修改格式  my\_alig = Alignment(vertical='center',horizontal='center') #垂直、水平居中  sheet['A3'].alignment = my\_alig  # 修改填充  my\_fill = fills.PatternFill('solid', fgColor='FFC0CB')  sheet['A3'].fill = my\_fill |

#### 画图

|  |
| --- |
| **from** openpyxl.chart **import** LineChart,Reference **from** openpyxl **import** load\_workbook   excel = load\_workbook(**'18.excel-demo.xlsx'**) sheet = excel[**'Sheet1'**]  *# 实例化图表* chart = LineChart()  *# 选择数据来源* data = Reference(sheet, min\_col=2, min\_row=1, max\_col=3, max\_row=9) *# 标签数据来源* categories = Reference(sheet, min\_col=1, min\_row=1, max\_col=1, max\_row=9) *# X轴标签来源  # 设置图表属性* chart.x\_axis.title = **"工龄"** chart.y\_axis.title = **"工资"** *# 设置图表属性 - 添加数据* chart.add\_data(data, titles\_from\_data=**True**, from\_rows=**False**) *# from\_rows:以行数据画图 # 设置图表属性 - 设置标签* chart.set\_categories(categories)  *# 添加图表* sheet.add\_chart(chart, **'A11'**) excel.save(**'18.excel-demo.xlsx'**)  **""" chart图表属性 # 创建LineChart对象 chart = LineChart() # 给图表设置标题 chart.title = "Report" # 给x轴和y轴设置标题 chart.x\_axis.title = "Date" chart.y\_axis.title = "Price" # 设置大小 chart.width = 15 chart.height = 12 # 设置图表的风格 chart.style = 15 # 设置数据点的分组方式 chart.grouping = "standard" # 设置数据点的形状和大小 chart.shape = "circle" chart.marker\_size = 8 # 设置曲线是否光滑 chart.smooth = True # 设置图例 chart.legend.position = "right" chart.legend.include\_in\_layout = False # 设置数据标签 chart.data\_labels.font = "Calibri" chart.data\_labels.position = "inside" # 添加数据 chat.add\_data(data,titles\_from\_data=True,from\_rows=False) """** |

### smtplib（邮件操作）

|  |
| --- |
| import smtplib  from email.mime.text import MIMEText  from email.header import Header  # 登录  email\_obj = smtplib.SMTP\_SSL("smtp.qq.com",465)  email\_obj.login("1604030114@qq.com","gwfxaascltvjhiia")  # 设置邮件内容  date = MIMEText('这是一封神秘的邮件，请你仔细阅读',"plain","utf-8") # 内容  date["From"] = Header("给我转2000我就告诉你",'utf-8') # 发送者  date['To'] = Header('你是我的有缘人','utf-8') # 收件人  date['Subject'] = Header('这不是广告，请注意，这不是广告！','utf-8') # 主题  # 发送  email\_obj.sendmail('1604030114@qq.com',['578734036@qq.com','1499426657@qq.com','1245194125@qq.com'],date.as\_string()) |

### pygame（游戏开发）

|  |
| --- |
| import pygame  from pygame.locals import \*  # 创建一个窗口  window = pygame.display.set\_mode((width,hight))  # 设置窗口标题  pygame.display.set\_caption("窗口标题")  # 设置窗口图标  pygame.display.set\_icon(image)  #设置背景  window.fill((255,255,255))  # 加载资源图片，返回图片对象  image = pygame.image.load("图片的路径")  # 指定坐标，将图片绘制到窗口  window.blit(image, (0, 0))  # 将图片对象按指定宽高缩放，返回新的图片对象  trans\_image = pygame.transform.scale(image, (WINDOWWIDTH, WINDOWHEIGHT))  # 获得图片矩形对象 -> Rect(x, y, width, height)  # 默认情况下左上角的坐标是 (0, 0)  rect = image.get\_rect(centerx=x, centery=y)  # 在原位置基础上，移动指定的偏移量 (x, y 增加)  rect.move\_ip(num1, num2)  # 判断两个矩形是否相交，相交返回True，否则返回False  flag = pygame.Rect.colliderect(rect1, rect2)  # 获得当前所有持续按键  # 获得所有事件的列表  event\_list = pygame.event.get()  for event in event\_list:  # 1. 鼠标点击关闭窗口事  if event.type == pygame.QUIT:  print("关闭了窗口")  exit()  # 2. 键盘按下事件  if event.type == pygame.KEYDOWN:  # 判断用户按下的键是否是a键  if event.key == pygame.K\_a:  print("按了 a ")  if event.key == pygame.K\_UP:  print("按了 方向键上")  # 3. 获得当前键盘所有按键的状态(按下，没有按下)，返回bool元组  pressed\_keys = pygame.key.get\_pressed()  if pressed\_keys[pygame.K\_w] or pressed\_keys[pygame.K\_UP]:  print("按了 w 键，或者 方向键上")  # 4.判断按键值  if keyboard.is\_pressed('w'):  # 监听按键，当没按指定键时进行堵塞  keyboard.wait('a')  # 热键，当监听的时候按下对应键执行相应函数  keyboard.add\_hotkey('f1', f1)  # 加载背景音乐  pygame.mixer.music.load("./res/音乐文件名")  # 循环播放背景音乐  pygame.mixer.music.play(-1)  # 停止背景音乐  pygame.mixer.music.stop()  # 加载音效  boom\_sound = pygame.mixer.Sound("./res/音效名")  # 播放音效  boom\_sound.play()  boom\_sound.stop()  # -------- 文字显示操作  text1 = pygame.font.Font('c:/Windows/Fonts/simhei.ttf', 30)  text1\_load = text1.render('显示的文本', True, (255, 0, 0))  self.window.blit(text1\_load, (0, 420))  # 设置文字矩形对象位置  textrect = textobj.get\_rect()  textrect.move\_ip(水平偏移量, 竖直偏移量)  # 在指定位置绘制指定文字对象  window.blit(textobj, textrect)  **# 更新界面**  **pygame.display.update()** |

### itertools（迭代器）

|  |
| --- |
| import itertools as it  # 去掉列表中嵌套的列表或者集合或者元组  b=[[1,2],[2,3],[3,4]]  print(it.chain(\*b)) # 输出[1,2,3,4]  # 排列和组合api  from itertools import combinations, permutations  list(permutations([1, 2, 3], 2)) #排列，有顺序区别  list(combinations([1, 2, 3], 2)) #组合，无顺序区别 |

### collections（容器数据类型）

|  |
| --- |
| import deque  d=deque(maxlen = 5) #生成最大长度5的序列，后添加的会挤掉前面的  d.append(‘a’)  d.appendleft(‘a’)  d.pop()  d.popleft()  from collections import defaultdict  d = defaultdict(set) #创建一个字典，值是集合  d['a'].add(1)  e = defaultdict(list)#创建一个字典，值是列表  e['a'].append(1)  from collections import OrderedDict  # OrderedDict 内部维护着一个根据键插入顺序排序的双向链表，内存大2倍  d = OrderedDict()  d['foo'] = 1  d['bar'] = 2  d['spam'] = 3  d['grok'] = 4  # Outputs "foo 1", "bar 2", "spam 3", "grok 4"  for key in d:  print(key, d[key])  ##############  from collections import defaultdict  # 创建字典，默认键的值为[0,0,0]，即当访问的键不存在时自动创建值  prices = defaultdict(lambda: [0, 0, 0]) |

### configparser(配置解析器)

|  |
| --- |
| ConfigParser对象的结构类似于字典，它主要包含以下部分：  1、Sections：配置文件的主要部分被划分为sections。每个section都是由一个方括号包围的标题，例如[section1]。在这个标题下面，你可以设置一些键值对。  2、Keys：每个section下面都有一些键值对。键和值之间可以用等号=或者冒号:分隔。  3、Values：每个键后面跟着一个值。这个值可以是任何字符串。 |

### heapy（堆）

|  |
| --- |
| heappush(heap, x) #将x压入堆中，最小的x会放在第一个，如果x是一个元组则根据元组的第一个值判断，如果第一个值相同则根据第二个值判断  heappop(heap) #从堆中弹出最小的元素  nlargest(n, iter， key) #返回iter中n个最大的元素  nsmallest(n, iter, key) #返回iter中n个最小的元素 |

### argparse(命令行解析)

|  |
| --- |
| import argparse  #实例化 arg\_test = argparse.ArgumentParser(description="this is test")  #添加 arg\_test.add\_argument('-n','--name',help='this is name')  #结果 res = arg\_test.parse\_args() print(res)  可在cmd运行 脚本.py –n ‘liusx’ 添加数据到name变量 |

### ddt(单元测试框架)

|  |
| --- |
| import unittest from ddt import ddt, data, unpack # 根据需要导入相应的模块   @ddt class TestDdt(unittest.TestCase):  def setUp(self):  print("setUp执行一次")   def tearDown(self):  print("tearDown执行一次")   @data(1,2,"hello")  def test\_single\_element(self, value):   print(value)  self.assertEqual(value,1)   if \_\_name\_\_ == '\_\_main\_\_':  unittest.main()  (将数据通过data装饰器导入，形成一个个测试用例，每个都得测试一遍)  输出:  Ran 3 tests in 0.001s  FAILED (failures=2)  setUp执行一次  1  tearDown执行一次  setUp执行一次  2  tearDown执行一次  setUp执行一次  hello  tearDown执行一次  进程已结束，退出代码为 1 |

### math

|  |
| --- |
| import math  math.gcd(12,8) #求最大公约数  math.log(x,base)  math.log10(x) # 可import math.log10 as lg  math.ceil 向上取整  math.floor 向下取整  pi = math.pi  n = 2 # 向上取整，保留小数的位竖 print(math.ceil(pi \* pow(10, n)) / pow(10, n))  math.perm(n, k) # 排列，n个项中选择k个项（不重复且有顺序）  math.comb(n, k) # 组合，获取从n个项目中选择k个项目（不重复且无顺序），  math.isnan(str\_v) # 判断是否是nan |

### traceback回溯

|  |
| --- |
| import traceback a = 1 try:  a.get(1) except:  traceback.print\_exc()  # 打印异常信息和文件函数和行数  traceback.format\_exc() # 返回异常信息，不打印  # 将异常打印到文件  traceback.print\_exc(file=open('tb.txt','w+')) |

### gc(垃圾回收)

|  |
| --- |
| gc.collect() #回收垃圾 |

### decimal(精确计算)

|  |
| --- |
| from decimal import \*  a = Decimal(“5.55”) #入参字符串或者整形，浮点本身就是不精确的  #保留有效数字几位  getcontext().prec = 6  Decimal(1)/Decimal(7)  # 结果为Decimal('0.142857')，六个有效数字  #四舍五入保留几位  Decimal('50.5679').quantize(Decimal('0.00'))  # 结果为Decimal('50.57')，结果四舍五入保留了两位小数 |

### pyinstaller (打包文件)

|  |
| --- |
| pip3 install pyinstaller  当前代码文件夹下cmd：Pyinstaller -F 1.py |

### sched（定时任务）

|  |
| --- |
| import sched  import time  def say\_hello(name1, name2):  print(f'{name1},{name2},hello')  sche\_obj = sched.scheduler(time.time, time.sleep) # 实例化对象  sche\_obj.enter(2, 1, say\_hello, ('lsx', 'ntt')) # 延时,优先级,函数,函数入参  s = time.strptime('2023:04:08:22:17:01','%Y:%m:%d:%H:%M:%S')  sche\_obj.enterabs(time.mktime(s), 1, say\_hello, ('lsx', 'ntt')) # 指定时间戳执行  sche\_obj.run() |

### bisect（二分查找）

|  |
| --- |
| # 查找有序的列表中，比6小或者比6大的位置。或者等于6的左边位置  a = bisect.bisect\_left([1, 3, 4, 5, 9, 55], 6) print(a)  # 查找有序的列表中，比6小或者比6大的位置。或者等于6的**右**边位置  a = bisect.bisect\_right([1, 3, 4, 5, 9, 55], 4) print(a) |

### Subprocess（cmd操作）

|  |
| --- |
| import subprocess # 执行CMD命令 result = subprocess.run('dir', shell=True, stdout=subprocess.PIPE, text=True) # 打印命令输出 print(result.stdout)  subprocess.run()函数执行了dir命令，shell=True参数表示命令将通过shell执行，stdout=subprocess.PIPE参数用于捕获标准输出，text=True参数指定输出为文本123。 |

### socket（网络服务）

|  |
| --- |
| def get\_host\_ip():  try:  s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  s.connect(('8.8.8.8', 80))  ip = s.getsockname()[0]  finally:  s.close()  return ip  import socket # 查看当前主机名 print('当前主机名称为 : ' + socket.gethostname()) # 根据主机名称获取当前IP print('当前主机的IP为: ' + socket.gethostbyname(socket.gethostname())) |

# 四．类和对象

## 1.属性

* 用来描述具有相同的属性和方法的对象的集合，它定义了该集合中每个对象所共有的属性和方法，对象是类的实例
* 类变量：在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用
* 数据成员：类变量或者实例变量, 用于处理类及其实例对象的相关的数据
* 方法重写：如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写
* 局部变量：定义在方法中的变量，只作用于当前实例的类
* 实例变量：在类的声明中，属性是用变量来表示的。这种变量就称为实例变量，是在类声明的内部但是在类的其他成员方法之外声明的
* 继承：即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个Dog类型的对象派生自Animal类，这是模拟"是一个（is-a）"关系（例图，Dog是一个Animal）
* 实例化：创建一个类的实例，类的具体对象
* 方法：类中定义的函数
* 对象：通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法

## 2.功能属性

|  |
| --- |
| # 类名  class Person:  # 属性…..  # 方法  def \_\_init\_\_(self,name,age):  print('init方法自动执行')  self.name = name  self.age = age  self.\_\_sex = 'girl' # 私有属性，只能在类内部被打印和修改  print('私有属性：',self.\_\_sex)  def \_\_msg(self):  print('这是私有方法')  def get\_msg(self): #调用私有方法  self.\_\_msg()  def \_\_str\_\_(self): # 当print这个类的对象时，会执行这个方法  return 'str方法被调用.'  def set\_age(self,age):  if age > 0:  self.age = age  else:  self.age = 0  def get\_age(self): # 隐藏self.age属性  return self.age  def eat(self):  print('----eat----')  #不用实例化该类也可以调用这个装饰器装饰的函数  @staticmethod  def info():  print(f'姓名是{self.name},年龄是{self.age}')  Person.info()  def Person2(Person):  # 类的继承  # 继承后可直接使用Person类的函数,或实例化后可使用  def info\_2(self):  self.eat()  # 创建对象  p1 = Person('nongtt',20)  p1.info()  # 调用  p1.eat()  #添加属性  p1.name = 'liusx'  p1.age = 21  p1.info()  # str方法  print(p1)  # 调用隐藏属性  p1.set\_age(-1)  print(p1.get\_age())  #调用私有方法  p1.get\_msg() |

|  |
| --- |
| **supper()的使用 直接调用父类**  class FooParent(object):  def \_\_init\_\_(self):  self.parent = 'I\'m the parent.'  print ('Parent')    def bar(self,message):  print ("%s from Parent" % message)    class FooChild(FooParent):  def \_\_init\_\_(self):  # super(FooChild,self) 首先找到 FooChild 的父类（就是类 FooParent），然后把类 FooChild 的对象转换为类 FooParent 的对象  super(FooChild,self).\_\_init\_\_()  print ('Child')    def bar(self,message):  super(FooChild, self).bar(message)  print ('Child bar fuction')  print (self.parent)    if \_\_name\_\_ == '\_\_main\_\_':  fooChild = FooChild()  fooChild.bar('HelloWorld')  res：  Parent  Child  HelloWorld from Parent  Child bar fuction  I'm the parent. |

|  |
| --- |
| @staticmetod  eg:  class Test:  @staticmethod  def b(x):  print(x)  from test import Test  print(Test.a(2)) |

|  |
| --- |
| # 继承多个类A,B  class C(A, B):  def foo(self):  super().A.foo()  super().B.foo() |

## 3.基本格式

|  |
| --- |
| class Test(object):  def \_\_init\_\_(self):  self.test = 'test'  def test\_def(self):  print(self.test)  def main():  r = Test()  if \_\_name\_\_ == '\_\_main\_\_':  main() |

## 4.重写特殊方法

|  |
| --- |
| \_\_getitem\_\_(self):允许实例用[取索引时调用此方法  \_\_len\_\_(self):用len()函数求实例时允许此方法  \_\_call\_\_(self):允许实例对象用obj()的方式执行时调用此方法  \_\_del\_\_(self)：析构函数，用于在类的实例被销毁时执行清理操作,比如程序结束  \_\_str\_\_(self)：定义类的实例的字符串表示，用于print和str函数。  \_\_repr\_\_(self)：定义类的实例的字符串表示，用于交互式解释器和repr函数。  str用于打印,repr用于调试. print(repr(obj))  \_\_eq\_\_(self, other)：定义类的实例之间的相等性测试，用于==运算符。  \_\_ne\_\_(self, other)：定义类的实例之间的不等性测试，用于!=运算符。  \_\_lt\_\_(self, other)：定义类的实例之间的小于关系，用于<运算符。  \_\_le\_\_(self, other)：定义类的实例之间的小于等于关系，用于<=运算符。  \_\_gt\_\_(self, other)：定义类的实例之间的大于关系，用于>运算符。  \_\_ge\_\_(self, other)：定义类的实例之间的大于等于关系，用于>=运算符。  \_\_add\_\_(self, other)：定义类的实例之间的加法运算，用于+运算符。  \_\_sub\_\_(self, other)：定义类的实例之间的减法运算，用于-运算符。  \_\_mul\_\_(self, other)：定义类的实例之间的乘法运算，用于\*运算符。  \_\_truediv\_\_(self, other)：定义类的实例之间的除法运算，用于/运算符。  \_\_floordiv\_\_(self, other)：定义类的实例之间的整除运算，用于//运算符。  \_\_mod\_\_(self, other)：定义类的实例之间的取模运算，用于%运算符。  \_\_pow\_\_(self, other[, modulo])：定义类的实例之间的幂运算，用于\*\*运算符。 |

# 五．函数

## 1.属性

说明文档：<https://docs.python.org/zh-cn/3/library/functions.html>

|  |  |  |
| --- | --- | --- |
| def str\_to\_float(位置参数，默认参数，\*args非固定参数元组，\*\*kwargs非固定参数字典):  str\_float = a[0:3]  return str\_float  str\_to\_float(位置参数，指定参数，””，”“，”张三”=[“男”])  1. 默认参数需要放最后  2. 调用可以用关键参数指定，但是必须放默认参数后  3. 有多个返回值时会形成元组，不指定return返回None  4. 遇到return代表程序结束  5. 函数内的变量都是局部变量，和主程序不冲突没关系。子函数内如果全局变量和局部变量命名一样，变量优先调用局部变量。子函数内不能修改全局变量的值，除非用globle声明一个全局变量替代。  6. 如果全局变量是列表和字典等，在子函数中可以改其中的值。因为字符串到子函数是copy进去，而字典等是传过去的地址  7. 返回值有2个时，可以用2个变量取值：x , y = 函数（）  8. 当第一个参数为\*时，表示不允许使用位置参数，只能用关键字入参  9. 入参不能是可变类型，如列表字典，否则有可能会改变原入参的值。   |  | | --- | | **def** a(b):  b[0] = 9   c = [1, 2] a(c) print(c) *# [9,1]* |   10. 函数的默认参数只会在函数定义时被计算一次，而不是每次调用函数时都重新计算。这意味着如果你使用一个可变的对象（如列表、字典或集合）作为默认参数，那么每次调用该函数时对该对象所做的更改都将在后续调用中保留。   |  | | --- | | **错误写法：**  **def** fun(num, ls=[]):  **for** i **in** range(num):  ls.append(i)  print(ls)  fun(2) fun(1)  打印：  [0, 1]  [0, 1, 0]  正确写法：  def fun(num, ls=None):  if ls is None:  ls = []  for i in range(num):  ls.append(i)  print(ls) |   闭包：  def text(num):  print('----------1-----------')  def text\_in(num\_in):  print('-------2---------')  return num + num\_in  print('------3---------')  return text\_in  s = text(10)  print(s(20)) |

## 2.内置函数

### self

* 类实例化后的对象的地址，可做入参给别的函数用，配合getattr()等使用

### all([0,1,2,3])

* 遍历所有数据，有一个为假则返回false，所有为真才返回Ture，为空返回Ture

### any([0,1,2,3])

* 遍历所有数据，有一个为真则返回ture

### abs()

* abs（-10） 取绝对值

### bool（-1）

* 判断真假，返回ture。除了None和0其他的都是ture
* 取反 a = bool(1-a)

### chr(97)

* 打印ascii码中的字符

### del

* 删除

### dict()

* 生成一个空字典

### exec(“”)

* 执行字符串形式的代码，如exec(“print(‘hello’)”)
* 批量创建变量并赋值：

|  |
| --- |
| for i in range(10):  exec ("temp%s=%d"%(i,i+2)) # 或直接用f””函数 |

### exit(“xxx”)

* 退出程序并打印xxx

### enumerate([“a”,”b”])

* 遍历列表返回每一个元素索引和值的地址

### enumerate()

* 枚举，enumerate(sequence, start=0) //以下标位置从0开始，数据源数据从第一个开始组合成一个索引序列。例：list=[‘a’,‘b’] -> list（enumerate(list)，start=1） -> 返回为 [(1, 'a'), (2, 'b')]

### eval()

* eval() 函数用来执行一个字符串表达式，并返回表达式的值
* eval(‘[1,2,3’] 类型转换，字符转列表
* 漏洞：脚本读取配置文件时，用eval去执行配置的数据有安全风险
* 规避：用if对配置文件的数据进行判断后再用eval
* 替代：ast.literal\_eval() 用此函数处理会进行判断执行后的结果是不是合法的python类型，不是则不进行运算且报错，大大提高安全性

### f“{}”

* 引用，字符串中的{}内的变量嵌入到字符串中
* 保留1位小数（4舍5入）print(f'{ans \* flag:.1f}')

### format

* '--{}'.format(1234) #生成一个字符串，把1234放到字符串的{}里
* ‘--{!r}’.format(123) #则为 “--‘123’”
* s=’name={},age={}’;s1=s.format(‘lsx’,’24’)
* s=’ age={age},name={name}’;s1=s.format(name=‘lsx’,age=’24’)
* format(1.23456, '.2f') -- 保留2位小数

### filter(表达式,list)

* 把列表的每一个值放到表达式中去判断，是真则返回

### float

* float(‘nan’) == float(‘nan’) # False

### id(x)

* 查询变量内存地址，返回内存地址

### type(x)

* 查询变量的数据类型，返回

### help(x)

* 查看帮助
* help（int）、help（数据. 函数）

### input(“”)

* 输入的值会被以字符串的类型返回
* 函数的（）中的字符串会被打印

### list()

* 强制转换为列表形式，用于字符串、字典等
* 同类函数比如str、int等

### isinstance

* isinstance('123',Iterable) //判断字符是不是可迭代
* isinstance('123',(str,dic)) //判断字符是不是字符串或者字典
* isinstance('123',Iterator) //判断字符是不是迭代器

### dir(obj)

* 返回当前对象所有的属性

### vars(obj)

* 返回当前对象所有的属性和值

### locals

* 返回当前程序中局部变量，比如某个函数或者某个py文件

### globals

* 返回当前程序中的全局变量，可以用globals()[‘x’] = 1修改全局变量的值

### lambda

* 自定义函数，例g = lambde x: x+1 ，即g(x)=x+1
* 配合map使用:map(lambde x,y,z:x+y+z, ls1,ls2,ls3),依次取3个列表的值给x,y,z，注意ls1里的值可以为变量，或者一个对象

### max(list)

* 求一个可迭代对象的最大值
* max(ls,key=len) 长度最大的元素

### map(函数，列表)

* 把列表里每一个值一次只能放一个到函数执行并把结果储存，注意函数不要加 ()
* map可以接受多个可迭代对象，map(fun,ls1,ls2)

### min(list)

* 求最小值

### sum(list)

* 求和

### round(3.1445,2)

* 保留2位小数
* round(x,0) 四舍五入

### ord(‘a’)

* 返回ascii表里ord的序号

### print()

* print(x,y,z,sep=' ?',end='\n',file=open(r‘a.txt’,’w’)) 格式输出x,y,z，以?间隔，结尾换行，输出写入到a,txt文件内
* print(x or '-') 输出变量x，如果x为0则输出-
* 字符串前面加 \033[1;31;40m ,则打印红色字体
* path = r"D:\file\python"

link = 'file:///' + path.replace('\\', '/') # 打印超链接文件路径

### pass

* 空语句

### range(0,10,2)

* 步长为2
* range(10,0,-1) #前大后小的，步长为负数

### zip(a,b)

* 列表a，b配对，列表a的第i个元素和b的第i个元素形成元组，要用list()输出

### hasattr (obj,name:str)

* 判断对象是否存在属性name

### getattr (obj,name:str)

* 获取对象的属性

### setattr (obj,name:str,value:any)

* 设置对象的属性

### delattr (obj,name:str)

* 删除对象设置的属性，删除后属性是默认值

# 六．文件操作

## 1.属性

## 2.功能

### 写

|  |
| --- |
| f = open("name\_list",mode="w", encoding='utf-8')  //告诉编译器打开的是什么编码的文件，windows默认打开文件使用gbk编码  f.write("张三2\n")  f.write("李四3\n") |

### 读

|  |
| --- |
| f = open("name\_list"，”r”, encoding='utf-8')  print(f.readline()) # 读一行  print(f.readlines()) # 读所有行且把每行内容形成元素，储存在列表返回。如果文件没有换行符则读到为空。读之前要f.seek(0)  print('------------')  print(f.read()) # 读出所有内容 |

### 追加

|  |
| --- |
| f = open("name\_list","a" ,encoding='utf-8')  f.write("heheh")  f.write("jack")  f.close() |

### 混合模式

|  |
| --- |
| w+ //写读  r+ //读写 读光标和写光标不一致，需要混合操作时要用seek定位  a+ //追加读 |

### 光标

|  |
| --- |
| f = open('二进制写文本', "w", encoding='utf-8')  # 注意打开文件编码和光标移动字节数，utf-8一个字3个字节，gbk一个字2个字节  f.write("刘双喜是人见人爱的大帅哥")  f.seek(12) //定位光标位置到字节后  # print(f.readline())  print("光标位置", **f.tell()**) //告诉光标位置  f.write("一个")  f.close() |

### 文件修改

|  |
| --- |
| f = open("文件文本", "r+", encoding="utf-8")  # 1.加载到内存，读取文本内容保持到变量，修改变量  date = f.read() # 可以指定读多少个字节  new\_date = date.replace("帅哥", "大帅哥", 1)  print(new\_date)  # 2.清空原文本  f.seek(0)  **f.truncate()** # 光标移动到开头，截断光标之后的字节，可以设置截断字数  # 3.把新内容写入  f.write(new\_date)  f.close() |

### 循环文件

|  |
| --- |
| f = open("嫩模联系方式", 'r', encoding='utf-8')  for line in f: # 每一行变成一个字符串，一行一行循环  line = line.split()  #print(line)  height = int(line[3])  weight = int(line[4])  if height >= 170 and weight <= 50:  print(line) |

### buffer强制存储

|  |
| --- |
| f = open("二进制写文本", "w", encoding="utf-8")  a = 2  while 1: #当指令正确才刷到硬盘显示  a = input("输入正确指令刷去硬盘：")  if a == "1":  f.write("刷去硬盘")  f.flush()  break  f.close() |

### 二进制模式操作图片

|  |
| --- |
| f = open("1313.jpg", 'rb') |

### 二进制模式操作文件

|  |
| --- |
| f = open("二进制写文本", 'wb')  s = '刘双喜'  f.write(s.encode("utf-8"))  f.close() |

# 七．推导式

## 1.属性

* 推导式 comprehensions（又称解析式），是 python 的一种独有特性。推导式是可以从一个数据序列构建另一个新的数据序列

## 2.功能

### 列表推导式

### 字典推导式

### 集合推导式

|  |
| --- |
| # 列表推导式  print([x\*x for x in range(10) if x%2 == 0])  print([x+y for x in 'abc' for y in 'def'])  dic = {'k1':'v1','k2':'v2'}  print([k+':'+v for k,v in dic.items()])  # 字典推导式  print({x:x\*2 for x in range(10)})  # 集合推导式  print({x for x in range(10)})  # 生成器  print(tuple((x for x in range(10))))  print(isinstance('123',(str,dic))) |

# 八．生成器和迭代器

## 1.生成器

### 1.1属性

* 在Python中，一边循环一边计算的机制，称为生成器：generator

### 1.2功能

|  |
| --- |
| def creat\_num():  a, b = 0, 1  for i in range(100000):  yield b  a, b = b, a + b  s = creat\_num()  print(next(s)) # 输出1 此时第一次代码执行到 yield b  print(next(s)) # 输出1 此时代码第二次执行到 yield b  print(next(s)) # 输出2...  print(next(s)) # 输出3...  print(next(s)) # 输出5... |

### 1.3应用

|  |
| --- |
| 并发运行：  def text1():  while True:  print('运行第一个程序')  yield None  def text2():  while True:  print('运行第二个程序')  yield None  t1 = text1()  t2 = text2()  while True:  next(t1)  next(t2) |

## 2.迭代器

### 2.1属性

* 迭代器是一个可以记住遍历位置的对象，因此不会像列表那样一次性全部生成，而是可以等到用的时候才生成，因此节省了大量的内存资源。迭代器对象从集合中的第一个元素开始访问，直到所有的元素被访问完。

### 2.2功能

|  |
| --- |
| lis = [1,2,3,4]  s = iter(lis)  print(next(s)) # 输出1  print(next(s)) # 输出2  print(next(s)) # 输出3  print(next(s)) # 输出4 |

# 九．装饰器

## 属性

* 修改其他函数的功能的函数

## 功能

|  |
| --- |
| def test1(s):  def test2():  print('----------2验证------')  s()  return test2  @test1 # @test1 加在def f1()前面 等价于f1=test1(f1)  def f1():  print('-------3功能---------')  # f1 = test1(f1)  f1()  图解： |

## 常用装饰器

### @property

@property 是一个内置装饰器，它用于将类方法转换为只读属性。调用方法时和调用属性一样，不需要加()，同时会执行方法的结果，并且只读属性不能改变值。

# 十.web开发

## 1.html标识语言

### 1.1属性

* HTML（Hypertext Markup Language，超文本标记语言）是一种文本类、解释执行的标记语言，用于编写要通过WWW显示的超文本文件，称为HTML文件，也叫Web页面或网页。扩展名为.html或.htm 。HTML文件的组成包含两部分内容：一是HTML标记；二是HTML标记所设置的内容。

### 1.2功能

#### 基本格式

|  |
| --- |
| <h1></h1> 一级标题  <h2></h2> 二级标题，共六级  <p></p > 段落  <a href=“网址”target=‘\_blank’>文字</a> //target新打开一个页面  <br>换行  <img src=“图片.png”width=“600px” height=“400px”>插入图片 |

#### 表格

|  |
| --- |
| <table border = ‘1’cellpadding=‘10px’> cellpadding边距  <tr>  <th></th>表头  <th></th>表头  </tr>  <tr> 行  <td></td>列  <td><b></b></td> 列  </tr>  <table>  <b></b>加粗 |

#### 列表

|  |
| --- |
| 有序  <ol start="50">  <li>Coffee</li>  <li>Tea</li>  <li>Milk</li>  </ol>  无序  <ul>  <li>Coffee</li>  <li>Tea</li>  <li>Milk</li>  </ul> |

#### div区块

|  |
| --- |
| <div style = ‘’>  </div> |

#### span内联元素

|  |
| --- |
| <span style= ‘color = red’ ></span> |

#### 表单

|  |
| --- |
| <form action="/test/url/"> /\* 定义表单 action为表单数据提交地址 \*/  用户名：<input type="text" name="username"> <br>/\*文本框\*/  爱好：<input id="hobbie" type="text" name="hobbie"> <br>  密码：<input type="password" name="password"><br>  年龄：<input class="class\_test" type="number" name="age"><br>  出生日期：<input type="date"><br>  个人介绍：<br> /\*文本域\*/  <textarea class="class\_test" cols="50" rows="5" placeholder=" 请输入不少于50字的个人介绍...."></textarea> <br> // placeholder提示符  Sex: <input type="radio">男 <input type="radio">女 /\*单选框\*/    <input type="submit" value="注册"> /\*按钮\*/  爱好： /\*复选框\*/  <input type="checkbox" name="hob" value="girl"> girl  <input type="checkbox" name="hob" value="car"> car  <input type="checkbox" name="hob" value="diving"> 潜水  <input type="checkbox" name="hob" value="python"> 编程  <fieldset> /\*标题嵌入边框的区块\*/  <legend>Personalia:</legend>  Name: <input class="class\_test" type="text"><br>  Email: <input class="class\_test" type="text"><br>  Date of birth: <input type="text">  喜欢的姑娘类型/\*下拉框\*/  <select name="favor\_girl\_type">  <option value="1">胸大腿长貌美</option>  <option value="2">小可爱</option>  <option value="3">微胖</option>  <optgroup label="区域">  <option>欧美</option>  <option>日韩</option>  <option>河南开封</option>  </optgroup>  </select>  </fieldset>  </form> |

## 2.css样式表

### 2.1属性

* 层叠样式表(英文全称：Cascading Style Sheets)是一种用来表现HTML（标准通用标记语言的一个应用）或XML（标准通用标记语言的一个子集）等文件样式的计算机语言

### 2.2功能

#### 内部样式表

|  |
| --- |
| 写在标题后  <style type = “text/css”>  div { // 前面加#号可以指定ID使用此样式  border :1px dash red;  width:80%;  heigth:500px  left:600px;  float:left; //浮动  clear:both;//清除之前的样式  }  </style> |

#### 外部样式表

1. 新建一个样式表文件

2. 在需要引用的html里的标题后加

<link rel=’stylesheet’ type=’text/css href=’外部样式表文件名’>

#### id选择器

内部样式表里用‘#id名’ { }，在代码里对应id会改样式

#### class类选择器

内部样式表里用‘ .类名’ { }，在代码里加入class = 类名就可以改

#### 后代选择器

在最外层id写样式，如“#id名 标签名 { }”

#### 子代选择器

在写样式的时候加上 > ,如 “#id名 > 标签名 { }”

#### 相邻兄弟选择器

在写样式的时候加上 >和+ ,如 “#id名 > 标签名+标签名2 { }”，注意是给id名相邻的下 1 个标签加

#### 组合选择器

直接写多个标签，写样式，如 标签1，标签2 { }

#### 常见格式

|  |
| --- |
| 内边距：padding，padding—left…  外边距：margin,margin-top…  边框：boder：1px dashed（solid） color  boder-bottom-color\style\width\ radius(边角) 下边框复合属性  背景：background:#ffffff url(图) no-repeat fixed center//颜色 图片 重复 位置  background-attachment-color-image-position-repeat-size  阴影：box-shadow：20 px 10px 10px #888888 |

#### 定位

|  |
| --- |
| 相对于页面定位  position:fix  top:50px  left:50px  相对于自己定位  position:relative  top:50px  left:50px |

#### font字体

|  |
| --- |
| font-weight:bold |

#### text文本

|  |
| --- |
| text-align（文字对齐） |

# 十一.爬虫

## 属性

* 网络爬虫（又称为网页蜘蛛，网络机器人，在FOAF社区中间，更经常的称为网页追逐者），是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。另外一些不常使用的名字还有蚂蚁、自动索引、模拟程序或者蠕虫

## 简例

|  |
| --- |
| import requests  import re  #爬取数据  url = 'https://movie.douban.com/chart'  header = {'User-agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.82 Safari/537.36'}  response = requests.get(url,headers=header)  # print(response.text)  date = response.text  #解析数据  date\_re = re.compile('<a.\*?nbg.\*?title="(.\*?)">',re.S)  date\_items = re.findall(date\_re,date)  print(date\_items)  #存储数据  with open('爬取数据.txt','w',encoding="utf-8") as f:  for item in date\_items:  f.write(item+'\n')  herder |

## 正则表达式

### 3.1字符匹配

#### 直接匹配

re.findall(‘abc’, ‘ddabc’) ->abc

#### . 匹配

refindall(‘a.c’, ‘acc’) ->acc 点可以代替任意字符（除\n）

refindall(‘a\.c’, ‘a.c’) -> a.c 点只当点使用

#### []匹配

refindall(‘a[bd]c’, ‘abc’) ->abc []中的都可以匹配

refindall(‘a\[b]c’, ‘abc’) ->abc 匹配a[b]c,，消除[]影响

#### 预定义的字符集

\d 匹配数字 \D匹配非数字 \s匹配空白字符（空格\t\r\n\f\v）

\S匹配非空白符 \w匹配大小写字母数字下划线中文 \W相反

#### 数量词

\* 匹配前一个字符0次或无限次

+ 匹配前一个字符1次或者无限次

？ 匹配前一个字符0次或1次

{m} 匹配前一个字符m次

{m,n} 匹配前一个字符m到n次

#### 贪婪匹配和懒惰匹配

贪婪匹配

\*

懒惰匹配

\*? 重复任意次，但尽可能少重复

+? 重复1次或更多次，但尽可能少重复

?? 重复0次或1次，但尽可能少重复

{n,m}? 重复n到m次，但尽可能少重复

{n,}? 重复n次以上，但尽可能少重复

### 3.2 re.findall()

* re.findall(正则表达式，str，flag=0)
* flag = re.DOTALL和re.S时，点匹配可以匹配换行符
* 分组的使用：当正则表达式中加了（）时，只返回（）里的内容
* 当()里加了(:?xx)则匹配xx但是不返回()里面()的内容

### 3.3元串的使用

* re.findall(r‘正则表达式，str，flag=0’) 消除转义字符的影响
* r元串可以解决不符合PEP8规范的问题

### 3.4 re. match

|  |
| --- |
| re.match尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match()就返回None。  re.match(pattern, string, flags=0)，其中pattern是匹配的正则表达式，string是要匹配的字符串，flags是标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。  如果匹配成功，re.match方法返回一个匹配的对象，否则返回None。我们可以使用group(num)或groups()匹配对象函数来获取匹配表达式。 |

### 3.4 re.seach

|  |
| --- |
| re.search扫描整个字符串并返回第一个成功的匹配。  re.search(pattern, string, flags=0)，其中pattern是匹配的正则表达式，string是要匹配的字符串，flags是标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。  如果匹配成功，re.search方法返回一个匹配的对象，否则返回None。我们可以使用group(num)或groups()匹配对象函数来获取匹配表达式。  **import** re s = **'我的姓名是刘双喜，我的性别是男，我的年龄是24岁'** res\_tuple = re.search(**'姓名是(.\*?)，.\*?性别是(.\*?)，.\*?年龄是(.\*?)岁'**, s) res\_dict = re.search(**'姓名是(?P<name>.\*?)，.\*?性别是(?P<sex>.\*?)，.\*?年龄是(?P<age>.\*?)岁'**, s) print(res\_tuple.groups()) print(res\_dict.groupdict())  输出：  ('刘双喜', '男', '24')  {'name': '刘双喜', 'sex': '男', 'age': '24'} |

### 3.5 re.sub

|  |
| --- |
| 作用：使用正则表达式替换字符串中的子字符串  re.sub() 函数的语法为：  re.sub(pattern, repl, string, count=0, flags=0)   * pattern - 正则表达式模式。 * repl - 替换字符串。可以是字符串或函数。 * string - 要搜索的输入字符串。 * count - 可选参数。模式匹配后替换的最大次数。默认值为 0，表示替换所有匹配项。 * flags - 可选参数。用于控制正则表达式匹配方式的标志。例如，可以使用 re.IGNORECASE 来执行不区分大小写的匹配。   re.IGNORECASE 或 re.I - 执行不区分大小写的匹配。  re.MULTILINE 或 re.M - 更改 ^ 和 $ 的行为，使它们分别匹配每行的开头和结尾，而不是整个字符串的开头和结尾。  re.DOTALL 或 re.S - 更改 . 的行为，使其匹配任何字符，包括换行符。  re.VERBOSE 或 re.X - 允许您在正则表达式中添加空格和注释，以使其更易于阅读。 |

### re.compile

|  |
| --- |
| import re content = 'Hello, I am Jerry, from Chongqing, a montain city, nice to meet you……' reg = re.compile('\\w\*o\\w\*') x = reg.findall(content) print(x) # 输出：['Hello', 'from', 'Chongqing', 'montain', 'to', 'you'] |

|  |
| --- |
| import re  # 编译一个正则表达式，匹配所有的单词 pattern = re.compile(r'[a-z]+', re.I)  # 使用findall()函数找到所有匹配的单词 result = pattern.findall('Hello, World! 123')  print(result) # 输出：['Hello', 'World'] |

# 十二.并发编程

一个进程中可以启动N个线程，一个线程中可以启动N个协程

## 多进程Process

* multiprocessing模块
* 多CPU并发
* 优点：可以利用多核CPU进行并行运算
* 缺点：占用资源最多，可启动数目比线程少
* 适用于IO密集型计算

### 1.GIL全局解释器锁

#### 1.python运行慢的原因

* 动态类型语言，边解释边执行
* GIL无法利用多核CPU并发执行

#### 2.GIL是什么

* 是计算机程序设计语言解释器用于同步线程的一种机制，它使得任何时刻仅有一个线程在执行
* 为了解决python设计初期多线程之间数据完整性和状态同步的问题
* 简化了对多线程的管理

#### 3.怎样规避GIL限制

* 在I/O期间，线程会释放GIL，实现cpu和io并行，因此多线程在IO密集计算仍然有用
* 使用multiprocessing模块

### 知识梳理

|  |
| --- |
|  |

### 示例

|  |
| --- |
| 、 |

## 2.多线程Thread

### 1.特点

* CPU和IO同时运行
* threading模块
* 优点：相比进程，更轻量级、占用资源少
* 缺点：相比进程，只能并发执行，不能利用多CPU；相比协程，启动数量有限制，占用内存资源，有线程切换开销
* 适用于IO密集型计算、同时运行的任务数目要求不多

### 2.使用

#### 1.基本使用

* 创建线程 t=threading.Thread(target=函数,args=参数)
* 执行线程t.start()
* 等待线程结束t.join()

|  |
| --- |
| import threading import time import requests  urls = [  f"https://www.shengxu5w.com/4\_4676/{1992837+page}.html"  for page in range(0,10) ] # 爬虫目标网址   def crew(url): # 爬取单个网页  r = requests.get(url)  print(url,len(r.text))   def single\_thread(): # 单线程for循环爬取  for url in urls:  crew(url)   def multi\_thread(): # for循环创建多个爬取线程，再for循环执行线程  threads = []  for url in urls:  threads.append(threading.Thread(target=crew,args=(url,)))   for thread in threads:  thread.start()   for thread in threads:  thread.join()   if \_\_name\_\_ == '\_\_main\_\_':  start = time.time()  multi\_thread()  end = time.time()  print(f"多线程时间是{end-start}s")   start = time.time()  single\_thread()  end = time.time()  print(f"单线程时间是{end-start}s") |

#### 2.线程通讯queue

* 导入库import queue
* 创建Queue ：q=queue.Queue()
* 添加元素q.put(item)
* 获取元素item = q.get()
* 查看元素多少q.qsize()
* 查看是否为空q.empty()
* 判断是否满q.full()

|  |
| --- |
| 创建线程3，html\_q.get一个爬取的文本进行解析，写入到文本f  创建线程3，html\_q.get一个爬取的文本进行解析，写入到文本f  打开文本文件为f  创建线程2，url\_q.get一个url进行爬取文本，添加到html\_q.put(r.text),循环  创建线程1，url\_q.get一个url进行爬取文本，添加到html\_q.put(r.text),循环  将所有网址添加到url\_q：  url\_q.put(url)  创建queue：url\_q和html—\_q |

#### 线程安全

* 线程安全指某个函数、函数库在多线程环境中被调用时，能够正确的处理多个线程之间的共享变量，使程序功能正确完成
* 在创建线程的target的执行内容里，将内容主体放在with Lock()：中

#### 线程池

|  |
| --- |
| ----------------------------------------------------------------    ----------------------------------------------------------------    -------------------------------示例----------------------------- |

* 提升性能：因为减去了大量新建、终止线程的开销，重用了线程资源
* 适用场景：适合处理突发性大量请求或需要大量线程完成任务但实际任务处理时间较短
* 防御功能：能有效避免系统因创建线程过多，而导致系统负荷过大响应变慢的问题
* 代码优势：使用线程池的语法比自己新建的语法更加简洁

## 3.多协程Coroutine

* asyncio模块
* 在单线程利用CUP和IO同时执行的原理，实现函数的异步运行
* 优点：内存开销最少，启动协程数目最多可达几万
* 缺点：支持的库有限，代码实现复杂
* 适用于IO密集型计算、需要超多任务运行，但有现成库支持的场景

|  |
| --- |
| -------------------------------示例----------------------------- |

## 4.python对并发编程的支持

* Lock：对资源加锁，防止同时访问产生的冲突访问
* Queue：实现不同进程/线程之间的数据通信，实现生产者-消费者模式
* Pool：线程池/进程池，简化线程/进程的任务提交、等待结果、获取结果
* subprocess：启动外部程序进程，并进行输入输出交互

# 十三.测试代码

## 测试函数

* unittest.main()会只执行类中以test开头的函数
* 断言函数会对测试进行结果对比
* 会先执行setUp函数里的内容，相当于初始化
* assertEqual(a,b) 核实a == b
* assertNotEqual(a,b) 核实a != b
* assertTrue(x) 核实x为Ture
* assertFalse(x) 核实x为False
* assertIn(item,list) 核实item在list里
* assertNotIn(item,list) 核实item不在list里

|  |
| --- |
| def get\_formatted\_name(first, last): #拼接姓和名  full\_name = first + ' ' + last  return full\_name.title()  import unittest  class NamesTestCase(unittest.TestCase):    def setUp(self):  self.formatted\_name = get\_formatted\_name('janis', 'joplin')  def test\_first\_last\_name(self):  self.assertEqual(formatted\_name, 'Janis Joplin')  unittest.main() |

# 十四.Linux

## 快捷键

* tab补全
* ctrl+l清空 c强制终止当前命令 a光标移动到行首 e光标到行尾 u删除光标行
* ctrl+z把命令放到后台 bg恢复
* ctrl+r在历史记录里搜索
* ctrl+shilt+t 打开第二个终端

## 常用命令

* ls当前目录里的清单
* ls –a 全部文件all
* ls –l详细信息 long
* ls -i 文件目录和标识 inum
* ls –d 目录 dir
* cd /到根目录；cd /xx/xx切换目录
* pwd当前目录路径
* cd .. 去上级目录
* touch 新建
* history查看历史 ！+历史命令序号->执行历史命令
* date 输出当前时间
* 连接符&&和||
* sudo apt install xx 安装
* sudo apt remove xx 卸载
* sudo apt upgrade xx 更新
* cal –y 查看日历
* df –h查看磁盘空间 du –h xx 查看xx文件大小
* 进程：ps a显示所有进程 ps u显示进程的详细状态 ps x 显示没有控制终端的进程
* top 动态显示运行中的进程；kill xxx 关掉pid为xxx的进程；kill -9 xxx 强制关闭
* tree 显示文件，以树状显示
* yum install -y lrzsz 复制文件到服务器输入：rz

## 文件操作

* 以.开头命名的文件是隐藏的
* 文件权限组成：第一位是文件类型：-是二进制文件，d是目录，l是链接
* whatis xx 告诉xx命令是干嘛的；whereis xx 告诉xx命令在哪；
* mkdir 创建文件夹；mkdir –p xx/xx 创建多重文件夹
* rmdir 删除空文件夹
* cp xx yy/yy/dd -a 复制xx文件的所有属性到yy/yy路径下重命名为dd
* cp –r xx yy 复制文件夹xx到yy
* mv xx yy/yy/dd 移动xx文件夹的所有属性到yy/yy路径下重命名为dd
* rm xx 删除文件；rm –r xx 删除文件夹
* cat查看文件；more 查看文件
* vim打开文件；i插入；esc，:w保存，q退出，x保存退出，n切换到下一个文件，prev切换到上一个文件，set nu显示行数，：3 跳到3行；vim +3 xx打开文件到第3行；命令模式下按dd删除当前行；4yy，复制4行，p粘贴；v视图，可多选
* location模糊查询
* sudo updatedb 更新数据库
* sudo find / -name xx 从根目录查询xx文件，xx文件可以用正则表达式 “xx\*？”
* **sudo find / -iname xx 从根目录查询xx文件，不区分大小写**
* find / -size +3M –a –size -4M 在根目录下找大于3M小于4M的文件，-a是和，-o或
* find / -size +3M –a –type f 查找大于3M的文件 type d目录l链接link
* find / -size +3M –exec ls –lh {}\; 查找大于1M的文件并显示详细信息
* find / -size +3M –exec rm –rf {}\; 查找大于1M的文件并显示详细信息
* find / -ctime +10 查找10天前改变过属性的文件 #atime访问过的 #ctime改变属性的
* #mtime改变过内容的 #amin 按分钟查找
* find / -inum 306413 按文件节点查找
* grep –i “xx” 1.txt 查找xx有没有在文件里 –i忽略大小写

## 用户和用户组

* su root 切换到管理员
* useradd xx –m 添加用户xx并且创建xx的根目录；sudo passwd xx 为xx用户设密码
* /etc/passwd下存放当前系统所有用户信息，格式为 用户名：密码占位符：用户编号：用户组编号：用户注释信息：用户主目录：shell类型
* su - xx 切换用户并且到根目录 exit退出用户
* sudo userdel xx 删除用户
* /etc/group 下存放当前所有组信息，格式为 组名称：组密码占位符：组编号：组中用户名列表
* sudo groupadd xx 添加用户组 sudo groupdel xx删除
* /etc/gshadow 下存放用户组密码 /etc/shadow 下存放用户密码
* 用户组命令：



* 用户命令;



## 权限管理

* 文件权限：第一位文件类型，第234位所有者（u）权限，第567位所属组（g）权限，第8910位其他人（o）权限。修改 chmod u+x xx 给xx文件加所有者可执行权限+-=
* r是4， w是2， x是1； 例755是rwxr-xr-x；； chmod 755 xx
* 权限对目录的作用：r可查询目录，w可增删目录里的文件，x可进入目录
* chown liusx:yy xx 把xx文件的所有者修改成liusx，所属组修改成yy
* chgrp liusx xx 把xx文件的所属组改变成liusx
* umask 文件默认权限 0022 文件权限=文件最大权限字母-默认权限字母

## 压缩和打包

* zip xxx.zip yy 把yy文件压缩，为xxx.zip文件
* zip –r xxx.zip yy 把yy目录压缩
* unzip xx.zip 解压缩
* gzip xx 把xx文件压缩，xx会被删除 gzip -c xx > xx.gz 把xx压缩，xx会保留
* gzip -r xx 把xx目录内的所有文件压缩，但是不压缩目录
* gzip –d xx.gz 解压缩
* gunzip xx.gz 解压缩
* bzip xx 把xx压缩成xx.bz2，xx会被删除 bzip –k xx 压缩xx不会被删除
* bzip2 –d xx.bz2 bunzip xx.bz2 解压缩 bunzip –k xx.bz2 解压缩，压缩文件会被保留
* tar –cvf xx.tar yy zz 把yy和zz打包成xx,tar
* tar –xvf xx.tar 解打包
* tar –zcvf xx.tar.gz yy zz 把yy和zz打包再压缩成xx.tar.gz; tar –zxvf xx.tar,gz 解打包
* tar –jcvf xx.tar.bz2 yy zz 把yy和zz打包再压缩成xx.tar.bz2; tar –xxvf xx.tar,bz2 解打包
* tar –ztvf xx.tar.gz 不解压，只查看

## vim文本操作

* vim命令模式：2dd删除两行；2yy复制光标向下4行；p粘贴；u撤销；alt+u反撤销；d0光标当前位置剪切到首行；x删除光标位置后面的，X删除光标前面
* vim输入模式：i当前位置输入；I当前行首输入；光标后输入；A行末输入；
* vim底行模式：w保存；q退出；ls显示当前打开的所有文本；n下一个；prew上一个；3光标停在第3行

# 十五.redis数据库

## radis

### 1.1属性

* 支持数据备份
* 非关系型数据库
* 定时从内存保存数据到硬盘
* 不仅仅支持key-value类型，还支持list、set、hash、hash、zset、str数据类型
* 主要用于快速存取
* 官网：<https://redis.io/>
* 命令地址：<http://doc.redisfans.com/>

### 1.2常用命令

* 打开服务：sudo service redis start
* 关闭服务：sudo service redis stop
* 连接：redis-cli
* 连接主服务器：redis-cli –p 6379 –h 192.168.30.130
* 连接从服务器：redis-cli –p 6378 –h 192.168.30.130
* 切换到数据库3：select 2
* 清空当前数据库中的key的数量：FLUSHDB
* 清空所有数据库的key：FLUSHALL
* 判断key是否存在：exists key

### 1.3五大数据类型

#### 1.3.1字符串string

##### 属性

* 一个key对应一个value
* 可以包含任何数据，最大不能超过512M

##### 常用命令

* 设置单个值set k1 v1
* 设置多个值mset k1 v1 k2 v2
* 索引2开始替换关键字的值setrange k1 2 xxx
* 获取值get k1
* 获取多个值mget k1 k2
* 0到2范围取值getrange k1 0 2
* 添加append k1 v2
* 删除del k1
* 求长度strlen k1
* 关键字的值加1递增incr k1
* 关键字的值减一递减decr k1
* 关键字的值加x递增incrby k1 x
* 关键字的值减x递减decrby k1 x

#### 1.3.2列表list

##### 属性

* 底层实际是个链表
* 可分为头部插入列表和尾部插入列表

##### 常用命令

* 从右部插入列表（正序排列）rpush list01 1 2 3 4 5
* 从左部插入列表（倒序排列）lpush list02 1 2 3 4 5
* 获取指定长度的值lrange list01 2
* 按索引取值lindex list01 2
* 获取列表中所有的值lrange list01 0 -1
* 删除最左/最右的值lpop/rpop list01
* 求长度llen list01
* 删除n个值为2命令lrem list01 n 2
* 按索引0到2截取列表的值ltrim list01 0 2
* 把list01的最后一位剪切添加到list02的第一位rpoplpush list01 list02
* 把索引2的值替换为xx值lset list01 2 xx
* 在x值之前/之后添加y值linset list01 before/after x y

#### 1.3.3哈希hash

##### 属性

* hash是键值对的一个集合，也就是一个key下多个field和value映射
* Redis hash 是一个 string 类型的 field（**字段**） 和 value（**值**） 的映射表，hash 特别适合用于存储对象
* Redis 中每个 hash 可以存储 232 - 1 键值对（40多亿）

##### 常用命令

* 设置单个字段的值hset **user** id 1
* 设置多个字段值hmset **user** id 1 name liusx age 18
* 取一个字段值hget **user** id
* 取多个字段值hmget **user** id name age
* 取所有字段和值hgetall **user**
* 单独取所有字段hkeys **user**
* 单独取所有值hvals **user**
* 删除del **user**
* 求长度（按照值来）hlen **user**
* 判断字段是是否在hash里hexists **user** id

#### 1.3.4集合set

##### 属性

* set是string类型的无序集合，是不重复的

##### 常用命令

* 添加集合元素sadd set01 1 2 3 4
* 查看集合元素smembers set01
* 查看集合元素个数scard set01
* 删除为3的元素srem set01 3
* 移动set01里的为2的元素到set02里smove set01 set02 2
* 判断元素3是否在集合内sismember set01 3
* 随机返回集合中2个元素srandmember set01 2
* 随机出栈，删除2个元素spop set01 2
* 差集sdiff set01 set02 返回在set01不在set02里的
* 交集sinter set01 set02
* 并集sunion set01 set02

#### 1.3.5有序集合

##### 属性

* zset的值是分数对的，一个分数一个值
* zset是按照分数排序的

##### 常用命令

* 添加元素zadd
* 查看所有值zrange z1 0 -1
* 查看带分数返回的值zrange z1 0 -1 withscores
* 根据开始和结束的分数取值zrangebyscore z1 60 80
* 据开始和结束的分数取值,不包括结束的zrangebyscore z1 60 （80
* 据开始和结束的分数取值，从索引1开始取3位zrangebyscore z1 60 80 limit 1 3
* 删除值z1和对应分数zrem z1 k1
* 求集合的元素的个数（分数对的数量）zcard z1
* 求分数60到90的值个数zcount z1 60 90
* 查看值的索引zrank z1 k1

## python操作redis数据库

### 2.1步骤

1. Ubuntu创建文件touch xx.py
2. 进入编辑vim xx.py
3. 另开一个窗口连接redis查看调试redis-cli
4. 写代码，调用redis模块赋值给一个对象
5. 通过对对象的操作来操作数据库
6. 另开一个窗口运行文件python3 xx.py

### 2.2代码基本格式

|  |
| --- |
| import redis    class TestString(object):  def \_\_init\_\_(self):  # 地址 端口号 数据库默认db=0  self.r = redis.StrictRedis(host = 'localhost', port = 6379)  # 设值  def set\_string(self):  # 返回值 布尔类型  res = self.r.set('name','liusx')  print(res)    # 设值  def set\_string(self):  # 返回值 布尔类型  res = self.r.set('name','liusx')  print(res)  # 取值 是字节类型  def get\_string(self):  res = self.r.get('name')  res\_str = res.decode()  print(res)  print(res\_str)  res = self.r.set('name','liusx')  print(res)  # 取值 是字节类型  def get\_string(self):  res = self.r.get('name')  res\_str = res.decode()  print(res)  print(res\_str)    # 取多个值  def get\_mstring(self):  l = ['age','sex']  res = self.r.mget(l)  print(res)  # 设置多个值  def set\_mstring(self):  d = {  def get\_string(self):  res = self.r.get('name')  res\_str = res.decode()  print(res)  print(res\_str)  # 取多个值  def get\_mstring(self):  l = ['age','sex']  res = self.r.mget(l)  print(res.decode)  # 设置多个值  def set\_mstring(self):  res\_str = res.decode()  print(res)  print(res\_str)  # 取多个值  def get\_mstring(self):  l = ['age','sex']  res = self.r.mget(l)  print(res)  # 设置多个值  def set\_mstring(self):  d = {  'age':'21',  'sex':'man'  }  res = self.r.mset(d)  # 删除  def del\_string(self,name):  res = self.r.delete(name)  print(res)  def main():  r = TestString()  r.set\_string()  print('取一个值')  r.get\_string()  r.set\_mstring()  print('取多个值')  r.get\_mstring()  if \_\_name\_\_ == '\_\_main\_\_':  main() |

## 设置主从数据库

### 3.1属性

* 主数据 库用于写入，也可以读取；从数据库只能读取
* 可以完成热备份

### 3.2步骤

1. 进入到/etc/redis，sudo vim打开redis.conf配置文件
2. 进入命令模式，输入：/bind查找bind，修改为本机ip
3. 复制主数据库配置文件，命名为从数据库cp redis.conf slave.conf
4. sudo vim打开slave.conf,查找port修改为6378
5. 查找slaveof，插入slaveof 主数据库ip 主数据库端口号
6. 分别连接主从数据库redis-cli –p 6379 –h 192.168.30.130

# 十六.MySQL数据库

## MySQL

### 1.1属性

* 持久化存储
* 读写速度极高
* 保证数据的有效性，可以存放各种数据类型
* 支持多个操作系统：Windows、Linux、Mac OS
* 支持多线程
* 为多种编译语言提供API，包括C、JAVE、Python
* MySQL属于关系型数据库

### 1.2window上操作mysql

1. 网络下载phpstudy2016，在其他选项菜单里的MySQL工具里的MySQL命令行打开，密码root，创建一个数据库db1；
2. 网络搜索下载SQLyog
3. 安装到D盘，打开创建新连接，填写密码和phpstudy创建的数据库名
4. 客户端操作增删表和数据：输入命令，ctrl+F9执行所有命令

### 1.3Ubuntu上操作mysql

1. 安装服务命令sudo apt-get install mysql-server
2. 安装客户端命令sudo apt-get install mysql-client
3. 连接数据库mysql -uroot –p
4. 通过命令操作

### 1.4常用命令

* 启动/停止服务sudo service mysql start/stop
* 查看进程中是否有mysql服务ps ajx | grep mysql
* 重启服务sudo service mysql restart
* 数据库连接mysql –uroot –p
* 退出数据库quit或exit
* 配置文件目录/etc/mysql
* 数据库文件存放目录/var/lib/mysql-files/
* 查看版本select version();
* 显示时间select now()
* ---------------------**SQL语句**---------------------
* -------------------**DDL数据定义语句**------------------
* 用哪个数据库use mysql;
* 创建数据库create database `xxx`
* 查看有哪些数据库show databases;
* 创建表create table xxx(id int auto\_increment primary key,name varchar(200) not null)default charset ‘utf8’;
* 查看有哪些表show tables;
* 查看表的具体属性信息desc xxx;
* 删除表drop table xxx
* 删除数据库drop database xxx
* 创建数据库指定字符编码create database xxx charset utf8;
* -------------------**DML数据管理语句**------------------
* ----------**增**-----------
* 表数据新增insert into table1 value(1,’刘双喜’);
* 表数据新增，序号自动递增insert into table1(`name`) value(‘刘双喜’);(!!注意name左右的是tab键上一个键的符号)
* 插入多条insert into table1(‘name’) value(‘刘双喜’),(‘喜双刘’);
* ----------**查**------------
* 查看表所有字段数据select \* from tab1;
* 查看表的id和name字段数据select id,name from tab1;（字段是别名，前面加as）
* 查看表name字段数据为liusx的数据select name from tab1 where name=’liusx’;
* 查看表name字段数据为liusx的数据,且id排序为正序/逆序select id,name from tab1 where name=’liusx’ order by id asc/desc;
* 查看表name字段数据为liusx的数据,且id排序为正序/逆序,只看从1开始，限制2条 select id,name from tab1 where name=’liusx’ order by id asc/desc limit 1,2;
* 模糊查询带sx的name字段里的数据（%相当于通配符，可以之加前或后）select name from tab1 where name like ‘%sx%’
* ----------**改**-----------
* 修改name字段id为1的数据update tab1 set name= ‘ntt’ where id = 1;
* ----------**删**-----------
* 删除delete from tab1 where id = 1;

## Python操作MySQl数据库

### 安装模块

1. sudo apt-get install mysql-server mysql-client
2. sudo apt-get install libmysqlclient-dev python3-dev
3. pip3 install mysqlclient
4. pip3 install pymysql

### 步骤

1. 导入pymysql模块
2. 连接数据库，赋予一个对象

|  |
| --- |
| try:  self.db = pymysql.connect(host='127.0.0.1', user='root', passwd='root', db='db1', charset='utf8')  except Exception as e:  print(e)  print('------error---------')  else:  print('连接成功')  self.conok = 1 |

1. 查

|  |
| --- |
| cur = self.db.cursor()  # SQL语句  cur.execute('select \* from tab1')  # 将数据库表中数据匹配拼接成字典，查一个fetchone（）  result = dict(cur.fetchall())  print('dict is :', result)  cur.close() |

1. 添加

|  |
| --- |
| cur = self.db.cursor()  cur.execute("insert into tab1(`name`) value('a1')")  # 提交事务  self.db.commit()  cur.close() |

1. 关闭数据库连接

|  |
| --- |
| try:  if self.conok == 1:  self.db.close()  except Exception as e:  print(e) |

### 代码实例

|  |
| --- |
| import pymysql   class MysqlDb(object):  def \_\_init\_\_(self):  self.conok = 0  self.connect()   def connect(self):  try:  self.db = pymysql.connect(host='127.0.0.1', user='root', passwd='root', db='db1', charset='utf8')  except Exception as e:  print(e)  print('------error---------')  else:  print('连接成功')  self.conok = 1   def get\_one(self):  # 创建游标，为了后续数据库操作  cur = self.db.cursor()  # SQL语句  cur.execute('select \* from tab1')  # print('cur.description:',cur.description)  # print('cur.fetchone:',cur.fetchone())  # 将数据库表中数据匹配拼接成字典  result = dict(zip([k[0] for k in cur.description], cur.fetchone()))  print('dict is :', result)  cur.close()  return result   def get\_more(self):  # 创建游标，为了后续数据库操作  cur = self.db.cursor()  # SQL语句  cur.execute('select \* from tab1')  # print('cur.description:',cur.description)  # print('cur.fetchall:',cur.fetchall())  # 将数据库表中数据匹配拼接成字典  result = dict(cur.fetchall())  print('dict is :', result)  cur.close()  return result   def add\_one(self):  try:  cur = self.db.cursor()  cur.execute("insert into tab1(`name`) value('a1')")  cur.execute("insert into tab1(`name`) values('a2')")  # 提交事务  self.db.commit()  cur.close()  except Exception as e:  print(e)  print('error')   def close(self):  try:  if self.conok == 1:  self.db.close()  except Exception as e:  print(e)   def main():  obj = MysqlDb()  # print(obj.get\_one()['name'])  obj.get\_more()  obj.add\_one()  obj.get\_more()  obj.close()   if \_\_name\_\_ == '\_\_main\_\_':  main() |

# 十七. MongoDB数据库

## MongoDB

### 属性

* 无数据结构限制，每条业务可以有完全不同的结构
* 中文社区[www.mongoing.com](http://www.mongoing.com)

### 在Ubuntu上安装

命令sudo apt-get install mongodb

### 常用命令

* 开启服务sudo service mongodb start
* 关闭服务sudo service mongodb stop
* 重启服务sudo service mongodb restart
* 查看服务是是否开启ps aux | grep mongodb
* 查看配置文件 cat /etc/mongodb.conf
* 连接mongodb数据库mongodb
* ----------DLL数据定义语句------------
* **查**看数据库show databases/dbs
* **查**看在哪个数据库下db
* **用**数据库use xxx
* **增加**数据库（数据库里必须要有数据才会保存数据库）use xxx
* **删**除数据库,当前在用哪个数据库就删除哪个数据库db.dropDatabase()
* **退**出数据库db.shutdownServer()
* **增**加集合,并设置字节上限（可不写）db.createCollection(‘xxx’,{capped:true,size:10})
* **查**看所有集合show collections
* **删除col**集合db.col.drop()
* ----------DML数据管理语句-------------
* **增**加集合col的数据db.col.insert({name:’liusx’})
* **增**加集合col的数据(按照id)db.col.insert({name:’machao’,\_id:1})
* **增**加多条for(i=1;i<3;i++)db.col.insert({x:i})
* **查**集合col所有数据db.col.find()
* **查**集合col一条数据db.col.findOne()
* **查**集合指定id的数据db.col.find({"\_id" : ObjectId("61a4e5c6e6117305ba9fe4ea")})
* **查**集合数据总数db.col.find().count()
* **查**指定数量2条集合col的数据db.col.find().limit(2)
* **查**集合col的跳过前2条数据剩余的数据db.col.find().skip(2)
* **查**集合col的跳过前2条剩余数据里的前2条数据db.col.find().limit(2).skip(2)
* **查**集合col的只显示指定变量x的所有数据db.col.find({},{x=1})
* **查**集合col的指定变量之外的数据db.col.find({},{\_id:0})
* **查**集合col的变量x为3的数据，且只显示它db.col.find({x:3},{x:1})
* **排序显示**(以变量x的值为准，1是升序，-1降序)db.col.find().sort({x:1})
* **比较运算符查询**（小于：$lt，小于等于：$lte，大于：$gt，大于等于：$gte，默认是等于）
* **查询**x值大于1的数据db.col.find({x:{$gt:1}})
* **查询**x值为1或者3的数据db.col.find({x:{$in:[1,3]}})
* **查询**x值不为1或者3的数据db.col.find({x:{$nin:[1,3]}})
* **查询**x的值为2或者id为2的数据db.col.find({$or:[{x:2},{\_id:2}]})
* **改变**数据,只改age（第一个参数是条件，第二个参数是修改的内容）db.col.update({name:’liusx’},{$set:{age:22}})
* **改变**数据,全部更新为输入的内容（第一个参数是条件，第二个参数是修改的内容）db.col.update({name:’liusx’},{{age:22}})
* **改变**数据,只改age，且匹配到的多条都改（第一个参数是条件，第二个参数是修改的内容）db.col.update({name:’liusx’},{$set:{age:22}},{multi:true})
* **改**变集合col的数据db.col.save({name:’liujiang’,\_id=1})
* **删除**集合所有数据db.col.remove({})
* **删除**集合特定数据db.col.remove({x:1})

### 数据类型

* string：字符串，必须是UTF-8
* boolea：布尔类型，true或者false
* integer：整数
* double：浮点数
* arrays：数组或列表
* object：嵌入式文档，相当于对象
* null：空
* objecID：是一个12字节的16进制数：前4个字节是时间戳，接下来3个是机器ID，接下来2个是mongoDB的服务器进程id，最后3个字节是增量

## Python操作MongoDB数据库

### 安装模块

命令pip3 install pymongo

### 在Ubuntu上操作MongoDB

#### 导入模块

|  |
| --- |
| from pymongo import MongoClient  from bson.objectid import ObjectId |

#### 连接

|  |
| --- |
| self.mc = MongoClient('localhost', 27017)  self.db = self.mc['db1'] |

#### 增加一个数据

|  |
| --- |
| res = self.db.col.insert({'name': 'liusx', 'age': 23, 'sex': 'man'}) |

#### 增加多个数据

|  |
| --- |
| date = [{'name': 'machao', 'age': 22, 'sex': 'man'}, {'name': 'liujb', 'age': 24, 'sex': 'man'}]  res = self.db.col.insert\_many(date) |

#### 查找一个数据

|  |
| --- |
| res = self.db.col.find\_one() |

#### 查找多个数据

|  |
| --- |
| res = self.db.col.find() |

#### 按id查找数据

|  |
| --- |
| res = self.db.col.find\_one({"\_id": ObjectId("61a4e5c6e6117305ba9fe4ea")}) |

#### 修改一个数据

|  |
| --- |
| res = self.db.col.update\_one({'name': 'liujb'}, {'$set': {'name': 'liujiang'}}) |

#### 修改多个数据

|  |
| --- |
| res = self.db.col.update\_many({}, {'$set': {'name': 'liuji'}}) |

#### 删除一个数据

|  |
| --- |
| res = self.db.col.delete\_one({'name': 'liuji'}) |

#### 删除多个数据

|  |
| --- |
| res = self.db.col.delete\_many({'name': 'liuji'}) |

#### 游标对象常用方法

|  |
| --- |
| next(): 返回游标中的下一个文档。当游标中没有更多文档时，会抛出 StopIteration 异常。  hasNext():判断是否还有下一个文档。  limit():限制游标返回文档的数量。  skip():跳过指定数量的文档。  sort():按照指定字段排序返回文档。  toArray():将游标中所有文档转为数组。  forEach():对游标中每个文档执行指定操作。  \_\_getitem\_\_() 或 []: 通过索引访问游标中的文档。索引可以是正数或负数，表示从末尾开始计数的位置。  \_\_iter\_\_(): 返回游标对象本身，使游标支持迭代。当使用 for 循环遍历游标时，会自动调用此方法。  count(): 返回游标中的文档数量。如果游标支持分页，可能需要使用 no\_cursor\_timeout() 方法来等待所有结果返回。  sort(): 对游标中的文档进行排序。可以按照一个或多个字段进行排序，并指定排序顺序（升序或降序）。  limit(): 限制游标返回的文档数量。可以指定一个整数作为限制的数量。  skip(): 跳过游标中的一定数量的文档。可以指定要跳过的文档数量。  distinct(): 返回指定字段的所有不同值。  find\_one(): 返回游标中的第一个文档。如果游标为空，则返回 None。  update\_many(): 更新满足条件的多个文档。可以使用更新操作符或更新文档来指定更新内容。  delete\_many(): 删除满足条件的多个文档。  find\_and\_modify(): 修改并返回满足条件的单个文档。可以同时执行更新和获取操作。  explain(): 返回查询的执行计划信息，帮助调试和优化查询性能。  no\_cursor\_timeout(): 防止游标超时。在某些情况下，查询可能需要很长时间才能完成，使用此方法可以确保游标保持打开状态直到所有结果返回。  close(): 关闭游标。一旦关闭游标，就不能再访问其中的文档。 |

#### 操作符

|  |
| --- |
|  |

# 十八. 综合应用

## git环境搭建和github下载代码

### 下载git软件

* <https://git-scm.com/download/win>
* 打开安装包，一直确定就行

### 配置git本地仓库

* 在**桌面**右键-git bash
* 输入以下命令：
* 设置用户名git config --global user.name ‘xxx’
* 设置邮箱git config --global user.email ‘xxx@qq.com’
* 查看git config --list

### 创建密匙

* ssh-keygen -t rsa -C [xxx@qq.com](mailto:xxx@qq.com)
* 一直回车

### 配置github账号

* <https://github.com>
* 创建账号，邮箱和邮件验证码就行

### 添加密匙



### 下载github上别人的项目

* 搜索项目名称，点进项目界面



* 复制项目地址



* 在自己电脑上创一个文件夹，将代码下载到创建的文件夹里
* 在文件夹右键git bash，输入git clone 和刚才复制的ssh地址



* 回车，遇到提示输入yes

### TortoiseGit使用

* 官网下载直接安装汉化包
* 配置设置里的Git的用户信息
* 网络里的ssh路径C:\Program Files (x86)\Git\usr\bin\ssh.exe

### push大文件

* 注意文件名不要有空格

|  |
| --- |
| 1. Git LFS下载<https://git-lfs.github.com/> 2. 检查安装git lfs install 3. 设置需要跟踪的文件git lfs track a.zip(要上传的文件) 4. 添加修改git add .gitattributes 5. 提交修改git commit -m ‘文件说明’ 6. 上传修改git push 7. 添加大文件git add a.zip(要上传的文件) 8. 提交大文件git commit -m ‘文件说明’ 9. 上传大文件git push   注:避免空格和特殊符号  注意压缩文件的后缀,有zip,有rar |

## 自动化脚本下载git代码

|  |
| --- |
| import time  from pynput.keyboard import Key, Controller  import win32api  import subprocess  def key\_done(string):  keyboard = Controller()  for i in string:  keyboard.press(i)  keyboard.release(i)  keyboard.press(Key.enter)  keyboard.release(Key.enter)  # 给命令生效时间  time.sleep(0.5)  # 打开git bash  git\_address="C:\Program Files (x86)\Git\git-bash.exe"  # win32api.ShellExecute(0, 'open', file, '', '', 0) # 后台执行  win32api.ShellExecute(0, 'open', git\_address, '', '', 1) # 前台打开  time.sleep(2)  # 在git窗口切换成英文输入  keyboard = Controller()  keyboard.press(Key.shift\_l)  keyboard.release(Key.shift\_l)  # 下载代码  cd\_apussvn = 'cd d:/apussvn'  apus\_clone\_cmd='clone avxddd'  cd\_iv = 'cd d:/apussvn/ApusAt/IV'  iv\_clone\_cmd = 'clone dds'  key\_done(cd\_apussvn)  key\_done(apus\_clone\_cmd)  key\_done(cd\_iv)  key\_done(iv\_clone\_cmd)  time.sleep(2)  # 关闭git进程  subprocess.call('taskkill /F /IM '+ 'git-bash.exe', creationflags=subprocess.CREATE\_NO\_WINDOW) |

## Git Push疑难杂症

|  |
| --- |
| 一个本地仓库不要push给多个远程仓库，上传太大的内容分步commit  例1：  fatal: sha1 file '<stdout>' write error: Broken pipe  fatal: the remote end hung up unexpectedly  改变提交缓存大小  git config --global http.postBuffer 1048576000  配置git的最低速度和最低速度时间  git config --global http.lowSpeedLimit 0  git config --global http.lowSpeedTime 999999  更改shh上传发送空包时间和数量  C:\Program Files (x86)\Git\etc\ssh\sshd\_config加上  ClientAliveInterval 30  ClientAliveCountMax 14400  例2：  git报错10054  git config --global http.sslVerify “false”  例3：  Connection reset by 20.205.243.166 port 22  fatal: Could not read from remote repository.  Please make sure you have the correct access rights  and the repository exists.  DNS被加速器等修改了。  解决方法1：  Cmd: ipconfig /flushdns  解决方法2：  换网络。关闭NAS等  解决方法3：  切换代理节点；  解决方法3：  [坑：ssh: connect to host github.com port 22: Connection refused - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/521340971) |