# 入门

## python速查表

### 数据类型

- \*\*Integer 整数\*\*：例子 25或 -15

- \*\*String 字符串\*\*：例子 "Hello, 你好"

- \*\*Boolean 布尔值\*\*：例子 True, False

- \*\*List 列表\*\*：例子 [value1, value2,...]

- \*\*Tuple 元组\*\*：例子 (value1, value2,...)

- \*\*Dictionary 字典\*\*：例子 {key1: value1, key2: value2,...}

- \*\*Set 集合\*\*：例子 {value1, value2,...}

### 语句

- \*\*if语句\*\*

- if 表达式:

- elif 表达式:

- else:

- \*\*while循环\*\*

- while 表达式:

- 语句

- \*\*for循环\*\*

- for var in 集合列表:

- 语句

- \*\*for与range循环\*\*

- for i in range(开始值, 结束值 [, 间隔]):

- 语句

- #结束值是不包含的，开始值包含

### 函数定义

- def name(arg1, arg2,...):

- return 返回值

### 算术运算符

- x + y ：加

- x - y ：减

- x \* y ：乘

- x / y ：除

- x % y ：取模,返回除法的余数

- x \*\* y ：幂,返回x为y次幂

- x // y ：整除

### 比较运算符

- x < y ：小于

- x <= y ：小于等于

- x > y ：大于

- x >= y ：大于等于

- x == y ：相等

- x != y ：不等于

### 布尔运算符

- not x ：非

- x and y ：且

- x or y ：或

### 转换函数

- int(exp) ：将exp转成整型

- float(exp) ：将exp转成浮点型

- str(exp) ：将exp转成字符串

- chr(num) ：将数字转成ASCII字符

### String字符串 / List列表 / Tuple操作

- \*\*len(s)\*\* ：长度

- \*\*s[i]\*\* ：取s中的第i个值 (从0开始)

- \*\*s[start:end]\*\* ：从开始 (包括) 到结束 (不包括) 的片段

- \*\*x in s\*\* ：如果x包含在s中则为True

- \*\*x not in s\*\* ：如果x不包含在s中，则为True

- \*\*s + t\*\* ：把t与s相连接

- \*\*s \* n\*\* ：将s重复n份

- \*\*sorted(s)\*\* ：对s进行排序

- \*\*s.index(item)\*\* ：item在s中的位置

### 字符串 String

#### 查找与替换

- \*\*count(sub)\*\* ：统计字符串里某个子字符串出现的次数

- \*\*find(sub)\*\* ：检测返回sub索引位置，否则返回-1

- \*\*rfind(sub)\*\* ：从后面开始查找返回sub索引位置

- \*\*index(sub)\*\* ：检测返回sub索引位置，否则抛出异常

- \*\*rindex(sub)\*\* ：从后面开始查找返回sub索引位置

- \*\*replace(old,new)\*\* ：用new替换old

- 前第五个方法可以接受start, end结束位置参数

#### 性质判定

- \*\*isalnum()\*\* ：是否全是字母和数字

- \*\*isalpha()\*\* ：是否全是字母

- \*\*isdigit()\*\* ：是否全是数字

- \*\*isdecimal()\*\* ：是否全是小数

- \*\*islower()\*\* ：是否全是小写

- \*\*isupper()\*\* ：是否全是大写

- \*\*isspace()\*\* ：是否全是空白字符

- \*\*istitle()\*\* ：是否每个单词首字母大写

- \*\*startswith(prefix)\*\* ：是否以指定字符串开头

- \*\*endswith(suffix)\*\* ：是否以指定字符串结尾

#### 分割与连接

- \*\*partition(exp)\*\* ：用指定字符串分割为元组

- \*\*rpartition(exp)\*\* ：从右边开始分割

- \*\*splitlines()\*\* ：用换行符分割为列表

- \*\*split(sep)\*\* ：用指定字符串分割为列表

- \*\*rsplit(sep)\*\* ：从右边开始分割

### 大小写转换

- str.lower() ：字符串转小写

- str.upper() ：字符串转大写

- str.capitalize()：首字母大写

- str.swapcase()：交换大小写

- str.title()：每个单词首字母大写

删减与填充

- \*\*str.strip([chars])\*\* ：移除头尾指定字符串 (默认空白)

- \*\*str.rstrip([chars])\*\* ：移除右边指定字符串 (默认空白)

- \*\*zfill(width)\*\* ：原字符串靠左，返回指定宽度字符串，前面用0填充

- \*\*center(width[,char])\*\* ：原字符串居中，返回指定宽度字符串，用char填充

- \*\*ljust(width[,char])\*\* ：原字符串靠左，返回指定宽度字符串，用char填充

- \*\*rjust(width[,char])\*\* ：原字符串靠右，返回指定宽度字符串，用char填充

- \*\*expandtabs([tabsize])\*\* ：把tab替换为空格，默认是8个

### 列表 List

#### 删除

- \*\*del list[i]\*\* ：删除指定位置的元素 (i可以是一个或多个)

- \*\*pop([i])\*\* ：移除并返回元素

- \*\*remove(elem)\*\* ：移除指定元素

#### 列表操作

- \*\*len(list)\*\* ：计算列表的长度

- \*\*min(list)\*\* ：返回列表最小值

- \*\*max(list)\*\* ：返回列表最大值

- \*\*list(seq)\*\* ：元组转换为列表

#### 列表排序

- \*\*sort()\*\* ：永久排序

- \*\*sorted(list)\*\* ：临时排序

- \*\*reverse()\*\* ：翻转列表

#### 列表切片 / range

- \*\*start:end\*\* ：从start提取到end - 1 ，每 step 个字符提取一个

- \*\*[start:end:step]\*\* ：从start提取到end - 1 ，每 step 个字符提取一个

- \*\*[start:]\*\* ：从start提取到结尾

- \*\*[:end]\*\* ：从开头提取到end - 1

- \*\*正序索引\*\* ：0 1 2 3 4 5 6

- \*\*逆序索引\*\* ：-7 -6 -5 -4 -3 -2 -1

- \*\*list[::2]\*\* ：[1, 2, 3, 4, 5, 6, 7]

- \*\*list[::-1]\*\* ：[7, 6, 5, 4, 3, 2, 1]

- \*\*list[1:4:2]\*\* ：[2, 4]

- \*\*list[0:7:2]\*\* ：[1, 3, 5, 7]

- \*\*list[start:]\*\* ：返回start索引以后的 [1, 2, 3, 4, 5, 6, 7]

- \*\*list[:end]\*\* ：返回end索引之前的 [1, 2, 3, 4, 5, 6, 7]

- \*\*list[start:end]\*\* ：返回start到end - 1的 [1, 2, 3, 4, 5, 6, 7]

- \*\*list[start:end:step]\*\* ：返回start到end - 1的 [1, 2, 3, 4, 5, 6, 7]

- \*\*range()函数\*\* ：range() 函数可创建一个整数数列，一般用在 for 循环中

- \*\*range(stop)\*\* ：计数从 0 到 stop 结束，默认不从 0 开始。

- \*\*range(start, stop[, step])\*\* ：计数从 start 到 stop 结束，默认不从 0 开始。

- \*\*step\*\* ：步长，默认为 1 例如：

- \*\*range(10)\*\* ：[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

- \*\*range(1, 10, 2)\*\* ：[1, 3, 5, 7, 9]

- \*\*range(5, 10, 1)\*\* ：[5, 6, 7, 8, 9]

- \*\*range(5, 30, 15)\*\* ：[5, 20]

### 字典 Dictionary

- \*\*pop(key)\*\* ：删除指定的键

- \*\*clear()\*\* ：清除所有的键与值

- \*\*dict.items()\*\* ：返回所有键值对

- \*\*setdefault(key[,default\_value])\*\* ：查找指定的键，没有则返回None

- \*\*update([dict2])\*\* ：合并更新字典

- \*\*keys()\*\* ：返回所有键

- \*\*values()\*\* ：返回所有值列表 并排序

### set集合

- \*\*add(item)\*\* ：增加一个值

- \*\*update(items)\*\* ：增加多个值

- \*\*remove(item)\*\* ：移除指定值

- \*\*discard(item)\*\* ：移除指定值

- \*\*&\*\* ：取两个集合相交值

- \*\*union\*\* ：合并两个集合

- \*\*difference\*\* ：两个集合差值

- \*\*symmetric\_difference\*\* ：两个集合对称差值

### 内置函数

#### 交互操作

- \*\*print(...)\*\* ：向标准输出打印对象输出

- \*\*input(...)\*\* ：读取用户输入值

#### 数学运算

- \*\*abs(...)\*\* ：返回数字的绝对值

- \*\*divmod(a, b)\*\* ：返回两个数的商和余数

- \*\*max(...)\*\* ：求最大值

- \*\*min(...)\*\* ：求最小值

- \*\*pow(...)\*\* ：返回两个数的幂运算值

- \*\*sum(...)\*\* ：元素求和

- \*\*round(...)\*\* ：四舍五入求值

#### 类型转换

- \*\*int(...)\*\* ：将一个字符串或数转换为整型

- \*\*dict(...)\*\* ：创建一个字典

- \*\*float(...)\*\* ：创建或转换为浮点数

- \*\*tuple(...)\*\* ：创建元组

- \*\*set(...)\*\* ：创建一个新的集合

- \*\*str(...)\*\* ：创建一个字符串

- \*\*list(...)\*\* ：创建一个列表

- \*\*bytearray(...)\*\* ：返回字节数组

- \*\*bytes(...)\*\* ：返回字节数组

- \*\*memoryview(...)\*\* ：创建内存对象

- \*\*complex(...)\*\* ：创建一个复数

- \*\*chr(...)\*\* ：返回整数对应的ASCII字符

- \*\*ord(...)\*\* ：将整数转换为ASCII字符

- \*\*bin(...)\*\* ：将整数转化为二进制数字符串

- \*\*hex(...)\*\* ：将整数转换为16进制字符串

- \*\*oct(...)\*\* ：将整数转换为8进制字符串

#### 迭代操作

- \*\*iter(...)\*\* ：根据可迭代对象创建一个新的迭代对象

- \*\*next(...)\*\* ：返回可迭代对象中的下一个元素

- \*\*super(...)\*\* ：创建一个子类和父类关系的代理对象

- \*\*object(...)\*\* ：创建一个新的object对象

#### 序列操作

- \*\*all(...)\*\* ：判断可迭代对象的元素是否都为True值

- \*\*any(...)\*\* ：判断可迭代对象的元素是否有True值

- \*\*filter(...)\*\* ：过滤可迭代对象的元素

- \*\*map(...)\*\* ：对可迭代对象中的每个元素应用函数

- \*\*reversed(...)\*\* ：返回可迭代对象的反向迭代器

- \*\*sorted(...)\*\* ：排序，返回一个新的列表

- \*\*zip(...)\*\* ：将传入的每个迭代器中相同位置的元素，返回一个新的元组类型

#### 其他函数

\_ \_import\_() 动态导入模块

- \*\*help(...)\*\* ：返回对象的帮助信息

- \*\*dir(...)\*\* ：返回对象的属性列表

- \*\*id(...)\*\* ：返回对象的ID值

- \*\*hasattr(...)\*\* ：判断对象是否有属性

- \*\*isinstance(...)\*\* ：判断对象是否是某个类的实例

- \*\*issubclass(...)\*\* ：判断对象是否是某个类的子类

- \*\*getattr(...)\*\* ：存取对象的属性值

- \*\*setattr(...)\*\* ：设置对象的属性值

- \*\*delattr(...)\*\* ：删除对象的属性值

- \*\*callable(...)\*\* ：检查对象是否可被调用

- \*\*globals(...)\*\* ：返回当前作用域的全局变量

- \*\*locals(...)\*\* ：返回当前作用域的局部变量

- \*\*importlib.import\_module(...)\*\* ：动态导入模块

Python的应用领域

WEB开发，数据分析与科学云计算，人工智能和机器学习，自动化测试和运维，网络爬虫，游戏开发

## ipo程序编写方法

（input输入,process处理,output输出）

### print输出函数

#print可以输出的内容

a=5

b=10

print(a+b)#输出运算结果

print('abc')#输出字符串

print("abc")#输出字符串

print("""abc""")#输出字符串

print('''abc''')#输出字符串

print(a,b,"abc")

print("b",chr(98))#输出的ASCLL码表当中的字符

ft=open("note.txt","w")

print("abc",file=ft)

ft.close()

end=" "后面没有换行符了，默认有

\n换行 +连接符连接两个字符串，字符串了整数不能连接

### Input输入函数

语法结构:x=input('提示文字')

无论输入是什么都是字符串

Int(变量)整数类型

mingzi=input('请输入你的名字：')

nianling=int(input('请输入你的年龄：'))#直接在获取的地方设置类型，或别重新用变量定义

print("你的名字是："+mingzi,nianling)

## 注释

**单行注释：**#后面的内容就是注释

**多行注释：**'''qqq

qqq'''

**#中文编码注释：**一定要写在第一行coding=utf-8

意思是就文件打开别存为就是会UTF-8

## 代码缩进

一个4个空格代表一个缩进量

## 数据类型

**保留字**

#查询python中的保留字

import keyword

print(keyword.kwlist)

print("python中的保留字有",len(keyword.kwlist),"个！")

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

python中的保留字有 35 个！

保留字是严格区别大小的。

**标示符：**1，不能用数字开头。2，不能用保留字。3，以\_\_下滑线开头的标识符一般是有特殊意义的。一般应避免。4，可以用中文作为标识符，但不建议。

**标示符的命名规范：**

**模块名**尽量短小，并且用小写字母，可以用下划线分隔。如：garme\_main

**包名**尽量短小，用小写，不推荐使用下划线。如com.ysjpython，不推荐com\_ysjpython

**类名**采用单词首字母大写，如MyClass

**模块内部的类**采用“\_”+pascal风格的类名组成，如MyClass内部类\_innerMyClass

**函数、类的属性和方法的命名**，用小写，多字母用下划线

**常量命名**时采用全部大写字母。可以用下划线

**使用单下划线”\_”开头的模块变量或函数是受保护的**，在使用from xxx import语句从模块中导入时，这些模块变量或函数不能被导入

**使用双下划线”\_\_”开头的实例变量或方法**是类私有的

**以双下划线开头和结尾的是Python的专用标识**，例如:\_init\_()表示初始化函数

### 变量与常量

变量名=value 举例：Number=8（整数型） 中间的等号是赋值符，变量名在栈内存，常量在堆内存。

My\_name=”胡”,字条串型的

用type（）可以查看变量的类型

a=5

b='b'

print("a是：",type(a),"b是：",type(b))

结果

a是： <class 'int'> b是： <class 'str'>

no=numb=1024 #变量可以赋同一个值

print(no,numb)

print(id(no),id(numb))#读取值的内存位置

结果：

1024 1024

2598304468688 2598304468688

**变量命名规则：**必须是一个有效的标识符，不能用保留字，慎用I和O，选择有意义的单词

**常量：**值不允许改变的量，全部使用大写字母和下划线命名

pi=3.1415#定义了一个变量

PI=3.1415#定义了一个常量

程序员之前默认不改大写的PI这个常量。

### 数值类型

**整数类型：int**（十，二，八，十六进制），

print(0.1+0.2) 结果：0.30000000000000004 #不确定尾数

**浮点数类型：float**，

如果只要保留后面1位尾数可以使用round

print(round(0.1+0.2,1))#round函数，保留尾数多少位，如后面1就是1位

结果：0.3

**复数：**由实部.real和虚部.imag。

pi=333+444j

print("实数：",pi.real)

print("虚数：",pi.imag)

结果:

实数： 333.0

虚数： 444.0

## 字符串类型

### 定义字符串：

单行定义：‘’ “”

多行定义：aaa='''sdfsd #三引号

sdfsdf'''

### 某字符串的索引：

-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 递减

A B C D E F G H I J 字符

0 1 2 3 4 5 6 7 8 9 递增

### 字符串切片：

1 正向搜索：

最左侧第一个字符，偏移量是0，第二个偏移量是1，以此类推。

直到len(str)-1为止。

2 反向搜索：

最右侧第一个字符，偏移量是-1，倒数第二个偏移量是-2，以此

类推，直到-len(str)为止。

**典型操作(三个量为正数的情况)如下：**

操作和说明 示例 结果

[:] 提取整个字符串 “abcdef”[:] “abcdef”

[start:] 从start索引开始到结尾 “abcdef”[2:] “cdef”

[:end] 从头开始知道end-1 “abcdef”[:2] “ab”

[start:end] 从start到end-1 “abcdef”[2:4] “cd”

[start:end:step] 从start提取到end-1，步长是step “abcdef”[1:5:2] “bd”**其他操作（三个量为负数）的情况：**示例 说明 结果

"abcdefghijklmnopqrstuvwxyz"[-3:] 倒数三个 “xyz”

"abcdefghijklmnopqrstuvwxyz"[-8:-3] 倒数第八个到倒数第三个(包头不包尾) 'stuvw'

"abcdefghijklmnopqrstuvwxyz"[::-1] 步长为负，从右到左反向提取

'zyxwvutsrqponmlkjihgfedcba'

### 布尔类型

只有两个值 Ture,False，每个一变量都可以是一个布尔值

测试布尔值：print(bool(10))#True print(bool(0.0))#False

**布尔值False的情况如下：**

1,False或者是None。2，数值是0（包含0.0，虚数0）。3，空列表、空字符串、空元组、空字典、空集合。4，自定义对象的实例，该对象的\_bool)（）方法返回False或\_len\_（）方法返回0。

### 数据类型之间的转换

数据类型分为隐式转换和显性转换，隐式是系统默认转换，显式是通过函数转换

bin(x) 将一个字符X转换为一个二进制字符串

int(x [,base]) 将x转换为一个整数

long(x [,base] ) 将x转换为一个长整数

float(x) 将x转换到一个浮点数

complex(real[,imag]) 创建一个复数

str(x) 将对象 x 转换为字符串

repr(x) 将对象 x 转换为表达式字符串

eval(str) 用来计算在字符串中的有效Python表达式,并返回一个对象

Complex(A) 将参数转换为复数型

tuple(s) 将序列 s 转换为一个元组

list(s) 将序列 s 转换为一个列表

set(s) 转换为可变集合

dict(d) 创建一个字典。d 必须是一个序列 (key,value)元组

frozenset(s) 转换为不可变集合

chr(x) 将一个整数转换为一个字符

unichr(x) 将一个整数转换为Unicode字符

ord(x) 将一个字符转换为它的整数值

hex(x) 将一个整数转换为一个十六进制字符串

oct(x) 将一个整数转换为一个八进制字符串

>>> ord('A') #65

>>> ord('高') #39640

>>> chr(66) #'B'

>>> ord('淇') #28103

**eval函数：**去掉字符串最外侧的引号，并按照python语句方式执行去掉引号后的字符串

使用方法：变量=eval(字符串)

如：

s="3+3"

print((s,type(s)))

x=eval(s)

print((x,type(x)))

结果：

('3+3', <class 'str'>)

(6, <class 'int'>)

eval()函数经常和input()函数一起使用用于获取用记输入的数值型。

age=eval(input("请输入你的年龄："))

print("你的年龄是：",age+1)

结果：

请输入你的年龄：20

你的年龄是： 21

### 转义字符

我们可以使用 \+特殊字符 ，实现某些难以用字符表示的效果。比如：换

行等。常见的转义字符有这些

转义字符 描述

(在行尾时) 续行符

\ 反斜杠符号

' 单引号

" 双引号

\b 退格(Backspace)

\n 换行

\t 横向制表符

\r 回车

**原字符：**使转义字符失效的字符，R或r

a = [10,20,30,40,\  
 50,60,70,\  
 80,90,100]  
b = 'abcdefg\  
hijklmn\  
opqrst\  
uvwxyz'  
print(a)  
print(b)

[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

abcdefghijklmnopqrstuvwxyz

### 字符串拼接

一、可以使用 + 将多个字符串拼接起来。例如： ’aa’+ ’bb’ 结果 是 ’aabb’

1 如果 + 两边都是字符串，则拼接。

2 如果 + 两边都是数字，则加法运算

3 如果 + 两边类型不同，则抛出异常

二、可以将多个字面字符串直接放到一起实现拼接。例 如： ’aa’’bb’ 结果是 ’aabb’

**字符串复制**

使用\*可以实现字符串复制：a = 'Sxt'\*3 #结果:'SxtSxtSxt'

### split()分割和join()合并

**split()可以基于指定分隔符将**字符串分隔成多个子字符串(存储到列表中)。如果不指定分隔符，则默认使用空白字符(换行符/空格/制表符)。示例代码如下：

>>> a = "to be or not to be"

>>> a.split()

['to', 'be', 'or', 'not', 'to', 'be']

>>> a.split('be')

['to ', ' or not to ', '']

**join()的作用和split()作用刚好相反**，用于将一系列子字符串连接起来。示例代码如下：

>>> a = ['sxt','sxt100','sxt200']

>>> '\*'.join(a)

'sxt\*sxt100\*sxt200'

拼接字符串要点： 使用字符串拼接符 + ，会生成新的字符串对象，因此不推荐使 用 + 来拼接字符串。推荐使用 join 函数，因为 join 函数在拼接字 符串之前会计算所有字符串的长度，然后逐一拷贝，仅新建一 次对象。

import time  
time01 = time.time() #起始时刻  
a = ""  
for i in range(100000):  
 a += "sxt"  
time02 = time.time() #终止时刻  
print("运算时间："+str(time02-time01))  
time03 = time.time() #起始时刻  
li = []  
for i in range(100000):  
 li.append("sxt")  
a = "".join(li)  
time04 = time.time() #终止时刻  
print("运算时间："+str(time04-time03))

运算时间：0.7792668342590332

运算时间：0.009430885314941406

### 字符串比较和同一性

我们可以直接使用 == != 对字符串进行比较，是否含有相同的字符。

我们使用 is not is ，判断两个对象是否同一个对象。比较的是对象的地址，即 id(obj1) 是否和 id(obj2) 相等。

**成员操作符判断子字符串**

in not in 关键字，判断某个字符(子字符串)是否存在于字符串中。

"ab" in "abcdefg" #true

### **不换行打印**

我们可以自己通过参数end = “任意字符串”。实现末尾添加任何内容：

print("sxt",end=' ')

print("sxt",end='##')

print("sxt")

sxt sxt##sxt

### 字符串常用方法：

#### 常用查找方法

a='''我是高淇,我在北京尚学堂科技上班。我的儿子叫高洛希，他6岁了。我是一个编程教育的普及者，希望影响6000万学习编程的中国人。我儿子现在也开始学习编程，希望他18岁的时候可以超过我'''

方法和使用示例 说明 结果

len(a) 字符串长度 96

a.startswith('我是高淇') 以指定字符串开头 True

a.endswith('过我') 以指定字符串结尾 True

a.find('高') 第一次出现指定字符串的位置 2

a.rfind('高') 最后一次出现指定字符串的位置 29

a.count("编程") 指定字符串出现了几次 3

a.isalnum() 所有字符全是字母或数字 False a.isalpha()\*\* ：是否全是字母

a.isdigit()\*\* ：是否全是数字

a.isdecimal()\*\* ：是否全是小数

a.islower()\*\* ：是否全是小写

a.isupper()\*\* ：是否全是大写

a.isspace()\*\* ：是否全是空白字符

a.istitle()\*\* ：是否每个单词首字母大写

#### 除首尾信息

我们可以通过strip()去除字符串首尾指定信息。通过lstrip()去除字符串左边指定信息rstrip()去除字符串右边指定信息。

【操作】去除字符串首尾信息

>>> "\*s\*x\*t\*".strip("\*")

's\*x\*t'

>>> "\*s\*x\*t\*".lstrip("\*")

's\*x\*t\*'

>>> "\*s\*x\*t\*".rstrip("\*")

'\*s\*x\*t'

>>> " s xt ".strip()

's xt'

#### 大小写转换

a = "gaoqi love programming, love SXT"示例 说明 结果

a.capitalize() 产生新的字符串,首字母大写 'Gaoqi love programming, love sxt'

a.title() 产生新的字符串,每个单词都首字母大写 'Gaoqi Love Programming, Love Sxt'

a.upper()产生新的字符串,所有字符全转成大写'GAOQI LOVE PROGRAMMING, LOVE SXT'

a.lower() 产生新的字符串,所有字符全转成小写'gaoqi love programming, love sxt'

a.swapcase() 产生新的,所有字母大小写转换 'GAOQI LOVE PROGRAMMING, LOVE sxt'

#### 格式排版

center() 、 ljust() 、 rjust() 这三个函数用于对字符串实现排版。示例如

下： >>> a="SXT"

>>> a.center(10,"\*") #'\*\*\*SXT\*\*\*\*'

>>> a.center(10) #' SXT '

>>> a.ljust(10,"\*") #'SXT\*\*\*\*\*\*\*'

### 特征判断方法

1 isalnum() 是否为字母或数字

2 isalpha() 检测字符串是否只由字母组成(含汉字)

3 isdigit() 检测字符串是否只由数字组成

4 isspace() 检测是否为空白符

5 isupper() 是否为大写字母

6 islower() 是否为小写字母

>>> "sxt100".isalnum()

True

>>> "sxt尚学堂".isalpha()

True

>>> "234.3".isdigit()

False

>>> "23423".isdigit()

True

>>> "aB".isupper()

False

>>> "A".isupper()

True

>>> "\t\n".isspace()

True

### 字符串的格式化

#### format() 基本用法

**基本语法是通过 {} 和 : 来代替以前的 % 。**

format() 函数可以接受不限个数的参数，位置可以不按顺序。

我们通过示例进行格式化的学习。

>>> a = "名字是:{0},年龄是：{1}"

>>> a.format("高淇",18)

'名字是:高淇,年龄是：18'

>>> a.format("高希希",6)

'名字是:高希希,年龄是：6'

>>> b = "名字是：{0}，年龄是{1}。{0}是个好小伙"

>>> b.format("高淇",18)

'名字是：高淇，年龄是18。高淇是个好小伙'

>>> c = "名字是{name}，年龄是{age}"

>>> c.format(age=19,name='高淇')

'名字是高淇，年龄是19'

#### 填充与对齐

1 填充常跟对齐一起使用

2 ^ 、 < 、 > 分别是居中、左对齐、右对齐，后面带宽度

3 : 号后面带填充的字符，只能是一个字符，不指定的话默认是用空格填充

我们可以通过{索引}/{参数名}，直接映射参数值，实现对字符串的 格式化，非常方便。

>>> "{:\*>8}".format("245")

'\*\*\*\*\*245'

>>> "我是{0},我喜欢数字{1:\*^8}".format("高

淇","666")

'我是高淇,我喜欢数字\*\*666\*\*\*'

#### 数字格式化

浮点数通过 f ，整数通过 d 进行需要的格式化。案例如下：

>>> a = "我是{0}，我的存款有{1:.2f}"

>>> a.format("高淇",3888.234342)

'我是高淇，我的存款有3888.23'

**其他格式，供大家参考：**

数字 格式 输出 描述

3.1415926 {:.2f} 3.14 保留小数点后两位

3.1415926 {:+.2f} 3.14 带符号保留小数点后两位

2.71828 {:.0f} 3 不带小数

5 {:0>2d} 05 数字补零 (填充左边, 宽度为2)

5 {:x<4d} 5xxx 数字补x (填充右边, 宽度为4)

10 {:x<4d} 10xx 数字补x (填充右边, 宽度为4)

1000000 {:,} 1,000,000 以逗号分隔的数字格式

0.25 {:.2%} 25.00% 百分比格式

1000000000 {:.2e} 1.00E+09 指数记法

13 {:10d} 13 右对齐 (默认, 宽度为10)

13 {:<10d} 13 左对齐 (宽度为10)

13 {:^10d} 13 中间对齐 (宽度为10)

### 可变字符串

1.Python中，字符串属于不可变对象，不支持原地修改，如果需要修改其中的值，只能创建新的字符串对象。

2.确实需要原地修改字符串，可以使用io.StringIO对象或array模块 import io

s = "hello, sxt"

sio = io.StringIO(s) #可变字符串

print(sio)

v1 = sio.getvalue()

print("v1:",v1)

char7 = sio.seek(7) #指针知道索引7这个位置

sio.write("gaoqi")

v2 = sio.getvalue()

print("v2:",v2)

## 运算符

### 算术运算符

+ 加法1+1=2，-减法1-1=0，\*乘法1\*1=1，/除法10/2=5.0，//整除10//3=3，%余数10%3=1,\*\*幂运算，2\*\*4=16（相当于2的4次方）

使用 divmod() 函数同时得到商和余数：

>>> divmod(13,3)

(4, 1)

特别注意，/不可以除0，否则报错

### 赋值运算符

算术运算符的优先级由高到低：最高级（）第一级\*\*。第二级\*，/，%，//。第三级+，-。

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 描述说明 | 示例 | 展开形式 |
| = | 简单的赋值运算 | x=y | x=y |
| += | 加赋值 | x+=y | x=x+y |
| -= | 减赋值 | x-=y | x=x-y |
| \*= | 乘赋值 | x\*=y | x=x\*y |
| /= | 除赋值 | x/=y | x=x/y |
| %= | 取余赋值 | x%=y | x=x%y |
| \*\*= | 幂赋值 | x\*\*=y | x=x\*\*y |
| //= | 整除赋值 | x//=y | x=x//y |

**链式赋值：**a=b=c=10 #a,b,c同时赋值10

**系列解包赋值：**a,b=10,20 #a和b分别被赋值成10和20，

字符串a,b,c,d="xyzl" 结果：x y z l

利用系列解包赋值交换两个变量的值：a,b=b,a

1 数字相加 3+2 结果是 5

2 字符串拼接 “3”+“2” 结果是 “32”

3 列表、元组等合并 [10,20,30]+[5,10,100] 结果是 [10,20,30,5,10,100]

乘法操作补充

1 数字相乘 3\*2 结果是 6

2 字符串复制 “sxt”\*3 结果是 ”sxtsxtsxt”

3 列表、元组等复制 [10,20,30]\*3 结果是 [10,20,30,10,20,30,10,20,30]

**增强赋值运算符(补充)**

复合赋值可以让程序更加精炼，提高效率。

运算符 描述 示例 等价于

+= 加法赋值字符串拼接 sum += na += “sxt” sum = sum + na = a + “sxt”

-= 减法赋值 num1 -= n num = num - n

\*= 乘法赋值 a \*= b a = a \* b

/= 浮点除赋值 a/=b a = a / b

//= 整数除赋值 a//=b a = a//b

%= 取余赋值 a%=b a = a % b

\*\*= 幂运算赋值 a\*\*=2 a = a\*\*2

<<= 左移赋值 a<<=2 a = a<<2

>>= 右移赋值 a>>=2 a = a>>2

&= 按位与赋值 a&=b a = a&b

|= 按位或赋值 a|=b a=a|b

^= 按位异或赋值 a^=b a = a^b

### 比较运算符

>,<,==,!=,>=,<=,结果布尔值

运算符 描述 is 是判断两个标识符是不是引用同一个对象

is not是判断两个标识符是不是引用不同对象

is 与 == 区别：

is 用于判断两个变量引用对象是否为同一个，既比较对象的地址。

== 用于判断引用变量引用对象的值是否相等，默认调用对象的 \_\_eq\_\_() 方法。

is 比较两个对象的 id 值是否相等，是否指向同一个内存地址

== 比较的是两个对象的内容是否相等，值是否相等

is 运算符比 == 效率高，在变量和 None 进行比较时，应该使用 is

a=20  
b=20  
c=30  
print("a和b是同一个对象",a is b) #执行结果:True  
print("a和c是同一个对象",a is c) #执行结果False  
print("a和c不是同一个对象",a is not c) #执行结果True

a = 1000  
b = 1000  
print(a == b) #True  
print(a is b ) #命令行下是False。 文件下执行是True  
c = 10  
d = 10  
print(c is d) #True

### 成员运算符

成员运算符测试实例中包含了一系列的成员，包括字符串，列表或 元组。

运算符 描述

in 如果在指定的序列中找到值返回 True，否则返回 False

not in 不在，则返回True

a = "python"  
b = "py"  
print(b in a) #True  
c = [10,20,30]  
print(10 not in c) #False

### 逻辑运算符

and,or,not结果就是布尔值，and,or是从左到友，not（逻辑非等表达计算完以后再用not取反）是从右到左，如：print(not True)

### 位运算符

把数字看作二进制数来进行计算的。

### 运算符的优先级

有括号取括号

1.\*\* 幂运算

2.~，+，- 取反（反来是0现在就是1），正号，负号

3.\*,/,%,// 算术运算符

4.+，- 算术运算符

5.<<,>> 位运算符中的左移位和右移位

6.& 位运算符中的按位与（1，1为1）

7.^ 位运算符中的异或（两个数位置相同为0不同为1）

8.| 位运算符中的按位或（1，0为1）

9.<,<=,>,>=,!=,== 比较运算符

10.= 赋值运算符

### 基本运算符总结

运算符 说明

and , or , not 布尔与、布尔或、布尔非

is , is not 同一性判断，判断是否为同一个对象

in , not in 判断是否在指定序列中

<,<=,>,>=,!=,== 比较值是否相当，可以连用

| ^ & 按位或，按位异或、按位与

《》 移位

- 按位翻转

+,-,\*,/,//,% 加，减，乘，浮点除、整数除、取余

\*\* 幂运算

#### 关于None和判断的总结

None是什么？

1 与C和JAVA不同，python中是没有 NULL 的，取而代之的是 None 。

2 None 是一个特殊的常量，表示变量没有指向任何对象。

3 在Python中， None 本身实际上也是对象，有自己的类型 NoneType 。

4 你可以将 None 赋值给任何变量，但我们不能创建 NoneType 类型的对象

obj = None

obj2 = None

print(type(None)) # <class 'NoneType'>

print(id(None)) #140717958924280

print(id(obj)) #140717958924280

print(id(obj2)) #140717958924280

⚠️None不是False，None不是0，None不是空字符串。None 和任何其他的数据类型比较永远返回False。

None和其他类型的比较

None和其他任何类型比较都会返回False

a = None

if a is None and a==None:

print("a是None") #会执行

if a==False or a==0:

print("None不等于False") #不会被打印

其他操作

1 dir(obj) 可以获得对象的所有属性、方法

2 obj.\_\_dict\_\_ 对象的属性字典

3 pass 空语句

4 isinstance（对象,类型） 判断“对象”是不是“指定类型”

#### 空列表、空字符串、0之间的比较

1 if语句判断时,空列表[]、空字典{}、空元组()、0等一系列代表 空和无的对象会被转换成False

a=[];b=();c={};d="";e=0;f=None

if (not a) and (not b) and (not c) and (not d) and (not e) and (not f):

print("if判断时，空列表[]、空字符串、0、None等代表空和无的对象会被转换成False")**2 == 和 is 判断时，空列表、空字符串不会自动转成 False**

a=[];b=();c={};d="";e=0;

if (a==False or d==False):

print("==时，空列表、空字符串不是 False！")   #不会执行

if(e==False):

print("==时，0会转成False")

from enum import show\_flag\_values  
class student:#创建了一个类  
 school='北京大学'#这就是一个变量，类属性  
 #实例属性  
 def \_\_init\_\_(self,xm,age):#nm,age方法的参数，局部变量作用域只能在\_\_init\_\_内  
 self.name=xm#等号的左侧是实例改属性，xm是局部变量,将局部变量的xm值赋值给了实例属性self.name  
 self.age=age#实例属性的变量名称和局部变量的名称可以相同  
 # 定义在类当中的函数就是实例方法  
 def show(self):  
 print(f'你的名字叫：{self.name}\b你的年龄：{self.age}')#self打点的实例属性可以在整个类中使用  
 #静态方法  
 @staticmethod  
 def sm():  
 print('这是一个静态方法,不能调用实例属性，也不能调用实例方法')  
 #类方法  
 @classmethod  
 def cm(cls):#cls是class的简写  
 print('这是一个类方法，类方法当中也不能调用实例属性和实例方法')  
#创建类的对象  
stu=student('genten',40)#将对象放到了stu这个变量中存储，为什么传两个参数，因为init中有两个形参  
#实例属性是通过对象打点调用的的  
print(stu.name,stu.age)#在类中用self打点调用，创建了对象名就需要用对象名打点调用  
#类属性，直接使用类名打点调用  
print(student.school)  
#实例方法，跟实例有关的就是对象打点调用  
stu.show()  
#类方法，调用方法直接使用类名打点调用  
student.cm()  
#静态方法的调用  
student.sm()

genten 40

北京大学

你的名字叫：gente你的年龄：40

这是一个类方法，类方法当中也不能调用实例属性和实例方法

这是一个静态方法,不能调用实例属性，也不能调用实例方法

## 程序的流程控制

控制语句：把语句组合成能完成一定功能的小逻辑模块。

分为三类：顺序、选择和循环。

### 程序的描述方式

**自然语言方式IPO**：input输入，process处理,output输出

**流程图：**是用一系统图形，流程线和文字说明描述程序的基本操作和控制流程。主要适用于较短的算法

开始》输入》运算》输出》结束

**伪代码：**是介于自然语言和编程语言之间的一种算法描述语言，如果程序比较小，可以直接使用代码描述。

### 程序的组织结构

#### 顺序结构：

按程序的自然顺序，从上到下，依次执行每条语句的程序。

**选择/分支结构：**选择结构if,双分支结构if else,多分支结构:if elif,elif

嵌套if的使用：

if zhi

if zhi

elif

else

else

#### 多个条件的连接,and，OR

密码验证

yonghum=()

mima=()

mimacishu=3

while yonghum !="guyue" and mima !="123456":

yonghum = input("请输入用户名：")

mima = input("请输入密码：")

print("你还可以输入：", mimacishu, "次")

mimacishu -= 1

if mimacishu==0:

print("输入次数过多请24小时以后再试")

break

else:

if yonghum=="guyue" and mima=="123456":

print("密码正确")

else:

print("密码错误")

python3.11后的新性，模式匹配：match case结构

a=eval(input("请入数字:"))

match a:

case 50:

print("a")

case 100:

print("b")

case 150:

print("c")

#### 循环结构：

##### 遍历循环for

遍历循环for的语句结构 for..else..语句结构

for 循环变量 in 遍历对象 for 循环变量 in 遍历对象

语句块 语句块1

else

语句块2

for i in range(1,5):

if i%2==0:

print(i,"偶数")

##### 无限循环while

无限循环while的语句结构 while..else..结构

while 表达式 while 表达式：

语句快 语句块1

else: #while执行完毕后就执行else

语句块2

while循环的四个步骤

1初始化变量，2.条件判断，3语句块，4.改变变量

#1到获取数字之间的累加和

suzi=int(input("1到？的累加和：") )#获取变量

s=0 #存储累加和

i=1 #1，初始化变量

while i<=suzi: #2，条件判断

s+=I #3，语句块

i+=1 #4，改变变量

print(s)

print("-"\*10,"用户登录系统","-"\*10)#创建界面

i=0#定义变量

#登陆只有三次机会的系统

while i<3:

mingzi=input("请输入用户名：")

mima=input("请输入密码：")

if mingzi=="guyue" and mima=="666666":

print("系统正在登录中。。。")

i=8#正确后需要退出改变变量

else:

if i<2:#第一次执行的是i=0,第二次是i=1,第三次不执行

print("密码错误，你还有",2-i,"次机会！")

i += 1#改变变量

if i==3:

print("密码错误次数过多，请24小时后再次输入")

#### 嵌套循环打印

循环结构也可以互相嵌套，在一个循环结构中嵌套另外一个完整的循环结构就称为嵌套循环。

while 表达式1： for循环变量1 in 遍历对象1：

while 表达式2： for循环变量2 in 遍历对象2：

语句块2 语句块2

#长方形的一个打印，用\*号横4个，竖3行

for i in range(1,4):

for j in range(1,5):

print("\*",end="")

print()#空的就是换行

\*\*\*\*

\*\*\*\*

\*\*\*\*

print("-"\*10,"打印一个三角形","-"\*10)

for i in range(1,6):

for j in range(1,i+1):

print("\*",end="")

print()

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

print("-"\*10,"打印倒三角形","-"\*10)

for i in range(1,6):

for j in range(1,7-i):

print("\*",end="")

print()

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

print("-" \* 10, "等腰三角形", "-" \* 10)

for i in range(1, 6):

for j in range(1, 6 - i):

print(" ", end="")

for k in range(1,i\*2):

print("\*",end="")

print()

---------- 等腰三角形 ----------

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

print("-"\*10,"菱形图","-"\*10)

#获取一个菱形的行数，因为菱形中间一行是分分割，所以只能是奇数

huoqu\_lx=int(input("请输入菱形行数:"))

#如果用户输入的是奇数则要求重新输入

while huoqu\_lx%2==0:

print("输入错误请输入奇数")

huoqu\_lx = int(input("请输入菱形行数:"))

#菱形的上半部分

shang\_hs=(huoqu\_lx+1)//2

for i in range(1,shang\_hs+1):

for j in range(1,shang\_hs+1-i):

print(" ",end="")

for k in range(1,i\*2):

print("\*",end="")

print()

#菱形的下半部分

for i in range(1,shang\_hs):

for j in range(1,i+1):

print(" ", end="")

for k in range(1,shang\_hs\*2-i\*2): #531用range就是642

print("\*",end="")

print()

---------- 菱形图 ----------

请输入菱形行数:7

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*

\*

print("-"\*10,"空心菱形图","-"\*10)

#获取一个菱形的行数，因为菱形中间一行是分分割，所以只能是奇数

huoqu\_lx=int(input("请输入菱形行数:"))

#如果用户输入的是奇数则要求重新输入

while huoqu\_lx%2==0:

print("输入错误请输入奇数")

huoqu\_lx = int(input("请输入菱形行数:"))

#菱形的上半部分

shang\_hs=(huoqu\_lx+1)//2

for i in range(1,shang\_hs+1):

for j in range(1,shang\_hs+1-i):

print(" ",end="")

for k in range(1,i\*2):

if k==1 or k==i\*2-1:

print("\*",end="")

else:

print(" ", end="")

print()

#菱形的下半部分

for i in range(1,shang\_hs):

for j in range(1,i+1):

print(" ", end="")

for k in range(1,shang\_hs\*2-i\*2): #531用range就是642

if k == 1 or k == shang\_hs\*2-i\*2-1:

print("\*", end="")

else:

print(" ", end="")

print()

---------- 空心菱形图 ----------

请输入菱形行数:5

\*

\* \*

\* \*

\* \*

\*

#### 程序的跳转语句break,continue

**break**break语句可用于while和for循环，用来结束整个循环。当有嵌套循环时，break语句只能跳出最近一层的循环。

**continue**: continue语句用于结束本次循环，继续下一次。多个循环嵌套时， continue也是应用于最近的一层循环。

#做一个偶数求和

a=int(input("请输入偶数求和的数字"))#定义变量

s=0#设定变量

i=1

for i in range(1,a+1):#判断条件

if i%2==1:#语句块如果是奇数

i += 1#执行i+1

print(i)

continue#不再执行后面的代码

s+=i#改变变量

i+=1

print(s)

#### 空语句pass

#### 列表推导式

列表推导式生成列表对象，语法如下：

[表达式 for item in 可迭代对象 ]

或者：{表达式 for item in 可迭代对象 if 条件判断}

[x for x in range(1,5)] #[1, 2, 3, 4]

[x\*2 for x in range(1,5)] #[2, 4, 6, 8]

[x\*2 for x in range(1,20) if x%5==0 ] #[10,20, 30]

[a for a in "abcdefg"] #['a', 'b', 'c','d', 'e', 'f', 'g']

#可以使用两个循环,使用zip并行迭代

cells = [(row,col) for row,col in zip(range(1,10),range(101,110))]

print(cells)

a = [x for x in range(1,10) if x%2==0]

print(a)

#不适用推导式，生成列表。二者对比。体现推导式的优势

b = []

for x in range(1,10):

if x%2==0:

b.append(x)

print(b)

#### 字典推导式

字典的推导式生成字典对象，格式如下： values = ["北京","上海","深圳","广州"]  
cities = {id\*100:city for id,city in zip(range(1,5),values)}  
print(cities)

{100: '北京', 200: '上海', 300: '深圳', 400: '广州'}

my\_text = ' i love you, i love sxt, i lovegaoqi'  
char\_count = {c:my\_text.count(c) for c in my\_text}  
print(char\_count)

{' ': 8, 'i': 4, 'l': 3, 'o': 5, 'v': 3, 'e': 3, 'y': 1, 'u': 1, ',': 2, 's': 1, 'x': 1, 't': 1, 'g': 1, 'a': 1, 'q': 1}

#### 集合推导式

集合推导式生成集合，和列表推导式的语法格式类似：

{表达式 for item in 可迭代对象 }

或者：{表达式 for item in 可迭代对象 if 条件判断} >>> {x for x in range(1,100) if x%9==0}

{99, 36, 72, 9, 45, 81, 18, 54, 90, 27, 63}

#### 生成器推导式（不直接生成元组）

很多同学可能会问：“都有推导式，元组有没有？”，能不能用小括

号呢？

>>> (x for x in range(1,100) if x%9==0) at 0x0000000002BD3048>

我们发现提示的是“一个生成器对象”。显然，元组是没有推导式 的。 一个生成器只能运行一次。第一次迭代可以得到数据，第二次迭代 发现数据已经没有了。

gnt = (x for x in range(1,100) if x%9==0)  
for x in gnt:  
 print(x,end=' ')  
for x in gnt:  
 print(x,end=' ')

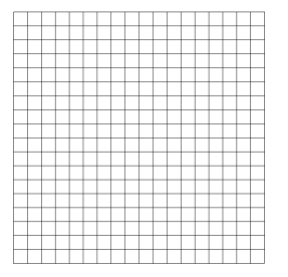
9 18 27 36 45 54 63 72 81 90 99

进程已结束，退出代码为 0

### 实例

绘制多个同心圆

import turtle  
p = turtle.Pen() #画笔对象  
radius = [x\*10 for x in range(1,11)]  
#10,20,30,40...  
my\_colors =("red","green","yellow","black")  
p.width(4)  
for r,i in zip(radius,range(len(radius))):  
 p.penup()  
 p.goto(0,-r)  
 p.pendown()  
 p.color(my\_colors[i%len(my\_colors)])  
 p.circle(r)  
turtle.done() #程序执行完毕，窗口在

#画棋盘  
import turtle  
width = 30  
num = 18  
x1 = [(-400,400),(-400+width\*num,400)]  
y1 = [(-400,400),(-400,400-width\*num)]  
t = turtle.Pen()  
t.speed(10)  
# t.goto(x1[0][0],x1[0][1])  
# t.goto(x1[1][0],x1[1][1])  
for i in range(num+1):  
 t.penup()  
 t.goto(x1[0][0],x1[0][1]-30\*i)  
 t.pendown()  
 t.goto(x1[1][0],x1[1][1]-30\*i)  
for i in range(num+1):  
 t.penup()  
 t.goto(y1[0][0] + 30 \* i, y1[0][1])  
 t.pendown()  
 t.goto(y1[1][0] + 30 \* i, y1[1][1])  
t.hideturtle() # 隐藏画笔  
turtle.done() # 保证运行窗口不被自动关闭

## 组合数据类型

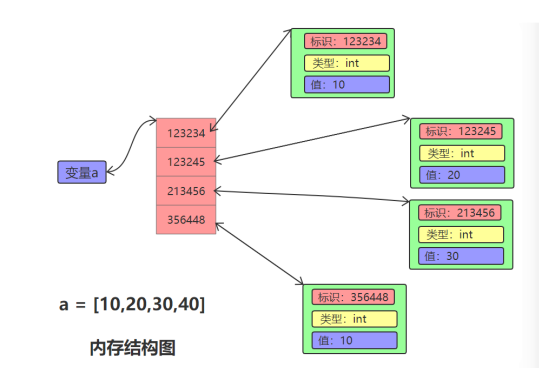
### 序列和索引

序列是一个用于存储多个值的连续空间，每个值都对应一个整数的编号称为索引

正向递增索引 0 1 2 3 4

无素 元素1 元素2 元素3 元素4 元素5

反向递减索引 -5 -4 -3 -2 -1



从图示中，我们可以看出序列中存储的是整数对象的地址，而不是整数对象的值。

#### 切换操作的语法结构

序列[开始：结束：步长]。

如：hello!,序列[0:5:2]，结果是hlo

省略了开始位置默认哦0，省略步长默认为1，序列[：结束]

如果序列[：：-1]。省略开始，结束，步长-1，则是逆向输出如：hello!结果!olleh

#### 序列的相关操作

操作符/函数 描述说明

x in s 如果x是s的元素，结果为True,否则False

x not in s 如果x不是s的元素，结果为True,否则False

len(s) 序列s中元素的个数（即序列的长数）

max(s) 序列s中元素的最大值(按ASK码计算)

min(s) 序列s中元素的最小值

s.index(x) 序列s中第一次出现元素x的位置 (英 /ˈɪndeks/:**标注；标记)**

s.count(x) 序列s中出现x的总次数 (英 /kaʊnt/**:总数**)

### list列表类型[]可变数据类型

列表是指一系列的按特定顺序排列的元素组成。使用[]定义列表，元素和元素间使用英文的逗号分隔，列表的元素可以是任意的数据类型。

#### 列表的相关操作方法

列表方法 要点 描述

list.append(x) 增加元素 将元素x增加到列表list尾部

list.extend(aList) 增加元素 将列表alist所有元素加到列表list尾部

list.insert(index,x) 增加元素 在列表list指定位置index处插入元素x

list.remove(x) 删除元素 在列表list中删除首次出现的指定元素x

list.pop([index]) 删除元素 删除并返回列表list指定为止index处的元素，默认是最后一个元素

list.clear() 删除所有元素 删除列表所有元素，并不是删除列表对象

list.index(x) 访问元素 返回第一个x的索引位置，若不存在x元素抛出异常

list.count(x) 计数 返回指定元素x在列表list中出现的次数

len(list) 列表长度 返回列表中包含元素的个数

list.reverse() 翻转列表 所有元素原地翻转

list.sort() 排序 所有元素原地排序

list.copy() 浅拷贝 返回列表对象的浅拷贝

例：

lb=["hallo","word","python"]

print(lb)

结果： ['hallo', 'word', 'python']

lb.append("sql")#在列表ist最后增加一个元素

print(lb)

结果： ['hallo', 'word', 'python', 'sql']

lb.insert(1,100)#在列表中第index位置增加一个元素

print(lb)

结果： ['hallo', 100, 'word', 'python', 'sql']

lb.remove(100)#将列表lst中出现的第一个元素x删除

print(lb)

结果： ['hallo', 'word', 'python', 'sql']

print(lb.pop(1))#将列表lst中第index位置的元素取出，并从列表中将其删除

print(lb)

结果： word

['hallo', 'python', 'sql']

xinlb=lb.copy()

print(xinlb)

结果： ['hallo', 'python', 'sql']

lb.reverse()#将列表lst中的元素反转

print(lb)

结果： ['sql', 'python', 'hallo']

lb.clear()#清除列表lst中所有的元素

print(lb)

结果： []

**继上修改列表**

xinlb[2]="world"

print(xinlb)

结果： ['hallo', 'python', 'world']

#### 列表的排序

1，列表对象的sort方法 2，内置函数sorted()英 /ˈsɔːtɪd/

lst.sort(key=None,reverse=False) sorted(iterable,key=none,rverse=False)

key表示排序的规则 iterable表示排序的对象

reverse表示排序方式(默认升序)

##### 修改原列表，不建新列表的排sort方法

>>> a = [20,10,30,40]

>>> id(a)

46017416

>>> a.sort() #默认是升序排列

>>> a

[10, 20, 30, 40]

>>> a = [10,20,30,40]

>>> a.sort(reverse=True) #降序排列

>>> a

[40, 30, 20, 10]

>>> import random

>>> random.shuffle(a) #打乱顺序

>>> a

[20, 40, 30, 10]

**字母升序排序是先排大写再排小写，降序是先排小写再排大写**

lb=["Dog","Cat","blu","age"]

print("原列表：",lb)

原列表： ['Dog', 'Cat', 'blu', 'age']

lb.sort()#默认升序排序reverse=False

print("升序：",lb)

升序： ['Cat', 'Dog', 'age', 'blu']

lb.sort(reverse=True)#reverse=True降序

print("降序：",lb)

**自己指定规则**

lb.sort(key=str.lower,reverse=True)#指定str字符串，按lower小写排序

print("按小写降序：",lb)

按小写降序： ['Dog', 'Cat', 'blu', 'age']

##### 建新列表的排序sorted()

英 /ˈsɔːtɪd/

内置函数sorted()将原列表生成一个新列表并排序

lb=["Dog","Cat","blu","age"]

print("原列表：",lb)

xinlb=sorted(lb)

print("新列表：",xinlb)

原列表： ['Dog', 'Cat', 'blu', 'age']

新列表： ['Cat', 'Dog', 'age', 'blu']

xinlb=sorted(lb,key=str.lower,reverse=True)#忽略大小写降序

print("新生成的列表：",xinlb)

新生成的列表： ['Dog', 'Cat', 'blu', 'age']

>>> a = [20,10,30,40]

>>> id(a)

46016008

>>> b = sorted(a) #默认升序

>>> b

[10, 20, 30, 40]

>>> id(b)

45907848

>>> c = sorted(a,reverse=True) #降序

>>> c

[40, 30, 20, 10]

通过上面操作，我们可以看出，生成的列表对象b和c都是完全新的 列表对象。

##### reversed()返回迭代器

内置函数reversed()也支持进行逆序排列，与列表对象reverse()方法不同的是，内置函数reversed()不对原列表做任何修改，只是返回一个逆序排列的迭代器对象。

a = [20,10,30,40]  
c = reversed(a)  
print(c)  
print(list(c))  
print(list(c))

<list\_reverseiterator object at 0x000002118D5AAD10>

[40, 30, 10, 20]

[]

我们打印输出c发现提示是：list\_reverseiterator。也就是一个迭代对象。同时，我们使用list(c)进行输出，发现只能使用一次。第一次输出了元素，第二次为空。那是因为迭代对象在第一次时已经遍历结束了，第二次不能再使用。

#### 列表的创建方式

##### 1.使用[]直接创建

语法结构如下：

列表名=[a,b,c]

如：s=["a","b","c"] 结果：['a', 'b', 'c']

##### 2.使用内置函数list()创建列表

语法结构如下

列表名=list(序号)

s=list("abc")结果：['a', 'b', 'c']

##### 3. range()创建整数列表

range()可以帮助我们非常方便的创建整数列表，这在开发中及其有

用。语法格式为：

range([start,] end [,step])

start参数：可选，表示起始数字。默认是0

end参数：必选，表示结尾数字。

step参数：可选，表示步长，默认为1

⚠️python3中range()返回的是一个range对象，而不是列表。 我们需要通过list()方法将其转换成列表对象。

典型示例如下：

a = list(range(3,15,2)) #结果：[3, 5, 7, 9,

11, 13]

b = list(range(15,9,-1)) #结果：[15, 14, 13,

12, 11, 10]

c = list(range(3,-4,-1)) #结果：[3, 2, 1, 0,

-1, -2, -3]

print(a,b,c)

##### enumerarte函数的使用语法结构

 /ɪˈnjuːməreɪt/ **逐一列举；枚举**

for index,item in enumerate(lst) 如：for a,b in enumerate(x)

输出index和item英 /ˈaɪtəm/通常用来指代容器（如列表、字典、元组等）中的元素

##### 4.推导式生成列表

#循环创建多个元素 [0, 2, 4, 6, 8]

a = [x\*2 for x in range(5)]

#通过if过滤元素[0, 18, 36, 54, 72, 90, 108,

126, 144, 162, 180, 198]

b = [x\*2 for x in range(100) if x%9==0]

print(a,b)

lst=[expression for item,range]

lst=[expression for item in range if condition]

expression英 /ɪkˈspreʃn/ 用于计算值、执行操作或表示某种逻辑关系等

lis=[item for item in range(1,10)]#生成1到9的列表

print(lis)

[1, 2, 3, 4, 5, 6, 7, 8, 9]

#生成10个1到100的随机数

lis=[random.randint(1,100) for \_ in range(10)]

print(lis)

[47, 77, 44, 37, 69, 92, 80, 53, 32, 69]

# “random” 读音为英 /ˈrændəm/ random是 Python 的一个内置模块，它提供了多种用于生成随机数和进行随机选择的函数

# “randint” 读音为 /ˈrændɪnt/randint 是该模块中的一个具体函数，其作用是生成一个包含下限和上限在内的随机整数。

#用法：random.randint(a, b)

生成2到10之间的偶数列表

lst=[i for i in range(1,11) if i%2==0]

print(lst)

[2, 4, 6, 8, 10]

#### 列表元素的增加

##### append()方法

原地修改列表对象，是真正的列表尾部添加新的元素，速度最快，

推荐使用。

a = [20,40]

a.append(80)

print(a) #结果：[20, 40, 80]

##### +运算符操作

并不是真正的尾部添加元素，而是创建新的列表对象；将原列表的

元素和新列表的元素依次复制到新的列表对象中。这样，会涉及大

量的复制操作，对于操作大量元素不建议使用。a = [20,40]

print(id(a))

a = a+[50]

print(id(a)) #两次地址不一样，创建了新的对象

通过如上测试，我们发现变量a的地址发生了变化。也就是创建了新

的列表对象。

##### extend()方法

将目标列表的所有元素添加到本列表的尾部，属于原地操作，不创 建新的列表对象。

a = [20,40] print(id(a))

b = [50,60] a.extend(b)     #原对象修改

print(id(a))

a = a+b         #产生新对象

print(id(a))

[20, 40] 1924707098944

[20, 40, 50, 60] 1924707098944

[20, 40, 50, 60, 50, 60] 1924707323520

##### insert()插入元素

使用 insert() 方法可以将指定的元素插入到列表对象的任意制定位置。这样会让插入位置后面所有的元素进行移动，会影响处理速度。涉及大量元素时，尽量避免使用。类似发生这种移动的函数还有：

remove() 、 pop() 、 del() ，它们在删除非尾部元素时也会发生操作位置后面元素的移动。

a = [10,20,30]

1. insert(2,100)
2. print(a)   #结果：[10, 20, 100, 30]

##### 乘法扩展

使用乘法扩展列表，生成一个新列表，新列表元素是原列表元素的多次重复。

a = ['sxt',100]

b = a\*3

print(a) #结果：['sxt', 100]

print(b) #结果：['sxt', 100, 'sxt', 100,'sxt', 100]适用于乘法操作的，还有：字符串、元组。例如：c = 'sxt'

d = c\*3 #结果：'sxtsxtsxt'

#### 列表元素的删除表

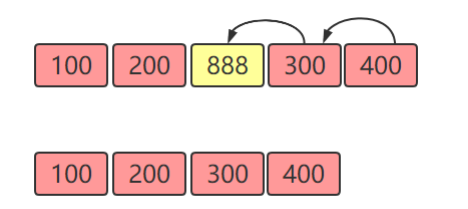
##### del 删除

删除列表指定位置的元素。

a = [100,200,888,300,400]

del a[2]

print(a) #结果：[100,200,300,400]



##### pop()方法

pop()删除并返回指定位置元素，如果未指定位置则默认操作列表最后一个元素。

a = [10,20,30,40,50]

b1 = a.pop() #结果：b1=50

print(a,b1) #结果：[10, 20, 30, 40] 50

b2 = a.pop(1)

print(a,b2) #结果：[10, 30, 40],20

##### remove()方法

删除首次出现的指定元素，若不存在该元素抛出异常。

a = [10,20,30,40,50,20,30,20,30]

a.remove(20) #[10, 30, 40, 50, 20, 30, 20,30]

a.remove(100) #报错：

ValueError:list.remove(x): x not in list

#### 列表元素访问和计数

##### 通过索引直接访问元素

我们可以通过索引直接访问元素。索引的区间在 [0, 列表长度-1] 这个范围。超过这个范围则会抛出异常。

a = [10,20,30,40,50,20,30,20,30]

print(a[2]) #结果：30

print(a[10]) #报错：IndexError: list index out of range

##### index()获得指定元素在列表中首次出现的索引

index() 可以获取指定元素首次出现的索引位置。语法是： index(value,[start,[end]]) 。其中， start 和 end 指定了搜索的范围。

>>> a = [10,20,30,40,50,20,30,20,30]

>>> a.index(20) #结果：1

>>> a.index(20,3) #结果：5 从索引位置3开始往后搜索的第一个20

>>> a.index(30,5,7) #结果：6 从索引位置5到7这个区间，第一次出现30元素的位置

##### count()获得指定元素在列表中出现的次数

count()可以返回指定元素在列表中出现的次数。

>>> a = [10,20,30,40,50,20,30,20,30]

>>> a.count(20)

3

##### len()返回列表长度

len()返回列表长度，即列表中包含元素的个数。

>>> a = [10,20,30]

>>> len(a)

3

#### 成员资格判断in,not in

判断列表中是否存在指定的元素，我们可以使用 count() 方法，返回0 则表示不存在，返回大于0则表示存在。但是，一般我们会使用更加 简洁的 in 关键字来判断，直接返回 True 或 False

>>> a = [10,20,30,40,50,20,30,20,30]

>>> 20 in a

True

>>> 100 not in a

True

>>> 30 not in a

False

#### 切片操作

类似字符串的切片操作，对于列表的切片操作和字符串类似。

切片是Python序列及其重要的操作，适用于列表、元组、字符串等等。

切片slice操作可以让我们快速提取子列表或修改。标准格式为：

[起始偏移量start:终止偏移量end[:步长step]]

##### 典型操作(三个量为正数的情况)如下：

操作和说明 示例 结果

[:] 提取整个列表 [10,20,30][:] [10,20,30]

[start:] 从start索引开始到结尾 [10,20,30][1:] [20,30]

[:end] 从头开始知道end-1 [10,20,30][:2] [10,20]

[start:end] 从start到end-1 [10,20,30,40][1:3] [20,30]

[start:end:step] 从start提取到end-1，步长是step

[10,20,30,40,50,60,70][1:6:2] [20, 40, 60]

##### 其他操作（三个量为负数）的情况：

示例 说明 结果

[10,20,30,40,50,60,70][-3:] 倒数三个 [50,60,70]

[10,20,30,40,50,60,70][-5:-3] 倒数第五个到倒数第三个(包头不包尾) [30,40]

[10,20,30,40,50,60,70][::-1] 步长为负，从右到左反向提取

[70, 60, 50, 40, 30, 20,10]

#### 列表的遍历操作

s=list("abcdef")#方法一，不带索引

for item in s:

print(item,end=" ")

结果：a b c d e f

for i in range(0,len(s)):#方法二，输出索引和值

print(i,s[i],end="\t\t")

结果：0 a 1 b 2 c 3 d 4 e 5 f

for index,item in enumerate(s,start=1): #方法三，输出值和修改后的索引

#方法三，start,设置索引号的起始值，可以省略不写，如enumerate(s,1)

print(index,item,end="\t\t")

结果：1 a 2 b 3 c 4 d 5 e 6 f

#### 复制列表所有的元素到新列表对象

如下代码实现列表元素的复制了吗？

list1 = [30,40,50]

list2 = list1

只是将list2也指向了列表对象，也就是说list2和list2持有地址值是相同的，列表对象本身的元素并没有复制。

我们可以通过如下简单方式，实现列表元素内容的复制：

list1 = [30,40,50]

list2 = [] + list1 #生成了新列表对象

#### 列表相关的其他内置函数汇总

##### max和min

用于返回列表中最大和最小值。

>>> a = [3,10,20,15,9]

>>> max(a)

20

>>> min(a)

3

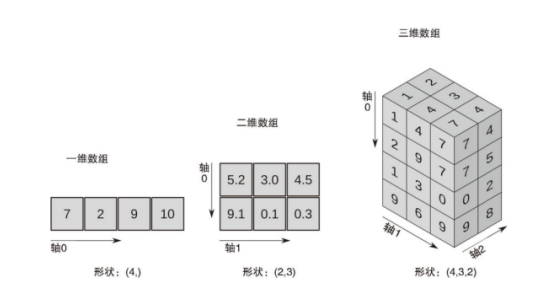
##### sum

>>> a = [3,10,20,15,9]

>>> sum(a)

57

#### 多维列表



##### 二维列表

一维列表可以帮助我们存储一维、线性的数据。

二维列表可以帮助我们存储二维、表格的数据。例如下表的数据：

姓名 年龄 薪资 城市

高小一 18 30000 北京

高小二 19 20000 上海

高小五 20 10000 深圳源码：

a = [

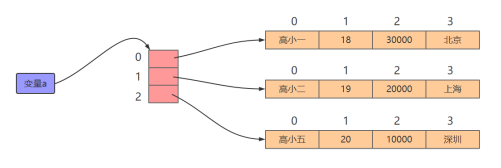
["高小一",18,30000,"北京"],

["高小二",19,20000,"上海"],

["高小五",20,10000,"深圳"],

]

内存结构图：



>>> print(a[1][0],a[1][1],a[1][2])

高小二 19 20000

###### 创建二维列表

lst=[

["城市","环比","对比"],

['北京',102,103],

['上海',104,504],

['深圳',100,39]

]

for row in lst:

for item in row:

print(item,end="\t")

print()

城市 环比 对比

北京 102 103

上海 104 504

深圳 100 39

# 用列表生成式生成一个4行5列的列表

lst2=[[j for j in range(5)]for i in range(4)]

print(lst2)

[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

###### 二维列表的遍历

for row in 二维列表

for item in row

pass

嵌套循环打印二维列表所有的数据

a = [

["高小一",18,30000,"北京"],

["高小二",19,20000,"上海"],

["高小一",20,10000,"深圳"],

]

for m in range(3):

for n in range(4):

print(a[m][n],end="\t")

print() #打印完一行，换行

运行结果：

高小一 18 30000 北京

高小二 19 20000 上海

高小一 20 10000 深圳

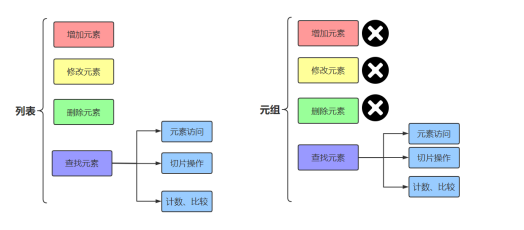
### tuple元组类型()不可变数据类型

英 /tjuːpl/

是python中内置的不可变序列

使用（）定义元组，元素与元素之间作用英文的逗号分隔

元组中只有一个元素的时候，逗号也不能省略



列表属于可变序列，可以任意修改列表中的元素。

元组属于不可变序列，不能修改元组中的元素。

因此，元组没有增加元素、修改元素、删除元素相关的方法。

因此，我们只需学元组的创建和删除，元素的访问和计数即可。元组支持如下操作：

1 索引访问

2 切片操作

3 连接操作

4 成员关系操作

5 比较运算操作

6 计数：元组长度len()、最大值max()、最小值min()、求和sum()等

#### 元组的两种创建方式

##### 使用()直接创建元组

通过()创建元组。小括号可以省略。a = (10,20,30) 或者 a = 10,20,30

如果元组只有一个元素，则必须后面加逗号。这是因为解释器会把(1)解释为整数1，(1,)解释为元组。a = 10,

语法结构如下：

元组名=(a,b,c)

如：

t=("a",[1,2,3],"python")

print(t)

('a', [1, 2, 3], 'python')

>>> a = (1)

>>> type(a)

<class 'int'>

>>> a = (1,) #或者 a = 1,

>>> type(a)

<class 'tuple'>

##### 使用内置函数tuple()创建元组

tuple(可迭代的对象)

语法结构如下：

元组名=tuple(序列)

t=tuple("hallo")

print(t)

('h', 'a', 'l', 'l', 'o')

t=tuple([10,20,30])

print(t)

(10, 20, 30)

a = tuple() #创建一个空元组对象

b = tuple("abc")

c = tuple(range(3))

d = tuple([2,3,4])总结：

1 tuple()可以接收列表、字符串、其他序列类型、迭代器等生成元组。2 list()可以接收元组、字符串、其他序列类型、迭代器等生成列表。

#### 元组的元素访问和计数

**1 元组的元素不能修改**

>>> a = (20,10,30,9,8)

>>> a[3]=33

Traceback (most recent call last):

File "<pyshell#313>", line 1, in

<module>

a[3]=33

TypeError: 'tuple' object does not support

item assignment

**2 元组的元素访问、index()、count()、切片等操作，和列表一样。**

>>> a = (20,10,30,9,8)

>>> a[1]

10

>>> a[1:3]

(10, 30)

>>> a[:4]

(20, 10, 30, 9)

3 列表关于排序的方法list.sorted()是修改原列表对象，元组没有该方法。如果要对元组排序，只能使用内置函数 sorted(tupleObj)，并生成新的列表对象。

a = (20,10,30,9,8)

b = sorted(a) #b是新对象，内容是：[8, 9,10, 20, 30]

#### zip

zip(列表1，列表2，...)将多个列表对应位置的元素组合成为元组，并返回这个zip对象。

如果各个迭代器的元素个数不一致，则返回列表长度与最短的对象相同a = [10,20,30]

b = [40,50,60]

c = [70,80,90,100]

d = zip(a,b,c)

print(d) #zip object

e = list(d) #列表：[(10, 40, 70), (20, 50,80), (30, 60, 90)]

print(e)

#### 生成器推导式创建元组

1 从形式上看，生成器推导式与列表推导式类似，只是生成器推导式使用小括号。

2 列表推导式直接生成列表对象，生成器推导式生成的不是列表也不是元组，而是一个生成器对象。

3我们可以通过生成器对象，转化成列表或者元组。也可以使用生成器对象的 \_\_next\_\_() 方法进行遍历，或者直接作为迭代器对象来使用。不管什么方式使用，元素访问结束后，如果需要重新访问其中的元素，必须重新创建该生成器对象。

【操作】生成器的使用测试

#列表推导式： [0, 2, 4, 6, 8]

s = (x\*2 for x in range(5))

print(s) #<generator object <genexpr> at 0x0000021C80BE2880>

b = tuple(s)

print(b) #(0, 2, 4, 6, 8)

c = tuple(s)

print(c) #()

s2 = (x for x in range(3))

print(s2.\_\_next\_\_()) #0

print(s2.\_\_next\_\_()) #1

print(s2.\_\_next\_\_()) #2

print(s2.\_\_next\_\_()) #报错：StopIteration

t=(item for item in range(1,4))

#元组生成式会生成生成器，如果需要使元素需要转换类型

print(tuple(t))#将T元组生成器转化成元组类型

结果：(1, 2, 3)

print(t.\_\_next\_\_())将生成器中的元素取出来，取出后生成器里被取出的元素就没有了

结果：1

#### 元组的遍历与列表相同

t=("a",[1,2,3],"python")

print(t[0])

print(t[0:3:2])#支持切片操作

元祖的三种遍历方式

for item in t:#遍历1，不带索引，只输出值

print(item,end=" ")

print()

a [1, 2, 3] python

for item in range(len(t)):#遍历2，带索引并且输出值

print(item,t[item],end=" ")

print()

0 a 1 [1, 2, 3] 2 python

for index,item in enumerate(t,1):#遍历3修改了索引值为1，输出索引和值

print(index,item, end=" ")

print()

元组总结

1 元组的核心特点是：不可变序列。

2 元组的访问和处理速度比列表快。

3 与整数和字符串一样，元组可以作为字典的键，列表则永远不能作为字典的键使用。

### 字典类型dict{}（可变数据类型）

英 /dɪkt/

字典类型是根据一个信息查找另一个信息的方式构成了“键值对”，它表示索引用的键和对应的值构成的成对关系。和列表和元组不一样。字典没有整数索引的概念，想要检索字典当中的元素通过key去检索元素。元组因数不可变数据可以做为键使用，列表是可变数据类型不能做为键

key 身份证号1 身份证号2 身价证号3 身份证号4

value 具体的对象1 具体的对象2 … …

#### 字典的创建方式

##### 1.使用{}直接创建字典

d={key1:value1,key2:value2…}

d={10:"cat",20:"dog",30:"pet",20:"zoo"}

print(d)

{10: 'cat', 20: 'zoo', 30: 'pet'}

#第二个20将第一个20的值覆盖了，说明键相同值会覆盖

##### 2.使用内置函数dict()创建字典

1,语法结构如下：dict(key1=value1,key2=value2…

dd=dict(cat=10,dog=20)

print(dd) 结果：{'cat': 10, 'dog': 20}

t=(10,20,30)

print({t:10})#元组做为键

{(10, 20, 30): 10}

t=[10,20,30]

print({t:10})#列表做为键，结果报错，因为列表是可变数据类型不能做为键

b =

dict(name='gaoqi',age=18,job='programmer')

a = dict([("name","gaoqi"),("age",18)])

c = {} #空的字典对象

d = dict() #空的字典对象

##### 3, 通过zip()创建字典对象

lis1=[10,20,30,40]

lis2=["cat","dog","pet","zoo","cat"]

c=zip(lis1,lis2)#创建了映射类型

print(type(c))

d=dict(c)#将映射类型转成字典类型

print(d)

结果：

<class 'zip'>zip就是映射类型

{10: 'cat', 20: 'dog', 30: 'pet', 40: 'zoo'}

##### 4.通过fromkeys创建值为空的字典

f = dict.fromkeys(['name','age','job'])

print(f) #结果：{'name': None, 'age':

None, 'job': None}

#### 字典元素的访问

1 通过 [键] 获得“值”。若键不存在，则抛出异常。

a =

{'name':'gaoqi','age':18,'job':'programmer

'}

b = a['name']

print(b)

2 通过get()方法获得“值”。❤️推荐使用。优点是：指定键不存在，返回None；也可以设定指定键不存在时默认返回的对象。推荐使用get()获取“值对象” a =

{'name':'gaoqi','age':18,'job':'programmer

'}

b = a.get('name')

c = a.get('gender','不存在')

print(b)

print(c)

d[key]或d.get(key) (key,其他)这是相当于自定义了一个其他的key

d={1: 'cat', 2: 'dog', 3: 'pet', 4: 'zoo'}

print("结果：",d[1])

print("结果：",d.get(1))

结果： cat

#### 列出所有的键值对

a ={'name':'gaoqi','age':18,'job':'programmer'}

b = a.items()

print(b) #dict\_items([('name', 'gaoqi'),('age', 18), ('job', 'programmer')])

#### 列出所有的键，列出所有的值

a ={'name':'gaoqi','age':18,'job':'programmer'}

k = a.keys()

v = a.values()

print(k) #dict\_keys(['name', 'age','job'])

print(v) #dict\_values(['gaoqi', 18,'programmer'])

#### len() 键值对的个数

a ={'name':'gaoqi','age':18,'job':'programmer'}

num = len(a)

print(num) #3

#### 检测一个“键”是否在字典中

a ={'name':'gaoqi','age':18,'job':'programmer'}

print("name" in a) #True

#### 字典元素添加、修改、删除

**1 给字典新增“键值对”。如果“键”已经存在，则覆盖旧的键值对；**

**如果“键”不存在，则新增“键值对**

a ={'name':'gaoqi','age':18,'job':'programmer'}

a['address']='西三旗1号院'

a['age']=16

print(a)

{'name': 'gaoqi', 'age': 16, 'job':'programmer', 'address': '西三旗1号院'}

**2 使用 update() 将新字典中所有键值对全部添加到旧字典对象上。如 果 key 有重复，则直接覆盖**

a ={'name':'gaoqi','age':18,'job':'programmer'}

b ={'name':'gaoxixi','money':1000,'gender':'男的'}

a.update(b)

print(a)

{'name': 'gaoxixi', 'age': 18, 'job':'programmer', 'money': 1000, 'gender': 男的'}

**3 字典中元素的删除，可以使用 del() 方法；或者 clear() 删除所有键值 对； pop() 删除指定键值对，并返回对应的“值对象**

a ={'name':'gaoqi','age':18,'job':'programmer'}

del(a['name'])

print(a) #{'age': 18, 'job':'programmer'}

age = a.pop('age')

print(age) #18

4 popitem() ：随机删除和返回该键值对。字典是“无序可变序列”， 因此没有第一个元素、最后一个元素的概念； popitem 弹出随机的 项，因为字典并没有"最后的元素"或者其他有关顺序的概念。若 想一个接一个地移除并处理项，这个方法就非常有效（因为不用 首先获取键的列表）

a ={'name':'gaoqi','age':18,'job':'programmer'}

r1 = a.popitem()

r2 = a.popitem()

r3 = a.popitem()

print(a) #{}

#### 序列解包

序列解包可以用于元组、列表、字典。序列解包可以让我们方便的 对多个变量赋值。

x,y,z=(20,30,10)

(a,b,c)=(9,8,10)

[m,n,p]=[10,20,30]

序列解包用于字典时，默认是对“键”进行操作； 如果需要对键值对操作，则需要使用items()；如果需要对“值”进行操作，则需要使用values()；s = {'name':'gaoqi','age':18,'job':'teacher'}

name,age,job=s #默认对键进行操作

print(name) #name

name,age,job=s.items() #对键值对进行操作

print(name) #('name', 'gaoqi')

name,age,job=s.values() #对值进行操作

print(name) #gaoqi

#### 表格数据使用字典和列表存储和访问

姓名 年龄 薪资 城市

高小一 18 30000 北京

高小二 19 20000 上海

高小五 20 10000 深圳r1 = {"name": "高小一","age":18,"salary":30000,"city":"北京"}

r2 = {"name": "高小二","age":19,"salary":20000,"city":"上海"}

r3 = {"name": "高小五","age":20,"salary":10000,"city":"深圳"}

tb = [r1, r2, r3]

# 获得第二行的人的薪资

print(tb[1].get("salary"))

# 打印表中所有的的薪资

for i in range(len(tb)): # i -->0,1,2

print(tb[i].get("salary"))

# 打印表的所有数据

for i in range(len(tb)):

print(tb[i].get("name"), tb[i].get("age"), tb

[i].get("salary"), tb[i].get("city"))

#### 字典核心底层原理(重要)

字典对象的核心是散列表。散列表是一个稀疏数组（总是有空白元素的数组），数组的每个单元叫做 bucket 。每个 bucket 有两部分：一个是键对象的引用，一个是值对象的引用。

由于，所有 bucket 结构和大小一致，我们可以通过偏移量来读取指定bucket 。

#### 字典元素的两种遍历

##### 遍历出key与value的元组

for element in d.items():

pass

for item in d.items():

print(item)

(1, 'cat')

(2, 'dog')

(3, 'pet')

(4, 'zoo')

##### 分别遍历出key和value

key: 键值；关键字value:数值

for key,value in d.items()

pass

for key,value in d.items():

print(key,value)

1 cat

2 dog

3 pet

4 zoo

#### 字典的相关操作方法

字典的方法 描述说明

d.keys() 获取所有的key数据

d.values() 获取所有的values数据

d.pop(key.default) key存在获取相应的values，同时删除key-value对，否则获取默认值

d.popitem() 随机从字典中取出一个key-value对，结果为元组类型，同时将该key-value从字典中删除

d.clear() 清空字典中所有的key-value对

d={1: 'cat', 2: 'dog', 3: 'pet', 4: 'zoo'}

print(d.keys())

dict\_keys([1, 2, 3, 4])

print(d.values())

dict\_values(['cat', 'dog', 'pet', 'zoo'])

print(list(d.values()))#将元素数值转列表类型打印

['cat', 'dog', 'pet', 'zoo']

添加字典元素没有方法直接赋值

d[5]='python'

print(d)

{1: 'cat', 2: 'dog', 3: 'pet', 4: 'zoo', 5: 'python'}

lis=list(d.items())#将字典的数据转换成键值对的形式

[(1, 'cat'), (2, 'dog'), (3, 'pet'), (4, 'zoo'), (5, 'python')]

d=dict(lis)#将lis转回字典类型

{1: 'cat', 2: 'dog', 3: 'pet', 4: 'zoo', 5: 'python'}

print(d.pop(1))#获取相应的values，同时删除key-value对

print(d)

cat

{2: 'dog', 3: 'pet', 4: 'zoo', 5: 'python'}

print(d.popitem()#随机从字典中取出一个key-value对

print(d)

(5, 'python')

{2: 'dog', 3: 'pet', 4: 'zoo'}

d.clear() 结果：{}

#### 字典生成式

##### 方法一：

d={key:value for item in range}

例：

d={item:random.randint(1,100) for item in range(4)}

#random.randint(1,100)生成1到100的随机值

print(d)

{0: 4, 1: 69, 2: 23, 3: 77}

##### 方法二

d={key:value for key,value in zip(lst1,lst2)}

例

lst1=[item for item in range(1,5)]

lst2=['guyue','yang','li','wang']

d={key:value for key,value in zip(lst1,lst2)}

print(d)

{1: 'guyue', 2: 'yang', 3: 'li', 4: 'wang'}

#### 字典用法总结：

1 字典在内存中开销巨大，典型的空间换时间。

2 键查询速度很快

3 往字典里面添加新键值对可能导致扩容，导致散列表中键的次序变化。因此，不要在遍历字典的同时进行字典的修改

4 键必须可散列

1 数字、字符串、元组，都是可散列的

2 自定义对象需要支持下面三点:(面向对象章节中再展开说)

1 支持 hash() 函数

2 支持通过 \_\_eq\_\_() 方法检测相等性

3 若 a==b 为真，则 hash(a)==hash(b) 也为真

### set集合类型{}

python中的集合与数学中集合的概念一致

集合是一个无序的不重复元素序列

集合中只能存储不可变数据类型

在python中集合使用{}定义

与列表、字典一样，都是python中的可变数据类型

#### 集合创建和删除

##### 使用{}直接创建对象，并使用add()方法添加元素

语法结构：s={element1,elementl2,…}

s={10,20,30,40} {40, 10, 20, 30}

a = {3,5,7}

a.add(9) #{9, 3, 5, 7}

##### 使用set()，将列表、元组等可迭代对象转成集合。如果原来数据存在重复数据，则只保留一个

语法结构：s=set(可迭代对象)

s=set("hallo") {'h', 'a', 'o', 'l'}#无序的不重复的元素，l少了一个

s2=set([10,20,30]) {10, 20, 30} #可以使用序列

a = ['a','b','c','b']

b = set(a) #{'b', 'a', 'c'}

##### 序列中的方法都可用

print(max(s2)) 30

print(min(s2)) 10

print(len(s2)) 3

print(10 in s2) True

##### 删除集合：del 集合名

del s2

#### 集合类型的操作符

a={10,20,30,40}

b={30,50,88,76,20}

a&b 交集:获取ab相交的部分 结果：{20, 30}

a|b 并集:获取ab全部的部分 结果： {40, 10, 76, 50, 20, 88, 30}

a-b差集 :获取a中不相交的部分 结果： {40, 10}

a^b补集：获取ab当中不相交的部分 结果： {50, 10, 88, 40, 76}

#### 集合的相关操作方法

s.add(x) 如果x不在集合中，则将x添加到集合s

s.remove(x0 如果x在集合中，将其删除，如果不在集合中，程序报错

s,clear 清除集合中所有元素 set()

#### 集合遍历

##### 方法1

s3= {40, 10, 76, 50, 20, 88, 30}

for item in s3:

print(item,end=" ")

结果：40 10 76 50 20 88 30

##### 方法2增加序号

for index,item in enumerate(s3,1):

print(index,item,end=" ")

1 40 2 10 3 76 4 50 5 20 6 88 7 30

#### 集合生成式

s={i for i in range(1,10)} #i作为集合获取的元素

print(s)

{1, 2, 3, 4, 5, 6, 7, 8, 9}

判断的方式

s={i for i in range(1,10) if i%2==1}#如果是奇数就给i

print(s)

{1, 3, 5, 7, 9}

### 列表、元组、字典、集合的区别

数据类型 序列类型 元素是否可重复 是否有序 定义符号

列表list 可变序列 可重复 有序 []

元组tuple 不可变序列 可重复 有序 ()

字典dict 可变序列 key不可重复，value可重复 无序 {key.value}

集合set 可变序列 不可重复 无序 {}

## phthon 3.11新特性

1. 结构模型匹配

语法结构如下：

match data:

case{}:

pass

case[]:

pass

case():

pass

case\_:

pass

2,字典合并运算符|

3,同步迭代：

match data1,data2:

case data1,dat2:

pass

例：

lis= [item for item in range(1,6)]

print(lis)

lis2=["a","b","c","d"]

print(lis2)

for l,l2 in zip(lis,lis2):

match l,l2:

case 1,"a":

print("1>>a")

case 2,"b":

print("2>>b")

case 3, "c":

print("3>>c")

case 4, "d":

print("4>>d")

结果：

[1, 2, 3, 4, 5]

['a', 'b', 'c', 'd']

1>>a

2>>b

3>>c

4>>d

### 实战

#### 千年虫

lis=[88,89,90,98,00,99]

print(lis)

#修改lis，将00前加200，其他中19

for i in range(len(lis)):#方法一

if str(lis[i]) != "0":

lis[i]="19"+str(lis[i])

else:

lis[i] = "200" + str(lis[i])

print(lis)

for item,value in enumerate(lis):#方法二

if str(value) != "0":

lis[item]="19"+str(value)

else:

lis[item] = "200" + str(value)

print(lis)

结果：

[88, 89, 90, 98, 0, 99]

['1988', '1989', '1990', '1998', '2000', '1999']

#### 模拟京东的购物流程

lst=[]#存添加商品

for item in range(1001,1006):

tianj\_shangp=input("请输入你需要添加的商品：")

lst.append(str(item)+tianj\_shangp)

print("已经添加的商品信息：")

for item in lst:

print(item)

lst2=[]#存购物车商品

while True:

flag=False#代表没有商品的情况

gouw\_che=str(input("请输入你购买的商品编号退出请输入Q:"))

for item in lst:

if item[0:4]==gouw\_che:

flag=True

lst2.append(item)

print(item[4::],"添加购物车成功")

break#退出了for循环

if not flag and gouw\_che.lower() !='q':#这里的not flag等于flag=false

print("商品不存在")

if gouw\_che.lower()=='q':

break

print("\_"\*50)

print("购物车的商品：")

lst2.reverse()

for item in lst2:

print(item[4::])

#### 模拟12306购票流程

che\_c={

'G1569':['北京南-天津南','18:06','18:39','00:33'],

'G1567':['北京南-天津南','18:15','18:49','00:34'],

'G8917':['北京南-天津西','18:20','19:19','00:59'],

'G203':[' 北京南-天津南','18:35','19:09','00:34']

}

print('车次\t\t出发站-到达站\t\t出发时间\t\t到达时间\t\t历时时长')

for key in che\_c.keys():

print(key,end=" ")

for item in che\_c.get(key):

print(item,end="\t\t")

print()

che\_m=input("请输入购票车次：")

info=che\_c.get(che\_m.upper(),"车次不存在")#info是一个列表类型

if info != "车次不存在":

xingming=input("请输入乘车人，如果是多位乘车人使用逗号分号：")

s=info[0]+' '+info[1]

print("您已购买了"+che\_m+" "+s+",请"+xingming.title()+'换取纸质车票。【铁路客服】')

else:

print('对不起！选择的车次可能不存在')

#### 模拟手机通讯录

因为手机号码不能重复所以用集合类型

tongxl=set()

for i in range(1,5):

into=input(f'请输入第{i}位好友的姓名和手机号码：')

tongxl.add(into)

print('已添加的号码')

for item in tongxl:

print(item)

## 字符串及正则表达式

### 字符串的常用操作

字符串是python中的不可变数据类型

方法名： 描述说明

str.lower() 将str字符串全部转成小写字母，结果为一个新的字符串

str.upper() 将str字符串全部转成大写字母，结果为一个新的字符串

str.split(sep=None) 把str按照指定的分隔符sep进行分隔，结果为列表类型

str.count(sub) 结果为sub这个字符串在str中出现的次数

str.find(sub) 查询sub这个字符串在str中是否存在，如果不存在结果为-1，如果 存在，结果为sub首次出现的索引

str.index(sub) 功能与find()相同，区别在于要查询的子串sub不存在时，程序报错

str.startswith(s) 查询字符串str是否以子串s开头

str.endswith(s) 查询字符串str是否以子串s结尾

str.replace(old,news) 使用new替换字符串s中所有的old字符串，结果是一个新的字符串

str.center(width,fillchar) 字符串str在指定的宽度范围内居中，可以使用fillchar进

行填充

str.join(iter) 在iter中的每个元素的后面都增加一个新的字符串str

str.trip(chars) 从字符串中去掉左侧和右侧chars中列出的字符串

str.lstrip(chars) 从字符串中去掉左侧chars中列出的字符串

str.rstrip(chars) 从字符串中去掉右侧chars中列出的字符串

s1="guyue@163.com"

lst=s1.split('@')

print('邮箱名',lst[0],'邮件服务器域名：',lst[1],)

以@为分隔，将字符分成了两个元素

print('python.py'.endswith('.py'))#判断字符串是不是以.py字符串结尾的

True

s1="guyue163.com"

print(s1.replace('u','o'))#默认替换所有

goyoe163.com

print(s1.replace('u','o',1))#设置替换一次

goyue163.com

print(s1.center(20))

guyue163.com

print(s1.center(20,'\*'))

\*\*\*\*guyue163.com\*\*\*\*

### 格式化字符串的三种方式

#### 占位符

%s:字符串格式。%d:十进制整数格式。%f:浮点数格式

name='马东梅'

age=18

score=98.5

print("姓名：%s,年龄：%d,成绩：%f"%(name,age,score))

姓名：马东梅,年龄：18,成绩：98.500000

print("姓名：%s,年龄：%d,成绩：%.1f"%(name,age,score))#符点占位符小数点为0.1

姓名：马东梅,年龄：18,成绩：98.5

#### f-string

python3.6引入的格式化字符串的方式，比{}标明被替换的字符

print(f'姓名：{name},年龄：{age}，成绩：{score}')

姓名：马东梅,年龄：18,成绩：98.5

#### str.format()方法

模板字符串.format(逗号分隔的参数)

print('姓名：{0},年龄：{1}，成绩：{2}'.format(name,age,score))

姓名：马东梅,年龄：18，成绩：98.5

format详细格式控制

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ： | 填充 | 对齐方式 | 宽度 | , | .精度 | 类型 |
| 引导符号 | 用于填充单个字符 | <左对齐  >右对齐  ^居中对齐 | 字符串的输出宽度 | 数字的千分位分隔符（只能用于整数和符点数） | 浮点数小数部分的精度或字符串的最大输出长度 | 整数类型  b\d\o\x\X  浮点整数型：e\E\f\% |

s='helloworld'

print('{0:\*<20}'.format(s))#:引导符号，\*是填充的字符，<左对齐，20是字符串的宽度

helloworld\*\*\*\*\*\*\*\*\*\*

print('{0:\*>20}'.format(s))#右对齐填充的字符填到了左侧

\*\*\*\*\*\*\*\*\*\*helloworld

print('{0:\*^20}'.format(s))#居中对齐

\*\*\*\*\*helloworld\*\*\*\*\*

print('{0:,}'.format(4562123))#引用的千分位分隔符

4,562,123

print('{0:,.2f}'.format(456463.141516))#.2f代表占位两位

456,463.14

print('{0:.5}'.format('helloworld'))#.5精度字串的最大输出长度

hello

print('二进制：{0:b},十进制：{0:d},八进制：{0:o},十六进制：{0:x},十六进制：{0:X}'.format(20))

二进制：10100,十进制：20,八进制：24,十六进制：14,十六进制：14

print('{0:.2e},{0:.2E},{0:.2f},{0:.2%}'.format(3.14))#符点数类型采用的是科学计数法

3.14e+00,3.14E+00,3.14,314.00%

### 字符串的编码和解码

你的电脑骗码是str》》》》》传输电脑是bytes》》》》》》别人的电脑解码str

#### 字符串的编码

将str类型转换成bytes类型，需要使用到字符串的encode()方法

语法格式：

str.encode(encoding='utf-8',

errors='strict/ignore/replace')

errors出错后ignore出错后忽略不编了，

ict是严格的出错会显示编码错误

replace编不了会替换成？问号

s='伟大的中国梦'

s.encode(errors='replace')#默认是的utf-8

b'\xe4\xbc\x9f\xe5\xa4\xa7\xe7\x9a\x84\xe4\xb8\xad\xe5\x9b\xbd\xe6\xa2\xa6'

ss=s.encode(encoding='gbk',errors='replace')#默认是的utf-8

#encodein改变了编码格式

#errors显示是出错后怎么样

b'\xce\xb0\xb4\xf3\xb5\xc4\xd6\xd0\xb9\xfa\xc3\xce'

#### 字符串的解码

将bytes类型转换成str类型，需要使用到bytes类型的decode()方法

语法格式：

bytes.decode(encoding='utf-8',

errors='strict/ignore/replace')

s.decode(encoding='gbk',errors='replace')

伟大的中国梦

### 数据的验证

数据的验证是指程序对用户输入的数据进行“合法”性验证

方法名： 描述说明

str.isdigit() 所有字符都是数字（阿拉伯数字）

str.isnumeric() 所有字符都是数字（罗马的，中国数字的大写都可以认）

str.isalpha() 所有字符都是字母（包含中文字符）

str.isalnum() 所有字符都是数字或字母（包含中文字符）

str.islower() 所有字符都是小写

str.isupper() 所有字符都是大字

str.istitle() 所有字符都是首字母大写

str.isspace() 所有字符都是空白字符（\n,\t等）

### 数据的处理

### 字符串的处理

#### 字符串的拼接操作

s='伟大的中国梦'

s1='我的'

print(s1+s)

我的伟大的中国梦

1. **使用str.join()方法进行拼接字符串**

print(''.join(s1+s))

print('\*'.join(['a','b','c']))

a\*b\*c#将逗号人前面的\*号拼接在一起了

1. **直接拼接**

print('你好''python')

你好python

1. **使用格式化字符串进行拼接**

用上面所学的三种占位符的方式

#### 字符串的去重操作

**1.遍历操作**

s='你好你好你好你好你好'

news=''

for item in s:

if item not in news:

news += item

print(news)

你好

1. **索引加not in**

s='你好你好你好你好你好'

for i in range(len(s)):

if s[i] not in news:

news +=s[i]

print(news)

**3.通过集合去重**

s='helloworld'

news2=set(s)#通过集合把内容去重了，但是集合是无序了

lis=list(news2)#将集合转成列表

lis.sort(key=s.index)#通过原来S的索引排序

print(lis)

结果：', 'e', 'l', 'o', 'w', 'r', 'd']

print(''.join(lis))#再使用join把列表字符串拼接回来

结果lowrd

### 正则表达式的使用

元字符：具有特殊意义的专用字符，例如 ^ $分别表示匹配的开始和结束

|  |  |  |  |
| --- | --- | --- | --- |
| 元字符 | 描述说明 | 举例 | 结果 |
| . | 匹配任意字符（除\n） | ‘p\nytho\tn’ | p,y,t,h,o,\t,n |
| \w | 匹配字母，数字，下划线 | ‘python\n123 | p,y,t,h,o,n,1,2,3 |
| \W | 匹配非字母，数字，下划线 | 'python\n123' | \n |
| \s | 匹配任意空白字符 | 'python\t123' | \t |
| \S | 匹配任意非空白字符 | 'python\t123' | python123 |
| \d | 匹配任意十进制数 | 'python\t123' | 123 |
| ? | 匹配前面的字符0次或1次 | colou?r | 可以匹配color或colour |
| + | 匹配前面的字符1次或多次 | colou+r | 可以匹配colour或colouu…r |
| \* | 匹配前面的字符0次或多次 | colou\*r | 可以匹配color或colouu…r |
| {n} | 匹配前面的字符n次 | colou{2}r | 可以匹配colouur |
| {n,} | 匹配前面的字符最少n次 | colou{2,}r | 可以匹配colouur或colouuu..r |
| {n,m} | 匹配前面的字符最小n次最多m次 | colou{2,4}r | 可以匹配colouur或colouuur或者colouuuur |
| 区间字符[] | 匹配[]中所指定的字符 | [.?!]  [0-9] | 匹配标点符号点、问号、感叹号  匹配0、1、2、3、…9 |
| ^（排除字符） | 匹配不在[]中指定的字符 | [^0-9] | 匹配除0、1、2…9的字符 |
| |（选择字符） | 用于匹配|左右的任意字符 | \d{18}|\d{15} | 匹配15位身份证或18位身份证 |
| \.（转义字符） | 同python中的转义字符 | \. | 将.作为普通字符使用 |
| [\u4e00-\u9fa5] | 匹配任意一个汉字 |  |  |
| 分组（） | 改变限定符的作用 | six|fourth  (six|four)th | 匹配six或fourth  匹配sixth或fourth |

### re模块

python中的内置模块，用于实现python中的正则表达式操作

|  |  |
| --- | --- |
| 函数 | 功能描述 |
| re.match(pattern,string,flags=0) | 从字符串的开头开始匹配正则表达式。如果匹配成功，返回一个match对象；否则返回None |
| re.search(pattern, string, flags=0) | 用于在整个字符串中搜索第一个匹配的值，如果匹配成功，结果为Match对象，否则结果为None |
| re.findall(pattern, string, flags=0) | 用于整个字符串搜索所有符合正则表达式的值，结果是一个列表类型 |
| re.sub(pattern, repl, string, count=0, flags=0) | 用于实现对字符串中指定子串的替换 |
| re.split(pattern, string, maxsplit=0, flags=0) | 字符串中的split()方法功能相同，都是分隔字符串 |

pattern：要匹配的正则表达式模式。英 /ˈpætn/

string：要进行匹配的字符串。

flags（可选）：标志位，用于控制正则表达式的匹配方式，如re.I（忽略大小写）、re.M（多行模式）等 英 /flæɡz/

例：

import re#导入

pattern='\d\.\d+'

#\d:0到9数字。\.转义字符将,作为普通字符。 +限定符定0到9的数字可以出现1次到多次

s2='3.11python i study every day'#待匹配字符串

match2=re.match(pattern,s2,re.I)

print(match2)

<re.Match object; span=(0, 4), match='3.11'>

3.11第一次\d是3,\.是中间的点，11是\d

print('匹配的起始位置：',match2.start())

print('匹配的结束位置：',match2.end())

print('匹配的区间位置：',match2.span())

print('匹配的字符串：',match2.string)

print('匹配的数据：',match2.group())

匹配的起始位置： 0

匹配的结束位置： 4

匹配的区间位置： (0, 4)

匹配的字符串： 3.11python i study every day

匹配的数据： 3.11

import re

pattern='\d\.\d+'#\d:0到9数字。\.转义字符将,作为普通字符。 +限定符定0到9的数字可以出现1次到多次

s='i study python every day4.22'#待匹配字符串

match=re.search(pattern,s,re.I)

print(match)#结果None

print(match.())#输出找到的数字

<re.Match object; span=(24, 28), match='4.22'>

4.22

import re

pattern='\d\.\d+'#\d:0到9数字。\.转义字符将,作为普通字符。 +限定符定0到9的数字可以出现1次到多次

s='3.11 i study python every day 4.22'#待匹配字符串

s2='i study python every day'#待匹配字符串

match=re.findall(pattern,s,re.I)

match2=re.findall(pattern,s2,re.I)

print(match)

print(match2)

['3.11', '4.22']

[]

import re

pattarn='黑客|破解|反爬'

s='我想学习python想破解一些VIP视频，python可以实现反爬吗？'

new\_s=re.sub(pattarn,"\*\*",s)

print(new\_s)

我想学习python想\*\*一些VIP视频，python可以实现\*\*吗？

import re  
s='token/MS4wLWFJ-EAKpHAIVJwA/source=list&log\_from=c2a1e'  
pattern2='[-|=|?|/]'  
spl=re.split(pattern2,s)  
print(s)  
print(spl)

token/MS4wLWFJ-EAKpHAIVJwA/source=list&log\_from=c2a1e

['token', 'MS4wLWFJ', 'EAKpHAIVJwA', 'source', 'list&log\_from', 'c2a1e']

### 实战2

#### 车版归属地

lis=['京A0000','沪A8888','苏A6666']

for i in lis:

if i[0:1]=='京':

print(i,'归属地是：北京')

if i[0:1] == '沪':

print(i, '归属地是：上海')

if i[0:1] == '苏':

print(i, '归属地是：苏州')

#### 统计字符串中出现指定字符的次数

a='HelloPython,HelloJava,Hellophp'

# #用记用键盘录入 要查询的字符（不区分大小写),要求统计出要查找的字符在字符串中出现的次娄

# tongji=0

shuru=input('请输入一个需要查询的字符：')

# for item in range(len(a)):#方法1

# if a[item].upper() == shuru.upper():

# tongji += 1

# print(shuru,'出现次数是：',tongji)

#方法2

print('{0}在{1}出现了{2}次'.format(shuru,a,a.count(shuru.upper())))

结果：loPython,HelloJava,Hellophp出现了3次

#### 格式化输出商品名称和单价

from dis import print\_instructions  
  
lis=[['01','电风扇','美的',500],  
 ['02','洗衣机','TCL',1000],  
 ['03','微波炉','老板',400]  
]  
#格式化  
print("编号\t\t\t商品\t\t\t品牌\t\t单价")  
for i in lis:  
 i[0] = '0000' + i[0]  
 i[3] = '￥{0:,.2f}'.format(i[3])  
  
for item in lis:  
 for i in item:  
 print(i,end="\t\t")  
 print()

结果：

编号 商品 品牌 单价

000001 电风扇 美的 ￥500.00

000002 洗衣机 TCL ￥1,000.00

000003 微波炉 老板 ￥400.00

#### 提取提出前面的成代

import re  
s='''146T1008 130/70R17 M/C 65H KM001F 1373440389 485 09A  
18063065 135/60R13 M/C 66S KR209(奔马) 10454489 486 20D  
K230B238 135/70R12 65Q KR23 096544 486 30A  
K230B204 135/70R12 65Q KR23(电动四轮汽车） 096544 486 30A  
33212069 135/80R13 70N KR209环 104544 486 30A  
3R212069 135/80R13 70N KR209 104544 06AE15 30A  
145T2008 140/70R17 M/C 66V KM001F 1373440389 485 09A  
133E1036 145/60R16 M/C 66T KR390 12855489 486 30A  
133E1069 145/60R16 M/C 66T KR390 12855489 486 30A  
13462031 145/60R16 M/C 66T KR392 12855489 486 30A  
13462080 145/60R16 M/C 66T KR392 12855489 486 30A  
'''  
pattern='\w{8}\s'  
s2=re.findall(pattern,s)  
s3=''  
for i in s2:  
 if i[-1]=='\t':  
 s3+=i  
print(s3)

结果：146T1008 18063065 K230B238 K230B204 33212069 3R212069 145T2008 133E1036 133E1069 13462031 13462080

## BUG的由来及分类

它指的是检测并并排除计算机程序/机器中的故障

### 异常是什么？

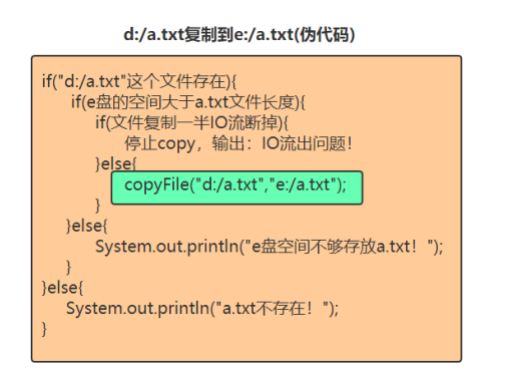
软件程序在运行过程中，非常可能遇到刚刚提到的这些问题，我们 称之为异常，英文是： Exception ，意思是例外。遇到这些例外情况， 或者叫异常，我们怎么让写的程序做出合理的处理，安全的退出， 而不至于程序崩溃呢？我们本章就要讲解这些问题。

工作中，程序遇到的情况不可能完美。比如：程序要打开某个 文件，这个文件可能不存在或者文件格式不对；程序在运行 着，但是内存或硬盘可能满了等等。

#### 伪代码说明异常机制

如果我们要拷贝一个文件，在没有异常机制的情况下，我们需要考

虑各种异常情况，伪代码如下：

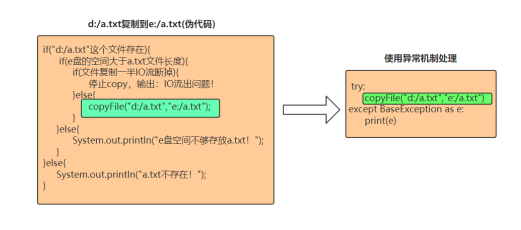


这种方式，有两个坏处：

1 逻辑代码和错误处理代码放一起！

2 程序员本身需要考虑的例外情况较复杂，对程序员本身要求较高！

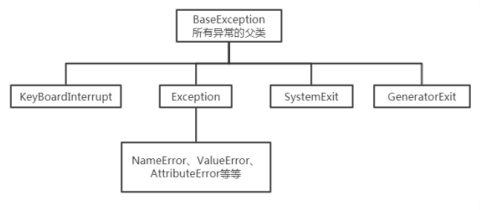
如上情况，如果是用Python的异常机制来处理，对比如下：



#### 异常机制本质：

当程序出现异常，程序安全的退出、处理完后继续执行的机制

python中，引进了很多用来描述和处理异常的类，称为异常类。异常类定义中包含了该类异常的信息和对异常进行处理的方法。下面较为完整的展示python中内建异常类的继承层次：



我们处理一下，遇到的第一个异常：

#测试简单的0不能做除数异常

a = 3/0

结果：

Traceback (most recent call last):

File "F:\gentel\learn\system\Python file\learning\python2\111.py", line 1, in <module>

a = 3/0

~^~

ZeroDivisionError: division by zero

python中一切都是对象，异常也采用对象的方式来处理。处理过程：

1.抛出异常：在执行一个方法时，如果发生异常，则这个方法生成代表该异常的一个对象，停止当前执行路径，并把异常对象提交给解释器。

2 捕获异常：解释器得到该异常后，寻找相应的代码来处理该异常

#### 异常的解决思路

解决异常问题的态度

1 学习完异常相关知识点，只是开始对异常有些认识，不意味着你会调试任何异常；

2调试异常，需要大量的经验作为基础。因此，大家不要在此停留，继续往后学习。碰到每个异常，都要花心思去解决而不要动不动张口问人。通过自己的努力无法解决，再去找老师同学帮助解决。

解决每一个遇到的异常，建议大家遵循如下三点：

1 不慌张，细看信息，定位错误。看清楚报的错误信息，并定位发生错误的地方。

2 百度并查看十个相关帖子。将异常类信息进行百度，至少查看十个以上的相关帖子。

3 以上两步仍然无法解决，找老师和同学协助解决。

正常情况，自己遵循如上步骤解决30个以上的错误，就能积累初步的调试经验，以后遇到的大部分错误都能独立完成。

#### 异常解决的关键：定位

当发生异常时，解释器会报相关的错误信息，并会在控制台打印出 相关错误信息。我们只需按照从上到下的顺序即可追溯 （Trackback）错误发生的过程，最终定位引起错误的那一行代码。

【示例】追溯异常发生的过程

### BUG的由来及分类

#### 粗心导致的语法错误：

1. 漏了末尾的冒号，如if语句、循环语句、else子句等
2. 缩进错误，该缩进的没有缩进，不该缩进的乱缩进
3. 把英文符号写成中文符号，例如：引号、冒号、括号
4. 字符串拼接的时候，把字符串和数字拼在一起
5. 没有定义变量，例如：while循环条件的变量没有定义
6. “==”比较运算符和“=”赋值运算符的混用

#### 思路不清导致的问题

解决思路不清导致的问题的思路有两种：

1. 第一种是使用print()函数
2. 第二种是使用#暂时注释部分代码

不同异常类型的处理方式

### python的异常处理机制

#### try…except

try:

可能会抛出异常的代码

except 异常类型：

异常处理代码（报错后执行的代码）

try:

被监控的可能引发异常的语句块

except BaseException [as e]:

异常处理语句块

1 try 块包含着可能引发异常的代码， except 块则用来捕捉和处理发生的异常。

2 执行的时候，如果 try 块中没有引发异常，则跳过 ecept 块继续执行后续代码；

3执行的时候，如果 try 块中发生了异常，则跳过 try 块中的后续代码，跳到相应的 except 块中处理异常；异常处理完后，继续执行后续代码。

【示例】遇到异常的执行顺序

#测试简单的0不能做除数异常  
try:  
 print("step1")  
 a = 3/0  
 print("step2")  
except BaseException as e:  
 print("step3")  
 print(e)  
print("step4")

step1

step3

division by zero

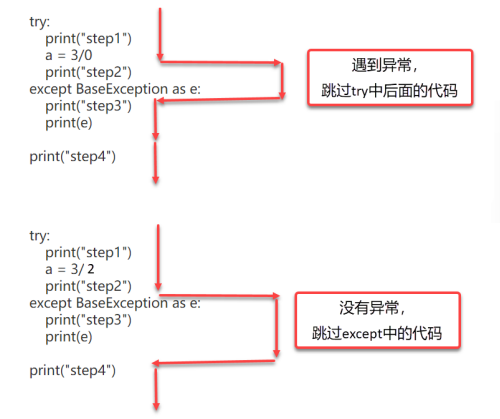
step4

#无异常的顺序  
try:  
 print("step1")  
 a = 3/2  
 print("step2")  
except BaseException as e:  
 print("step3")  
 print(e)  
print("step4")

step1

step2

step4



【示例】循环输入数字，如果不是数字则处理异常；直到输入 88 ， 则结束循环。

while True:  
 try:  
 x = int(input("请输入一个数字："))  
 print("您入的数字是",x)  
 if x==88:  
 print("退出程序")  
 break  
 except BaseException as e:  
 print("异常：输入的不是一个数字")  
 print(e)

执行结果：

请输入一个数字：11

您入的数字是 11

请输入一个数字：rd

异常：输入的不是一个数字

invalid literal for int() with base 10: 'rd'

请输入一个数字：88

您入的数字是 88

退出程序

#### try...多个except结构

上面的结构可以捕获所有的异常，工作中也很常见。但是，从经典 理论考虑，一般建议尽量捕获可能出现的多个异常（按照先子类后父类的顺序），并且针对性的写出异常处理代码。为了避免遗漏可 能出现的异常，可以在最后增加 BaseException 。结构如下：

try…except..except的语法结构为

try:

被监控的、可能引发异常的语句块

except Exception1:

处理Exception1的语句块

except Exception2:

处理Exception2的语句块

[...]

except BaseException：

处理可能遗漏的异常的语句块

**举例：如果用户输入10/0，程序就会报错**

try:  
 num1=int(input("输入"))  
 num2 = int(input("输入"))  
 print(num1/num2)  
except ZeroDivisionError:#代码中出现了除以零的情况，提醒你代码中存在问题。  
 print('除数不能为0')  
except ValueError:#代码出现除法出现字母的错误，提醒你  
 print('字母文字不能为除')  
except BaseException:  
 print('未知异常')

结果：输入10 输入0

#### try …except…else的语法结构为：

try...except...else 结构增加了 else块 。如果 try 块中没有抛出异常，则执行 else 块。如果 try 块中抛出异常，则执行 except 块，不执行 else 块。

try:

可能会抛出异常的代码

except 异常类型：

异常处理代码（报错后执行的代码）

else:

没有异常要执行的代码

try:  
 num1=int(input("输入"))  
 num2 = int(input("输入"))  
except ZeroDivisionError:#代码中出现了除以零的情况，提醒你代码中存在问题。  
 print('除数不能为0')  
except ValueError:#代码出现除法出现字母的错误，提醒你  
 print('字母文字不能为除')  
except BaseException:  
 print('未知异常')  
else:  
 print(num1/num2)

#### try...except...finally结构

try …except…else的语法结构为：

try:

可能会抛出异常的代码

except 异常类型：

异常处理代码（报错后执行的代码）

else:

没有异常要执行的代码

finally:

无论是否产生异常都要执行的代码

try:  
 num1=int(input("输入"))  
 num2 = int(input("输入"))  
except ZeroDivisionError:#代码中出现了除以零的情况，提醒你代码中存在问题。  
 print('除数不能为0')  
except ValueError:#代码出现除法出现字母的错误，提醒你  
 print('字母文字不能为除')  
except BaseException:  
 print('未知异常')  
else:#没有异常要执行的代码  
 print(num1/num2)  
finally:#无论是否产生异常都要执行的代码  
 print('程序执行结束')

【示例】读取文件， finally 中保证关闭文件资源

try:

f = open("d:/a.txt",'r')

content = f.readline()

print(content)

except BaseException as e:

print(e)

finally:

f.close() #释放资源。此处也可能会发生异常。若发生异常，则程序终止,不会继续往下执行

print("step4")

#### return语句和异常处理问题

由于 return 有两种作用：结束方法运行、返回值。我们一般不把 return 放到异常处理结构中，而是放到方法最后。

【示例】 return 和异常结构的正确处理方式

def test01():  
 print("step1")  
 try:  
 x = 3 / 0  
 # return "a"  
 except:  
 print("step2")  
 print("异常：0不能做除数")  
 # return "b"  
 finally:  
 print("step4")  
 # return "d"  
 print("step5")  
 return "e" # 一般不要将return语句放到try、except、 else 、finally块中，会发生一些意想不到的错误。建议放到方法最后。  
print(test01())

step1

step2

异常：0不能做除数

step4

step5

e

### 自定义异常类raise

抛出一个异常，从而提醒程序出现了异常情况，程序能够正确地处理这些异常情况

语法结构为：

自定义异常由 raise 语句主动抛出。

raise[Exception类型（异常描述信息）]

try:  
 xingbie=input('请输入性别：')  
 if xingbie != '男' and xingbie != '女':  
 raise Exception('性别只能写男或女')#Exception抛出异常  
 else:  
 print('你的性别是：',xingbie)  
except Exception as e:#捕获Exception异常  
 print(e)#输出异常

【示例】自定义异常类和raise语句

#测试自定义异常类  
class AgeError(Exception): #继承Exception  
 def \_\_init\_\_(self,errorInfo):  
 Exception.\_\_init\_\_(self)  
 self.errorInfo = errorInfo  
 def \_\_str\_\_(self):  
 return str(self.errorInfo)+",年龄错误！应该在1-150之间"  
############测试代码################  
if \_\_name\_\_ == "\_\_main\_\_": #如果为True，则模块是作为独立文件运行，可以执行测试代码  
 age = int(input("输入一个年龄:"))  
 if age<1 or age>150:  
 raise AgeError(age)  
 else:  
 print("正常的年龄：",age)

Traceback (most recent call last):

File "F:\gentel\learn\system\Python file\learning\python2\111.py", line 12, in <module>

raise AgeError(age)

AgeError: 200,年龄错误！应该在1-150之间

自定义异常类一般都是运行时异常，通常继承 Exception 或其子类即 可。命名一般以 Error 、 Exception 为后缀。 自定义异常由 raise 语句主动抛出。

### python当中常见的异常类型

异常类型 描述说明

ZeroDivisionError 当除数为0时，引发的异常

IndexError 索引超出范围所引发的异常

KeyError 字典所值时key不存在的异常

NameError 使用一个没有声明的变量时引发的异常

SyntaxError Python中的语法错误

ValueError 传入的值错误

AttributeError 属性或方法不存的异常

TypeError 类型不合适引发的异常

IndentationError 不正确的缩进引发的异常

Python中的异常都派生自 BaseException 类，本节我们测试和列出常见 的一些异常，方便初学者掌握。

1 SyntaxError ：语法错误

int a =3

^

SyntaxError: invalid syntax

2 NameError ：尝试访问一个没有申明的变量

print(a)

NameError: name 'a' is not defined

3 ZeroDivisionError ：除数为0错误（零除错误）

a = 3/0

ZeroDivisionError: division by zero

4 ValueError ：数值错误

float("gaoqi")

ValueError: could not convert string to

float: 'gaoqi'

5 TypeError ：类型错误

123+"abc"

TypeError: unsupported operand type(s) for

+: 'int' and 'str'

6 AttributeError ：访问对象的不存在的属性

a=100

a.sayhi()

AttributeError: 'int' object has no

attribute 'sayhi'

7 IndexError ：索引越界异常

a = [4,5,6]

a[10]

IndexError: list index out of range

8 KeyError ：字典的关键字不存在

a = {'name':"gaoqi",'age':18}

a['salary']

KeyError: 'salary'

#### 常见异常汇总

异常名称 说明

ArithmeticError 所有数值计算错误的基类

AssertionError 断言语句失败

AttributeError 对象没有这个属性

BaseException 所有异常的基类

DeprecationWarning 关于被弃用的特征的警告

EnvironmentError 操作系统错误的基类

EOFError 没有内建输入,到达EOF 标记

Exception 常规错误的基类

FloatingPointError 浮点计算错误

FutureWarning 关于构造将来语义会有改变的警告

GeneratorExit 生成器(generator)发生异常来通知退出

ImportError 导入模块/对象失败

IndentationError 缩进错误

IndexError 序列中没有此索引(index)

IOError 输入/输出操作失败

KeyboardInterrupt 用户中断执行(通常是输入^C)

KeyError 映射中没有这个键

LookupError 无效数据查询的基类

MemoryError 内存溢出错误(对于Python 解释器不是致命的)

NameError 未声明/初始化对象 (没有属性)

NotImplementedError 尚未实现的方法

OSError 操作系统错误

OverflowError 数值运算超出最大限制

OverflowWarning 旧的关于自动提升为长整型(long)的警告

PendingDeprecationWarning 关于特性将会被废弃的警告

ReferenceError 弱引用(Weak reference)试图访问已经垃圾回收了的对象

RuntimeError 一般的运行时错误

RuntimeWarning 可疑的运行时行为(runtime behavior)的警告

StandardError 所有的内建标准异常的基类

StopIteration 迭代器没有更多的值

SyntaxError Python 语法错误

SyntaxWarning 可疑的语法的警告

SystemError 一般的解释器系统错误

SystemExit 解释器请求退出

TabError Tab 和空格混用

TypeError 对类型无效的操作

UnboundLocalError 访问未初始化的本地变量

UnicodeDecodeError Unicode 解码时的错误

UnicodeEncodeError Unicode 编码时错误

UnicodeError Unicode 相关的错误

UnicodeTranslateError Unicode 转换时错误

UserWarning 用户代码生成的警告

ValueError 传入无效的参数

Warning 警告的基类

WindowsError 系统调用失败

ZeroDivisionError 除(或取模)零 (所有数据类型)

### with上下文管理

finally 块由于是否发生异常都会执行，通常我们放释放资源的代码。 其实，我们可以通过 with 上下文管理，更方便的实现释放资源的操作。

with 上下文管理的语法结构如下：

with context\_expr [ as var]：

语句块

with 上下文管理可以自动管理资源，在 with 代码块执行完毕后自动还原进入该代码之前的现场或上下文。不论何种原因跳出 with 块，不论是否有异常，总能保证资源正常释放。极大的简化了工作，在文件操作、网络通信相关的场合非常常用。

【示例】 with 上下文管理文件操作

with open("d:/bb.txt") as f:

for line in f:

print(line)

执行结果：

gaoqi

sxt

baizhan

### traceback模块和生成异常日志

【示例】使用 traceback 模块打印异常信息

import traceback

try:

print("step1")

num = 1/0

except:

traceback.print\_exc()

运行结果：

step1

Traceback (most recent call last):

File "...mypro\_exception/my01.py", line 7,

in <module>

num = 1/0

ZeroDivisionError: division by zero

Process finished with exit code 0

【示例】使用 traceback 将异常信息写入日志文件

#coding=utf-8

import traceback

try:

print("step1")

num = 1/0

except:

with open("d:/a.log","a") as f:

traceback.print\_exc(file=f)

### PyCharm的调式模式

1 进行调试的核心是设置断点。

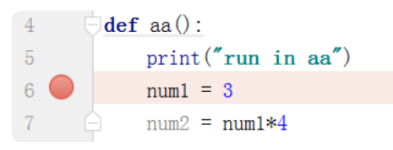
2程序执行到断点时，暂时挂起，停止执行。就像看视频按下停止一样，我们可以详细的观看停止处的每一个细节。

#### 断点

程序运行到此处，暂时挂起，停止执行。我们可以详细在此时观察 程序的运行情况，方便做出进一步的判断。

设置断点：

在行号后面单击即可增加断点。在断点上再单击即可取消断点。



#### 进入调试视图点击小虫子进行调试

我们通过如下三种方式都可以进入调试视图：

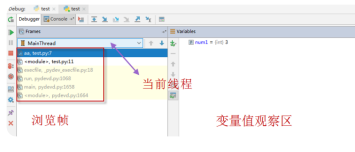
1 单击工具栏上的按钮



2 右键单击编辑区，点击： debug ‘模块名’3 快捷键： shift+F9



#### 进入调试视图后，布局如下：



1 左侧为“浏览帧”：

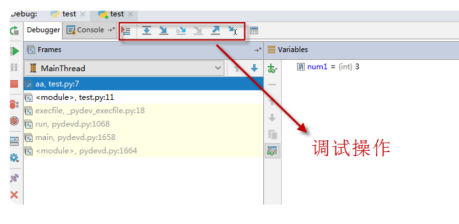
调试器列出断点处，当前线程正在运行的方法，每个方法对应一个“栈帧”。最上面的是当前断点所处的方法。

2 变量值观察区：

3 调试器列出了断点处所在方法相关的变量值。我们可以通过它，

查看变量的值的变化。

#### 调试操作区



我们通过上图中的按钮进行调试操作，它们的含义如下：



### 实例3

判断是不是三角形

try:  
 a=int(input('请输入边长：'))  
 b=int(input('请输入边长：'))  
 c=int(input('请输入边长：'))  
 if a==b==c:  
 print(f'你分别输入的{a},{b},{c}是等边三角形')  
 elif a+b>c and a+c>b and b+c>a:  
 print(f'你分别输入的{a},{b},{c}是三角形')  
 else:  
 raise Exception(f'你分别输入的{a},{b},{c}不是三角形')  
except Exception as a:  
 print(a)

## 函数的定义及调用

### 函数的定义及调用

函数是将一段实现功能的完整代码，使用函数名称进行封装，通过函数名称进行调用。以此达到一次编写，多次调用的目的

#### Python函数分为如下几类：

1 内置函数

我们前面使用的 str() 、 list() 、 len() 等这些都是内置函数，我们可以

拿来直接使用。

2 标准库函数

我们可以通过 import 语句导入库，然后使用其中定义的函数

3 第三方库函数

Python社区也提供了很多高质量的库。下载安装这些库后，也

是通过 import 语句导入，然后可以使用这些第三方库的函数

4 用户自定义函数 用户自己定义的函数，显然也是开发中适应用户自身需求定义的 函数。今天我们学习的就是如何自定义函数。

#### 自定义函数

def 函数名（参数列表）：

函数体

[return返回值列表]

def get\_sum(num):#定义一个函数(num是形式参数)  
 s=0  
 for i in range(1,num+1):#1到num的累加和  
 s+=i  
 print(f'1到{num}的累加和是：{s}')  
  
#调用函数  
get\_sum(10)#10是实际参数  
get\_sum(100)  
get\_sum(1000)

1到10的累加和是：55

1到100的累加和是：5050

1到1000的累加和是：500500

def add(a,b,c):

'''完成三个数的加法，并返回他们的和'''

sum = a+b+c

print("{0}、{1}、{2}三个数的和是：{3}".format(a,b,c,sum))

return sum

add(10,20,30)

add(30,40,50)

要点：

1 我们使用 def 来定义函数，然后就是一个空格和函数名称； Python执行 def 时，会创建一个函数对象，并绑定到函数名变量上。

2 参数列表 圆括号内是形式参数列表，有多个参数则使用逗号隔开 定义时的形式参数不需要声明类型，也不需要指定函数返回值类型 调用时的实际参数必须与形参列表一一对应 3 return 返回值 如果函数体中包含 return 语句，则结束函数执行并返回值； 如果函数体中不包含 return 语句，则返回 None 值。

4 调用函数之前，必须要先定义函数，即先调用 def 创建函数对象 内置函数对象会自动创建 标准库和第三方库函数，通过 import 导入模块时，会执行模块中的def语句

#### 形参和实参

形参和实参的要点：

圆括号内是形式参数列表，有多个参数则使用逗号隔开

定义时的形式参数不需要声明类型，也不需要指定函数返回值类型

调用时的实际参数必须与形参列表一一对应

【操作】定义一个函数，实现两个数的比较，并返回较大的值

def printMax(a,b):

'''实现两个数的比较，并返回较大的值'''

if a>b:

print(a,

'较大值')

return a

else:

print(b,

'较大值')

return b

printMax(10,20)

printMax(30,5)

执行结果：20 较大值

30 较大值

上面的 printMax 函数中，在定义时写的 printMax(a,b) 。 a 和 b 称为“形式参

数”，简称“形参”。也就是说，形式参数是在定义函数时使用的。 形

式参数的命名要符合“标识符”命名规则。

在调用函数时，传递的参数称为“实际参数”，简称“实参”。上面代码

中， printMax(10,20) ， 10 和 20 就是实际参数。

#### 函数的使用总结：

函数定义：

使用关键字def

确定函数名称、参数名称，参数个数，编写函数体（用于实现函数功能的代码

##### 函数调用

通过函数名称进行调用函数，对函数的个个参数进行实际的赋值

函数名（参数列表）

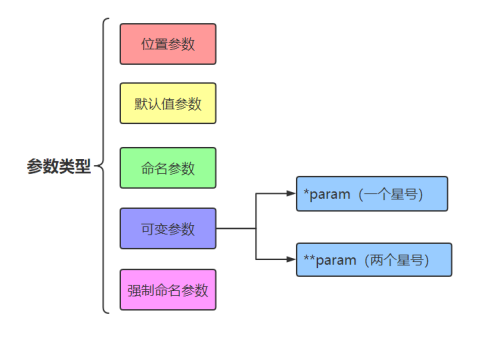
##### 函数执行

使用实际参数与函数功能的实现

##### 函数返回结果

函数执行结束后，如果使用teturn进行返回结果，则结果被返回到函数的调用处

### 参数的几种类型



#### 位置参数

是指调用时的参数个数和顺序必须与定义的参数个数和顺序相同

def birthdy(name,age):  
 print(f'祝{name}{str(age)}岁生日快乐')  
birthdy('胡',20)

祝胡20岁生日快乐

def f1(a,b,c):

print(a,b,c)

f1(2,3,4)

f1(2,3) #报错，位置参数不匹配

#### 默认值参数

是在函数定义时，直接对形式参数进行赋值，在调用时如果该参数不传值，将使用默认值，如果该参数传值，则使用传递的值

def birthdy(name='胡',age=20):  
 print(f'祝{name}{str(age)}岁生日快乐')  
birthdy()  
birthdy('李',21)

祝胡20岁生日快乐

祝李21岁生日快乐

def f1(a,b,c=10,d=20): #默认值参数必须位于普通位置参数后面

print(a,b,c,d)

f1(8,9)

f1(8,9,19)

f1(8,9,19,29)

结果：

8 9 10 20

8 9 19 20

8 9 19 29

#### 命名参数

我们也可以按照形参的名称传递参数，称为“命名参数”，也称“关键字参数”。

def f1(a,b,c):

print(a,b,c)

f1(8,9,19) #位置参数

f1(c=10,a=20,b=30) #命名参数

执行结果：

8 9 19

20 30 10

def birthdy(name,age):  
 print(f'祝{name}{str(age)}岁生日快乐')  
birthdy(age=20,name='胡')#通过关键字的赋可以改变形参的位置

#### 可变参数

可变参数指的是“可变数量的参数”。分两种情况：

1 \*param （一个星号），将多个参数收集到一个“元组”对象中。

2 \*\*param （两个星号），将多个参数收集到一个“字典”对象中。

又分为个数可变的位置参数和个数可变的关键字参数两种，其中个数可变的位置参数是在参数前加一个颗星（\*pare）,pare形式参数的名称，函数调用时可接收任意个数的实际参数，并放到一个元组中，个数可变的关键字参数是在参数前加两颗星（\*\*pare）,在函数调用时可接收任意多个“参数=值”形式的参数，并放到一个字典中。

def fun(\*pare):#位置参数  
 print(type(pare))  
 for i in pare:  
 print(i,end="\t")  
fun(10,20,30,40)  
print()  
fun([11,22,33,44])  
print()  
fun(\*[11,22,33,44])#列表前面加一个\*就会把列表解表操作

<class 'tuple'>

10 20 30 40

<class 'tuple'>

[11, 22, 33, 44]

<class 'tuple'>

11 22 33 44

def fun(\*\*pare):#关键字参数  
 print(type(fun))  
 for key,value in pare.items():  
 print(f'{key},,,{value}')  
fun(name='hu',age=18,hight=170)  
print("-"\*30)  
d={'name':'李','age':20,'hight':180}  
fun(\*\*d)#前面两个\*\*将字典解包输出

<class 'function'>

name,,,hu

age,,,18

hight,,,170

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

<class 'function'>

name,,,李

age,,,20

hight,,,180

**【操作】测试可变参数处理（元组、字典两种方式）**

def f1(a,b,\*c):

print(a,b,c)

f1(8,9,19,20)

def f2(a,b,\*\*c):

print(a,b,c)

f2(8,9,name='gaoqi',age=18)

def f3(a,b,\*c,\*\*d):

print(a,b,c,d)

f3(8,9,20,30,name='gaoqi',age=18)执行结果：

8 9 (19, 20)

8 9 {'name': 'gaoqi', 'age': 18}

8 9 (20, 30) {'name': 'gaoqi', 'age': 18}

#### 强制命名参数

在带星号的“可变参数”后面增加新的参数，必须在调用的时候“强制 命名参数”。

def f1(\*a,b,c):

print(a,b,c)

#f1(2,3,4) #会报错。由于a是可变参数，将2,3,4全部收集。造成b和c没有赋值。

f1(2,b=3,c=4)

执行结果：

(2,) 3 4

### 函数的返回值return

如果函数的运行结果需要在其它函数中使用，那么这个函数就应该被定义为带返回值的函数。

函数的运行结果使用return关键字进行返回

return可以出现在函数中的任意一个位置，用于结束函数

返回值可以是一个值，或多个值，如果返回的值是多个，结果是一个元组类型。

#单个返回值举例  
def abc(a,b):  
 c=a+b  
 return c  
aaa=abc(1,2)#返回函数的C  
bbb=abc(abc(1,2),3)  
print(aaa,bbb)

3 6

#多个返回值举例  
def abc(num):  
 a=0#接收奇数和  
 b=0#接收偶数和  
 abc=0#累加和  
 for i in range(1,num+1):  
 if i%2 != 0:#算奇数  
 a+=i  
 else:#偶数和  
 b+=i  
 abc+=i#总和  
 return a,b,abc  
print(abc(10))  
  
a,b,abc=abc(10)#元组解包  
print(f'{a}\t\t{b}\t\t{abc}')

(25, 30, 55)

25 30 55

return 返回值要点：

1 如果函数体中包含 return 语句，则结束函数执行并返回值

2 如果函数体中不包含 return 语句，则返回 None 值

3 要返回多个值，使用列表、元组、字典、集合将多个值“存起来”即可

### 变量的作用域

是指变量起作用的范围，根据范围作用的大小可分为局部变量和全局变量

#### 局部变量

1 在函数体中（包含形式参数）声明的变量。

2 局部变量的引用比全局变量快，优先考虑使用

3 如果局部变量和全局变量同名，则在函数内隐藏全局变量，只使用同名的局部变量

def sum(a,b):  
 c=a+b  
 return c#函数内的a,b,c就是局部变量  
abc=sum(10,20)  
print(abc)

#### 全局变量

1 在函数和类定义之外声明的变量。作用域为定义的模块，从定义位置开始直到模块结束。 2 全局变量降低了函数的通用性和可读性。应尽量避免全局变量的使用。

3 要在函数内改变全局变量的值，使用 global 声明一下

c=100#全局变量  
  
def sum(a,b):  
 return c+a+b#函数内的a,b,c就是局部变量  
  
abc=sum(10,20)  
print(c)  
print(abc)

100

130

全局变量可以在函数中使用。

c=100#全局变量  
def sum(a,b):  
 global c#全局变量的声明，只能单行声明  
 c=200#将c定义为全局变量  
 return c+a+b#函数内的a,b,c就是局部变量  
abc=sum(10,20)  
print(c)#函数外调用C仍然是200  
print(abc)

200

230

a = 100 #全局变量

def f1():

print(a) #打印全局变量a的值

global a #如果要在函数内改变全局变量的值，增加global关键字声明

a = 300

f1()

f1()

print(a)

结果：

100

300

300

a=100

def f1():

a = 3 #同名的局部变量

print(a)

f1()

print(a) #a仍然是100，没有变化

结果：

3

100

#### 局部变量和全局变量效率测试

局部变量的查询和访问速度比全局变量快，优先考虑使用，尤其是在循环的时候。

在特别强调效率的地方或者循环次数较多的地方，可以通过将全局 变量转为局部变量提高运行速度。

【操作】测试局部变量和全局变量效率

a = 1000  
def test01():  
 start = time.time()  
 global a  
 for i in range(100000000):  
 a += 1  
 end = time.time()  
 print("耗时{0}".format((end-start)))  
  
def test02():  
 c = 1000  
 start = time.time()  
 for i in range(100000000):  
 c += 1  
 end = time.time()  
 print("耗时{0}".format((end - start)))  
  
test01()  
test02()

耗时6.563061475753784 #全局变量的用时

耗时3.975112199783325 #局部变量的用时

#### 匿名的函数lambba的使用

lambda 表达式可以用来声明匿名函数。 lambda 函数是一种简单的、在

同一行中定义函数的方法。 lambda 函数实际生成了一个函数对象。

lambda 表达式只允许包含一个表达式，不能包含复杂语句，该表达式

的计算结果就是函数的返回值。

是指没有名字的函数，这种函数只能使用一次，一般是在函数的函数体只有一句代码且只有一个返回值时，可以使用匿名函数来简化。

语法结构：

result=lambda 参数列表：表达式

lambda arg1,arg2,arg3... : <表达式>

arg1 arg2 arg3 为函数的参数。<表达式>相当于函数体。运算结果是：表达式的运算结果。

s=lambda a,b:a+b  
print(s(10,20))

匿名函数的使用场景1

lis=[10,20,30,40]  
for i in range(len(lis)):  
 ss=lambda x:x[i]  
 print(ss(lis),end=" ")

10 20 30 40

匿名函数的使用场景2

#字典排序  
scores=[  
 {'name':'王天一','srore':90},  
 {'name':'李元霸','srore':100},  
 {'name':'陈美美','srore':95}  
]  
scores.sort(key=lambda x:x.get('srore'),reverse=True)#按srore值的降序排列  
print(scores)

[{'name': '李元霸', 'srore': 100}, {'name': '陈美美', 'srore': 95}, {'name': '王天一', 'srore': 90}]

g = [lambda a:a\*2,lambda b:b\*3,lambda c:c\*4]

print(g[0](6),g[1](7),g[2](8))

12 21 32

#### 递归函数recursion

1 递归(recursion)是一种常见的算法思路，在很多算法中都会用到。比如：深度优先搜索（DFS:Depth First Search）等。

2 递归的基本思想就是“自己调用自己”

在一个函数的函数体内调用该函数本身，该函数就是递归函数

一个完整的递归操作由两部分组成，一部分是递归调用，一部分是递归终止条件，一般可使用if-else结构来判断递归的调用和递归的终止。

5!=5\*4\*3\*2\*1 = 5!=5\*4!

4!=4\*3\*2\*1 = 4!=4\*3!

3!= 3\*2\*1 = 3!=3\*2! >>> n\*afc(n-1)#计算阶乘的公式

2!= 2\*1 = 2!=2\*1!

1!=1 = 1!=1

def fac(n):  
 if n==1:  
 return 1  
 else:  
 return n\*fac(n-1)  
print(fac(5))

120

def my\_recursion(n):  
 print("start:" + str(n))  
 if n == 1:  
 print("recursion over!")  
 else:  
 my\_recursion(n - 1)  
 print("end:" + str(n))  
my\_recursion(3)

start:3

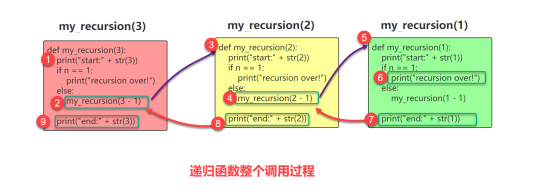
start:2

start:1

recursion over!

end:2

end:3



#### 嵌套函数(内部函数)

嵌套函数：在函数内部定义的函数！

def outer():

print('outer running...')

def inner():

print('inner running...')

inner()

outer()

outer running...

inner running...一般在什么情况下使用嵌套函数？

1 封装 - 数据隐藏

外部无法访问“嵌套函数”。

2 贯彻 DRY(Don’t Repeat Yourself) 原则

3 嵌套函数，可以让我们在函数内部避免重复代码。

4 闭包

【操作】使用嵌套函数避免重复代码

def printChineseName(name,familyName):

print("{0} {1}".format(familyName,name))

def printEnglishName(name,familyName):

print("{0} {1}".format(name, familyName))

使用1个函数代替上面的两个函数

def printName(isChinese,name,familyName):

def inner\_print(a,b):

print("{0} {1}".format(a,b))

if isChinese:

inner\_print(familyName,name)

else:

inner\_print(name,familyName)

printName(True,"小七","高")

printName(False,"George","Bush")

#### nonlocal关键字

nonlocal 用来在内部函数中，声明外层的局部变量。global 函数内声明全局变量，然后才使用全局变量【操作】使用nonlocal声明外层局部变量

#测试nonlocal、global关键字的用法

a = 100  
def outer():  
 b = 10  
 def inner():  
 nonlocal b #声明外部函数的局部变量  
 print("inner b:",b)  
 b = 20  
 global a #声明全局变量  
 a = 1000  
 inner()  
 print("outer b:",b)  
outer()  
print("a：",a)

inner b: 10

outer b: 20

a： 1000

#### 斐波那契数列

又称黄金分割线，指的是这样一个数列：1、1、2、3、5、8、13、21、34、。。。从第三项开始，每项都等前两项之各

公式为：f(n)=f(n-1)+f(n-2)

#斐波那契数列计算  
def fac(n):  
 if n==1 or n==2:  
 return 1  
 else:  
 return fac(n-1)+fac(n-2)  
  
for i in range(1,10):  
 print(fac(i),end="\t")  
print()

1 1 2 3 5 8 13 21 34

#### LEGB规则

Python在查找“名称”时，是按照LEGB规则查找的：



Local 指的就是函数或者类的方法内部

Enclosed 指的是嵌套函数（一个函数包裹另一个函数，闭包）

Global 指的是模块中的全局变量

Built in 指的是Python为自己保留的特殊名称

如果某个 name 映射在局部 local 命名空间中没有找到，接下来就会在闭包作用域 enclosed 进行搜索，如果闭包作用域也没有找到，Python就会到全局 global 命名空间中进行查找，最后会在内建built-in 命名空间搜索 （如果一个名称在所有命名空间中都没有找到，就会产生一个 NameError ）

### 常用的内置函数

#### 数据类型转换函数

函数名称 描述说明

bool(boj) 获取指定对象obj的布尔值

str(obj) 将指定对像obj转成字符串类型

int(x) 将x转成int类型

float(x) 将x转成float类型

list(sepuence) 将序列转成列表类型

tuple(sepuence) 将序列转成元组类型

set(sepuence) 将序列转成集合类型

#### 数学函数

函数名称 描述说明

abs(x) 获取x的绝对值

divmod(x,y) 获取x与y的商和余数

min(sequence) 获取sepuence的最小值

sum(iter) 对可迭代对象进行求和运算

pow(x,y) 获取x的y次幂

round(x,d) 对x进行保留d位小数，结果四舍五入

# 获取C列数据范围（假设数据从第2行开始，跳过表头）  
c\_column = ws['C'][1:] # 索引1表示从第2行开始  
# 计算C列数据总和  
total = sum(cell.value for cell in c\_column if cell.value is not None)

#### 迭代器操作函数

函数名称 描述说明

sorted(ter) 对可迭代对象进行排序

reversed(sequence) 反转序列生成新的迭代器对象

zip(iter1,iter2) 将iter1与iter2打包成元组并返回一个可迭代的zip对象

enumerate(iter) 根据iter对象创建一个enumerate对象

all(iter) 判断可迭代对象iter中所有元素的布尔值是否都是为True

any(iter) 判断可迭代对象iter中所有元素的布尔值是否都是为False

next(iter) 获取迭代器的下一个元素

filter(function,iter)通过指定条件过滤序列并返回一个迭代器对象

map(function,iter)通过函数function对可迭代对象iter的操作返回一个迭代器对象

s=[214,99,88,220]  
desc=sorted(s,reverse=True)  
print(desc)

[220, 214, 99, 88]

s=[214,99,88,220]  
fanzhuan=reversed(s)#生成一个反转的迭代器对象  
print(type(fanzhuan))  
print(list(fanzhuan))#转换成list类型，将列表反转

<class 'list\_reverseiterator'>

[220, 88, 99, 214]

s=[214,99,88,220]  
x=['a','b','c','d']  
zipobj=zip(x,s)#打包成可迭代的zip对象  
print(type(zipobj))#对像  
print(list(zipobj))#将对象转成list类型

<class 'zip'>

[('a', 214), ('b', 99), ('c', 88), ('d', 220)]

x=['a','b','c','d']  
enum=enumerate(x,start=1)#元素前面加了索引  
print(type(enum))#生成的是一个对象  
print(tuple(enum))#转成元组类型输出

<class 'enumerate'>

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

s=[214,99,0,220]  
x=['a','b','c','d']  
print(all(s))#s中有0为false  
print(all(x))

False

True

x=['a','b','c','d']  
s=[214,99,0,220]  
zipobj=zip(x,s)  
# lis=tuple(zipobj)#如果转化了类型，next就不能用  
# print(lis)  
print(next(zipobj))#遍历的是对象  
print(next(zipobj))

('a', 214)

('b', 99)

def fun(num):  
 return num%2==1#如果是奇数返回Ture  
obj=filter(fun,range(10))#过虑fun函数将结果为True的留下来，生成对象  
print(list(obj))#将对象转成列表输出

[1, 3, 5, 7, 9]

def upper(x):  
 return x.upper()#将x全部转换成大写  
lis=['hello','world','gentel']  
mapp=map(upper,lis)#把lis的数值给了upper，执行后生成对象  
print(list(mapp))

['HELLO', 'WORLD', 'GENTEL']

#### 其他函数

format(value.format\_spec) 将value以format\_spec格式进行显示

数值型右对齐，字符串左对齐

les(s) 获取s的长度或s元素的个数

id(obj) 获取对象的内存地址

type(x) 获取x的数据类型

eval(s) 执s这个字符串所表示的python代码，去掉引号参与运算

print(format(3.14,"20"))  
print(format("hello","20"))  
print(format(3.14,"\*<20"))#\*代表填充符，小于20代表左对齐，20代表字符的宽度  
print(format(3.14,"\*>10"))#\*代表填充符，大于代表右对齐，10代表字符的宽度  
print(format(3.14,"\*^6"))#\*代表填充符，^代表居中对齐，6代表字符的宽度  
print(format(3.1415926,".2f"))#.2f代表保留两位小数  
print(format(200,'b'))#b代表2进制  
print(format(200,'o'))#b代表8进制  
print(format(200,'x'))#x代表16进制  
print(format(200,'X'))#X代表16进制

3.14

hello

3.14\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*3.14

\*3.14\*

3.14

11001000

310

c8

C8

### 实战

**随机生成一个10位数的列表，不用max内置函数选出最大**

import random  
def get\_max(lis):  
 s=lis[0]  
 for i in range(len(lis)):  
 if lis[i]>s:  
 s=lis[i]  
 return s  
lst=[random.randint(1,100) for i in range(10)]  
print(lst)  
max=get\_max(lst)  
print(max)

[86, 98, 49, 60, 82, 1, 21, 94, 71, 54]

98

**输入入一个字符串将中间的数字变字列表，并加总数字**

def get\_ls(x):  
 s=0  
 lis=[]  
 for item in x:  
 if item.isdigit():  
 lis.append(int(item))  
 s=sum(lis)  
 return lis,s  
i=input("请输入内容")  
lis,s=get\_ls(i)  
print(lis)  
print(s)

请输入内容hell123world456

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

21

**#编写函数实现将字符串中字母的大小写转换**

**#编写函数实现将字符串中字母的大小写转换**  
def convert(x):  
 lis=[]  
 for item in x:  
 if 'A'<=item<='Z':  
 lis.append(chr(ord(item)+32))#小 写字母比大写字母在unicode大32

#ord将字母转成unicode码，chr小Unicode码转成字符  
 elif 'a'<=item<='z':  
 lis.append(chr(ord(item)-32))#小写转大写  
 else:  
 lis.append(item)#非大小写直接 添加  
 return ''.join(lis)#在lis中的每个元素的后面都增加一个新的字符串str,字了符串的拼接  
#调用函数  
i=input("请输入内容：")  
print('已将大小写格式转换',convert(i))

请输入内容：hello123WORLD

已将大小写格式转换 HELLO123world

**#实现in的功能，如果x在lis中就为真，否就是为假**

#实现in的功能，如果x在lis中就为真，否就是为假  
def chazhao(x,lis):  
 for item in lis:  
 if item==x:  
 return True  
 return False  
#调用  
i=input('请输入查找内容：')  
k=['a','b','c','d']  
s=chazhao(i,k)  
print('存在'if s else '不存在')#三元表达法

请输入查找内容：b

存在

## 面向对象程序设计

### 面向过程和和面向对象两大编程思想

面向过程：功能上的封装，python中写的函数就是功能上的封装。典型代表c语言

面向对象：属性和行为上的封装，典型代表python,java

两者的异同点：

面向过程 面向对象

异：事物比较简单，可以用线性思维去解决。 事物比较复杂，使用简单的线性思维无解决

同：面向过程和面向对象都是解决实际问题的一种思维方式

两者相辅想成，并不是对立的。

解决复杂问题，通过面向对象方式便于我们从宏观上把握事物之间复杂的关系，方便我们分析整个系统，具体到微观操作，仍然使用面向过程方式来处理

### 类的定义

由n多个对象抽取出“像”的属性和行为从而归纳总结出来的一种类别

类相当于一个图纸。如汽车是一个图级，里面的轿车，SUV就是具体的对象

类：我们叫做 class 。 对象：我们叫做 object , instance (实例)。以后 我们说某个类的对象，某个类的实例。是一样的意思。





自定义数据类型的语法结构

class 类名（）：#（）这个可以不写

类体

要点如下：

1 类名必须符合“标识符”的规则；一般规定，首字母大写，多个单词使用“驼峰原则”。 2 类体中我们可以定义属性和方法 3 属性用来描述数据，方法(即函数)用来描述这些数据相关的操作

【操作】一个典型的类的定义

class Student:  
 def \_\_init\_\_(self,name,score): #构造方法第一个参数必须为self  
 self.name = name #实例属性  
 self.score = score  
 def say\_score(self): #实例方法  
 print("{0}的分数是{1}".format(self.name,self.score))  
s1 = Student('张三',80) #s1是实例对象，自动调用\_\_init\_\_()方法  
s1.say\_score()

张三的分数是80

pass 为空语句。就是表示什么都不做，只是作为一个占位符存 在。当你写代码时，遇到暂时不知道往方法或者类中加入什么 时，可以先用pass占位，后期再补上。

\_\_init\_\_ 构造方法和 \_\_new\_\_ 方法

初始化对象，我们需要定义构造函数 \_\_init\_\_() 方法。构造方法用于执 行“实例对象的初始化工作”，即对象创建后，初始化当前对象的相关属性，无返回值。



构造方法是负责初始化（装修），不是建对象 (房子)

\_\_init\_\_() 的要点如下：

1 名称固定，必须为： \_\_init\_\_()

2 第一个参数固定，必须为： self 。 self 指的就是刚刚创建好的实例对象

3 构造函数通常用来初始化实例对象的实例属性，如下代码就是初始化实例属性： name 和 score

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

self.name = name #实例属性

self.score = score

4 通过“类名(参数列表)”来调用构造函数。调用后，将创建好的对 象返回给相应的变量。 比如： s1 = Student('张三', 80)

5 \_\_init\_\_() 方法：初始化创建好的对象，初始化指的是：“给实例属性赋值”

6 \_\_new\_\_() 方法: 用于创建对象，但我们一般无需重定义该方法

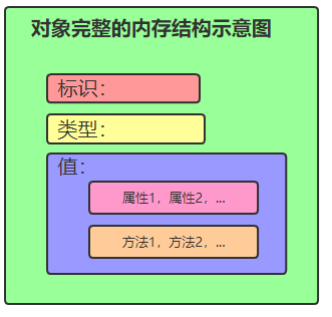
7 如果我们不定义 \_\_init\_\_ 方法，系统会提供一个默认的 \_\_init\_\_ 方法。 如果我们定义了带参的 \_\_init\_\_ 方法，系统不创建默认的 \_\_init\_\_ 方法

### 类对象、类属性、类方法、静态方法

#### 类对象

对象完整内存结构

类是抽象的，也称之为“对象的模板”。我们需要通过类这个模板， 创建类的实例对象，然后才能使用类定义的功能。 我们前面说过一个Python对象包含三个部分： id （identity识别 码）、 type （对象类型）、 value （对象的值）。 现在，我们可以更进一步的说，一个Python对象包含如下部分：



对象是类的具体实体，一般称为“类的实例”。类看做“饼干模具”，对象就是根据这个“模具”制造出的“饼干”。 从一个类创建对象时，每个对象会共享这个类的行为（类中定义的 方法），但会有自己的属性值（不共享状态）。更具体一点：“方法 代码是共享的，属性数据不共享”。

对象名=类名

class cat:#创建了一个类  
 pass  
c=cat()#c是对象属于cat类  
print(type(c))#显示c的数据类型

<class '\_\_main\_\_.cat'>

class Student:

pass #空语句

print(type(Student)) #显示的是类<class 'type'>

print(id(Student)) #51686328

Stu2 = Student

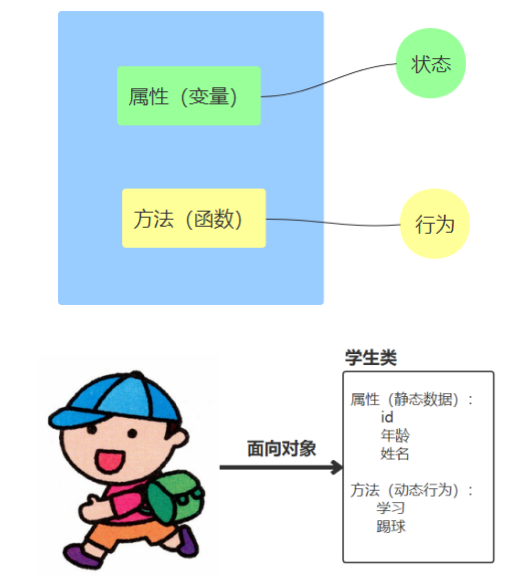
s1 = Stu2()

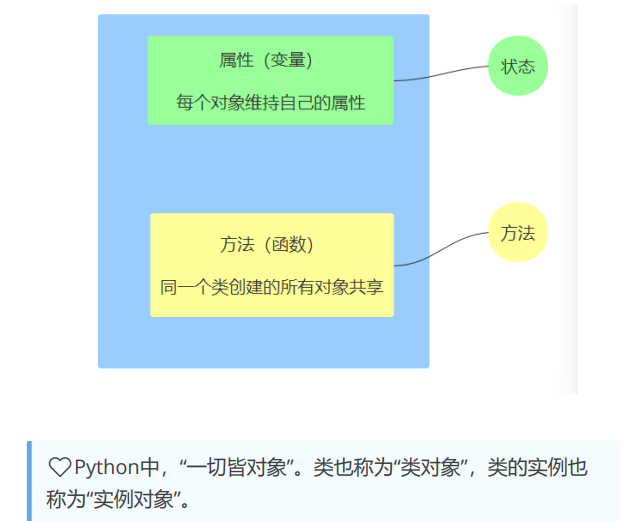
print(s1) #<\_\_main\_\_.Student object at 0x0000000002B5FDD8>

我们可以看到实际上生成了一个变量名就是类名 Student 的对象。我们通过赋值给新变量 Stu2 ，也能实现相关的调用。说明，确实创建 了“类对象”。

#### 类的组成及属性和方法的调用

我们通过类定义数据类型的属性（数据）和方法（行为）,也就是说，“类将行为和状态打包在一起”。





类属性，直接定义在类中，方法外的变量

实例属性：定义在\_init\_方法中，使用self打点的变量

实例方法：定义在类中的函数，而且自带参数self

静态方法：使用装饰器@staticmethod修饰的方法

类方法：使用装饰器@classmethod修饰的方法

#### 类属性

类属性是从属于“类对象”的属性，也称为“类变量”。由于，类属性从属于类对象，可以被所有实例对象共享。

类属性的定义方式：class 类名：

类变量名= 初始值

在类中或者类的外面，我们可以通过： 类名.类变量名 来读写

内存分析实例对象和类对象创建过程（重要）

我们以下面代码为例，分析整个创建过程，让大家对面向对象概念

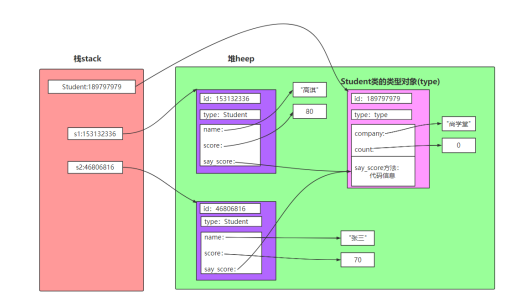
掌握更加深刻：

class Student:  
 company = "尚学堂" # 类属性  
 count = 0 # 类属性  
 def \_\_init\_\_(self, name, score):  
 self.name = name # 实例属性  
 self.score = score  
 Student.count = Student.count + 1  
 def say\_score(self): # 实例方法  
 print("我的公司是：", Student.company)   
 print(self.name, '的分数是：',self.score)   
s1 = Student('高淇', 80) # s1是实例对象，自动调用\_\_init\_\_()方法  
s2 = Student('张三', 70)  
s1.say\_score()  
print('一共创建{0}个Student对象'.format(Student.count))

我的公司是： 尚学堂

高淇 的分数是： 80

一共创建2个Student对象



实例属性是从属于实例对象的属性，也称为“实例变量”。他的使用 有如下几个要点：

1 实例属性一般在 \_\_init\_\_() 方法中通过如下代码定义： self.实例属性名 = 初始值

2 在本类的其他实例方法中，也是通过 self 进行访问： self.实例属性名

3 创建实例对象后，通过实例对象访问：

obj01 = 类名() #创建和初始化对象，调用 \_\_init\_\_() 初始化属性

obj01.实例属性名 = 值 #可以给已有属性赋值，也可以新加属性

举例：

class Student:  
 def \_\_init\_\_(self,name,score):  
 self.name = name #增加name属性  
 self.score = score #增加score属性  
 def say\_score(self):  
 self.age = 18 #增加age属性  
 print("{0}的分数是{1}".format(self.name,self.score))  
s1 = Student("张三",80)  
s1.say\_score()  
print(s1.age)  
s1.salary = 3000 #s1对象增加salary属性  
s2 = Student("李四",90)  
s2.say\_score()  
print(s2.age)

张三的分数是80

18

李四的分数是90

18

#### 类方法

实例方法是从属于实例对象的方法。实例方法的定义格式如下：

def  方法名(self [, 形参列表])：

函数体

要点：

1 定义实例方法时，第一个参数必须为 self 。和前面一样， self 指当前的实例对象。

2 调用实例方法时，不需要也不能给 self 传参。 self 由解释器自动传参

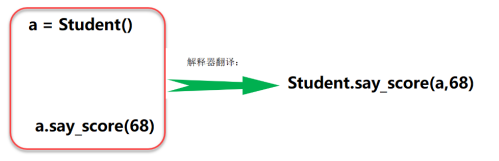
函数和方法的区

1 都是用来完成一个功能的语句块，本质一样。

2 方法调用时，通过对象来调用。方法从属于特定实例对象，普通函数没有这个特点

3 直观上看，方法定义时需要传递self，函数不需要

实例对象的方法调用本质



【操作】类方法使用测试

class Student:

company = "SXT" #类属性

@classmethod

def printCompany(cls):

print(cls.company)

Student.printCompany()

#### 静态方法

静态方法通过装饰器@staticmethod来定义，格式如下：

@staticmethod

def 静态方法名([形参列表]) ：

方法体

要点如下：

1 @staticmethod 必须位于方法上面一行

2 调用静态方法格式： 类名.静态方法名(参数列表)

3 静态方法中访问实例属性和实例方法会导致错误

【操作】静态方法使用测试

class Student:  
 company = "SXT" # 类属性  
 @staticmethod  
 def sum(a, b): # 静态方法  
 print("{0}+{1}={2}".format(a,b,(a+b)))  
 return a+b  
Student.sum(20,30)

Python中允许定义与“类对象”无关的方法，称为“静态方法”。 “静态方法”和在模块中定义普通函数没有区别，只不过“静态方法”放 到了“类的名字空间里面”，需要通过“类调用”。

#### \_\_del\_\_方法 (析构函数)和垃圾回收机制

\_\_del\_\_() 称为“析构方法”，用于实现对象被销毁时所需的操作。比如：释放对象占用的资源，例如：打开的文件资源、网络连接等。

Python实现自动的垃圾回收，当对象没有被引用时（引用计数为 0），由垃圾回收器调用 \_\_del\_\_() 。

我们也可以通过 del语句 删除对象，从而保证调用 \_\_del\_\_() 。

系统会自动提供 \_\_del\_\_方法 ，一般不需要自定义析构方法。

#析构函数

class Person:

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

print("销毁对象：{0}".format(self))

p1 = Person()

p2 = Person()

del p2

print("程序结束")

运算结果：

销毁对象：<\_\_main\_\_.Person object at 0x02175610>

程序结束

销毁对象：<\_\_main\_\_.Person object at 0x021755D0>

\_\_call\_\_ 方法和可调用对象

1 Python 中，凡是可以将 () 直接应用到自身并执行，都称为可调 用对象。

2 可调用对象包括自定义的函数、Python 内置函数、以及本节所 讲的实例对象。

3 定义了 \_\_call\_\_() 的对象，称为“可调用对象”，即该对象可以像函数 一样被调用。 4 该方法使得实例对象可以像调用普通函数那样，以“对象名()”的 形式使用。

def f1():  
 print("f1")  
f1() #本质也是调用了\_\_call\_\_()方法  
class Car:  
 def \_\_call\_\_(self, age,money):  
 print("\_\_call\_\_方法")  
 print("车龄：{0},金额：{1}".format(age,money))  
f2 = Car()  
f2(3,200000) #像调用函数那样调用，本质也是调用了\_\_call\_\_()

f1

\_\_call\_\_方法

车龄：3,金额：200000

#### 方法没有重载

如果我们在类体中定义了多个重名的方法，只有最后一个方法有效。

建议：不要使用重名的方法！Python中方法没有重载。 #Python中没有方法的重载。定义多个同名方法，只有最后

一个有效

class Person:  
 def say\_hi(self):  
 print("hello")  
 def say\_hi(self,name):  
 print("{0},hello".format(name))  
p1 = Person()  
# p1.say\_hi() #不带参，报错：TypeError:say\_hi() missing 1 required positional argument: 'name'  
p1.say\_hi("高淇")

高淇,hello

#### 方法的动态性

Python是动态语言，我们可以动态的为类添加新的方法，或者动态 的修改类的已有的方法

#测试方法的动态性  
class Person:  
 def work(self):  
 print("努力上班！")  
def play\_game(self):  
 print("玩游戏")  
  
def work2(s):  
 print("好好工作，努力上班！")  
  
Person.play = play\_game  
Person.work = work2  
p = Person()  
p.play()  
p.work()

玩游戏

好好工作，努力上班！

我们可以看到， Person 动态的新增了 play\_game 方法，以及用 work2 替换 了 work 方法

#### @property 装饰器

@property 可以将一个方法的调用方式变成“属性调用”。

@property 主要用于帮助我们处理属性的读操作、写操作。对于某一个属性，我们可以直接通过：

emp1.salary = 30000

如上的操作读操作、写操作。但是，这种做法不安全。比如，我需要限制薪水必须为 1-10000 的数字。这时候，我们就需要通过使用装饰器 @property 来处理。

#测试@property  
class Employee:  
 def \_\_init\_\_(self,name,salary):  
 self.name = name  
 self.\_\_salary = salary  
 @property #相当于salary属性的getter方法  
 def salary(self):  
 print("月薪为{0},年薪为{1}".format(self.\_\_salary,(12\*self.\_\_salary)))  
 return self.\_\_salary;  
 @salary.setter  
 def salary(self,salary): #相当于salary属性的setter方法  
 if(0<salary<1000000):  
 self.\_\_salary = salary  
 else:  
 print("薪水录入错误！只能在0-1000000之间")  
emp1 = Employee("高淇",100)  
print(emp1.salary)  
emp1.salary = -200

月薪为100,年薪为1200

100

薪水录入错误！只能在0-1000000之间

#### 属性和方法命名总结

1 \_xxx ：保护成员，不能用 from module import \* 导入，只有类对象和子类对象能访问这些成员。（⚠️讲完模块再看）

2 \_\_xxx\_\_ ：系统定义的特殊成员

\_\_xxx ： 类中的私有成员，只有类对象自己能访问，子类对象也不能访问。（但，在类外部可以通过 对象名. \_类名\_\_xxx 这种特殊方式访问。Python不存在严格意义的私有成员）

#### 类编码风格

1 类名首字母大写，多个单词之间采用驼峰原则。

2 实例名、模块名采用小写，多个单词之间采用下划线隔开

3 每个类，应紧跟“文档字符串”，说明这个类的作用

4可以用空行组织代码，但不能滥用。在类中，使用一个空行隔开方法；模块中，使用两个空行隔开多个类

### 使用类模板创建N多个对象并调用

from enum import show\_flag\_values  
class student:#创建了一个类  
 school='北京大学'#这就是一个变量，类属性  
 #实例属性  
 def \_\_init\_\_(self,xm,age):#nm,age方法的参数，局部变量作用域只能在\_\_init\_\_内  
 self.name=xm#等号的左侧是实例改属性，xm是局部变量,将局部变量的xm值赋值给了实例属性self.name  
 self.age=age#实例属性的变量名称和局部变量的名称可以相同  
 # 定义在类当中的函数就是实例方法  
 def show(self):  
 print(f'你的名字叫：{self.name}\t你的年龄：{self.age}')#self打点的实例属性可以在整个类中使用  
#创建对象  
stu1=student('gentel',40)  
stu2=student('tom',28)  
stu3=student('jimi',30)  
#给类属性赋值  
student.school='音乐学院'  
lis=[stu1,stu2,stu3]#列表当中的元素是student类型的对象  
for item in lis:#item是列表当中的元素是student类型的对象  
 item.show()#用对象名打点调用实例方法

你的名字叫：gentel 你的年龄：40

你的名字叫：tom 你的年龄：28

你的名字叫：jimi 你的年龄：30

### 动态绑定属性和方法

from enum import show\_flag\_values  
class student:#创建了一个类  
 school='北京大学'#这就是一个变量，类属性  
 #实例属性  
 def \_\_init\_\_(self,xm,age):#nm,age方法的参数，局部变量作用域只能在\_\_init\_\_内  
 self.name=xm#等号的左侧是实例改属性，xm是局部变量,将局部变量的xm值赋值给了实例属性self.name  
 self.age=age#实例属性的变量名称和局部变量的名称可以相同  
 # 定义在类当中的函数就是实例方法  
 def show(self):  
 print(f'你的名字叫：{self.name}\t你的年龄：{self.age}')#self打点的实例属性可以在整个类中使用  
  
stu1=student('gent',18)  
print(stu1.name,stu1.age)  
stu1.show()  
  
stu2=student('tom',21)  
stu2.gender='男'#动态绑定属性  
print(stu2.name,stu2.age,stu2.gender)  
  
# 动态绑定实例方法  
def show2():  
 print('这是一个普通的函数，但是可以被动态定义成stu2对象的实例方法')  
stu2.s=show2  
stu2.s()

gent 18

你的名字叫：gent 你的年龄：18

tom 21 男

这是一个普通的函数，但是可以被动态定义成stu2对象的实例方法

### 面向对象的三大特征

Python对于类的成员没有严格的访问控制限制，这与其他面向对象 语言有区别。关于私有属性和私有方法，有如下要点：

1 通常我们约定，两个下划线开头的属性是私有的(private)。其他为公共的(public)。

2 类内部可以访问私有属性(方法)

3 类外部不能直接访问私有属性(方法)

4 类外部可以通过 \_类名\_\_私有属性(方法)名 ”访问私有属性(方法)

【测试】私有属性和公有属性使用测试

#测试私有属性、私有方法  
class Employee:  
 \_\_company = "百战程序员" #私有.通过dir查到\_Employee\_\_company  
 def \_\_init\_\_(self,name,age):  
 self.name = name  
 self.\_\_age = age # 私有实例属性  
  
 def say\_company(self):  
 print("我的公司是：",Employee.\_\_company) #类内部可以直接访问私有属性  
 print(self.name,"的年龄是：",self.\_\_age)  
 self.\_\_work()  
 def \_\_work(self): #私有实例方法，通过dir可查到\_Employee\_\_work  
 print("工作！好好工作，好好赚钱，娶个媳妇！")  
p1 = Employee("高淇",32)  
print(p1.name)  
print(dir(p1)) #  
p1.say\_company()  
print(p1.\_Employee\_\_age) #通过这种方式可以直接访问到私有属性 。通过dir可以查到属性：\_Employee\_\_age  
#print(p1.\_\_age) #直接访问私有属性，报错  
#p1.\_\_sleep() #直接访问私有方法，报错高淇

高淇

['\_Employee\_\_age', '\_Employee\_\_company', '\_Employee\_\_work', '\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_firstlineno\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_static\_attributes\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'name', 'say\_company']

我的公司是： 百战程序员

高淇 的年龄是： 32

工作！好好工作，好好赚钱，娶个媳妇！

32

#### 封装

隐藏内部细节，对外提供操作方式。

隐藏对象的属性和实现细节，只对外提供必要的方法。相当于 将“细节封装起来”，只对外暴露“相关调用方法”。 通过前面学习的“私有属性、私有方法”的方式，实现“封装”。 Python追求简洁的语法，没有严格的语法级别的“访问控制符”，更多的是依靠程序员自觉实现。

##### 权限控制

是通过对属性或方法添加单下划线、双下划线以及首尾双下划线来实现

**单下划线开头：**

以单下划线开头的属性或方法表示protected受保护的成员，这类成员被视为仅供内部使用，允许类本身和子类进行访问，但实际上它可以被外部代码访问。

双

**双下划线开头:**

表示private私有的成员，这类成员只允许定义该属性或方法的类本身进行访问

**首尾双下划线：**一般表示特殊的方法

class student():  
 def \_\_init\_\_(self,name,age,gender):#创建实例属性  
 self.\_name=name#受保护的，只本类和子类访问  
 self.\_\_age=age#表示私用，只给类本身访问  
 self.gender=gender#普通的属性，类的内部，子类，类的外部都可以访问  
 def \_fun1(selfs):#实义的一个实例方法，有一个下划线，本类和子类可以使用  
 print('本类和子类可以使用')  
  
 def \_\_fun2(self):  
 print('私有的只给类本身访问')  
  
 def fun3(self):#变通的实例方法  
 self.\_fun1()#在类本身可以去使用受保护的方法  
 self.\_\_fun2()#类本身可以去访问私有方法  
 print(self.\_name)#访问受保护的属性  
 print(self.\_\_age)#访问私有的属性  
  
stu=student('gentel',40,'man')  
#类的外部调用类的属性  
print(stu.\_name)#访问爱保护的可以成功  
#print(stu.\_\_age)#在类外部访问私有的属性报错  
  
#类的外部调用实例方法  
stu.\_fun1()#访问受保护的高用成功  
stu.fun3()#访问普通的全部显示  
#stu.\_\_fun2()#访问私有的报错  
stu.\_student\_\_fun2()#通过这种方法在外部就可以访问私有的了  
print(stu.\_student\_\_age)#通过这种方法在外部就可以访问私有类属性  
#为什么要这样表达看下面  
print(dir(stu))#将对像当中所有的属性和方法，私有的age,就是\_student\_\_age,fun2也是一样

gentel

本类和子类可以使用

本类和子类可以使用

私有的只给类本身访问

gentel

40

私有的只给类本身访问

40

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_firstlineno\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_static\_attributes\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', '\_fun1', '\_name', '\_student\_\_age', '\_student\_\_fun2', 'fun3', 'gender']

##### 属性的设置

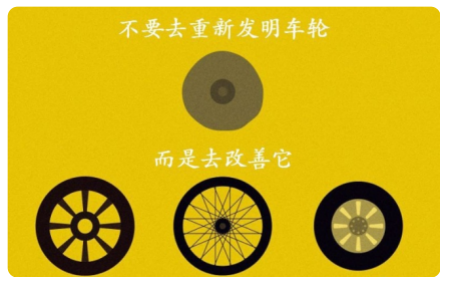
class student:  
 def \_\_init\_\_(self,name,gender):  
 self.name=name#普通属性  
 self.\_\_gender=gender#私有属性  
 #用装饰器将方法转成属性@property,让他去修饰方法  
 @property  
 def gender(self):#通过调用他把属性数值转给私有属性  
 return self.\_\_gender#把属性当方法来使用，只能查看值不能修改值  
 #将gender设置成可写属性  
 @gender.setter#设置一个可写入方法  
 def gender(self,value):  
 if value!='men' and value!='wumen':  
 print('性别有误，已将性别默认设置为women')  
 self.\_\_gender='women'  
 else:  
 self.\_\_gender=value  
  
stu=student('gentel','men')  
# print(stu.gender)  
print(stu.name,'性别是私有的无法访问但是可以访问方法',stu.gender)  
#修改gender属性的值  
stu.gender='other'  
print(stu.gender)

gentel 性别是私有的无法访问但是可以访问方法 men

性别有误，已将性别默认设置为women

women

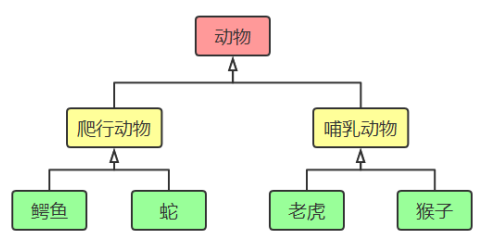
#### 继承



继承可以让子类具有父类的特性，提高了代码的重用性。

从设计上是一种增量进化，原有父类设计不变的情况下，可以 增加新的功能，或者改进已有的算法。

如果一个新类继承自一个设计好的类，就直接具备了已有类的特 征，就大大降低了工作难度。已有的类，我们称为“父类或者基 类”，新的类，我们称为“子类或者派生类”。



是在函数调用时，使用“形参名称=值”的方式进行传参，传递参数顺序可以与定义时参数的顺序不同。

**继承的语法结构：**

class 类名（父类1，父类2....父类n）

pass

在python中一个子类可以继承n多个父类；一个父类也可以拥有n多个子类；如果一个类没有继承任何类，那么这个类默认继承的是object类

关于构造函数：

1 子类不重写 \_\_init\_\_ ，实例化子类时，会自动调用父类定义的 \_\_init\_\_ 。

2 子类重写了 \_\_init\_\_ 时，实例化子类，就不会调用父类已经定义的 \_\_init\_\_

3如果重写了 \_\_init\_\_ 时，要使用父类的构造方法，可以使用 super 关键字，也可以使用如下格式

调用： 父类名.\_\_init\_\_(self, 参数列表)

class Person:  
 def \_\_init\_\_(self,name,age):  
 print("Person的构造方法")  
 self.name = name  
 self.age = age  
 def say\_age(self):  
 print(self.name,"的年龄是：",self.age)  
class Student(Person):  
 def \_\_init\_\_(self,name,age,score):  
 # 子类并不会自动调用父类的\_\_init\_\_()，我们必须显式的调用它。  
 Person.\_\_init\_\_(self, name, age)  
 #super(Student,self).\_\_init\_\_(name,age)  
 #上面两个方式都可能调用父类的初始化方法。  
 print("Student的构造方法")  
 # self.name = name  
 # self.age = age  
 self.score = score  
s1 = Student("张三",15,85)  
s1.say\_age()  
print(dir(s1))

Person的构造方法

Student的构造方法

张三 的年龄是： 15

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_firstlineno\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_static\_attributes\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'age', 'name', 'say\_age', 'score']

##### 一个父类拥有n多个子类

class person:#没有加（）默认是object的父类  
 def \_\_init\_\_(self,name,age):  
 self.name=name  
 self.age=age  
 def show(self):  
 print(f"大家好我叫：{self.name},我今年是{self.age}岁")  
  
#接下来建立一个子类student去继承父类person  
class student(person):#继承了父类拥有了当中公有的内容和受保护的东西  
 def \_\_init\_\_(self,name,age,id):#name,age父类有可以继承，子类中新建了id  
 super().\_\_init\_\_(name,age)#调用父类的初始化方法  
 self.id=id  
class doctor(person):#新建一个Doctor类  
 def \_\_init\_\_(self,name,age,departmen):#建了一个部门的属性  
 super().\_\_init\_\_(name,age)  
 self.departmen=departmen  
  
 #创建子类对象  
stu=student('tom',20,'1001')  
stu.show()  
doc=doctor('gentel',30,'surgery')  
#调用父类公有的方法  
doc.show()

大家好我叫：tom,我今年是20岁

大家好我叫：gentel,我今年是30岁

##### 子类继承n多个父类

class fatherA:  
 def \_\_init\_\_(self,name):  
 self.name=name  
 def showA(self):  
 print("fatherA的方法")  
class fatherB:  
 def \_\_init\_\_(self,age):  
 self.age=age  
 def showB(self):  
 print("fatehrB的方法")  
#多继承子类继承了AB两个父类  
class son(fatherA,fatherB):  
 def \_\_init\_\_(self,name,age,gender):  
 fatherA.\_\_init\_\_(self,name)#调用多个父类需要用类名区分不能用super了  
 fatherB.\_\_init\_\_(self,age)  
 self.gender=gender  
s=son('gentel',40,'man')  
s.showA()  
s.showB()

fatherA的方法

fatehrB的方法

##### 方法重写

1 成员继承：子类继承了父类除构造方法之外的所有成员。

⚠️(私有属性、私有方法也被继承)

2 方法重写：子类可以重新定义父类中的方法，这样就会覆盖父类的方法，也称为“重写”

【操作】继承和重写的案例

class Person:  
 def \_\_init\_\_(self,name,age):  
 self.name = name  
 self.age = age  
 def say\_age(self):  
 print(self.name,"的年龄是：",self.age)  
 def say\_name(self):  
 print("我是",self.name)  
class Student(Person):  
 def \_\_init\_\_(self,name,age,score):  
 Person.\_\_init\_\_(self,name,age)  
 self.score = score  
 def say\_score(self):  
 print(self.name,"的分数是：",self.score)  
 def say\_name(self): #重写父类的方法  
 print("报告老师，我是",self.name)  
s1 = Student("张三",15,85)  
s1.say\_score()  
s1.say\_name()  
s1.say\_age()

张三 的分数是： 85

报告老师，我是 张三

张三 的年龄是： 15

子类继承了父类就拥有了父类中公有成员和受保护的成员

父类的方法并不能完全适合子类的要求，这个时候子类就可以重写父类的方法

子类在重写父类的方法时，要求方法的名称必须与父类方法的名称相同，在子类重写后的方法中可以通过super().xxx()调用父类中的方法

class person:#没有加（）默认是object的父类  
 def \_\_init\_\_(self,name,age):  
 self.name=name  
 self.age=age  
 def show(self):  
 print(f"大家好我叫：{self.name},我今年是{self.age}岁")  
  
#接下来建立一个子类student去继承父类person  
class student(person):#继承了父类拥有了当中公有的内容和受保护的东西  
 def \_\_init\_\_(self,name,age,id):#name,age父类有可以继承，子类中新建了id  
 super().\_\_init\_\_(name,age)#调用父类的初始化方法  
 self.id=id  
 def show(self):  
 super().show()  
 print(f"我来自北京大学，我的学号是{self.id}")  
class doctor(person):#新建一个Doctor类  
 def \_\_init\_\_(self,name,age,departmen):#建了一个部门的属性  
 super().\_\_init\_\_(name,age)  
 self.departmen=departmen  
 def show(self):  
 # super().show()  
 #上面的父类的方法，不合适，于是下面重写了父类的方法  
 print(f"各位好我叫{self.name},我现在{self.age},我的工作是{self.departmen}")  
  
 #创建子类对象  
stu=student('tom',20,'1001')  
stu.show()  
doc=doctor('gentel',30,'surgery')  
#调用父类公有的方法  
doc.show()

大家好我叫：tom,我今年是20岁

我来自北京大学，我的学号是1001

各位好我叫gentel,我现在30,我的工作是surgery

##### 查看类的继承层次结构

通过类的方法 mro() 或者类的属性 \_\_mro\_\_ 可以输出这个类的继承层次 结构。

【操作】 查看类的继承层次结构

class A:pass

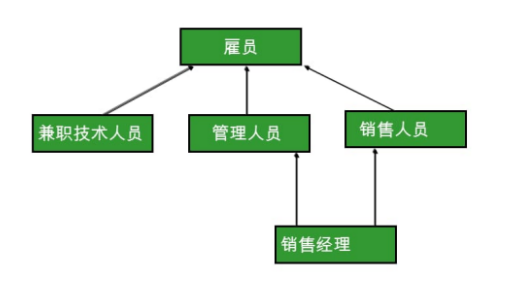
class B(A):pass

class C(B):pass

print(C.mro())

[<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.B'>,<class '\_\_main\_\_.A'>, <class 'object'>]

##### 多重继承



Python支持多重继承，一个子类可以有多个“直接父类”。这样，就 具备了“多个父类”的特点。但是由于，这样会被“类的整体层次”搞的 异常复杂，尽量避免使用。

class A:

def aa(self):

print("aa")

class B:

def bb(self):

print("bb")

class C(B,A):

def cc(self):

print("cc")

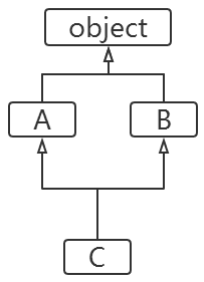
c = C()

c.cc()

c.bb()

c.aa()

类结构为：



##### MRO方法解析顺序

Python支持多继承，如果父类中有相同名字的方法，在子类没有指 定父类名时，解释器将“从左向右”按顺序搜索。

MRO（Method Resolution Order）：方法解析顺序。 我们可以通 过 mro() 方法获得“类的层次结构”，方法解析顺序也是按照这个“类的 层次结构”寻找的。

class A:  
 def aa(self):  
 print("aa")  
 def say(self):  
 print("say AAA!")  
class B:  
 def bb(self):  
 print("bb")  
 def say(self):  
 print("say BBB!")  
class C(B, A):  
 def cc(self):  
 print("cc")  
c = C()  
print(C.mro()) # 打印类的层次结构  
c.say() # 解释器寻找方法是“从左到 右”的方式寻找，此时会执行B类中的say()

[<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.A'>, <class 'object'>]

say BBB!

##### super()获得父类定义

在子类中，如果想要获得父类的方法时，我们可以通过 super() 来做。 super() 代表父类的定义，不是父类对象。

❤️想调用父类的构造方法： super(子类名称,self).\_\_init\_\_(参数列表)

class A:  
 def \_\_init\_\_(self):  
 print("A的构造方法")  
 def say(self):  
 print("A: ", self)  
 print("say AAA")  
class B(A):  
 def \_\_init\_\_(self):  
 super(B, self).\_\_init\_\_() # 调用父类的构造方法  
 print("B的构造方法")  
 def say(self):  
 # A.say(self) 调用父类的say方法  
 super().say() # 通过super()调用父类的方法  
 print("say BBB")  
b = B()  
b.say()

A的构造方法

B的构造方法

A: <\_\_main\_\_.B object at 0x0000025AE6136E40>

say AAA

say BBB

#### 多态

多态是指同一个方法调用由于对象不同会产生不同的行为。生 活中这样的例子比比皆是：同样是休息方法，人不同休息方法 不同。张三休息是睡觉，李四休息是玩游戏，程序员休息是“敲 几行代码”。

关于多态要注意以下2点：

1 多态是方法的多态，属性没有多态。

2 多态的存在有2个必要条件：继承、方法重写

#多态  
class Animal:  
 def shout(self):  
 print("动物叫了一声")  
  
class Dog(Animal):  
 def shout(self):  
 print("小狗，汪汪汪")  
  
class Cat(Animal):  
 def shout(self):  
 print("小猫，喵喵喵")  
  
def animalShout(a):  
 a.shout() # 传入的对象不同，shout方法对应的实际行为也不同。  
  
animalShout(Dog())  
animalShout(Cat())

小狗，汪汪汪

小猫，喵喵喵

是在函数定义时，直接对形式参数进行赋值，在调用时如果该参数不传值，将使用默认值，如果该参数传值，则使用传递的值。

多态指的就是多种形态，即使不知道一个变量所引用的对象到底是什么类型，仍然可以通过这个变量调用对象的方法。

在程序运行过程中根据变量所引用对象的数据类型，动态决定调用哪个对象中的方法。

python语言中的多态，根本不关心对象的数据类型，也不关心类之间是否存在继承关系，只关心对象的行为（方法）。只要不同的类中有同名的方法，即可实现多态。

class person():  
 def eat(self):  
 print('人，吃五谷杂粮')  
class cat():  
 def eat(self):  
 print('猫，吃猫粮')  
class dog:  
 def eat(self):  
 print('狗，吃骨头')  
#这个三个都是一个共同点有一个同名的方法  
  
#编写函数  
def fun(obj):#obj形参  
 obj.eat()#通过obj调用eat方法  
  
#创建三个类的对象  
per=person()  
cat=cat()  
dog=dog()  
  
#调用fun函数  
fun(per)#这就是python当中的多态，不关心你的对象数据类型，只关心你这个对象是否具有同名的方法  
fun(cat)  
fun(dog)

人，吃五谷杂粮

猫，吃猫粮

狗，吃骨头

### object类的常用方法

object 类是所有类的父类，因此所有的类都有 object 类的属性和方法。 我们显然有必要深入研究一下 object 类的结构。对于我们继续深入学 习Python很有好处。

#### dir() 查看对象属性

为了深入学习对象，先学习内置函数 dir() ，他可以让我们方便的看到 指定对象所有的属性

【测试】查看对象所有属性以及和 object 进行比对

class Person:

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

self.name = name

self.age = age

def say\_age(self):

print(self.name,"的年龄是：",self.age)

obj = object()

print(dir(obj))

s2 = Person("高淇",18)

print(dir(s2))

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_']

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'age', 'name', 'say\_age']

从上面我们可以发现这样几个要点：

1 Person 对象增加了六个属性：

\_\_dict\_\_ \_\_module\_\_ \_\_weakref\_\_ age name say\_age

2object 的所有属性， Person 类作为 object 的子类，显然包含了所有的

属性

3 我们打印 age 、 name 、 say\_age ，发现 say\_age 虽然是方法，实际上也是属性。只不过，这个属性的类型是 method 而已。

\_new\_() 由系统调用，用于创建对象

\_int\_() 创建对象时手动调用，用于初始化的对象属性值

\_str\_() 对象的描述，返回值是str类型，默认输出对象的内存地址

class person():  
 def \_\_init\_\_(self,name ,age):  
 self.name=name  
 self.age=age  
 def show(self):  
 print(f"我叫{self.name},{self.age}岁")  
  
#创建对象  
per=person('gente',20)#创建对象时自动调用init的方法  
print(dir(per))#显示对象所有存在的方法  
print(per)#自动调用str的内存地址信息  
  
class son(person):  
 def \_\_str\_\_(self):  
 return "改变str的内存为自定义的描述"  
s=son('guyue',15)  
print(s)

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_firstlineno\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_static\_attributes\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'age', 'name', 'show']

<\_\_main\_\_.person object at 0x00000243FBFE7CB0>

改变str的内存为自定义的描述

1 object 有一个 \_\_str\_\_() 方法，用于返回一个对于“对象的描述”。内置函数 str(对象) ,调用的就是\_\_str\_\_()

2 \_\_str\_\_() 经常用于 print() 方法，帮助我们查看对象的信息。 \_\_str\_\_() 可以重写

#### 特殊方法和运算符重载

Python的运算符实际上是通过调用对象的特殊方法实现的。

a=10  
b=20  
print(dir(a))  
print(a+b)  
print(a.\_\_add\_\_(b))  
print(a.\_\_sub\_\_(b))  
print(a.\_\_lt\_\_(b))  
print(a.\_\_mul\_\_(b))

['\_\_abs\_\_', '\_\_add\_\_', '\_\_and\_\_', '\_\_bool\_\_', '\_\_ceil\_\_', '\_\_class\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_divmod\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_float\_\_', '\_\_floor\_\_', '\_\_floordiv\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getnewargs\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_index\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_int\_\_', '\_\_invert\_\_', '\_\_le\_\_', '\_\_lshift\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_neg\_\_', '\_\_new\_\_', '\_\_or\_\_', '\_\_pos\_\_', '\_\_pow\_\_', '\_\_radd\_\_', '\_\_rand\_\_', '\_\_rdivmod\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rfloordiv\_\_', '\_\_rlshift\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_ror\_\_', '\_\_round\_\_', '\_\_rpow\_\_', '\_\_rrshift\_\_', '\_\_rshift\_\_', '\_\_rsub\_\_', '\_\_rtruediv\_\_', '\_\_rxor\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_sub\_\_', '\_\_subclasshook\_\_', '\_\_truediv\_\_', '\_\_trunc\_\_', '\_\_xor\_\_', 'as\_integer\_ratio', 'bit\_count', 'bit\_length', 'conjugate', 'denominator', 'from\_bytes', 'imag', 'is\_integer', 'numerator', 'real', 'to\_bytes']

30

30

-10

True

200

**常见的特殊方法统计如下：**

方法 说明 例子

\_\_init\_\_ 构造方法 对象创建和初始化： p = Person()

\_\_del\_\_ 析构方法 对象回收

\_\_repr\_\_ , \_\_str\_\_ 打印，转换 print(a)

\_\_call\_\_ 函数调用 a()

\_\_getattr\_\_ 点号运算 a.xxx

\_\_setattr\_\_ 属性赋值 a.xxx = value

\_\_getitem\_\_ 索引运算 a[key]

\_\_setitem\_\_ 索引赋值 a[key]=value

\_\_len\_\_ 长度 len(a)

**每个运算符实际上都对应了相应的方法，统计如下：**

|  |  |  |
| --- | --- | --- |
| 运算符 | 特殊方法 | 功能描述 |
| + | \_\_add\_\_ () | 执行加法运算 |
| - | \_\_sub\_\_() | 执行减法运算 |
| <,<=,== | \_\_lt\_\_(),\_\_le\_\_(),\_\_eq\_\_() | 执行比较运算 |
| >,>=,!= | \_\_gt\_\_(),\_\_ge\_\_ (),\_\_ne\_\_ () | 执行比较运算 |
| \* / % // | \_\_mul\_\_ \_\_truediv\_\_ \_\_mod\_\_ \_\_floordiv\_\_ | 乘、浮点除、模运算（取余）、整 数除 |
| %,// | \_\_mod\_\_ (),\_\_floordiv\_\_ () | 执行取余，整除运算 |
| | ^ & | \_\_or\_\_ \_\_xor\_\_ \_\_and\_\_ | 或、异或、与 |
| << >> | \_\_lshift\_\_ \_\_rshift\_\_ | 左移、右移 |
| \*\* | \_\_pow\_\_() | 执行幂运算 |

**我们可以重写上面的特殊方法，即实现了“运算符的重载”。**

#测试运算符的重载  
class Person:  
 def \_\_init\_\_(self,name):  
 self.name = name  
 def \_\_add\_\_(self, other):  
 if isinstance(other,Person):  
 return "{0}--{1}".format(self.name,other.name)  
 else:  
 return "不是同类对象，不能相加"  
 def \_\_mul\_\_(self, other):  
 if isinstance(other,int):  
 return self.name\*other  
 else:  
 return "不是同类对象，不能相乘"  
p1 = Person("高淇")  
p2 = Person("高希希")  
x = p1 + p2  
print(x)  
print(p1\*3)

高淇--高希希

高淇高淇高淇

#### 特性属性

|  |  |
| --- | --- |
| 特殊属性 | 功能描述 |
| obj.\_dict\_() | 对象的属性字典 |
| obj.\_class\_() | 对象的所属的类 |
| class.\_bases\_() | 类的父类元组 |
| class.\_base\_() | 类的父类 |
| class.\_mro\_() | 类的层次结构 |
| class.\_subclasses\_() | 类的子类列表 |

class A:  
 pass  
class B:  
 pass  
class C(A,B):  
 def \_\_init\_\_(self,name,age):  
 self.name=name  
 self.age=age  
a=A()  
b=B()  
c=C('gentel',20)  
print(a.\_\_dict\_\_,b.\_\_dict\_\_,c.\_\_dict\_\_)  
print(a.\_\_class\_\_,b.\_\_class\_\_,c.\_\_class\_\_)  
print(A.\_\_bases\_\_)  
print(B.\_\_bases\_\_)  
print(C.\_\_bases\_\_)  
print()  
print(A.\_\_base\_\_)  
print(B.\_\_base\_\_)  
print(C.\_\_base\_\_)  
print()  
print(A.\_\_mro\_\_)  
print(B.\_\_mro\_\_)  
print(C.\_\_mro\_\_)  
print()  
print(A.\_\_subclasses\_\_())  
print(B.\_\_subclasses\_\_())  
print(C.\_\_subclasses\_\_())

{} {} {'name': 'gentel', 'age': 20}

<class '\_\_main\_\_.A'> <class '\_\_main\_\_.B'> <class '\_\_main\_\_.C'>

(<class 'object'>,)

(<class 'object'>,)

(<class '\_\_main\_\_.A'>, <class '\_\_main\_\_.B'>)

<class 'object'>

<class 'object'>

<class '\_\_main\_\_.A'>

(<class '\_\_main\_\_.A'>, <class 'object'>)

(<class '\_\_main\_\_.B'>, <class 'object'>)

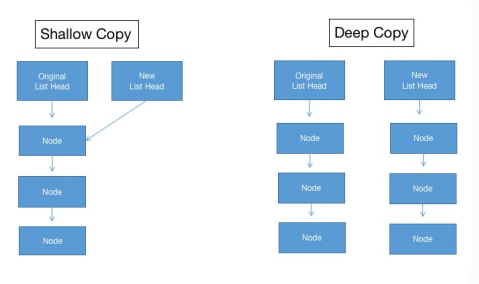
(<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <class '\_\_main\_\_.B'>, <class 'object'>)

[<class '\_\_main\_\_.C'>]

[<class '\_\_main\_\_.C'>]

[]

### 对象的深拷贝与浅拷贝



变量的赋值：只是形成两个变量，实际上还是指同一个对象 com1=com

浅拷贝：拷贝时，对象包含的子对象内容不拷贝，因此，源对象与拷贝对象会引用同一个子

浅拷贝 Python拷贝一般都是浅拷贝。 浅拷贝：拷贝时，拷贝源对象，

对象 com2=copy.copy(com)

深拷贝：使用copy模块的deepcopy函数，递归拷贝对象中包含的子对象，源对象和拷贝对象所有的子对象也不相同 com3=copy.deepcopy(com)

#测试对象的引用赋值、浅拷贝、深拷贝  
import copy  
class MobilePhone:  
 def \_\_init\_\_(self,cpu):  
 self.cpu = cpu  
class CPU:  
 pass  
c = CPU()  
m = MobilePhone(c)  
print("----浅拷贝-------")  
m2 = copy.copy(m) #m2是新拷贝的另一个手机对象  
print("m：",id(m))  
print("m2：",id(m2))  
print("m的cpu：",id(m.cpu))  
print("m2的cpu：",id(m2.cpu)) #m2和m拥有了一样的cpu对象  
print("----深拷贝--------")  
m3 = copy.deepcopy(m)  
print("m：",id(m))  
print("m3：",id(m3))  
print("m的cpu：",id(m.cpu))  
print("m3的cpu：",id(m3.cpu)) #m3和m拥有不一样的cpu对象

----浅拷贝-------

m： 1632872725008

m2： 1632873961488

m的cpu： 1632872724672

m2的cpu： 1632872724672

----深拷贝--------

m： 1632872725008

m3： 1632873961808

m的cpu： 1632872724672

m3的cpu： 1632873962128

class CPU():  
 pass  
class DISK:  
 pass  
class computer:  
 #计算机由cpu和disk组装成的  
 def \_\_init\_\_(self,cpu,disk):  
 self.cpu=cpu  
 self.disk=disk  
cpu=CPU()  
disk=DISK()  
#创建一个计算机对象  
com=computer(cpu,disk)  
#变量对象的赋值  
com1=com  
print(com,"子对象：",com.cpu,com.disk)  
print(com1,"子对象：",com1.cpu,com1.disk)#输出显示出两个对象地址和子对象地址都相同  
  
#类对像的浅拷贝  
import copy#导入模块  
com2=copy.copy(com)#com2是新产生的对象，copy的com  
print('-'\*50)  
print(com,"子对象：",com.cpu,com.disk)  
print(com2,"子对象：",com2.cpu,com2.disk)#通过浅拷贝的对象，对象的内存地址变了，但是子对象不变  
#深拷贝  
com3=copy.deepcopy(com)  
print('-'\*50)  
print(com,"子对象：",com.cpu,com.disk)  
print(com3,"子对象：",com3.cpu,com3.disk)#通过深拷贝的对象，对象及子对象的内存地址都变了

<\_\_main\_\_.computer object at 0x0000016F73868D70> 子对象： <\_\_main\_\_.CPU object at 0x0000016F736D7CB0> <\_\_main\_\_.DISK object at 0x0000016F736D7E00>

<\_\_main\_\_.computer object at 0x0000016F73868D70> 子对象： <\_\_main\_\_.CPU object at 0x0000016F736D7CB0> <\_\_main\_\_.DISK object at 0x0000016F736D7E00>

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

<\_\_main\_\_.computer object at 0x0000016F73868D70> 子对象： <\_\_main\_\_.CPU object at 0x0000016F736D7CB0> <\_\_main\_\_.DISK object at 0x0000016F736D7E00>

<\_\_main\_\_.computer object at 0x0000016F738079D0> 子对象： <\_\_main\_\_.CPU object at 0x0000016F736D7CB0> <\_\_main\_\_.DISK object at 0x0000016F736D7E00>

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

<\_\_main\_\_.computer object at 0x0000016F73868D70> 子对象： <\_\_main\_\_.CPU object at 0x0000016F736D7CB0> <\_\_main\_\_.DISK object at 0x0000016F736D7E00>

<\_\_main\_\_.computer object at 0x0000016F73807B10> 子对象： <\_\_main\_\_.CPU object at 0x0000016F73807C50> <\_\_main\_\_.DISK object at 0x0000016F73807D90>

### 组合

结婚就是组合。两人组合后，可以复用对方的属性 和方法！

除了继承，“组合”也能实现代码的复用！“组合”核心是“将父类对象 作为子类的属性”。

1、is-a关系，我们可以使用“继承”。从而实现子类拥有的父类的方法和属性。 is-a 关系指的是 类似这样的关系：狗是动物，dog is animal。狗类就应该继承动物类。

2 、has-a 关系，我们可以使用“组合”，也能实现一个类拥有另一个类的方法和属性。 has-a 关 系指的是这样的关系：手机拥有CPU。 MobilePhone has a CPU

#组合测试  
class MobilePhone:  
 def \_\_init\_\_(self,cpu,screen):  
 self.cpu = cpu  
 self.screen = screen  
class CPU:  
 def calculate(self):  
 print("计算，算个12345")  
class Screen:  
 def show(self):  
 print("显示一个好看的画面，亮瞎你的钛合金大眼")  
c = CPU()  
s = Screen()  
m = MobilePhone(c,s)  
m.cpu.calculate() #通过组合，我们也能调用cpu对象的方法。相当于手机对象间接拥有了“cpu的方法”  
m.screen.show()

计算，算个12345

显示一个好看的画面，亮瞎你的钛合金大眼

#### 设计模式\_工厂模式实现

设计模式是面向对象语言特有的内容，是我们在面临某一类问题时候固定的做法，设计模式有很多种，比较流行的是：GOF（Goup Of Four）23种设计模式。

当然，我们没有必要全部学习，学习几个常用的即可。

对于初学者，我们学习两个最常用的模式：工厂模式和单例模式。 工厂模式实现了创建者和调用者的分离，使用专门的工厂类将选择实现类、创建对象进行统一的管理和控制。

#工厂模式  
class CarFactory:  
 def createCar(self,brand):  
 if brand == "奔驰":  
 return Benz()  
 elif brand == "宝马":  
 return BMW()  
 elif brand == '比亚迪':  
 return BYD()  
 else:  
 return "未知品牌，无法创建"  
class Benz:  
 pass  
class BMW:  
 pass  
class BYD:  
 pass  
factory = CarFactory()  
c1 = factory.createCar("奔驰")  
c2 = factory.createCar("宝马")  
print(c1)  
print(c2)

<\_\_main\_\_.Benz object at 0x00000253FD3C6F90>

<\_\_main\_\_.BMW object at 0x00000253FD3C70E0>

#### 设计模式\_单例模式实现

单例模式（Singleton Pattern）的核心作用是确保一个类只有一个实例，并且提供一个访问该实例的全局访问点。

单例模式只生成一个实例对象，减少了对系统资源的开销。当一个对象的产生需要比较多的资源，如读取配置文件、产生其他依赖对象时，可以产生一个“单例对象”，然后永久驻留内存中，从而极大的降低开销。

⚠️单例模式有多种实现的方式，我们这里推荐重写 \_\_new\_\_() 的方 法。

#单例模式  
class MySingleton:  
 \_\_obj = None  
 \_\_init\_flag = True  
 def \_\_new\_\_(cls,\*args,\*\*kwargs):  
 if cls.\_\_obj == None:  
 cls.\_\_obj = object.\_\_new\_\_(cls)  
 return cls.\_\_obj  
  
 def \_\_init\_\_(self, name):  
 if MySingleton.\_\_init\_flag:  
 print("init....")  
 self.name = name  
 MySingleton.\_\_init\_flag = False  
  
a = MySingleton("aa")  
print(a)  
b = MySingleton("bb")  
print(b)

init....

<\_\_main\_\_.MySingleton object at 0x0000018E2BC46E40>

None

#### 工厂和单例模式结合

设计模式称之为“模式”，就是一些固定的套路。我们很容易用到其 他场景上，比如前面讲的工厂模式，我们需要将工厂类定义成“单例”，只需要简单的套用即可实现：

#测试工厂模式和单例模式的整合使用  
class CarFactory:  
 \_\_obj = None #类属性  
 \_\_init\_flag = True  
 def create\_car(self,brand):  
 if brand =="奔驰":  
 return Benz()  
 elif brand =="宝马":  
 return BMW()  
 elif brand == "比亚迪":  
 return BYD()  
 else:  
 return "未知品牌，无法创建"  
 def \_\_new\_\_(cls,\*args,\*\*kwargs):  
 if cls.\_\_obj == None:  
 cls.\_\_obj = object.\_\_new\_\_(cls)  
 return cls.\_\_obj  
 def \_\_init\_\_(self):  
 if CarFactory.\_\_init\_flag:  
 print("init CarFactory....")  
 CarFactory.\_\_init\_flag = False  
class Benz:  
 pass  
class BMW:  
 pass  
class BYD:  
 pass  
factory = CarFactory()  
c1 = factory.create\_car("奔驰")  
c2 = factory.create\_car("比亚迪")  
print(c1)  
print(c2)  
factory2 = CarFactory()  
print(factory)  
print(factory2)

init CarFactory....

<\_\_main\_\_.Benz object at 0x000001E5B8896F90>

<\_\_main\_\_.BYD object at 0x000001E5B88970E0>

<\_\_main\_\_.CarFactory object at 0x000001E5B8896E40>

<\_\_main\_\_.CarFactory object at 0x000001E5B8896E40>

### 实战

#### 算圆的面积和周长

#算圆的面积和周长 面积的公式S=3.14\*r的2次方 圆的周长公式C=2\*3.14\*r  
class count():  
 def \_\_init\_\_(self,r):  
 self.r=r  
 def s(self):  
 return 3.14\*pow(self.r,2)  
 def c(self):  
 return 2\*3.14\*self.r  
r=eval(input("请输入圆的半径:"))  
cou=count(r)  
s=cou.s()  
c=cou.c()  
print(f"圆的面积：{s}\t圆的周长：{c}")

请输入圆的半径:4

圆的面积：50.24 圆的周长：25.12

#### 定义学生类录入5个学生信息存储到列表中

from traceback import print\_tb  
class student:  
 def \_\_init\_\_(self,name,age,genden,fraction):  
 self.name=name  
 self.age=age  
 self.genden=genden  
 self.fraction=fraction  
 def info(self):  
 print(self.name,self.age,self.genden,self.fraction)  
lst=[]  
print("请输入5位学生信息，（姓名#年龄#性别#成绩）")  
for i in range(1,6):  
 s=input(f"输入第{i}位学生信息：")  
 s\_lst=s.split('#')#将#号分割  
 stu=student(s\_lst[0],s\_lst[1],s\_lst[2],s\_lst[3],)  
 lst.append(stu)#这里将stu这个对象装进了lst这个列表里了  
for item in lst:  
 item.info()

请输入5位学生信息，（姓名#年龄#性别#成绩）

输入第1位学生信息：gentel#18#men#100

输入第2位学生信息：tainn#17#women#70

输入第3位学生信息：tom#18#men#88

输入第4位学生信息：tim#18#women#95

输入第5位学生信息：jiayue#16#women#100

gentel 18 men 100

tainn 17 women 70

tom 18 men 88

tim 18 women 95

jiayue 16 women 100

#### 使用面向对象思想实现乐器弹奏

class instrument:  
 def make\_sound(self):  
 pass  
class erhu(instrument):  
 def make\_sound(self):  
 print('二胡在演奏')  
class piano(instrument):  
 def make\_sound(self):  
 print('钢琴在演奏')  
class violin(instrument):  
 def make\_sound(self):  
 print('小提琴在演奏')  
#定义一个函数弹奏各种乐器  
def play(obj):  
 obj.make\_sound()  
#测试  
erhu=erhu()  
piano=piano()  
violin=violin()  
#调用  
play(erhu)  
play(piano)  
play(violin)

二胡在演奏

钢琴在演奏

小提琴在演奏

#### 面向对象的思想设计自定义类

class vehicle:  
 def \_\_init\_\_(self,model,license\_plate):  
 self.model=model  
 self.license\_plate=license\_plate  
 def start(self):  
 print('启动')  
 def stop(self):  
 print('停止')  
  
class taxi(vehicle):  
 def \_\_init\_\_(self,model,license\_plate,company):  
 super().\_\_init\_\_(model,license\_plate)  
 self.company=company  
 #重写父类方法  
 def start(self):  
 print(f'乘客你好我是{self.company}公司的，我的车牌是{self.license\_plate},您要去哪里？')  
 def stop(self):  
 print('目的地快到了，请您付款下车，欢饮下次乘坐')  
class family\_car(vehicle):  
 def \_\_init\_\_(self, model, license\_plate, name):  
 super().\_\_init\_\_(model, license\_plate)  
 self.name=name  
 # 重写父类方法  
 def start(self):  
 print(f'我是{self.name},我的汽车我做主')  
 def stop(self):  
 print('目的地快到了我们去玩吧')  
  
t=taxi('福克斯','京：888888','长城')  
t.start()  
t.stop()  
f=family\_car('byd','苏：666666','gentel')  
f.start()  
f.stop()

乘客你好我是长城公司的，我的车牌是京：888888,您要去哪里？

目的地快到了，请您付款下车，欢饮下次乘坐

我是gentel,我的汽车我做主

目的地快到了我们去玩吧

# 提高

## 模块及常用的第三方模块

在python中一个后缀名为.py的python文件就是模块

模块中可以定义函数、类等

模块也可以避免函数、类、变量等名称相冲突的问题

模块不仅提高了代码的可维护性，同进还提高了代码的可重用性

在给模块命名的时候要求全部使用小写字母，多个单词之间使用下划线进行分隔

如果自定义模块名称与系统内置模块名称相同，那么在导入时会优先导入自定义的模块。

**模块：**

系统内置模块：由开发人员编写好的模块，在安装python解释器时一同安装成计算机

自定义模块：一个以.py结尾的文件就是一个模块，新建python文件，实际上就是在新建模块

**自定义模块的作用：**

一个规范代码，将功能相同的函数、类等封装到一个模块中，让代码更易于阅读，

另外一个目的与系统内置模块相同，即可以被其他模块调用，提高开发的效率。

### 自定义模块的创建

用小写字母建一个.py的文件，就是一个模块,myself.py，我新建了一个自我介绍的模块

myself.py

name='gentel'  
  
def info():  
 print(f"我叫{name}")

这就是一个输出我叫什么的模块

myself2.py

name='guyue'  
def info():  
 print(f"大家好我叫{name}")

### 模块的导入

模块缩写守成就可以被其他模块进行调用并使用被调用模块中的功能

import导入方式 的语法结构: import 模块名称[as别名]

如果模块名称太长，可以通过as自己改一个名字

from...import导入方式的语法结构： from 模块名称 import 变量/函数/类/\*

from的方式导入，调用的时候就不用再写模块名字做前缀，里面的变量函数类可以直接使用，\*代表导入所有的，代表通配符

#### 第一种方式:import

导入上面自定义的模块

import myself  
print(myself.name)  
myself.info()

import myself as my#在本文件中给模块起别名  
print(my.name)  
my.info()

#### 第二种方式form

from myself import name,info#name导入的是变量，info导入的函数  
print(name)  
info()

from myself import \*#\*可以导入这个模块的所有  
print(name)  
info()

gentel

大家好我叫gentel

import math,time,random#同时导入多个模块

如果导入的模块中，模块有相同的函数，后面的函数给把前面的函数覆盖，所以只能通过打点名称去使用。

import myself,myself2  
myself.info()  
myself2.info()

大家好我叫gentel

大家好我叫guyue

### python中包的定义

包

含有\_init\_.py文件的文件夹（目录）

可以避免模块名称冲突的问题

主程序运行

if\_name\_ == ‘\_main\_’:

pass

#### 包的创建

在pccharm中新建文件的地主新建python包，建好以后，下面就会有一个\_init\_.py的文件，可以在包里创建模块

print('版权归属gentel')  
print('自学爱好者')

对\_init\_py进行了一个简单的编辑

mybao.py

def info():  
 print('大家好我叫gentel')  
name="gentel"

包里创建了一个模块

#### 包的导入

在包外面创建了一个load\_bao.py

import bao.mybao as my#导入包bao下面的mybao模块，并起别名叫my  
my.info()  
  
print('-'\*60)  
from bao import mybao as my #导入包名bao，下面的mybao模块并改别名为my  
my.info()  
  
print('-'\*60)  
from bao.mybao import info #导入bao名下mybao模块的info  
info()#直接可以调用函数

print('-'\*60)  
from bao.mybao import \* #用通配符导入  
info()

版权归属gentel

自学爱好者 #首次执行会默认执行\_init\_py的文件

大家好我叫gentel

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

大家好我叫gentel #二次执行就不会执行这个文件了

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

大家好我叫gentel

#### 主程序运行

举例，如创建了一个mod1.py

print("welcome beijing")

又创建了一个mod2.py

import mod1 #导入mod1模块

welcome beijing #会自动输出mod1的信息

为了不让mod1自动输出修为mod1.py

if \_\_name\_\_ == '\_\_main\_\_':#将输出放在主程序运行里面，调用的时候就不会默认输出了  
 print("welcome beijing")

把不希望自动执行的代码放在主程序运行里面。

### python中常用的内置模块

在安装python解释器时与解释器一起安装进来的模块被称为系统内置模块也被称为标准模块或标准库，大概270个左右，在python安装目录的Python\Python313\Lib中

|  |  |
| --- | --- |
| 标准库名称 | 功能描述 |
| os模块 | 与操作系统和文件相磁操作有关的模块 |
| re模块 | 用于在python的字符串执行正则表达式的模块 |
| random模块 | 用于产生随机数的模块 |
| json模块 | 用于对高维数据进行编码和解码的模块 |
| time模块 | 与时间相关的模块 |
| datetime模块 | 与日期时间相关的模块，可以方便的显示日期并对日期进行运算 |

#### random模块

是python中用于产生随机数的标准库

|  |  |
| --- | --- |
| 函数名称 | 功能描述 |
| seed(x) | 初始化给定的随机数种子，默认为当前系统时间 |
| random() | 产生一个[0.0,1.0]之间的随机小数 |
| randint(a,b) | 生成一个[a,b]之间的整数 |
| randrange(m,n,k) | 生成一个[m,n]之间步长为k的随机整数 |
| uniform(a,b) | 生成一个[a,b]之间的随机小数 |
| choice(seq) | 从序列中随机选择一个元素 |
| shuffle(seq) | 将序列seq中元素随机排列，返回打乱后的序列 |

举例：

import random  
#设置随机数的种子  
random.seed(10)#随机种相同所产生的随机数也相同  
print(random.random())#0.0到1.0但是不包含1。[0, 1)  
print(random.random())#按ctrl点击函数可以看到函数源码的解释  
  
random.seed(10)  
print(random.randint(1,100))#含1和100的数[a, b]  
  
for i in range(10):  
 print(random.randrange(1,10,3),end="\t")  
  
print()  
print(random.uniform(1,100))#[a, b]  
  
lst=[i for i in range(1,11)]  
print(random.choice(lst))  
  
#随机的排序  
print(lst)  
random.shuffle(lst)  
print(lst)  
random.shuffle(lst)  
print(lst)

0.5714025946899135

0.4288890546751146

74

1 4 4 7 1 1 4 4 4 7

81.25126013057475

1

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

[5, 4, 10, 7, 3, 2, 1, 6, 8, 9]

[3, 4, 5, 6, 8, 10, 7, 1, 2, 9]

#### time模块

time模块是python中提供的用于处理时间的标准库，可以来进行时间处理、时间格式化和计时等

|  |  |
| --- | --- |
| 函数名称 | 功能描述 |
| time() | 获取当前时间戳 |
| localtime(sec) | 获取指定时间戳对应的本地时间的struct\_time对象 |
| ctime() | 获取当前时间戳对应的易读字符串 |
| strftime() | 格式化时间，结果为字符串 |
| strptime() | 提取字符串的时间，结果为struct\_time对象 |
| sleep(sec) | 休眠sec秒 |

##### 格式化字符串

|  |  |  |
| --- | --- | --- |
| 格式化字符串 | 日期/时间 | 取值范围 |
| %Y | 年份 | 0001-9999 |
| %m | 月份 | 01-12 |
| %B | 月名 | january-december |
| %d | 日期 | 01-31 |
| %A | 星期 | Monday-Sunday |
| %H | 小时（24h制） | 00-23 |
| %l | 小时（12h制） | 01-12 |
| %M | 分钟 | 00-59 |
| %S | 秒 | 00-59 |

import time  
now=time.time()  
print(now)  
  
obj=time.localtime()  
print(obj)#对象可以看到看月日等全部信息  
  
obj2=time.localtime(60)#60代表60秒  
print(obj2)  
print("year:",obj2.tm\_year)  
  
print(time.ctime())#易读的字符串  
print(time.strftime('%Y年%m月%d日 %H：%M：%S',time.localtime()))#日期时间格式化  
print(time.strptime('2008-8-8','%Y-%m-%d'))#将所提供的字符串转成对象  
time.sleep(2)#程序暂停2秒  
print('helloworld')

1743475246.6275158

time.struct\_time(tm\_year=2025, tm\_mon=4, tm\_mday=1, tm\_hour=10, tm\_min=40, tm\_sec=46, tm\_wday=1, tm\_yday=91, tm\_isdst=0)

time.struct\_time(tm\_year=1970, tm\_mon=1, tm\_mday=1, tm\_hour=8, tm\_min=1, tm\_sec=0, tm\_wday=3, tm\_yday=1, tm\_isdst=0)

year: 1970

Tue Apr 1 10:40:46 2025

2025年04月01日 10：40：46

time.struct\_time(tm\_year=2008, tm\_mon=8, tm\_mday=8, tm\_hour=0, tm\_min=0, tm\_sec=0, tm\_wday=4, tm\_yday=221, tm\_isdst=-1)

helloworld

#### datetime模块

|  |  |
| --- | --- |
| 类名 | 功能描述 |
| datetime.datetime | 表示日期时间的类 |
| datetime.timedelta | 表示时间间隔的类 |
| datetime.date | 表示日期的类 |
| datetime.time | 表示时间的类 |
| datetime.tzinfo | 时区相关的类 |

datetime.now() 获取当前的时间

from datetime import datetime#从datetime模块中导入datetime类  
dt=datetime.now()#获取当前系统的时间和时间  
print(dt)  
#datetime是一个类，手动创建这个类的对象  
dt2=datetime(2020,8,8,20,8)  
print(dt2,type(dt2))  
print(dt2.year,dt2.month,dt2.day)#将dt2的年月日单独取出来  
  
#比较datetime对象的大小  
print(dt<dt2)  
  
#datetiem类型与字符串转换  
d3=dt.strftime('%Y %m %d')  
print(d3,type(d3))  
  
#将字符串类型转成datetiem类型  
d4=datetime.strptime(d3,'%Y %m %d')  
print(d4,type(d4))

2025-04-02 10:19:22.631770

2020-08-08 20:08:00 <class 'datetime.datetime'>

2020 8 8

False

2025 04 02 <class 'str'>

2025-04-02 00:00:00 <class 'datetime.datetime'>

from datetime import datetime  
from datetime import timedelta  
#创建datetime的对象  
delta=datetime(2020,10,1)-datetime(2020,5,1)#日期的运算  
print(delta,type(delta))  
print('2020年5月1号+delta天等于哪一天',datetime(2020,5,1)+delta)#  
  
#通过一个参数创建timedelta对象  
td1=timedelta(10,11)  
print(td1)

153 days, 0:00:00 <class 'datetime.timedelta'>

2020年5月1号+delta天等于哪一天 2020-10-01 00:00:00

10 days, 0:00:11

#### 海龟画图import turtle

import turtle #导入turtle模块

turtle.showturtle() #显示箭头

turtle.write("高淇") #写字符串

turtle.forward(300) #前进300像素

turtle.color("red") #画笔颜色改为red

turtle.left(90) #箭头左转90度

turtle.forward(300)

turtle.goto(0,50) #去坐标（0,50）

turtle.goto(0,0)

turtle.penup() #抬笔。这样，路径就不会画出来

turtle.goto(0,300)

turtle.pendown() #下笔。这样，路径就会画出来

turtle.circle(100) #画圆

turtle.done() #程序结束，保持窗口存在

### python第三方模块的安装与卸载

第三方模块由全球python爱好者、程序员、各行各业的专家进行开发并进行维护

pip默认的下载超时时间可能较短，可以通过修改pip配置文件增加超时时间。在命令行中运行以下命令：pip config set global.timeout 60

上述命令将下载超时时间设置为 60 秒，你可以根据实际情况调整该数值。

清理 pip 缓存：有时，pip的缓存可能会导致下载问题，可以先清理缓存再重新安装：

pip cache purge

#### 在pycharm中安装第三方模块

PyCharm：在 PyCharm 中，点击 “File”（文件）->“Settings”（设置）->“Project: [项目名称]”->“Python Interpreter”（Python 解释器），然后点击右上角的 “+” 按钮，在弹出的窗口中搜索并选择要安装的模块，点击 “Install Package”（安装包）即可。

#### 安装第三方模块的语法：

pip install模块名称

例如：装一个爬虫的模块

pip install requests

pip install 模块名 –i 这种安装方式是提定其他的国内的镜像源

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

清华大学的镜像源<https://pypi.tuna.tsinghua.edu.cn/simple>

阿里云镜像源https://mirrors.aliyun.com/pypi/simple

#### 卸载第三方模块的语法结构：

pip uninstall 模块名称

#### 升级pip命令的语句结构

python –m pip install --upgrade pip

### 第三方模块的使用

#### requests模块的使用

requests库，是用于处理http请求的第三方库，该库在爬虫程序中应用非常广泛

使用requests库中的get（）函数可以找开一个网络请求，并获取一个response响应对象，响应结果中的字符串数据可以通过响应对象的text属性获取，响应结果中除了有字符串数据也有二进制数据，响应结果中的二进制数据可以通过响应对象的content属性获取。

爬取天气网站的关键字-----用谷歌浏览器的检查功能

import re  
import requests  
ur='https://www.weather.com.cn/weather1d/101010100.shtml#search' #爬虫打开的浏览器上网的网做网页  
resp=requests.get(ur)#打开浏览器并打开网址  
resp.encoding='utf-8'#设置一下编码格式  
# print(resp.text)#响应对象  
#将上面所有的数据进行提取用正则表达式  
city=re.findall('<span class="name">([\u4e00-\u9fa5]\*)</span>',resp.text,)  
weather=re.findall('<span class="weather">([\u4e00-\u9fa5]\*)</span>',resp.text,)  
wd=re.findall('<span class="wd">(.\*)</span>',resp.text,)  
zs=re.findall('<span class="zs">([\u4e00-\u9fa5]\*)</span>',resp.text,)  
print(city)  
print(weather)  
print(wd)  
print(zs)  
#将数据打包  
lst=[]  
for a,b,c,d in zip(city,weather,wd,zs):  
 lst.append([a,b,c,d])  
for item in lst:  
 print(item)

['景区', '三亚', '九寨沟', '大理', '张家界', '桂林', '青岛']

['天气', '多云', '多云转小雨', '多云', '晴转小雨', '阴', '多云转小雨']

['气温', '31/23℃', '28/9℃', '23/11℃', '32/16℃', '24/18℃', '18/9℃']

['旅游指数', '适宜', '适宜', '适宜', '适宜', '适宜', '适宜']

['景区', '天气', '气温', '旅游指数']

['三亚', '多云', '31/23℃', '适宜']

['九寨沟', '多云转小雨', '28/9℃', '适宜']

['大理', '多云', '23/11℃', '适宜']

['张家界', '晴转小雨', '32/16℃', '适宜']

['桂林', '阴', '24/18℃', '适宜']

['青岛', '多云转小雨', '18/9℃', '适宜']

爬取百度网站的log图标

import requests  
url='https://www.baidu.com/img/PCtm\_d9c8750bed0b3c7d089fa7d55720d6cf.png'  
resp=requests.get(url)  
  
#将图片保存到本地  
with open('logo.png','wb')as file:  
 file.write(resp.content)

#### openpyxl模块的使用

openpyxl模块是 用于处理microsoft excel文件的第三方库

可以对excel文件中的数据进行写入和读取

|  |  |
| --- | --- |
| 函数/属性名称 | 功能描述 |
| load\_workbook(filename) | 打开已存在的表格，结果为工作薄对象 |
| workbook.sheetnames | 工作薄对象的sheetnames属性，用于获取所有工作表的名称，结果为列表类型 |
| sheet.append(lst) | 向工作表中添加一行数据，新数据接在工作表已有数据的后面 |
| workbook.save(excelname) | 保存工作簿 |
| workbook() | 创建新的工作簿对象 |
| workbook.create\_sheet('景区天气') | 创建工作表 |

将天气爬虫数据打包

import requests  
  
#定义函数  
def get\_html():  
  
 ur='https://www.weather.com.cn/weather1d/101010100.shtml#search' #爬虫打开的浏览器上网的网做网页  
 resp=requests.get(ur)#打开浏览器并打开网址  
 resp.encoding='utf-8'#设置一下编码格式  
 # print(resp.text)#响应对象  
 return resp.text  
def parse\_html(html\_str):  
 #将上面所有的数据进行提取用正则表达式  
 city=re.findall('<span class="name">([\u4e00-\u9fa5]\*)</span>',html\_str,)  
 weather=re.findall('<span class="weather">([\u4e00-\u9fa5]\*)</span>',html\_str,)  
 wd=re.findall('<span class="wd">(.\*)</span>',html\_str,)  
 zs=re.findall('<span class="zs">([\u4e00-\u9fa5]\*)</span>',html\_str,)  
 #将数据打包  
 lst=[]  
 for a,b,c,d in zip(city,weather,wd,zs):  
 lst.append([a,b,c,d])  
 return lst

将爬虫数据写和excel

from operator import itemgetter  
  
import weather  
import openpyxl  
html=weather.get\_html()#发送请求，得响应结果  
lst=weather.parse\_html(html)#解析数据  
#创建一个新的excel工作薄  
workbook=openpyxl.Workbook()#创建对象  
#在excel当中去创建工作表  
sheet=workbook.create\_sheet('景区天气')  
#向工作表中添加数据  
for item in lst:  
 sheet.append(item)#一次添加一行  
  
workbook.save('景区天气.xlsx')

读取excel表的数据

import openpyxl  
workbook=openpyxl.load\_workbook('景区天气.xlsx')  
sheet=workbook['景区天气']  
#景区数据是一个二维列表  
lst=[]#存储行数数所  
for row in sheet.rows:  
 sublst=[]#存储单元格数据  
 for cell in row:  
 sublst.append(cell.value)  
 lst.append(sublst)  
  
for item in lst:  
 print(item)

['景区', '天气', '气温', '旅游指数']

['三亚', '多云', '31/23℃', '适宜']

['九寨沟', '小雨转多云', '25/7℃', '适宜']

['大理', '多云转晴', '25/11℃', '适宜']

['张家界', '小雨转晴', '30/14℃', '适宜']

['桂林', '小雨', '24/19℃', '适宜']

['青岛', '多云', '12/8℃', '适宜']

#### pdfplumber

可以于从pdf文件中读取内容

import pdfplumber  
#打开pdf文件  
with pdfplumber.open('Python From Beginner to Expert.pdf') as pdf:#重命名为pdf  
 for page\_num, page in enumerate(pdf.pages, start=1):#遍历页enumerate函数会给pdf.pages中的每个元素配上一个序号，start=1表示序号从 1 开始  
 if page\_num > 2:#显示两页  
 break  
 print(page.extract\_text())#extract\_text()方法提取内容  
 print(f'.............{page\_num}页')

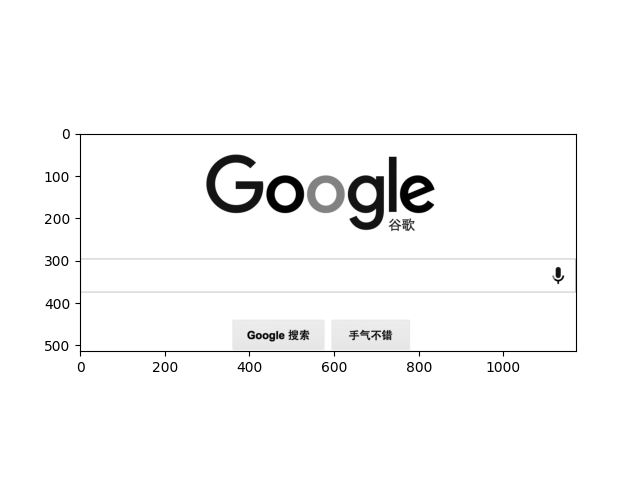
显示两页的内容

#### Numpy模块的使用

是python数据分析方向和其它库的依赖库，用于处理数组、矩阵等数据

numpy的模块图像的灰度处理

import numpy as np  
import matplotlib.pyplot as plt#  
#读取图片  
m1=plt.imread('google.png')#数据是三维数据，分别是宽，高 RGB颜色  
# print(m1)  
plt.imshow(m1)  
#灰度处理  
m2=np.array([0.299,0.587,0.114])#创建灰度的数组  
# 将数组mi和m2进行数组的点乘运算  
x=np.dot(m1,m2)  
# 传入数组  
plt.imshow(x,cmap='gray')  
# 显示图片  
plt.show()



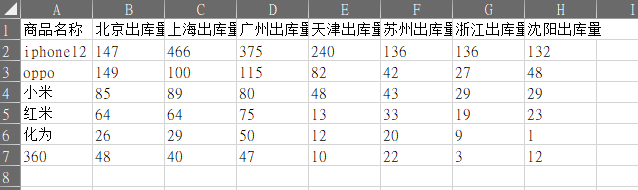
#### Pandas模块与matplotlib模块的使用

pandas是基于Numpy模块扩展的一个非常重要的数据分析模块，使用Pandas读取Excel数据更加的方便

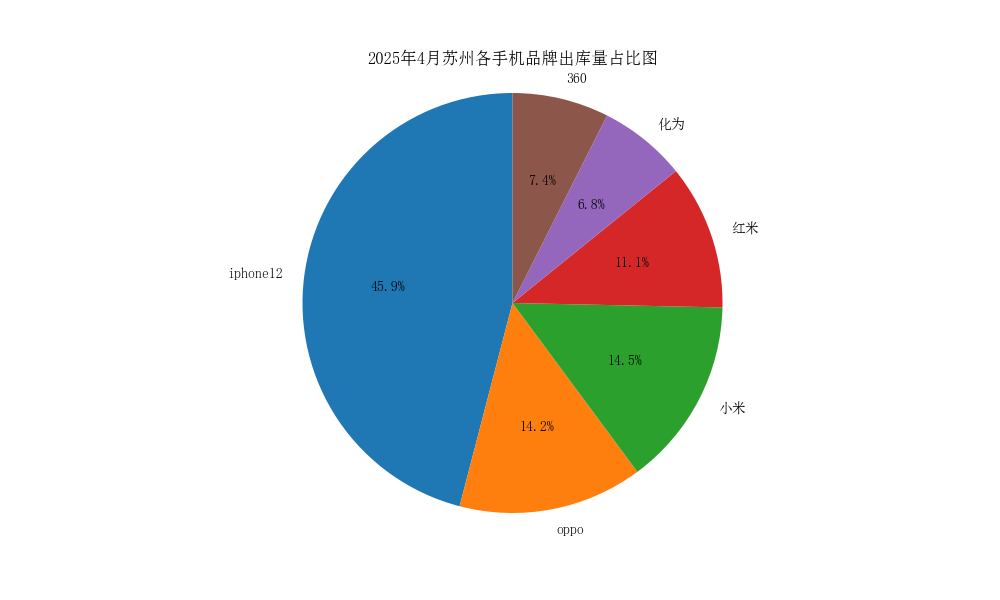
Matplotlib是用于数据可视化的模块，使用Matplotlib.pyplot可以非常方便的绘制饼图，柱形图，折线图等。

**用pandas读取excel数据并用matplotlib画图**

表格名为JQ手机销售数据,在相对目录excel文件下



import pandas as pd  
import matplotlib.pyplot as plt  
#读取excel文件  
df=pd.read\_excel('excel/JQ手机销售数据.xlsx')  
# print(df)  
#绘图会产生中中文乱码，所以先解决中文乱码的问题，  
plt.rcParams['font.sans-serif'] = ['SimSun']  
#绘图需要设置画布的大小  
plt.figure(figsize=(10,6))  
labels=df['商品名称']  
y=df['苏州出库量']  
# print(labels)  
# print(y)  
plt.pie(y,labels=labels,autopct='%1.1f%%',startangle=90)  
# #设置一下x,y轴一致，保证饼图是正圆  
plt.axis('equal')  
plt.title('2025年4月苏州各手机品牌出库量占比图')  
# #显示图表  
plt.show()



#### PyEcharts模块的使用

pyecharts是由百度开源的数据可视化库，它对流行图的支持度比较高，它给用户提供了30多种图形，如柱形渐变图，k线周期图等

中文帮助文档：https://pyecharts.org/#/zh-cn/

pyecharts的使用可以分四个步骤实现：

1. 导入pyecharts包
2. 找到相应图形模板
3. 准备相应数据
4. 对图表进行个性化修饰

#### PIL模块图像的处理

是用于图像处理的第三方库，它支持图像存储、处理和显示等操作

安装是pip install pillow

pil模块图像颜色交换.py

from PIL import Image  
#加载图片  
im=Image.open('image/google.PNG')  
# print(im)  
#提取RGB图像的颜色通道  
r,g,b=im.split()  
# print(r,g,b)  
#合并通道  
om=Image.merge(mode='RGB',bands=(r,b,g))  
om.save('image/new\_google.jpg')

#### jieba模块实现中文分词

jieba是python中用于对中文进行分词的模块，它可以将一段中文文本分隔成中文词组的序列

用分词统计评论关键字的次数

import jieba  
#读取文件  
with open('txt/京东iPad 11英寸 A16芯片2025年款用户评价.txt','r',encoding='utf-8') as file:  
 s=file.read()  
# print(s)  
#分词  
lst=jieba.lcut(s)  
# print(lst)  
#去重操作  
setl=set(lst)#用集合去重  
#统计词出现的频率  
d={}#使用字典的词，volue做为出现的频率  
for item in setl:  
 if len(item)>=2:  
 d[item]=0  
# print(d)  
for item in lst:  
 if item in d:  
 d[item]=d.get(item)+1  
# print(d)  
new\_lis=[]  
for item in d:  
 new\_lis.append([item,d[item]])  
# print(new\_lis)  
  
#列表排序  
new\_lis.sort(key=lambda x:x[1],reverse=True)  
print(new\_lis[0:11])#切片只显示前10项

[['2025', 105], ['03', 88], ['非常', 53], ['屏幕', 45], ['速度', 43], ['轻薄', 40], ['运行', 40], ['不错', 36], ['外观', 30], ['流畅', 28], ['喜欢', 27]]

#### PyInstaller模块打包源文件

可以在windows操作系统中将python源文件打包成.exe的可执行文件。还可以在linux和mac os操作系统中对源文件进行打包操作

注意事项：在进行文件打包时，需要打包的文件尽量不要有中文，而且需要打包的文件路径也尽量不要有中文，路径中包含中文有可能导致打包失败

打包的语法结构为：

pyinstaller –F 源文件文件名

如果文件路径存在空格则用引号如下面

pyinstaller -F "F:\gentel\learn\system\Python file\learning\jieba\_rabao.py"

### 实战

#### 高铁售票系统.py

import prettytable as pt  
#显示座席  
def show\_ticket(row\_num):  
 tb=pt.PrettyTable()#创建一张表格  
 #设置标题行  
 tb.field\_names=['行号','座位1','座位2','座位3','座位4','座位5']  
 #遍历票  
 for i in range(1,row\_num+1):  
 lst=[f'第{i}行','有票','有票','有票','有票','有票']  
 tb.add\_row(lst)  
 print(tb)  
#订票  
def order\_ticket(row\_num,row,column):  
 tb = pt.PrettyTable() # 创建一张表格  
 # 设置标题行  
 tb.field\_names = ['行号', '座位1', '座位2', '座位3', '座位4', '座位5']  
 # 遍历票  
 for i in range(1, row\_num + 1):  
 if int(row)==i:  
 lst = [f'第{i}排', '有票', '有票', '有票', '有票', '有票']  
 lst[int(column)]= '已售'  
 tb.add\_row(lst)  
 else:  
 lst = [f'第{i}行', '有票', '有票', '有票', '有票', '有票']  
 tb.add\_row(lst)  
 print(tb)  
if \_\_name\_\_ == '\_\_main\_\_':  
 row\_num=6  
 show\_ticket(row\_num)  
 #开始售票  
 choose\_num=input("请输入您先选择的座位：比如4.3表示第四排第3列：")  
 row,column=choose\_num.split(".")#系列解包赋值  
 order\_ticket(row\_num,row,column)

+-------+-------+-------+-------+-------+-------+

| 行号 | 座位1 | 座位2 | 座位3 | 座位4 | 座位5 |

+-------+-------+-------+-------+-------+-------+

| 第1行 | 有票 | 有票 | 有票 | 有票 | 有票 |

| 第2行 | 有票 | 有票 | 有票 | 有票 | 有票 |

| 第3行 | 有票 | 有票 | 有票 | 有票 | 有票 |

| 第4行 | 有票 | 有票 | 有票 | 有票 | 有票 |

| 第5行 | 有票 | 有票 | 有票 | 有票 | 有票 |

| 第6行 | 有票 | 有票 | 有票 | 有票 | 有票 |

+-------+-------+-------+-------+-------+-------+

请输入您先选择的座位：比如4.3表示第四排第3列：4.2

+-------+-------+-------+-------+-------+-------+

| 行号 | 座位1 | 座位2 | 座位3 | 座位4 | 座位5 |

+-------+-------+-------+-------+-------+-------+

| 第1行 | 有票 | 有票 | 有票 | 有票 | 有票 |

| 第2行 | 有票 | 有票 | 有票 | 有票 | 有票 |

| 第3行 | 有票 | 有票 | 有票 | 有票 | 有票 |

| 第4排 | 有票 | 已售 | 有票 | 有票 | 有票 |

| 第5行 | 有票 | 有票 | 有票 | 有票 | 有票 |

| 第6行 | 有票 | 有票 | 有票 | 有票 | 有票 |

+-------+-------+-------+-------+-------+-------+

#### 推算几天后的日期.py

import datetime  
#插入日期函数  
def input\_date():  
 inputdate=input('请输入开始日期：(20281001)后按回车：')  
 datestr=inputdate[0:4]+'-'+inputdate[4:6]+'-'+inputdate[6:8]#将输入的日期切片  
 #类型转换  
 dt=datetime.datetime.strptime(datestr,'%Y-%m-%d')  
 return dt  
#主程序运行  
if \_\_name\_\_ == '\_\_main\_\_':  
 # print(input\_date())  
 date=input\_date()  
 #输入间隔的天数  
 in\_num=eval(input('请输入间隔的天数：'))  
 date=date+datetime.timedelta(days=in\_num)  
 print(f'你推算的日期是{date}')

请输入开始日期：(20281001)后按回车：20201020

请输入间隔的天数：10

你推算的日期是2020-10-30 00:00:00

#### 词云图

from traceback import print\_tb  
import jieba  
from wordcloud import WordCloud  
with open('txt/京东iPad 11英寸 A16芯片2025年款用户评价.txt','r',encoding='utf-8') as file:  
 s=file.read()  
#中文分词  
lst=jieba.lcut(s)  
#排除一些不好的词  
stopword=['运行','散热','2025','03', '轻薄', '运行速度','流畅','屏幕效果','外观材质', '轻薄程度' ,  
 '音质音效','屏幕清晰','系统流畅','其他特色']  
txt=''.join(lst)  
#绘制词云图  
wordcloud=WordCloud(background\_color='white',font\_path='msyh.ttc',stopwords=stopword,  
 width=800,height=600)  
#txt生成词云图  
wordcloud.generate(txt)  
#保存图片  
wordcloud.to\_file('image/ipad评价词云图.png')



## 文件及IO操作

一个完整的程序一般都包括数据的存储和读取；我们在前面写的程 序数据都没有进行实际的存储，因此python解释器执行完数据就消 失了。实际开发中，我们经常需要从外部存储介质（硬盘、光盘、 U盘等）读取数据，或者将程序产生的数据存储到文件中，实现“持 久化”保存。

### 文本文件和二进制文件

按文件中数据组织形式，我们把文件分为文本文件和二进制文件两 大类。

1 文本文件

文本文件存储的是普通“字符”文本，python默认为 unicode 字符集 （两个字节表示一个字符，最多可以表示：65536个），可以使 用记事本程序打开。

2 二进制文件

二进制文件把数据内容用“字节”进行存储，无法用记事本打开。必 须使用专用的软件解码。常见的有：MP4视频文件、MP3音频文 件、JPG图片、doc文档等等。

文件：存储在计算机的存储设备中的一组数据序列就是文件

不同类型的文件通过后缀名进行区分

文本文件：由于编码格式的不同，所占磁盘空间的字节数不同

二进制文件：没有统一的编码，文件直接由0或1组成需要使用指定的软件才能打开

### 文件操作相关模块概述

Python标准库中，如下是文件操作相关的模块，我们会陆续给大家 介绍。

名称 说明

io模块 文件流的输入和输出操作 input output

os模块 基本操作系统功能，包括文件操作

glob模块 查找符合特定规则的文件路径名

fnmatch模块 使用模式来匹配文件路径名

fileinput模块 处理多个输入文件

filecmp模块 用于文件的比较

csv模块 用于csv文件处理

pickle和cPickle 用于序列化和反序列化

xml包 用于XML数据处理

bz2、gzip、zipfile、zlib、tarfile 用于处理压缩和解压缩文件（分别对应不同的算法）

### 文件读写的基本操作

文本文件的写入一般就是三个步骤： 1 创建文件对象 2 写入数据 3 关闭文件对象

打开或创建文件： 变量名=open(filename,mode,encoding)

文件名及路径，模式：读或写，编码格式

操作文件： 变量名.red() 变量名.write(s)

关闭文件： 变量名.close()

【操作】文本写入操作简单测试

f = open(r"a.txt","a")

s = "itbaizhan\nsxt\n"

f.write(s)

f.close()

或者使用with上下文管理：

with open(r"a.txt","a") as f:

s = "itbaizhan\nsxt\n"

f.write(s)

def my\_write():  
 #创建或找开文件  
 file=open('txt/a.txt','w',encoding='utf-8')  
 #写入文件  
 file.write('伟大的中国梦')#写入字符  
 #关闭文档  
 file.close()  
#读取  
def my\_read():  
 # 创建或找开文件  
 file = open('txt/a.txt', 'r', encoding='utf-8')  
 #操作文件  
 s=file.read()  
 print(type(s),s)  
 #关闭文件  
 file.close()  
#主程序运行  
if \_\_name\_\_ == '\_\_main\_\_':  
 # my\_write()#调用函数  
 my\_read()

<class 'str'> 伟大的中国梦

文件的状态，当未操作的时候是存储状态，当被file=open的时候是内存占用状态，当占有用进是不允许其他程序使用的，当file.close（）关闭文件，才会回到存储状态。

#### 文件的基本操作

文件的开打模式

|  |  |
| --- | --- |
| 文件的打开模式 | 模式说明 |
| r | 以只读模式找开，文件指针在文件的开头，如果文件不存在，程序抛异常 |
| rb | 以只读模式打开二进制文件，如图片文件 |
| w | 覆盖写模式，文件不存在创建，文件存在则内容覆盖 |
| wb | 覆盖写入二进制的数据，文件不存在创建，文件存在则内容覆盖 |
| a | 追加写模式，文件不存在创建，文件存在，则在文件最后追加内容 |
| + | 现w/r/a等一同使用，在原功能的基础上增加同时读写功能 |

文件的读写方法

|  |  |
| --- | --- |
| 读写方法 | 描述说明 |
| file.read(size) | 从文件中读取size个字符或字节，如果没有给定参数，则读取文件中的全部内容 |
| file.readline(size) | 读取文件中的一行数据，如果给定参数，则为读取这一行中的size个字符或字节 |
| file.readlines() | 从文件中读取所有内容，结果为列表类型 |
| file.write(s) | 将字符串s写入文件 |
| file.writelines(lst) | 将内容全部为字符串的列表lst写入文件 |
| file.seek(offset) | 改变当前文件操作指针的位置，英文占一个字节，中文gbk编码占有两个字节，utf-8编码占有三个字节 |

def my\_write(s):  
 file=open('txt/b.txt','a',encoding='uft-8')#打开文件追加模式  
 file.write(s)#写入s  
 file.write('\n')#写入换行  
 file.close()#关闭文件  
def my\_write\_list(file,lst):  
 f=open(file,'a',encoding='utf-8')  
 #操作文件  
 f.writelines(lst)  
 f.close()  
  
#主程序运行  
if \_\_name\_\_ == '\_\_main\_\_':#输入main按tab键可以自动补全  
 # my\_write('伟大的中国梦')  
 # my\_write('北京欢迎你')  
 lst=['姓名\t','年龄\t','成绩\n','张三\t','30\t','98']  
 my\_write\_list('txt/c.txt',lst)

txt/c.txt

姓名 年龄 成绩

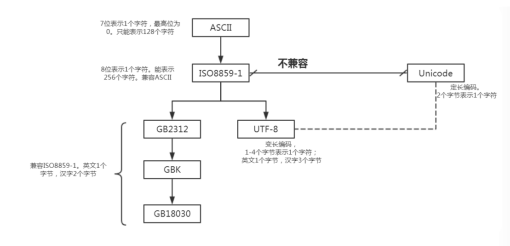
张三 30 98

def my\_read(filename):  
 file=open(filename,'w+',encoding='utf-8')  
 file.write('你好啊')#写入完成后文件的指针会在最后  
 #读取文件，修改字针的位置  
 file.seek(0)  
 # s=file.read()#读取全部内部  
 # s=file.read(2)#读取的是两个字符  
 # s=file.readline()#读取一行  
 # s=file.readline(2)#读取一行中的两个字符:你好  
 # s=file.readlines()#读取所有内容，结果为列表类型:<class 'list'> ['你好啊']  
 #调整文件指针位置  
 file.seek(3)#指针移动到一个汉字以后，utf三个字节一个字  
 s=file.read()#虽然是读取所有，但是把光标调整后把你的这个汉字给跳过了，只显示好啊  
 print(type(s),s)  
 file.close()  
if \_\_name\_\_ == '\_\_main\_\_':  
 my\_read('txt/d.txt')

### 常用编码介绍

在操作文本文件时，经常会操作中文，这时候就经常会碰到乱码问 题。为了让大家有能力解决中文乱码问题，这里简单介绍一下各种 编码之间的关系。

常用编码之间的关系如下：



#### ASCII

全称为 American Standard Code for Information Interchange ，美国信息交换标准代 码，这是世界上最早最通用的单字节编码系统，主要用来显示现代 英语及其他西欧语言。

ASCII 码用7位表示，只能表示128个字符。只定义了27=128个字 符，用7bit即可完全编码，而一字节8bit的容量是256，所以一 字节 ASCII 的编码最高位总是0。

0～31表示控制字符如回车、退格、删除等；32～126表示打印 字符即可以通过键盘输入并且能显示出来的字符； 其中48～57 为0到9十个阿拉伯数字，65～90为26个大写英文字母，97～ 122号为26个小写英文字母，其余为一些标点符号、运算符号 等，具体可以参考 ASCII 标准表（大家自行百度，不在此赘 述）。

#### ISO8859-1

ISO-8859-1 又称 Latin-1 ，是一个8位单字节字符集，它把 ASCII 的最高位也 利用起来，并兼容了 ASCII ，新增的空间是128，但它并没有完全用 完。 在 ASCII 编码之上又增加了西欧语言、希腊语、泰语、阿拉伯语、希 伯来语对应的文字符号，它是向下兼容 ASCII 编码

#### GB2312,GBK,GB18030

GB2312 全称为信息交换用汉字编码字符集，是中国于1980年发布， 主要用于计算机系统中的汉字处理。 GB2312 主要收录了6763个汉 字、682个符号。GB2312 覆盖了汉字的大部分使用率，但不能处理像古汉语等特殊的 罕用字，所以后来出现了像 GBK 、 GB18030 这种编码。GB2312 完全兼容 ISO8859-1

#### GBK

现在最新的内码字集于2000年发布，并于2001年强制执行，包含了 中国大部分少数民族的语言字符，收录汉字数超过70000余个。 它主要采用单字节、双字节、四字节对字符编码，它是向下兼容 GB2312 和 GBK 的，虽然是我国的强制使用标准，但在实际生产中很少 用到，用得最多的反而是 GBK 和 GB2312

#### Unicode

Unicode 编码设计成了固定两个字节，所有的字符都用16位 (2^16=65536)表示，包括之前只占8位的英文字符等，所以会造成 空间的浪费， UNICODE 在很长的一段时间内都没有得到推广应用。 7 Unicode 完全重新设计，不兼容 iso8859-1 ，也不兼容任何其他编码。

#### UTF-8

对于英文字母， unicode 也需要两个字节来表示。所以 unicode 不便于 传输和存储。因此而产生了 UTF编码 ， UTF-8 全称是（ 8-bit Unicode Transformation Format ）。 UTF 编码兼容 iso8859-1 编码，同时也可以用来表示所有语言的字 符，不过， UTF 编码是不定长编码，每一个字符的长度从1-4个字节 不等。其中，英文字母都是用一个字节表示，而汉字使用三个字 节。

❤️老鸟建议 一般项目都会使用 UTF-8 。

#### 中文乱码问题

windows 操作系统默认的编码是 GBK ， Linux 操作系统默认的编码是 UTF8 。当我们用 open() 时，调用的是操作系统打开的文件，默认的编码 是 GBK

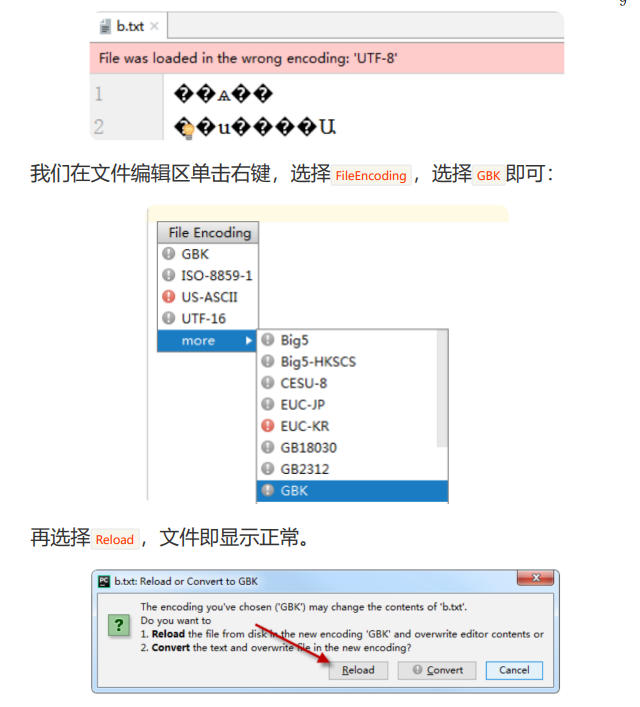
【示例】中文字符文件，乱码出现测试

#测试写入中文

f = open(r"b.txt","w")

f.write("尚学堂\n百战程序员\n")

f.close()运行结果（ Linux 环境中不存在这个问题）:



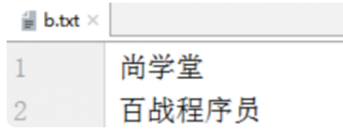
【示例】通过指定文件编码解决中文乱码问题

#测试写入中文

f = open(r"b.txt","w",encoding="utf-8")

f.write("尚学堂\n百战程序员\n")

f.close()

运行结果：

### write()/writelines()写入数据

write(a) ：把字符串 a 写入到文件中

writelines(b) ：把字符串列表写入文件中，不添加换行符

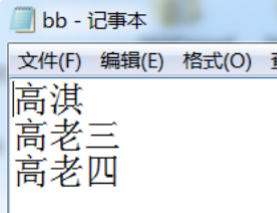
【操作】添加字符串列表数据到文件中

f = open(r"d:\bb.txt","w",encoding="utf-8")

s = ["高淇\n","高老三\n","高老四\n"]

f.writelines(s)

f.close()执行结果：



### close()关闭文件流

由于文件底层是由操作系统控制，所以我们打开的文件对象必须显式调用 close() 方法关闭文件对象。当调用 close() 方法时，首先会把缓冲区数据写入文件(也可以直接调用 flush() 方法)，再关闭文件，释放文件对象。

为了确保打开的文件对象正常关闭，一般结合异常机制的 finally 或者with 关键字实现无论何种情况都能关闭打开的文件对象。

【操作】结合异常机制的 finally ，确保关闭文件对象

try:

f = open(r"my01.txt","a")

s = "gaoqi"

f.write(s)

except BaseException as e:

print(e)

finally:

f.close()

### with语句的使用

with语句：又称上下文管理器，在处理文件时，无论是否产生异常，都能保证with语句执行完毕后关闭已经打开的文件，这个过程是自动的，无需手机操作

语法结构：

with open(…) as file: file是自己起的别名

pass

with语句的使用读写复制关闭操作

def write\_fun():  
 with open('txt/aa.txt','a',encoding='utf-8') as file:  
 file.write('冬奥会')  
def read\_fun():  
 with open('txt/aa.txt', 'r', encoding='utf-8') as file:  
 print(file.read())  
def copy\_fun(src\_file,target\_file):  
 with open(src\_file, 'r', encoding='utf-8') as file:  
 with open(target\_file, 'w', encoding='utf-8') as file2:  
 file2.write(file.read())#将读取的文件直接写入文件  
if \_\_name\_\_ == '\_\_main\_\_':  
 write\_fun()  
 read\_fun()  
 copy\_fun('txt/aa.txt','txt/bb.txt')

结果：生成aa,bb.txt文件

【操作】使用 with 管理文件写入操作

s = ["高淇\n","高老三\n","高老五\n"]

with open(r"d:\bb.txt","w") as f:

f.writelines(s)

### 文本文件的读取

文件的读取一般使用如下三个方法：

1 read([size]) 从文件中读取 size 个字符，并作为结果返回。如果没有 size 参 数，则读取整个文件。 读取到文件末尾，会返回空字符串。

2 readline() 读取一行内容作为结果返回。读取到文件末尾，会返回空字符串

3 readlines() 文本文件中，每一行作为一个字符串存入列表中，返回该列表

【操作】 读取一个文件前4个字符

with open(r"a.txt","r",encoding="utf-8") as f:

print(f.read(4))

【操作】文件较小，一次将文件内容读入到程序中

with open(r"d:\bb.txt","r") as f:

print(f.read())

【操作】按行读取一个文件

with open(r"bb.txt","r") as f:

while True:

fragment = f.readline()

if not fragment:

break

else:

print(fragment,end="")

【操作】使用迭代器（每次返回一行）读取文本文件

with open(r"d:\bb.txt","r") as f:

for a in f:

print(a,end="")

【操作】为文本文件每一行的末尾增加行号

with open("b.txt","r",encoding="utf-8") as f:  
 lines = f.readlines()  
 lines2 = [line.rstrip()+" #"+str(index)+"\n" for index,line in zip(range(1,len(lines)+1),lines)]  
with open("b.txt","w",encoding="utf-8") as f:  
 f.writelines(lines2)

执行前文件内容：

我love u!

尚学堂

百战程序员

执行程序后文件内容：

我love u! #1

尚学堂 #2

百战程序员 #3

### 数据的组织维度

数据的组织方式或存储方式，在python中常用的数据组织方式可分为一维数据、二维数据和高维数据。

**一维数据：**通常采用线性方式组织数据，一般使用python中的列表、元组或者集合进行存储数据。

**二维数据：**称表格数据，由行和列组成，类似于excel表格，在pyhon中使用二维列表进行存储

#一维数据的写入和读取，注意的是写入需要转换，读取需要转换  
def my\_write():  
 #一维数据，可以使用列表，元组，集合  
 lst=['张三','李四','王五']  
 with open('excel/lst.csv','w',encoding='utf-8') as file:  
 file.write(','.join(lst))#列表写不进去，只有转成字符串才能写进去  
def my\_read():  
 with open('excel/lst.csv','r',encoding='utf-8') as file:  
 s=file.read()  
 lst=s.split(',')#将字符串转成列表  
 print(lst)  
  
#二维数据  
def my\_write\_table():  
 lst=[  
 ['商品名称','单价','采购数量'],  
 ['水杯','98.5','20'],#写入的数据要求都是字符串类型  
 ['鼠标', '89', '100'],  
 ]  
 with open('excel/table.csv','w',encoding='utf-8') as file:  
 for item in lst:  
 line=','.join(item)  
 file.write(line)  
 file.write('\n')  
def my\_read\_table():  
 date=[]  
 with open('excel/table.csv','r',encoding='utf-8') as file:  
 lst=file.readlines()#读取的是每一行，是列表中的元素  
 # print(type(lst),lst)  
 for item in lst:  
 new\_lst=item[:len(item)-1].split(',')  
 date.append(new\_lst)  
 print(date)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # my\_write()  
 # my\_read():  
 # my\_write\_table()  
 my\_read\_table()

**高维数据：**高级数据则是使用key-value方式进行组织数据，在python中使用字典进行存储数据。在python中内置的json模块专门用于处理JSON（javaScript Object Notation）格式的数据

#### json模块的常用函数

|  |  |
| --- | --- |
| 函数名称 | 描述说明 |
| json.dumps(obj) | 将python数据类型转成json格式过程，编码过程 |
| json.loads(s) | 将json格式字符串转成python数据类型，解码过程 |
| json.dump(obj,file) | 与dumps()功能相同，将转换结果存储在文件file中 |
| json.load(file) | 与loads()功能相同，从文件file中读入数据 |

import json  
#准备数据  
lst=[  
 {'name':'gentel','age':20,'score':90},  
 {'name':'tom','age':21,'score':99},  
 {'name':'jeke','age':19,'score':89}  
]  
print(lst)  
#编码  
s=json.dumps(lst,ensure\_ascii=False,indent=4)#ensure\_ascii=False正常显示为中文、indent=4增加数据的缩进使json格式的字符串更有可读性  
print(type(s))#编码的过程，将字典转成了字符串类型  
print(s)  
#解码  
lst2=json.loads(s)  
print(type(lst2))  
print(lst2)  
  
#编码到文件中  
with open('txt/高维数据的存储.txt','w') as file:  
 json.dump(lst,file,ensure\_ascii=False,indent=4)  
  
#解码到程序  
with open('txt/高维数据的存储.txt','r') as file:  
 lst3=json.load(file)  
 print(type(lst3))  
 print(lst3)

[{'name': 'gentel', 'age': 20, 'score': 90}, {'name': 'tom', 'age': 21, 'score': 99}, {'name': 'jeke', 'age': 19, 'score': 89}]

<class 'str'>

[

{

"name": "gentel",

"age": 20,

"score": 90

},

{

"name": "tom",

"age": 21,

"score": 99

},

{

"name": "jeke",

"age": 19,

"score": 89

}

]

<class 'list'>

[{'name': 'gentel', 'age': 20, 'score': 90}, {'name': 'tom', 'age': 21, 'score': 99}, {'name': 'jeke', 'age': 19, 'score': 89}]

<class 'list'>

[{'name': 'gentel', 'age': 20, 'score': 90}, {'name': 'tom', 'age': 21, 'score': 99}, {'name': 'jeke', 'age': 19, 'score': 89}]

### 目录与文件的相关操作

#### os模块

python内置的与操作系统文件相关的模块，该模块中语句的执行结果通常与操作系统有关，即有些函数的运行效果在windows操作系统和macOS系统中不一样。

|  |  |
| --- | --- |
| 函数名称 | 描述说明 |
| getcwd() | 获取当前的工作路径 |
| listdir(path) | 获取path路径下的文件和目录信息，如果没有指定path,则获取当前路径下的文件和目录信息 |
| mkdir(path) | 在指定路径下创建目录（文件夹） |
| makedirs(path) | 创建多级目录 |
| rmdir(path) | 删除path下的空目录 |
| removedirs(path) | 删除多级目录 |
| chdir(path) | 把path设置为当前目录 |
| walk(path) | 遍历目录树，结果为元组，包含所有路径名、所有目录列表和文件列表 |
| remove(path) | 删除path指定的文件 |
| rename(old,new) | 将old重名名为new |
| stat(path) | 获取path指定的文件信息 |
| startfile(path) | 启动path指定的文件 |

import os  
# print("当前的工作路径",os.getcwd())  
# print("当前路径下的所有目录及文件",os.listdir())  
# print("指定的目录所有的文件",os.listdir('txt'))  
#创建目录  
# os.mkdir('case')#如果创建的文件夹存在程序报错  
#创建多级目录  
# os.makedirs('case/a/b')  
#删除path下的空目录  
# os.rmdir('case/a/b')  
#删除多级目录  
# os.removedirs('./case/a')  
# 把path设置为当前目录  
#os.chdir('txt')#改完成以，再写代码就是会时入txt这个文件夹  
# 遍历目录树，结果为元组，包含所有路径名、所有目录列表和文件列表，相当于递归操作  
# for dirs,dirlst,filelst in os.walk('F:/gentel/learn/system/Python file'):  
# print(dirs)  
# print(dirlst)  
# print(filelst)  
# print('\*'\*50)  
#删除指定的文件  
# os.remove('./AAA.py')  
# 将old重名名为new  
# os.rename('aaa.py','bbb.py')  
# 获取path指定的文件信息,文件的操作时间信息，所以需要用时间函数翻译  
import time  
def date\_format(longtiem):  
 s=time.strftime('%Y-%m-%d',time.localtime(longtiem))  
 return s  
  
i=os.stat('./txt/京东iPad 11英寸 A16芯片2025年款用户评价.txt')  
print(type(i),i)  
print('最近的一次访问时间：',date\_format(i.st\_atime))  
print('文件的创建时间：',date\_format(i.st\_ctime))  
print('文件的大小：',i.st\_size)  
  
# 启动path指定的文件  
# os.startfile('cmd.exe')#windows+r，输入命令可以找开的程序  
#启动python.exe  
os.startfile(r'C:\Users\KCP2110\AppData\Local\Programs\Python\Python313\python.exe')#\是python当中的提示符前面加个r让所有\提示符失效

<class 'os.stat\_result'> os.stat\_result(st\_mode=33206, st\_ino=844424930144293, st\_dev=8408279064841096156, st\_nlink=1, st\_uid=0, st\_gid=0, st\_size=27834, st\_atime=1744613139, st\_mtime=1744423548, st\_ctime=1744423548)

最近的一次访问时间： 2025-04-14

文件的创建时间： 2025-04-12

文件的大小： 27834

#### path模块

是os模块的子模块，也提供了一些目录或文件的操作函数

|  |  |
| --- | --- |
| 函数名称 | 描述说明 |
| abspath(path) | 获取目录或文件的绝对路径 |
| exists(path) | 判断目录或文件在磁盘上是否存在，结果为bool类型，如果目录或文件在磁盘上存在，结果为True，否则为False |
| join(path,name) | 将目录与目录名或文件名进行拼接，相当于字符串的“+”操作 |
| splitext() | 分别获取文件名和后缀名 |
| basename(path) | 从path中提取文件名 |
| dirname(path) | 从path中提取路径（不包含文件名） |
| isdir(path) | 判断path是否是有效路径 |
| isfile(path) | 判断file是否是有效文件 |

import os.path  
print('获取目录或文件的绝对路径:',os.path.abspath('./bbb.py'))  
print('判断目录或文件在磁盘上是否存在:',os.path.exists('./bbb.py'))  
print('判断目录或文件在磁盘上是否存在:',os.path.exists('./txt'))  
print('将目录与目录名或文件名进行拼接:',os.path.join('F:\gentel\learn\system\Python file\learning','bbb.py'))  
print('分别获取文件名和后缀名:',os.path.splitext('bbb.py'))  
print('从path中提取文件名:',os.path.basename(r'F:\gentel\learn\system\Python file\learning\bbb.py'))  
print('从path中提取路径（不包含文件名）:',os.path.dirname(r'F:\gentel\learn\system\Python file\learning\bbb.py'))  
print('判断path是否是有效路径:',os.path.isdir(r'F:\gentel\learn\system\Python file\learning'))  
print('判断file是否是有效文件:',os.path.isfile(r'F:\gentel\learn\system\Python file\learning\bbb.py'))

获取目录或文件的绝对路径: F:\gentel\learn\system\Python file\learning\bbb.py

判断目录或文件在磁盘上是否存在: True

判断目录或文件在磁盘上是否存在: True

将目录与目录名或文件名进行拼接: F:\gentel\learn\system\Python file\learning\bbb.py

分别获取文件名和后缀名: ('bbb', '.py')

从path中提取文件名: bbb.py

从path中提取路径（不包含文件名）: F:\gentel\learn\system\Python file\learning

判断path是否是有效路径: True

判断file是否是有效文件: True

### 实战

#### 实战一 批量创建文件

import random  
import os  
import os.path  
  
#函数编程  
def create\_filename():  
 filename\_list=[]  
 lit=['水果','蔬菜','烟酒','肉蛋','粮油']  
 code=['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']  
 for i in range(1,13):#创建12个文件  
 filename=''  
 if i<10:#小于10的文件名前面加一个0  
 filename+='0'+str(i)  
 else:  
 filename+=str(i)  
 #物资的类别  
 filename+= '\_'+random.choice(lit)  
 #拼接识别码  
 s=''  
 for item in range(9):  
 s+=random.choice(code)  
 filename+='\_'+s  
 filename\_list.append(filename)  
 return filename\_list  
#创建文件  
def create\_file(filename):  
 with open(filename,'w') as file:  
 pass  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 指定创建文件的目录  
 path='./txt/批量创建文件'  
 # 创建批量创建的目录  
 if not os.path.exists(path):  
 os.mkdir(path)  
 lst=create\_filename()#获取文件名  
  
 for item in lst:#批量生成文件  
 create\_file(os.path.join(path,item)+'.txt')  
 # print(create\_filename())

#### 实战二 批量创建文件夹

import os  
import os.path  
  
def mkdirs(path,num):  
 for item in range(1,num+1):#批量创建  
 os.mkdir(path+'/'+str(item))#他奸文件夹  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 path='./mkdir'  
 if not os.path.exists(path):  
 os.mkdir(path)  
 num=eval(input("请输入创建目录的个数："))  
 mkdirs(path,num)

#### 实战三 记录用户登录日志并查看

import time  
def show\_info():  
 print('输入提示数字，执行相应的操作：0.退出 1.查看登陆日志')  
  
#记录日志  
def write\_loginfo(username):  
 with open('./txt/log.txt','a',encoding='utf-8') as file:  
 s=f'用户名：{username},登陆时间：{time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(time.time()))}'  
 file.write(s)  
 file.write('\n')  
  
#读取日志的操作  
def read\_loginfo():  
 with open('./txt/log.txt','r',encoding='utf-8') as file:  
 while True:  
 line=file.readline()  
 if line == '':  
 break  
 else:  
 print(line)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # write\_loginfo('admin')  
 username=input('请输入用户名：')  
 pwd=input('请输入密码：')  
 if username=='admin' and pwd=='admin':  
 print('登陆成功')  
 #将登陆信息写入登陆文件  
 write\_loginfo(username)  
 #提示用户操作  
 show\_info()  
 num=eval(input('请输入要操作的数字：'))  
 while True:  
 if num==0:  
 print('退出成功！')  
 break  
 elif num==1:  
 print('查看登陆日记')  
 read\_loginfo()  
 show\_info()  
 else:  
 print("输入的数量有误")  
 show\_info()  
 num = eval(input('请输入要操作的数字：'))  
 else:  
 print('用户名或密码不正确！')

请输入用户名：admin

请输入密码：admin

登陆成功

输入提示数字，执行相应的操作：0.退出 1.查看登陆日志

请输入要操作的数字：1

查看登陆日记

用户名：admin,登陆时间：2025-04-18 14:21:47

输入提示数字，执行相应的操作：0.退出 1.查看登陆日志

请输入要操作的数字：0

退出成功！

#### 实战四 淘宝自动回复

def find\_answer(question):  
 with open('./txt/automatic\_reply.txt','r',encoding='utf-8') as file:  
 while True:  
 line=file.readline()  
 if line=='':  
 break  
 # 分割字符串  
 parts = line.split('|')  
 if len(parts) > 1:  
 keyword = parts[0]  
 reply = parts[1]  
 if keyword in question:  
 return reply  
 return False  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 question=input("Hi.XXX你好，小蜜在此等主人很长时间了，有什么麻烦和小蜜说说吧：")  
 while True:  
 if question=='BEY':  
 break  
 else:  
 reply=find\_answer(question)#返回值有两种可能找到答案为True，找不到为Flase  
 if reply==False:#没有答案  
 question=input('小蜜不知道您在说什么，您可以问我一些，订单、物流、账户、支付方面的问题，退出请输BEY：').upper()  
 else:  
 print(reply)  
 question=input('小主您还可以问一些关于订单、物流、账户、支付方面的问题，退出请输BEY：').upper()  
 print('小主再见')

Hi.XXX你好，小蜜在此等主人很长时间了，有什么麻烦和小蜜说说吧：顶替

小蜜不知道您在说什么，您可以问我一些，订单、物流、账户、支付方面的问题，退出请输BEY：我的钱支付多了怎么办

如果您有任何支付问题，可以联系支付宝客服，QQ：XXXXXXXX

小主您还可以问一些关于订单、物流、账户、支付方面的问题，退出请输BEY：bey

小主再见

## 网络编程

### 网络编程的基本概念

#### 网络编程与通信协议

**通信协议：**就就是规则，为了使全世界不同类型的计算机可以连接起来，所以制定了全球通用的internet协议，只要支持这个协议，就可以接入互联网。

层次关系图

### 了解七层协议与四层协议

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 四层协议 | 七层协议 | POP3 |  |  |
| 应用层 | 应用层 | FTP HTTP | DHCP TFTP |
| 表示层 | Telnet SMTP | SNMP DNS |
| 传输层 | 会话层 | TCP |  | UDP |
| 传输层 |
| 网际层 | 网络层 | IP ICMP | IGMP | ARP RARP |
| 网络接口层 | 数据链路层 | CSMA/CD | TokingRing |  |
| 物理层 |  |  |  |

### 掌握TCP/IP协议

#### IP协议

IP协义是整个TCP/IP协议族的核心

IP地址就是互联网上计算机的唯一标识

目前的IP地址有两种表示方式，即IPV4和IPV6（ipv4十进制的点分制，ipv6是十六进制冒号分隔的）

在命令行下使用ipconifg命令可以查看本机的ip地址

#### TCP协议

TCP(Transmission Control Protocol)协议即传输控制协议，是建立在IP协议基础之上的。TCP协议负责两台计算机之间建立可靠连接，保证数据包按顺序发送到。它是一种可靠的、一对一的、面向有连接的通信协议

#### TCP/IP协议中的四个层次

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TCP/IP协议 | | 右  图  上  到  下 |  |  |  | 用户数据 |  |
| 应用层 | 应用程序 |  |  | APPI首部 | 用户数据 |  |
| 传输层 | TCP |  |  | TCP首部 | 应用数据 |  |
| 网际层 | IP |  | IP首部 | TCP首部 | 应用数据 |  |
| 网络接口层 | 以太网驱动 | 以太网首部 | IP首部 | TCP首部 | 应用数据 | 以太网尾部 |

#### TCP/IP协议数据发送和数据接收

发送方，装包打包

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 发  送  方 | 应用层 | HTTP数据 |  |  |  | HTTP报文 | 从  上  往  下 |
| 传输层 | HTTP数据 | TCP首部 |  |  | TCP首部 |
| 网络层 | HTTP数据 | TCP首部 | IP首部 |  | IP数据包 |
| 链路层 | HTTP数据 | TCP首部 | IP首部 | 以太网首部 | 网络架构 |

接收方，拆包

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 接收  方 | 应用层 | HTTP数据 |  |  |  | HTTP报文 | 从  下  往  上 |
| 传输层 | HTTP数据 | TCP首部 |  |  | TCP首部 |
| 网络层 | HTTP数据 | TCP首部 | IP首部 |  | IP数据包 |
| 链路层 | HTTP数据 | TCP首部 | IP首部 | 以太网首部 | 网络架构 |

#### TCP协议的三次握手

TCP client客户端 SYN 1(同步信号) TCP server服务器

TCP server服务器 SYN K/ACK（同步+确认） J+1 TCP client客户端

TCP client客户端 ACK K+1（确认）

### 掌握UDP协议

UDP协议又被称为用户数据包协议（user datagram protocol）,它是面向无连接的协议，只要知道对方的ip地址和端口，就可以直接发送数据包，由于是面向无连接的，所以无法保证数据一定会到达接收方。

TCP可以比喻成打电话，请求连接，对方没有接听是无法连接的

UDP可以比喻成发短信，不需要知道对方的的机号码直接发就可以了，但无法保证数据能发的过去。

#### 端口号

区分计算机中的运行的应用程序的整数

端口号的取值范围是0到65535，一共65536个，其中80这个端口号分配给了http服务，21这个端口号分配给了ftp服务

如MYSQL的端口号是3306，oracle数据库1521，SQL Server数据库1433

#### TCP协议与UDP协议的区别

|  |  |  |
| --- | --- | --- |
|  | TCP协议 | UDP协议 |
| 连接方面 | 面向连接的 | 面向无连接 |
| 安全方面 | 传输消息可靠，不丢失，按顺序到达 | 无法保证不丢包 |
| 传输效率方面 | 传输效率相对较低 | 传输效率高 |
| 连接对象数量方面 | 只能是点对点、一对一 | 支持一对一，一对多、多对多的交互通信 |

### 掌握Socket套接字

Socket是描述ip地址和端口号的

#### socket对象的常用方法

|  |  |
| --- | --- |
| 方法名称 | 功能描述 |
| bind(ip,port) | 绑定ip地址和端口 |
| listen(N) | 开始TCP监听，N表示操作系统挂起最大连接数量，取值范围1-5之间，一般设置为5 |
| accept() | 被动接收TCP客户端连接，阻塞式 |
| connect((ip,port)) | 主动初始化TCP服务器连接 |
| recv(size) | 接收TCP数据，返回值为字符串类型，size表示要接收的最大数据量 |
| send(str) | 发送tcp数据，返回值是要发送的字节数量 |
| sendall(str) | 完整发送TCP数据，将str中的数据发送到连接的套接字，返回之前尝试发送所有数据，如果成功为None,失败抛出异常 |
| recvfrom() | 接收UDP数据，返回值为一个元组（data.address）,data表示接收的数据，address表示发送数据的套接字地址 |
| sendto(data,(ip,port)) | 发送UDP数据，返回值是发送的字节数 |
| close() | 关闭套接字 |

### 应用TCP编程

#### TCP服务器端流程如下：

1. 使用socket类创建一个套接字对象
2. 使用bind(（ip,port）)方法绑定ip地址和端口号，这IP 地址和端口号是元组数据
3. 使用listen()方法开始TCP监听
4. 使用accept()方法等待客户端的连接
5. 使用recv()/send()方法接收/发送数据
6. 使用close()关闭套接字
7. #第一步创建socket对象  
   server\_socket=socket(AF\_INET,SOCK\_STREAM)  
   #第二步绑定ip地址和端口号  
   ip='199.1.8.41'#本机的ip  
   port=80#自定义的端口  
   server\_socket.bind((ip,port))#绑定了ip地址和端口号  
   #第三步开始TCP监听  
   server\_socket.listen(5)#5代表排队数量  
   print('服务器已启动')  
   #第四步等等客户端的连接  
   client\_socket,client\_addr=server\_socket.accept()#客户端的发送内容,系列解包赋值  
   #第五步接收来自客户端的数据  
   data=client\_socket.recv(1024)#接收1024个数据  
   print('客户端发送过来的数据：',data.decode('utf-8'))#这里给date数据做了个转码，因为客户端发过来有编码  
   #第六步关闭socket  
   client\_socket.close()  
   server\_socket.close()

#### TCP客户端代码的编写

1. 使用socket类创建一个套接字对象
2. 使用connect((host,port))设置连接的主机ip和主机设置的端口号
3. 使用recv()/send()方法接收/发送数据
4. 使用close()关闭套接字

import socket  
from base64 import encode  
  
#创建socket对象  
client\_socket=socket.socket()  
#连接哪个主机  
ip='199.1.8.41'  
port=80  
client\_socket.connect((ip,port))  
print('.........与服务器的连接成功............')  
#发送数据  
client\_socket.send('welcome to python world'.encode('utf-8'))  
#闭socket  
client\_socket.close()  
print('发送完毕')

#### TCP编程服务器端的多次通信

import socket  
socket\_obj=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)#创建对象  
socket\_obj.bind(('199.1.8.41',80))#绑定ip  
socket\_obj.listen(5)#设置最大的连接数量  
clinet\_socket,clinet\_addr=socket\_obj.accept()#等待客户端的连接  
print('服务器已启动等待连接')  
#接收数据  
info=clinet\_socket.recv(1024).decode('utf-8')#初始化变量  
while info!='q':#循环体  
 if info!='':  
 print('接收到的数据是：',info)  
 #准备发送的数据  
 data=input('请输入要发送的数据关闭通信请输入Q：')  
 #服务器回复客户端  
 clinet\_socket.send(data.encode('utf-8'))  
 if data.lower()=='q':  
 break  
 info = clinet\_socket.recv(1024).decode('utf-8')#改变变量  
  
#关闭socket对象  
clinet\_socket.close()  
socket\_obj.close()

#### TCP编程服务器端的流程

import socket  
#创建socket对象  
clinet\_socket=socket.socket()  
clinet\_socket.connect(('199.1.8.41',80))#连接ip  
print('..............已连接..................')  
#客户端发送数据  
info=''  
while info!='q':  
 #准备发送的数量  
 send\_data=input("请输入要发送的数据退了请输入q:")  
 clinet\_socket.send(send\_data.encode('utf-8'))  
 if send\_data=='q':  
 break  
 #从服务器端接收数据  
 info=clinet\_socket.recv(1024).decode('utf-8')  
 print('收到的数据：',info)  
clinet\_socket.close()#关闭对象

### 应用UDP编程

客户端：socket>>sendto()数据请求>>recvfrom()数据响应>>close()

服务器：socket>>bind(ip,port)>>recvfrom()阻塞直到客户端连接传达>>sendto()>>close()

#### UDP发送方

from socket import socket,AF\_INET,SOCK\_DGRAM  
#一、create obj  
send\_socket=socket(AF\_INET,SOCK\_DGRAM)  
#二、准备send data  
data=input('请输入要发送的数据：')  
#三、接收方ipAddress  
ip\_port=('199.1.8.41',80)  
#四、send data  
send\_socket.sendto(data.encode('utf-8'),ip\_port)  
#五、接收方的回复数据  
recv\_data,addr=send\_socket.recvfrom(1024)  
print('接收到的数据为：',recv\_data.decode('utf-8'))  
#close socket  
send\_socket.close()

#### UDP接收方

from socket import socket,AF\_INET,SOCK\_DGRAM  
#1.创建socket对象  
recv\_socket=socket(AF\_INET,SOCK\_DGRAM)  
#2.绑定ip地址和端口  
recv\_socket.bind(('199.1.8.41',80))  
#3.接收来自发送方的数据  
recv\_data,addr=recv\_socket.recvfrom(1024)  
print('接收到的数据为',recv\_data.decode('utf-8'))  
#4.准备回复对方的数据  
data=input('请输入要回复的数据：')  
#5.回复  
recv\_socket.sendto(data.encode('utf-8'),addr)  
#6.关闭  
recv\_socket.close()

### 实战：UDP模拟客服咨询小程序

服务端

from socket import socket,AF\_INET,SOCK\_DGRAM  
#1.创建socket对象  
recv\_socket=socket(AF\_INET,SOCK\_DGRAM)  
#2.绑定IP和端口  
recv\_socket.bind(('199.1.8.41',666))  
while True:  
 #3.接收数据  
 recv\_data,addr=recv\_socket.recvfrom(1024)  
 print('客户说：',recv\_data.decode('utf-8'))  
 if recv\_data.decode('utf-8')=='q':  
 break  
 #准备回得对方的信息  
 data=input('回复信息：')  
 #回复  
 recv\_socket.sendto(data.encode('utf-8'),addr)  
#关闭  
recv\_socket.close()

客户端

from socket import socket,AF\_INET,SOCK\_DGRAM  
send\_socket=socket(AF\_INET,SOCK\_DGRAM)  
#开始多次通信  
while True:  
 #准备发送的数据  
 data=input('请输入发送信息输入q关闭通话：')  
 #发送  
 send\_socket.sendto(data.encode('utf-8'),('199.1.8.41',8080))  
 if data.lower()=='q':  
 break  
 #接收来自客服人员的回得信息  
 recv\_data,addr=send\_socket.recvfrom(1024)  
 print('客服说：',recv\_data.decode('utf-8'))  
#关闭  
send\_socket.close()

## 程序与进程的概念

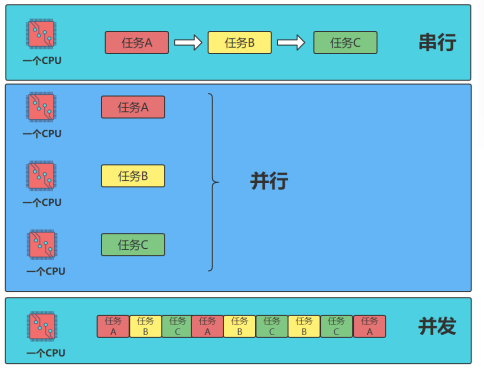
### 什么是程序与进程

程序：program，指一系列有序指令的集合，使用编程语言所编写，用于实现一定的功能

程序是安装在来硬盘当中，进程是程序在运行状态

### 并发编程介绍

#### 串行、并行与并发的区别



1 串行(serial)：一个CPU上，按顺序完成多个任务

2 并行(parallelism)：指的是任务数小于等于cpu核数，即任务真的是一起执行的

3并发(concurrency)：一个CPU采用时间片管理方式，交替的处理多个任务。一般是是任务数多余cpu核数，通过操作系统的各种任务调度算法，实现用多个任务“一起”执行（实际上总有一些任务不在执行，因为切换任务的速度相当快，看上去一起执行而已）

#### 进程、线程、协程的区别

乔布斯想开工厂生产手机，费劲力气，制作一条生产线，这个生产线上有很多的器件以及材料。一条生产线就是一个进程。

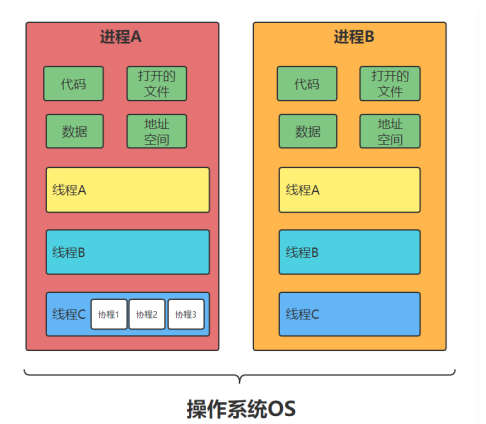
只有生产线是不够的，所以找五个工人来进行生产，这个工人能够利用这些材料最终一步步的将手机做出来，这五个工人就是五个线程。

为了提高生产率，想到3种办法：

1 一条生产线上多招些工人，一起来做手机，这样效率是成倍増长，即单进程多线程方式

2 多条生产线，每个生产线上多个工人，即多进程多线程

3乔布斯深入一线发现工人不是那么忙，有很多等待时间。于是规定：如果某个员工在等待生产线某个零件生产时 ，不要闲着，干点其他工作。也就是说：如果一个线程等待某些条件，可以充分利用这个时间去做其它事情，这就是：协程方式。



1 线程是程序执行的最小单位，而进程是操作系统分配资源的最小单位；

2 一个进程由一个或多个线程组成，线程是一个进程中代码的不同执行路线；

3进程之间相互独立，但同一进程下的各个线程之间共享程序的内存空间(包括代码段、数据集、堆等)及一些进程级的资源(如打开文件和信号)，某进程内的线程在其它进程不可见；

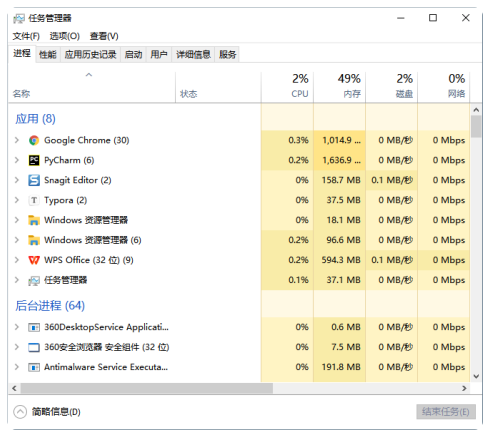
4 调度和切换：线程上下文切换比进程上下文切换要快得多。

##### 进程是什么？

进程（Process）是一个具有一定独立功能的程序关于某个数据集合的一次运行活动

现代操作系统比如Mac OS X，Linux，Windows等，都是支持 “多任务”的操作系统，叫“多任务”呢？简单地说，就是操作系统 可以同时运行多个任务。打个比方，你一边在用逛淘宝，一边 在听音乐，一边在用微信聊天，这就是多任务，至少同时有3个任务正在运行。还有很多任务悄悄地在后台同时运行着，只是 桌面上没有显示而已。

对于操作系统来说，一个任务就是一个进程（Process），比如打开 一个浏览器就是启动一个浏览器进程，就启动了一个记事本进程， 打开两个记事本就启动了两个记事本进程，打开一个Word就启动了 一个Word进程。



##### 线程是什么？

线程（Thread）是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。

有些进程还不止同时干一件事，比如微信，它可以同时进行打字聊天，视频聊天，朋友圈等事情。在一个进程内部，要同时干多件事，就需要同时运行多个“子任务”，我们把进程内的这些 “子任务”称为线程（Thread）。

并发编程解决方案：

多任务的实现有3种方式：

1 多进程模式

2 多线程模式

3 多进程+多线程模式

启动多个进程，每个进程虽然只有一个线程，但多个进程可以一块执行多个任务

启动一个进程，在一个进程内启动多个线程，这样，多个线程也可以一块执行多个任务

启动多个进程，每个进程再启动多个线程，这样同时执行的任务就更多了，当然这种模型更复杂，实际很少采用。

##### 协程是什么？

协程，Coroutines，也叫作纤程(Fiber)，是一种在线程中，比线程更加轻量级的存在，由程序员自己写程序来管理。

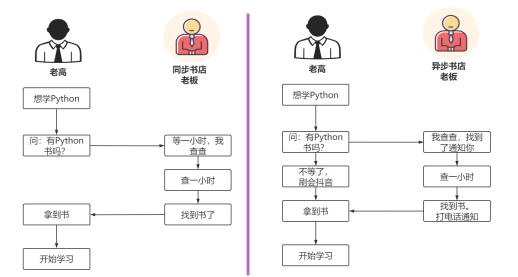
当出现IO阻塞时，CPU一直等待IO返回，处于空转状态。这时候用协程，可以执行其他任务。当IO返回结果后，再回来处理数据。充分利用了IO等待的时间，提高了效率。

#### 同步和异步介绍

同步和异步强调的是消息通信机制 (synchronous communication/ asynchronous communication)。

同步(synchronous)：A调用B，等待B返回结果后，A继续执行

异步(asynchronous )：A调用B，A继续执行，不等待B返回结果；B 有结果了，通知A，A再做处理。



### 创建进程的方式

#### 第一种函数式创建进子进程

from multiprocessing import Process

创建进程的语法结构：process(group=None,target，name,args,kwargs)

参数说明：

1. group:表示分组，实际上不使用，值默认为None即可
2. target:表示子进程要执行的任务，支持函数名
3. name:表示子进程的名称
4. args:表示调用函数的位置参数，以元组的形式笔传递
5. kwargs:表示调用函数的关键字参数，以字典的形式进行传递

import os  
import time  
from multiprocessing import Process  
  
#我的进程要执行的代码  
def test():  
 print(f'我是子进程，我的pip是：{os.getpid()},我的父进程是：{os.getpid()}')  
 time.sleep(1)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('主进程开始执行')  
 lis=[]  
 #创建五个子进程  
 for i in range(5):  
 #创建子进程  
 p=Process(target=test)  
 #启动子进程  
 p.start()  
 #将启动中的进程加入列表当中  
 lis.append(p)  
 #遍历列表，lis中有个子进程，当子进程执行完毕后，再结束主进程  
 for item in lis:  
 item.join()#阻塞主进程  
 print('主进程执行结束')

主进程开始执行

我是子进程，我的pip是：14112,我的父进程是：14112

我是子进程，我的pip是：9144,我的父进程是：9144

我是子进程，我的pip是：14044,我的父进程是：14044

我是子进程，我的pip是：14912,我的父进程是：14912

我是子进程，我的pip是：10452,我的父进程是：10452

主进程执行结束

##### process类的方法

|  |  |
| --- | --- |
| 方法/属性名称 | 功能描述 |
| name | 当前进程实例别名，默认为process-N |
| pid | 当前进程对象的pid值 |
| is\_alive() | 进程是否执行完，没执行完结果为True，否则为False |
| join(timeout) | 等待结束或等待timeout秒 |
| start() | 启动进程 |
| run() | 如果没有指定target参数，则启动进程后，会调用父类中的run方法 |
| terminate() | 强制终止进程 |

###### process类常的方法1.py

from multiprocessing import Process  
import os,time  
#函数调用子进程  
def sub\_process(name):  
 print(f'子进程的PID：{os.getpid()},父进程的PID：{os.getppid},..........{name}')  
 time.sleep(1)  
def sub\_process2(name):  
 print(f'子进程的PID：{os.getpid()},父进程的PID：{os.getppid},..........{name}')  
 time.sleep(1)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 #主进程  
 print('父进程开始执行')  
 for i in range(5):  
 # 创建第一个子进程  
 p1=Process(target=sub\_process,args=('ysj',))  
 # 创建第二个子进程  
 p2 = Process(target=sub\_process2, args=(18,))  
 #用start启动  
 p1.start()  
 p2.start()  
 print(p1.name,'是否执行完毕',p1.is\_alive())  
 print(p2.name,'是否执行完毕',p2.is\_alive())  
  
 print(p1.name,'PID是：',p1.pid)  
 print(p2.name,'PID是：',p2.pid)  
  
 p1.join()#主程序要等待p1执行结束  
 p2.join()#主程序要等待p2执行结束  
  
 print('父进程执行完毕')

C:\Users\KCP2110\AppData\Local\Programs\Python\Python313\python.exe "F:\gentel\learn\system\Python file\learning\process类常的方法1.py"

父进程开始执行

Process-1 是否执行完毕 True

Process-2 是否执行完毕 True

Process-1 PID是： 13880

Process-2 PID是： 13160

子进程的PID：13880,父进程的PID：<built-in function getppid>,..........ysj

子进程的PID：13160,父进程的PID：<built-in function getppid>,..........18

Process-3 是否执行完毕 True

Process-4 是否执行完毕 True

Process-3 PID是： 6720

Process-4 PID是： 3568

子进程的PID：6720,父进程的PID：<built-in function getppid>,..........ysj

子进程的PID：3568,父进程的PID：<built-in function getppid>,..........18

Process-5 是否执行完毕 True

Process-6 是否执行完毕 True

Process-5 PID是： 14104

Process-6 PID是： 12400

子进程的PID：14104,父进程的PID：<built-in function getppid>,..........ysj

子进程的PID：12400,父进程的PID：<built-in function getppid>,..........18

Process-7 是否执行完毕 True

Process-8 是否执行完毕 True

Process-7 PID是： 14472

Process-8 PID是： 10988

子进程的PID：14472,父进程的PID：<built-in function getppid>,..........ysj

子进程的PID：10988,父进程的PID：<built-in function getppid>,..........18

Process-9 是否执行完毕 True

Process-10 是否执行完毕 True

Process-9 PID是： 14340

Process-10 PID是： 13916

子进程的PID：14340,父进程的PID：<built-in function getppid>,..........ysj

子进程的PID：13916,父进程的PID：<built-in function getppid>,..........18

父进程执行完毕

进程已结束，退出代码为 0

###### process类常的方法2.py

from multiprocessing import Process  
import os,time  
#函数调用子进程  
def sub\_process(name):  
 print(f'子进程的PID：{os.getpid()},父进程的PID：{os.getppid},..........{name}')  
 time.sleep(1)  
def sub\_process2(name):  
 print(f'子进程的PID：{os.getpid()},父进程的PID：{os.getppid},..........{name}')  
 time.sleep(1)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('主进程开始执行')  
 for i in range(5):  
 p1=Process(target=sub\_process,args=('jsy',))  
 p2=Process(target=sub\_process,args=(18,))  
  
 p1.start()  
 p2.start()  
  
 #强制终止进程  
 p1.terminate()#子进程刚开始就被结束了，所有没有了进程

p2.terminate()  
 print('主进程执行结束')

主进程开始执行

主进程执行结束

#### 第二种创建进程的语法

class 子进程（process）:

pass

from multiprocessing import Process  
import os,time  
#自定义一个类  
class subprocess(Process):  
 #编写初始化方法  
 def \_\_init\_\_(self,name):  
 super().\_\_init\_\_()  
 self.name=name  
 #重写父类中的run方法  
 def run(self):  
 print(f'子进程的名称：{self.name},PID,{os.getpid()},父进程的PID：,{os.getppid()}')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('父进程开始执行')  
 lst=[]  
 for i in range(1,6):  
 p1=subprocess(f'进程：{i}')  
 #启动进程  
 p1.start()  
 lst.append(p1)  
 #阻塞一下主进程  
 for item in lst:  
 item.join()  
 print('父进程结束')

父进程开始执行

子进程的名称：进程：4,PID,8688,父进程的PID：,3512

子进程的名称：进程：5,PID,13200,父进程的PID：,3512

子进程的名称：进程：2,PID,6864,父进程的PID：,3512

子进程的名称：进程：3,PID,14560,父进程的PID：,3512

子进程的名称：进程：1,PID,13708,父进程的PID：,3512

父进程结束

#### 进程池的使用

创建一个进程池，并设置进程池中最大的进程数量。假设进程池中最大的进程数为3，现在有10个任务需要执行，那么进程池一次可以执行3个任务，4次即可完成全部任务的执行。

创建进程池的语法结构：

进程池对象=Pool(N)

|  |  |
| --- | --- |
| 方法名 | 功能描述 |
| apply\_async(func,args,kwargs) | 使用非阻塞方式调用函数func |
| apply(func,args,kwargs) | 使用阻塞方式调用函数func |
| close() | 关闭进程池，不再接收新任务 |
| terminate() | 不管任务是否完成，立即终止 |
| join() | 阻塞主进程，必须在terminate()或close()之后使用 |

**进程池非阻塞方式执行.py**

from multiprocessing import Pool  
import time,os  
#编写函数  
def task(name):  
 print('子进程的pid',os.getpid(),'执行的任务',name)  
 time.sleep(1)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 #主进程  
 start=time.time()  
 print('父进程开始执行')  
 #创建进程池  
 p=Pool(3)  
 #创建任务  
 for i in range(10):  
 #以非阻塞的方式执行  
 p.apply\_async(func=task,args=(i,))  
 p.close()#关闭进程池  
 p.join()#阻塞一下主进程，子进程结束以后才会执行父进程的代码  
 print('所有的子进程执行完毕，父进程执行结束')  
 print(int(time.time()-start),'秒')

父进程开始执行

子进程的pid 8856 执行的任务 0

子进程的pid 15260 执行的任务 1

子进程的pid 8956 执行的任务 2

子进程的pid 8856 执行的任务 3

子进程的pid 15260 执行的任务 4

子进程的pid 8956 执行的任务 5

子进程的pid 8856 执行的任务 6

子进程的pid 15260 执行的任务 7

子进程的pid 8956 执行的任务 8

子进程的pid 8856 执行的任务 9

所有的子进程执行完毕，父进程执行结束

4 秒

**进程池阻塞方式执行.py**

from multiprocessing import Pool  
import time,os  
#编写函数  
def task(name):  
 print('子进程的pid',os.getpid(),'执行的任务',name)  
 time.sleep(1)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 #主进程  
 start=time.time()  
 print('父进程开始执行')  
 #创建进程池  
 p=Pool(3)  
 #创建任务  
 for i in range(10):  
 #以阻塞的方式执行  
 p.apply(func=task,args=(i,))  
 p.close()#关闭进程池  
 p.join()#阻塞一下主进程，子进程结束以后才会执行父进程的代码  
 print('所有的子进程执行完毕，父进程执行结束')  
 print(int(time.time()-start),'秒')

父进程开始执行

子进程的pid 9328 执行的任务 0

子进程的pid 10228 执行的任务 1

子进程的pid 12608 执行的任务 2

子进程的pid 9328 执行的任务 3

子进程的pid 10228 执行的任务 4

子进程的pid 12608 执行的任务 5

子进程的pid 9328 执行的任务 6

子进程的pid 10228 执行的任务 7

子进程的pid 12608 执行的任务 8

子进程的pid 9328 执行的任务 9

所有的子进程执行完毕，父进程执行结束

10 秒

上面两个案例可以看出非阻塞运行程序时间短一些

#### 并发和并行

**并发：**是指两个或多个事件同一时间间隔发生，多个任务被交替轮换着执行，比如A事件是吃苹果，在吃青苹果的过程中有快递员敲门让你收下快递，收快递就是B事件，那么收完快递继续吃没有吃完的苹果。这就是并发。

**并行:**指两个或多个事件在同一时刻发生，多个任务在同一时刻在多个处理器上同时执行。比如A事件是泡脚，B事件是打电话，C事件是记录电话内容，这三件事则可以在同一时刻发生，这就是并行。

#### 进程之间的通信

**多个进程之间数据是否共享.py**

from multiprocessing import Process  
a=100  
  
def add():  
 print('子进程1开始执行')  
 global a  
 a+=30  
 print('a=',a)  
 print('子进程1执行完毕')  
  
def sub():  
 print('子进程2开始执行')  
 global a  
 a-=50  
 print('a=',a)  
 print('子进程2执行完毕')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("父进程开始执行")  
 print('a=',a)  
 p1=Process(target=add)  
 p2=Process(target=sub)  
 #启进子进程  
 p1.start()  
 p2.start()  
 #阻塞一下主进程  
 p1.join()  
 p2.join()  
 print('父进程执行结束')  
 print('a=',a)  
#说明多个进程之间数据不共享

父进程开始执行

a= 100

子进程1开始执行

a= 130

子进程1执行完毕

子进程2开始执行

a= 50

子进程2执行完毕

父进程执行结束

a= 100

说明：多个进程之间数据不能共享

#### 队列Queue

进程之间可以通过队列（Queue）进行通信，队列是一种先进先出（first in first out）的数据结构

创建队列的语法结构：

队列对象=Queue(N)

|  |  |
| --- | --- |
| 方法名称 | 功能描述 |
| qsize() | 获取当前队列包含的消息数量 |
| empty() | 判断队列是否为空，为空结果为True，否则为False |
| full() | 判断队列是否满了，满结果为True,否则为False |
| get(block=True) | 获取队列中的一条消息，然后从队列中移除，block默认值为True |
| get\_nowait() | 相当于get(blok=False),消息队列为空时，抛出异常 |
| put(item,block=True) | 将item消息放入队列，block默认为True |
| put\_nowait(item) | 相当于put(itme,block=False) |

队列的基本使用

from multiprocessing import Queue  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 #创建一个队列  
 q=Queue(3)#3表示最多可以接受3条信息  
 print('队列是否为空',q.empty())  
 print('队列是否为满',q.full())  
 #向队列中添加消息  
 q.put('hello')  
 q.put('world')  
 print('队列是否为空', q.empty())  
 print('队列是否为满', q.full())  
 q.put('python')  
 print('队列是否为空', q.empty())  
 print('队列是否为满', q.full())  
 print('队列中有多少条消息',q.qsize())  
  
 #出队  
 print(q.get())  
 print('队列中有多少条消息', q.qsize())  
 #入队  
 q.put\_nowait('html')  
 # q.put\_nowait('sql')#输入这一条就报错了，因为队列已满，出了一个，入队两个  
 # q.put('sql')#输入这个不会报错，但是处于等待中，等队列有空位，才可以添加进去  
 #遍历队列中的元素  
 if not q.empty():  
 for i in range(q.qsize()):  
 print(q.get\_nowait())  
 print('队列是否为空：',q.empty())  
 print('队列是否为满：',q.full())  
 print('队列中消息的个数：',q.qsize())

C:\Users\KCP2110\AppData\Local\Programs\Python\Python313\python.exe "F:\gentel\learn\system\Python file\learning\队列的基本使用.py"

队列是否为空 True

队列是否为满 False

队列是否为空 True

队列是否为满 False

队列是否为空 True

队列是否为满 True

队列中有多少条消息 3

hello

队列中有多少条消息 2

队列是否为空： True

队列是否为满： True

队列中消息的个数： 3

进程已结束，退出代码为 0

#### 向已满的队列中添加元素

from multiprocessing import Process  
from queue import Queue  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 q=Queue(3)  
 #向队列中添加元素  
 q.put('hello')  
 q.put('world')  
 q.put('python')  
  
 q.put('html',block=True,timeout=2)#等2秒后还没有空位置就抛出异常

File "F:\gentel\learn\system\Python file\learning\向已满的队列中添加元素.py", line 11, in <module>

q.put('html',block=True,timeout=2)#等2秒后还没有空位置就抛出异常

~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

File "C:\Users\KCP2110\AppData\Local\Programs\Python\Python313\Lib\queue.py", line 172, in put

raise Full

queue.Full

#### 使用队列实现进程之间的通信

from multiprocessing import Queue,Process  
import time  
  
a=100  
def write\_msg(q): #q表示要操作的队列  
 global a  
 if not q.full():  
 for i in range(6):  
 a-=10  
 q.put(a)  
 print('a入队的值：',a)  
  
def read\_msg(q):  
 time.sleep(1)  
 while not q.empty():  
 print('出队时a的值：',q.get(a))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('父进程开始执行')  
 q=Queue()#同父进程创建队列，没有批定说明你指定的消息个数没有上限  
 #创建两个进程一个负责入队一个负责出队  
 p1=Process(target=write\_msg(q,))  
 p2=Process(target=read\_msg(q,))  
 #启动两个子进程  
 p1.start()  
 p2.start()  
 #等待写的进程结束后，再去执行主进程  
 p1.join()  
 p2.join()  
 print('...............父进程执行完毕..................')

父进程开始执行

a入队的值： 90

a入队的值： 80

a入队的值： 70

a入队的值： 60

a入队的值： 50

a入队的值： 40

出队时a的值： 90

出队时a的值： 80

出队时a的值： 70

出队时a的值： 60

出队时a的值： 50

出队时a的值： 40

...............父进程执行完毕..................

### 创建线程的方式

Python的标准库提供了两个模块： \_thread 和 threading ， \_thread 是低级 模块， threading 是高级模块，对 \_thread 进行了封装。绝大多数情况 下，我们只需要使用 threading 这个高级模块。

线程是CPU可调度的最小单们，被包含在进程中，是进程中实际的动作单位。一个进程中可以拥有n多个线程并发执行。而每个线程并执行不同的任务

进程1

进程3

进程2

线程C

程序》》》

线程B

线程A

线程的创建可以通过分为两种方式：

1. 方法（函数）包装

2. 类包装

线程的执行统一通过 start() 方法

#### 函数式创建线程的语法结构：

t=Thread(group,target,name,args,kwargs)

参数说明：

1. group:创建线程对象的进程组
2. target:创建的线程对象所要执行的目标函数
3. name:创建线程对象的名称，默认为”Tread-n”
4. args:用元组以位置参数的形式传入target对应函数的参数
5. kwargs:用字典以关键字参数的形式传入target对应函数的参数

函数式创建线程

import threading  
from threading import Thread  
import time  
#编写函数  
def test():  
 for i in range(3):  
 time.sleep(1)  
 print(f'线程：{threading.current\_thread().name}正在执行{i}')  
if \_\_name\_\_ == '\_\_main\_\_':  
 start=time.time()  
 print('主线程开始执行')  
  
 #线程  
 lst=[Thread(target=test) for i in range(2)]  
 for item in lst:#item的数据类型就是Thread类型  
 #启动线程  
 item.start()  
 for item in lst:  
 item.join()  
 print(f'用时：{int(time.time()-start)}秒')  
#主线程负责执行main中的代码，thread-1线程它要执行三次循环，thread-2他也要执行三次循环  
# 他们之间是并发执行的，谁先执行的不一定

主线程开始执行

线程：Thread-1 (test)正在执行0

线程：Thread-2 (test)正在执行0

线程：Thread-1 (test)正在执行1

线程：Thread-2 (test)正在执行1

线程：Thread-1 (test)正在执行2

线程：Thread-2 (test)正在执行2

用时：3秒

#encoding=utf-8  
#方法方式创建线程  
from threading import Thread  
from time import sleep  
def func1(name):  
 for i in range(3):  
 print(f"thread:{name} :{i}")  
 sleep(1)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("主线程，start")  
 #创建线程  
 t1 = Thread(target=func1,args=("t1",))  
 t2 = Thread(target=func1,args=("t2",))  
 #启动线程  
 t1.start()  
 t2.start()  
 print("主线程，end")

主线程，start

thread:t1 :0

主线程，end

thread:t2 :0

thread:t1 :1

thread:t2 :1

thread:t1 :2

thread:t2 :2

运行结果可能会出现换行问题，是因为多个线程抢夺控制台输出的IO流。输出换行就没有按照预想的显示

#### 继承式创建线程

使用Thread子类创建线程的操作步骤是：

1. 自定义类继承threading模块下的Thread类
2. 实现run方法
3. import threading,time  
   from threading import Thread  
   class subThread(Thread):  
    def run(self):  
    for i in range(3):  
    time.sleep(1)  
    print(f'线程：{threading.current\_thread().name}正在执行{i}')  
     
   if \_\_name\_\_ == '\_\_main\_\_':  
    print('主线程开始执行')  
    lst = [subThread() for i in range(2)]  
    for item in lst:#item的数据类型就是Thread类型  
    #启动线程  
    item.start()  
    for item in lst:  
    item.join()  
    print(f'主线程执行完毕')

主线程开始执行

线程：Thread-1正在执行0

线程：Thread-2正在执行0

线程：Thread-1正在执行1

线程：Thread-2正在执行1

线程：Thread-1正在执行2

线程：Thread-2正在执行2

主线程执行完毕

#### 线程之间的通信方式

线程之间的数据可以共享吗？ 答案：可以共享

from threading import Thread  
a=100 #全局变量  
def add():  
 print('加线程开始执行：')  
 global a  
 a+=30  
 print(f'a的值为:{a}')  
 print('加的线程执行结束')  
  
  
def sub():  
 print('减线程开始执行：')  
 global a  
 a-=50  
 print(f'a的值为:{a}')  
 print('减的线程执行结束')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('主线程开始执行')  
 print(f'..........a的全局变量是：{a}')  
 #线程  
 add=Thread(target=add)  
 sub=Thread(target=sub)  
  
 add.start()  
 sub.start()  
 add.join()  
 sub.join()  
 print('主线程执行结束')

主线程开始执行

..........a的全局变量是：100

加线程开始执行：

a的值为:130

加的线程执行结束

减线程开始执行：

a的值为:80

减的线程执行结束

主线程执行结束

结论：线程之间的数据或要共享

a+=30 a-=50

add线程

sub线程

#### 线程操作共享数据的安全性问题

import threading  
import time  
from threading import Thread  
ticket=50 #代表50张票  
  
def sale\_ticket(): #售票窗口  
 global ticket  
 for i in range(100):#相当于这个线程循环了100次  
 if ticket>0:  
 print(f'{threading.current\_thread().name}正在售票{ticket}张票')  
 ticket-=1  
 time.sleep(1)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 for i in range(3): #循环3次，内容创建三个线程，分别代表三个窗口  
 t=Thread(target=sale\_ticket)  
 t.start()

Thread-3 (sale\_ticket)正在售票19张票Thread-1 (sale\_ticket)正在售票19张票

问题：线程3和1同时出现出售同一张票

##### 解决办法lock锁

线程a访问共享数据的时候用acquire()方法锁定数据，别的线程无法使用，当非锁定状态release()方法的时候线程才可以访问数据

import threading  
import time  
from threading import Thread,Lock  
ticket=50 #代表50张票  
lock\_obj=Lock()#创建锁对象  
  
def sale\_ticket(): #售票窗口  
 global ticket  
 for i in range(100):#相当于这个线程循环了100次  
 lock\_obj.acquire()#在判断之前上锁  
 if ticket>0:  
 print(f'{threading.current\_thread().name}正在售票{ticket}张票')  
 ticket-=1  
 time.sleep(1)  
 lock\_obj.release()#使用完数据后解锁  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 for i in range(3): #循环3次，内容创建三个线程，分别代表三个窗口  
 t=Thread(target=sale\_ticket)  
 t.start()

Thread-3 (sale\_ticket)正在售票3张票

Thread-1 (sale\_ticket)正在售票2张票

Thread-2 (sale\_ticket)正在售票1张票

#### 生产者与消费者模式

是线程模型中的经典问题，与编程语言无关。当程序中出现了明确的两类任务，一个任务负责生产数据，一个任务负责生产的数据时就可以使用该模式

生产数据放入仓库 从仓库取出数据

消费者线程

生产者线程

##### python内置模块queue中的Queue类

|  |  |
| --- | --- |
| 方法名称 | 功能描述 |
| put(item) | 向队列中放置数据，如果队列为满，则阻塞 |
| get() | 从队列中取走数据，如果队列为空，则阻塞 |
| join() | 如果队列不为空，则等待队列变为空 |
| task\_done() | 消费者从队列中取走一项数据，当队列变为空时，唤醒调用join()的线程 |

使用队列实现消费者与生产者

from queue import Queue  
from threading import Thread  
import time  
  
from jinja2.utils import consume  
  
  
#创建一个生产者类  
class producar(Thread):  
 def \_\_init\_\_(self,name,queue):  
 Thread.\_\_init\_\_(self,name=name)  
 self.queue=queue  
 def run(self):  
 for i in range(1,5):  
 print(f'{self.name}的产品{i}放入队列')  
 self.queue.put(i)  
 time.sleep(1)  
 print('生产者完成了所有数据的库放')  
  
#创建一个消费者类  
class Consumer(Thread):  
 def \_\_init\_\_(self,name,queue):  
 Thread.\_\_init\_\_(self,name=name)  
 self.queue=queue  
 def run(self):  
 for \_ in range(1,5):  
 value=self.queue.get()  
 print(f'消费者线程:{self.name}取出了{value}')  
 time.sleep(1)  
 print('..........消费者线程完成了把有数据的取出........')  
if \_\_name\_\_ == '\_\_main\_\_':  
 #创建队列  
 queue=Queue()  
 #创建生产者线程  
 p=producar('producar',queue)  
 #创建消费者线程  
 c=Consumer('Consumer',queue)  
 #启动线程  
 p.start()  
 c.start()  
 #  
 p.join()  
 c.join()  
 print('主线程执行结束')

producar的产品1放入队列

消费者线程:Consumer取出了1

producar的产品2放入队列

消费者线程:Consumer取出了2

producar的产品3放入队列

消费者线程:Consumer取出了3

producar的产品4放入队列

消费者线程:Consumer取出了4

生产者完成了所有数据的库放

..........消费者线程完成了把有数据的取出........

主线程执行结束

## 多人聊天室

### 使用第三方库wxPython绘制客户端界面

#coding:utf-8  
import wx  
  
class ysjclient(wx.Frame):  
 def \_\_init\_\_(self, client\_name):  
 # 调用父类的初始化方法  
 # None代表没有父级窗口  
 # id自己定义的表示当前窗口的编号  
 # 窗体的标题  
 # pos窗体的打开位置，采用的默认  
 # size窗体的大小，单位为像素400是宽度，450为高度  
 wx.Frame.\_\_init\_\_(self, None, id=1001, title=client\_name + '的客户端界面', pos=wx.DefaultPosition, size=(400, 450))  
 # 创建面板对象  
 pl = wx.Panel(self)  
 # 面板上的盒子  
 box = wx.BoxSizer(wx.VERTICAL) # vertical垂直布局  
 # 创建可伸缩的风格布局  
 fqz1 = wx.FlexGridSizer(wx.HSCROLL) # 水平方向布局  
 # 创建两个按钮对象  
 conn\_btn = wx.Button(pl, size=(200, 40), label='连接') # 按钮的大小  
 dis\_btn = wx.Button(pl, size=(200, 40), label='断开') # 按钮的大小  
 # 把两个按钮放到可伸缩网络布局中  
 fqz1.Add(conn\_btn, 1, wx.TOP | wx.LEFT) # 顶部左对齐  
 fqz1.Add(dis\_btn, 1, wx.TOP | wx.RIGHT) # 顶部右对齐  
 # 将网格布局添加到box中  
 box.Add(fqz1, 1, wx.ALIGN\_CENTRE) # wx.ALIGN\_CENTRE居中对齐  
  
 # 只读文本框，对于显示聊天内容  
 self.show\_text = wx.TextCtrl(pl, size=(400, 210), style=wx.TE\_MULTILINE | wx.TE\_READONLY)  
 # 将文本框放入box中  
 box.Add(self.show\_text, 1, wx.ALIGN\_CENTRE)  
  
 # 创建聊天内容的文本框  
 self.chat\_text = wx.TextCtrl(pl, size=(400, 120), style=wx.TE\_MULTILINE)  
 # 将文本框放入box中  
 box.Add(self.chat\_text, 1, wx.ALIGN\_CENTRE)  
  
 # 创建可伸缩的风格布局  
 fqz2 = wx.FlexGridSizer(wx.HSCROLL) # 水平方向布局  
 # 创建两个按钮对象  
 reset\_btn = wx.Button(pl, size=(200, 40), label='重置') # 按钮的大小  
 send\_btn = wx.Button(pl, size=(200, 40), label='发送') # 按钮的大小  
 fqz2.Add(reset\_btn, 1, wx.TOP | wx.LEFT) # 顶部左对齐  
 fqz2.Add(send\_btn, 1, wx.TOP | wx.RIGHT) # 顶部右对齐  
 # 将网格布局添加到box中  
 box.Add(fqz2, 1, wx.ALIGN\_CENTRE) # wx.ALIGN\_CENTRE居中对齐  
  
 # 将盒子放在面板中  
 pl.SetSizer(box)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 初始化app  
 app = wx.App()  
 # 创建自己的客户端对象  
 client = ysjclient('python娟子姐')  
 client.Show() # 可以改成ysjclient('python娟子姐'.Show（)）  
 # 循环刷新显示  
 app.MainLoop()

### 掌握网络编程与多线程的综合应用

# 办公自动化

### excel

#### 获取表格数据

from openpyxl import load\_workbook  
from pandas.core.config\_init import pc\_max\_info\_rows\_doc  
  
  
#打开文件函数  
def open():  
 wb=load\_workbook('./excel/bead\_plan.xlsx')  
 pcm=wb['PCM']  
 print(pcm)  
#显示表格里有哪些子文档  
def show\_sheels():  
 wb = load\_workbook('./excel/bead\_plan.xlsx')  
 print(wb.sheetnames)  
 for sh in wb:  
 print(sh.title)  
#获取一个单元格  
def get\_one\_value():  
 wb = load\_workbook('./excel/bead\_plan.xlsx')  
 pcm = wb['PCM']  
 value1 = pcm['b2']  
 print(value1)  
  
#获取多个单元格  
def get\_many\_value():  
 wb = load\_workbook('./excel/bead\_plan.xlsx')  
 pcm = wb['PCM']  
 cells1 = pcm['b2':'c6']#获取多个单元格  
 # print(cells1)  
 #整行整列  
 cells\_row2= pcm[2]#整行  
 cells\_col2= pcm['b']#整列  
 # print(cells\_row2)  
 # print(cells\_col2)  
 #多行多列  
 cells\_rwo2\_5=pcm[2:5]#2到5行  
 # print(cells\_rwo2\_5)  
  
 #通过迭代获取几行到几行之间，前几列数据  
 # for row in pcm.iter\_rows(min\_row=6,max\_row=9,max\_col=3):  
 # for cell in row:  
 # print(cell.value,'\t',end='')  
 # print()  
  
 # 通过迭代获取几行到几行之间，几列到几列之间的数据  
 for row in pcm.iter\_rows(min\_row=6, max\_row=9,min\_col=2,max\_col=2):  
 for cell in row:  
 print(cell.value,'\t',end='')  
 print()  
#获取所有数据  
def get\_all\_data():  
 wb = load\_workbook('./excel/bead\_plan.xlsx')  
 pcm = wb['PCM']  
 #获取表格中所有行  
 # for row in pcm.rows:  
 # for cell in row:  
 # print(cell.value,'\t',end='')  
 # print()  
  
 # 获取表格中所有列  
 for cloumn in pcm.columns:  
 for cell in cloumn:  
 print(cell.value, '\t', end='')  
 print()  
#获取表格中有多少行多少列  
def get\_num():  
 wb = load\_workbook('./excel/bead\_plan.xlsx')  
 pcm = wb['PCM']  
 print(pcm.max\_row)  
 print(pcm.max\_column)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # open()  
 # show\_sheels()  
 # get\_one\_value()  
 # get\_many\_value()  
 # get\_all\_data()  
 get\_num()

#### 创建excel数据

from openpyxl import Workbook  
from openpyxl.styles import Font,Alignment,colors  
  
def new():  
 wb=Workbook()  
 sh1=wb.active#默认激活创建好的sheet1  
 ss2=wb.create\_sheet('数据')#创建一个数据表  
 ss2=wb.create\_sheet('人员',0)#将人员数据放在最前面  
 wb.save('./excel/create.xlsx')  
  
def set\_value():  
 hold\_italic\_30\_font=Font(name='宋体',size=30,italic=True,bold=True,color=colors.BLUE)  
 #italic斜体 bold加粗  
 wb=Workbook()  
 sh1=wb.active  
 sh1['a1']='hello'  
 sh1['b2']='python'  
 sh1['b2'].font=hold\_italic\_30\_font  
 wb.save('./excel/create.xlsx')  
  
def set2\_value():  
 wb=Workbook()  
 sh1=wb.active  
 hold\_italic\_30\_font = Font(name='宋体', size=30, italic=True, bold=True, color=colors.BLUE)  
 data=['python','hello','world']  
 for i,d in enumerate(data):  
 sh1.cell(i+1,1).value=d  
 sh1.cell(i+1,1).font= hold\_italic\_30\_font  
 wb.save('./excel/create.xlsx')  
  
def set\_merge():  
 wb=Workbook()  
 sh1=wb.active  
 sh1.merge\_cells('a1:c1')#合并单元格  
 sh1.merge\_cells('d2:e5')  
 sh1['a1']='横向合并'  
 sh1['d2']='纵向合并'  
 wb.save('./excel/create.xlsx')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # new()  
 # set\_value()  
 # set2\_value()  
 set\_merge()

#### 多个文件合并成多个工作表.py

from openpyxl import load\_workbook,Workbook  
import os  
# print(os.listdir('./excel'))#显示目录下面有哪些文件  
def copy\_data():  
 wb =Workbook()  
 for name in os.listdir('./excel/合并数据'):  
 path=f'./excel/合并数据/{name}'  
 tmp\_wb=load\_workbook(path)  
 tmp\_sh=tmp\_wb.active  
 sh=wb.create\_sheet(name[:-5])#文件名会带后缀xlsx,通过切片把后面的后.xlsx去掉  
 for r in range(1,tmp\_sh.max\_row+1):  
 #获取整行数据  
 row\_value=[]  
 for c in range(1,tmp\_sh.max\_column+1):  
 value=tmp\_sh.cell(r,c).value  
 row\_value.append(value)  
 sh.append(row\_value)  
 del wb['Sheet']#删除sheet这个表格  
 wb.save('./excel/合并数据/合并excel成多个工作表.xlsx')  
if \_\_name\_\_ == '\_\_main\_\_':  
 copy\_data()

#### 多个表格合并数据excel.py

from openpyxl import load\_workbook,Workbook  
import os  
# print(os.listdir('./excel'))#显示目录下面有哪些文件  
def copy\_data():  
 wb =Workbook()  
 sh =wb.active  
 all\_data=[]  
 for name in os.listdir('./excel/合并数据'):  
 path=f'./excel/合并数据/{name}'  
 tmp\_wb=load\_workbook(path)  
 tmp\_sh=tmp\_wb.active  
 for r in range(1,tmp\_sh.max\_row+1):  
 #获取整行数据  
 row\_value=[]  
 for c in range(1,tmp\_sh.max\_column+1):  
 value=tmp\_sh.cell(r,c).value  
 row\_value.append(value)  
 #获取整行数据，加到全局数据里  
 if row\_value not in all\_data:  
 all\_data.append(row\_value)  
 for data in all\_data:  
 sh.append(data)  
 wb.save('./excel/合并数据/多个表格合并数据.xlsx')  
if \_\_name\_\_ == '\_\_main\_\_':  
 copy\_data()

#### 工资表综合数据生成个人数据.py

from numpy.ma.core import append  
from openpyxl import load\_workbook,Workbook  
  
def create\_excel():  
 wb=load\_workbook('./excel/工资表分成个人表格数据/工资条.xlsx',data\_only=True)  
 sh=wb.active  
 title=['工号','姓名','部门','基本工资','提成','加班工资','社保扣除','考勤扣除','应发工资','邮箱']  
 for i,row in enumerate(sh.rows):  
 if i==0:  
 continue  
 else:  
 temp\_wb=Workbook()  
 temp\_sh=temp\_wb.active  
 temp\_sh.append((title))  
 row\_data=[cell.value for cell in row]  
 temp\_sh.append(row\_data)  
 temp\_wb.save(f'./excel/工资表分成个人表格数据/{row\_data[1]}.xlsx')  
 # print(row\_data)  
if \_\_name\_\_ == '\_\_main\_\_':  
 create\_excel()

#### 表格隔行背景变色

from openpyxl import Workbook,load\_workbook  
from openpyxl.utils import rows\_from\_range  
from openpyxl.styles import Protection, PatternFill  
#创建数据  
def create\_excel():  
 wb=Workbook()  
 sh=wb.active  
 rows=[  
 ['工号', '姓名', '部门', '基本工资', '提成', '加班工资', '社保扣除', '考勤扣除', '应发工资', '邮箱'],  
 [10000, 'A', '甲', 8000, 7000, 1000, 680, 0, 15320, '1@163.com'],  
 [10001, 'B', '乙', 4000, 3000, 1000, 680, 0, 7320, '1@164.com'],  
 [10002, 'C', '丙', 6000, 5000, 1000, 680, 300, 11020, '1@165.com'],  
 [10003, 'D', '丁', 7000, 6000, 1000, 680, 600, 12720, '1@166.com'],  
 [10004, 'E', '戊', 5000, 4000, 1000, 680, 0, 9320, '1@167.com']  
 ]  
 for row in rows:  
 sh.append(row)  
 #修改样式  
 bak\_color=PatternFill('solid',fgColor='AEEEEE')  
 for i in range(1,sh.max\_row+1):  
 if i%2==0:#如果被2整除为真，就是偶数  
 for cell in range(1,sh.max\_column+1):  
 sh.cell(i,cell).fill=bak\_color  
 # print(sh.cell(i,cell))  
 wb.save('./excel/工资表分成个人表格数据/隔行变色.xlsx')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 create\_excel()

#### excel标记重复数据.py

from openpyxl import load\_workbook,Workbook  
from openpyxl.styles import PatternFill  
  
  
def dum():  
 wb=load\_workbook('./excel/打卡时间.xlsx')  
 sh=wb.active  
 index=[] #存储哪一行是重复数据  
 tmp=[] #没有重复的数据  
 for i,c in enumerate(sh['b'],1):  
 flag=c.value not in tmp  
 # print(flag,f'==={c}==={tmp}')  
 # print(i)  
 if flag:  
 tmp.append(c.value)  
 else:  
 index.append(i)  
 # print(tmp,index)  
 fill=PatternFill('solid',fgColor='AEEEEE')  
 for i,r in enumerate(sh.rows,1):  
 if i in index:  
 for c in r:  
 c.fill=fill  
 # print(f'第{i}条数据是重数据')  
 wb.save('./excel/去重.xlsx')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 dum()

#### 提取身份证的信息

from openpyxl import load\_workbook  
from datetime import datetime  
  
def create\_time():  
 wb=load\_workbook('./excel/person.xlsx')  
 sh=wb.active  
 now\_year=datetime.now().year #获取当前年  
 max\_column=sh.max\_column #获取最后一列的数字  
 for i,cell in enumerate(sh['c'],1):  
 per=cell.value  
 year=per[6:10]  
 mouth=per[10:12]  
 day=per[12:14]  
 # print(year,mouth,day)  
 age=now\_year-int(year) #计算年龄  
 sh.cell(i,max\_column+1).value=year  
 sh.cell(i,max\_column+2).value=mouth  
 sh.cell(i,max\_column+3).value=day  
 sh.cell(i,max\_column+4).value=age  
 wb.save('./excel/提取身份证信息.xlsx')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 create\_time()