## 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'''

**转义字条：**\n换行 \t制表符 \“输出一个双引号 \‘ 单引号 \\斜杠

**原字符：**使转义字符失效的字符，R或r

#### 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

**字符串切处：**print(aaa[2:5]第3到第5位, aaa[:5]默认从0开始

aaa[-1]最后一位,aaa[0])第一位数

#### 常用的字符串操作：

X+Y连接

X\*N或 N\*X复制N次字符串X

X IN S 如果X是S的子串，结果为True,否则为False

aaa='123456'

print("1" in aaa)#True

print("10" in aaa)#False

#### 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

#### 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

### 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

int(x)将X转换成整数类型

float(x) 将X转换成浮点类型

str(x) 将X转换成字符串类型

chr(x)和ord(x)两者相返一个是看unicode中的字符,一个是看整数值.

print(chr(32993))#32993在unicode表中所对应的字符

print(ord("胡"))#胡在unicode表中的整数值

hex(x) 将一个字符X转换为一个十六进制字符串

oct(x) 将一个字符X转换为一个八进制字符串

bin(x) 将一个字符X转换为一个二进制字符串

**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

## 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）

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

### 比较运算符

>,<,==,!=,>=,<=,结果布尔值

### 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

## 程序的流程控制

### 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**用于退出循环结构，通常与IF一起搭配使用

**continue**:用于跳过本次循环的后续代码，而继续执行下一次循环操作，通常与IF一起搭配使用

#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

## 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

#### 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。

#### 列表的创建方式用两种

##### 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']

#### 列表的删除del列表名

语法结构如下，**del**列表名

#### 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

#### 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

#### 列表的相关操作方法

列表方法： 描述说明

lst.append(x) 在列表ist最后增加一个元素

lst.insert(index,x) 在列表中第index位置增加一个元素

lst.clear() 清除列表lst中所有的元素

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

lst.remove(x) 将列表lst中出现的第一个元素x删除

lst.reverse(x) 将列表lst中的元素反转

lst.copy() 拷贝列表lst中的所有元素，生成一个新的列表

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']

#### 列表的两种排序

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)

##### 1，列表对象的sort方法

例子：

lb=[5,2,3,4,1]

print("原列表：",lb)

原列表： [5, 2, 3, 4, 1]

lb.sort()#默认升序排序

print("升序：",lb)

升序： [1, 2, 3, 4, 5]

lb.sort(reverse=True)#reverse=True降序

print("降序：",lb)

降序： [5, 4, 3, 2, 1]

**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']

##### 2，内置函数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']

#### 列表生成式的语法结构及二维列表

##### 列表生成式

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]

#### 二维列表

##### 二维列表的遍历

for row in 二维列表

for item in row

pass

##### 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]]

### 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

#### 元组的两种创建方式

##### 1.使用()直接创建元组

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')

##### 2.使用内置函数tuple()创建元组

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)

#### 删除元组：del 元组名

元组属于组合数据可以用序列的操作

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)

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

print(t)

(10, 20, 20, 30)

print(10 in t) True

print(10 not in t) False

print(len(t)) 4

print(max(t)) 30

print(min(t)) 10

print(t.index(20)) 1

print(t.count(20)) 2

print(t,type(t)) (10, 20, 20, 30) <class 'tuple'> tuple显示是元组类型

t=(10)

print(t,type(t)) 10 <class 'int'> 只有一个数字显示的是整数类型

#### 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()

#### 元组生成式

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

### 元组和列表的区别

元组 列表

不可变序列 可以

无法实现添加，删除和修改元素等操作 append(),insert(),remove(),pop()等方法

支持切片访问元素，不能修改操作 支持切片访问和修改元素

访问和处理速度快 访问和处理速度慢

可以作为字典的键 不能作为字典的键

### 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

##### 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

##### 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

##### 2,通过映射函数创建字典,zip(lst1,lst2)

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'}

#### 删除字典：del 字典名

##### 字典可以用序列的相关操作

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

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

print(max(d)) 结果：4

print(min(d)) 结果：1

print(len(d)) 结果：4

print(1 in d) 结果： True

print(1 not in d) 结果： False

#### 字典元素的访问及遍历

##### 字典元素的取值get()

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

##### 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'}

### 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

#### 集合的创建方式有两种

##### 使用{}直接创建集合

语法结构：s={element1,elementl2,…}

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

##### 使用内置函数set()创建集合

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

##### 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

### 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）

try…except..except的语法结构为

try:

能会抛出异常的代码

except 异常类型a:

异常处理代码（报错后执行的代码）

except 异常类型b

异常处理代码（报错后执行的代码）

**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:

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…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')

#### 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：

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

### 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

### PyCharm的调式模式

短点设置：在pycharm程序行的左侧点击一下，出现红点就设置短点成功



点击小虫子进行调试



### 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

#### 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

#### 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

### 函数的参数传递

**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

**关键字参数**：是在函数调用时，使用“形参名=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

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

**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

**可变参数：**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

### 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

### 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

定义：在函数定义处的参数和函数内部定义的变量

作用范围：仅在函数内部，函数执行结束，局部变量的生命周期也结束

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

定义：在函数外定义的变量或函数内部使用global关键字修饰的变量

作用范围：整个程序，程序运行结束，全局变量的生命周期才结束

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

#### 匿名的函数lambba的使用

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

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}]

#### 递归函数

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

#### 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

### 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

#### 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

自定义数据类型的语法结构

class 类名（）：#（）这个可以不写

pass

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

### 对象的创建

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'>

### 类的组成及属性和方法的调用

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

举例：

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

### 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

#### 封装

隐藏内部细节，对外提供操作方式。

##### 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

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。

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

继承的语法结构：

class 类名（父类1，父类2....父类n）

pass

##### 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

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

#### 多态

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

\_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

#### 特殊方法

|  |  |  |
| --- | --- | --- |
| 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\_\_() | 执行乘法，非整除运算 |
| %,// | \_\_mod\_\_ (),\_\_floordiv\_\_ () | 执行取余，整除运算 |
| \*\* | \_\_pow\_\_() | 执行幂运算 |

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

#### 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'>]

[]

### 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)

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>

### 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

## 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 | 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

### python中常用的第三方模块