修改编码

python默认使用ascii编码，因此在处理中文时需要在文件头加入：

```
# coding=utf-8
```

来让编译器以utf8进行编码，从而正常处理中文字符。

基础语法

标识符

python的标识符遵循以下规则：

- 由字母、数字、下划线组成
- 不能以数字开头
- 区分大小写
- 不能是python的关键字
- 以下划线开头的标识符有特殊意义：
  - 单下划线开头`_xxx`：不能直接访问的类属性，需通过类提供的接口进行访问，不能用`from xxx import *`导入
  - 双下划线开头`__xxx`：类的私有成员
  - 双下划线开头结尾`__xxx__`：python中的特殊方法

缩进和换行

python依靠缩进来表示代码块，需要在同一个文件中统一缩进，通常为一个tab或四个空格。

可以使用反斜杠`\`将单行语句拆分成多行，若使用了括号则不需要反斜杠，如：

```
total = item_one + \
        item_two + \
        item_three

days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

引号

python可以使用单引号'、双引号"、三引号'''或"""来表示字符串，一个字符串的开始和终止由相同类型的引号组成，内部可以有其他类型的引号，其中由三引号括起来的字符串可以由多行组成，如：

```
word = 'word'
sentence = "这是一个句子。"
paragraph = """这是一个段落。
包含了多个语句"""
```

注释

单行注释由单个引号#开头，多行注释由三引号'''或"""括起来，如：

```
# 单行注释

'''
这是多行注释，使用单引号。
这是多行注释，使用单引号。
'''

"""
这是多行注释，使用双引号。
这是多行注释，使用双引号。
"""
```

输出 print

```
# 输出（组合的）字符串和表达式的值
print('100 +', '200 =', 100 + 200)

# 使用end控制换行
# 在print的参数末尾加入 end = 'xxx'，可在输出的末尾加入xxx，可以使用转义字符，默认值为
\n
print('example', end = '@gmail.com')
print("123", end = '') # 不换行

# 使用sep控制分隔符
print("Hello", "World", 123, sep="-")
# 输出：Hello-World-123
```

输出的格式化：

```
name = "Charlie"
age = 28

print(f"My name is {name} and I am {age} years old.")
print("My name is %s and I am %d years old." % (name, age))
```

输入 input

使用input函数进行输入，其参数为一个字符串，输出在控制台用于提示用户进行输入，默认返回值为string，如：

```
name = input()
print('Hello,', name)
```

可以对input获取到的输入进行类型转换，如：

```
age = float(input())
print('Your age is', age)
```

数据类型

python的标准数据类型包括：

- Numbers 数字
  - int
  - float
  - complex
- String
- List 列表
- Tuple 元组
- Dictionary 字典

对变量赋值不需要显式指定变量的类型，也可以连续赋值，如：

```
counter = 100 # int
miles = 1000.0 # float
name = "John" # String

a = b = c = 1
a, b, c = 1, 2, "john"
```

已创建的变量不允许修改数据类型

字符串 string

对字符串中元素的索引有两种计数方式：

```
 A  B  C  D  E  F
 0  1  2  3  4  5
-6 -5 -4 -3 -2 -1
```

使用加号+连接字符串，使用型号*重复字符串，使用[头下标,尾下标:步长]来截取字符串，包括头下标不包括尾下标，尾下标留空表示一直到字符串末尾，步长默认为1，可以是负数，如：

```
str1 = 'ABCD'
str2 = 'EFGH'

print(str1 + str2) # 输出'ABCDEFGH'
print(str1 * 2) # 输出'ABCDABCD'
print((str1 + str2)[1:5]) # 输出'BCDE'
```

可以使用转义字符：

转义字符	解释
\'	单引号
\"	双引号
\\	反斜杠
\n	换行符
\t	制表符
\0	空字符
\uxxxx	Unicode字符，xxxx为16进制数
\Uxxxxxxxx	Unicode字符，xxxxxxxx为8位16进制数
\oXXX	八进制字符，XXX为三位八进制数
r"\r"	原始字符串，r后的字符串不作转义处理

字符串也可以格式化输出，如：

```
name = "Chara"
age = 18
print("My name is %s, I'm %d years old." % (name, age))
```

```python
# 使用字典格式化字符串
person = {
    'name': 'Alice',
    'age': 30,
    'city': 'New York'
}
print ("Name: %(name)s, Age: %(age)d, City: %(city)s" % person)
```

格式控制字符	解释
%s	字符串
%d	整数
%f	浮点数
%x	十六进制整数
%o	八进制整数
%e	科学计数法
%g	自动选择 %f 或 %e 格式
%%	转义为百分号 %
%<width>d	指定整数宽度
%<width>.<precision>f	指定浮点数宽度和小数位数
%(<name>)s	使用字典中的键进行格式化

python对字符串有一些内建函数：

函数	描述
string.capitalize()	首字母大写，其余字母小写。
string.lower()	所有字母转换为小写。
string.upper()	所有字母转换为大写。
string.title()	每个单词的首字母大写。
string.strip([chars])	移除字符串两端的空白字符（或指定字符）。
string.split([sep[, maxsplit]])	根据指定分隔符将字符串分割为多个部分。
string.join(iterable)	将序列中的元素连接为一个字符串，使用指定的分隔符。
string.replace(old, new[, count])	将字符串中的指定子串替换为另一个子串。
string.find(sub[, start[, end]])	返回指定子串第一次出现的位置，未找到时返回 -1。
string.count(sub[, start[, end]])	返回指定子串在字符串中出现的次数。

列表 list

列表是可变的对象的有序集合，以方括号[]来表示，元素之间使用逗号,分隔，可以储存字符、字符串、列表（嵌套），如：

```
list1 = ['abcd', 786, ')', 70.2, ["a", "b", "c"]]
```

列表的索引方式和连接、重复、截取与字符串相同，但是列表在截取时可以指定补偿如：

```
list = [ 'runoob', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list)
#['runoob', 786, 2.23, 'john', 70.2]

print (list[0])
# runoob

print (list[1:3])
# [786, 2.23]

print (list[2:])
# [2.23, 'john', 70.2]

print (tinylist * 2 )
# [123, 'john', 123, 'john']

print (list + tinylist)
# ['runoob', 786, 2.23, 'john', 70.2, 123, 'john']

print (list[::2])
# ['runoob', 2.23, 70.2]
```

- 列表名.append(value)：在列表末尾追加元素
- 列表名.insert(索引,value)：在索引处插入元素
- 列表名.remove(value)：删除第一个指定值的元素
- 列表名.pop(索引)：删除并返回索引处的元素，不指定索引则删除并返回最后一个元素
- 列表名.sort()：对元素可比较的列表进行排序
- 列表名.reverse()：反转列表
- 列表名.count(value)：统计某个元素出现的个数
- 列表名.index(value)：返回某个元素第一次出现的索引
- del 列表名[索引]：删除索引处的元素
- len(列表名)：返回列表的长度

元组 Tuple

元组与列表相似，以小括号()表示，元素之间使用逗号,分隔，但元组不可以二次赋值，相当于只读列表，截取、连接等语法与列表相同。

```
tuple1 = ('abcd', 786, 2.23, 'john', 70.2, [1, 2, 3])

# tuple1[1] = 200
# 这个是非法的 因为元组不允许修改
```

由于元组不可变，其操作方法较少，列表的count、index、len可以用于元组

字典 Dictionary

字典是键值对的集合，是无序的对象集合，以大括号{}表示，键值对之间使用逗号,分隔，键和值之间使用冒号:分隔，键必须是不可变对象，值可以是任意对象，如：

```
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"

tinydict = {
    'name': 'runoob',
    'code':6734,
    'dept': 'sales'
}

print dict['one']
# This is one

print dict[2]
# This is two

print tinydict
# {'name': 'runoob', 'code': 6734, 'dept': 'sales'}

print tinydict.keys()      # 输出所有键
# ['name', 'code', 'dept']

print tinydict.values()    # 输出所有值
# ['runoob', 6734, 'sales']
```

- 字典名.keys()：返回字典所有的键
- 字典名.values()：返回字典所有的值
- 字典名.items()：返回字典所有的键值对
- 字典名.get(key)：返回指定键的值，如果键不存在则返回None
- 字典名.pop(key)：删除指定键的键值对，并返回值
- 字典名.popitem()：删除并返回字典最后一个键值对

- `字典名.clear()`：清空字典
- `del 字典名[key]`：删除指定键的键值对
- `if value in 字典名`：检查字典中是否存在指定值
- `字典1 | 字典2`：合并字典，对于相同的键，则字典2的值会覆盖字典1的值

集合 Set

集合是用于存储多个元素的无序、不重复的集合体，其中元素不可重复，不可通过索引访问。

使用花括号`{}`定义集合，其中元素用逗号`,`分隔。特殊的，空集合使用`set()`定义，因为`{}`表示空字典。

```
set1 = {1, 2, 3, 4, 5}
set2 = set()
```

- `集合名.add(value)`：添加元素，若元素已存在则集合不变
- 删除元素
  - `集合名.remove(value)`：若元素不存在则抛出异常
  - `集合名.discard(value)`：若元素不存在则集合不变，不会抛出异常
- 集合运算
  - 并集：`set1 | set2`
  - 交集：`set1 & set2`
  - 差集：`set1 - set2`，存在于第一个但不存在于第二个元素中
  - 对称差集：`set1 ^ set2`，两个集合中不重复的元素
- 检查元素是否存在：`if value in/not in 集合名`
- `集合名.len()`：返回集合的长度
- `集合名.clear()`：清空集合

数据类型转换

函数	描述
`int()`	将数据转换为整数，浮点数取整，字符串转换
`float()`	将数据转换为浮点数
`str()`	将数据转换为字符串
`chr()`	将数据转换为字符
`hex()`	将数据转换为十六进制
`oct()`	将数据转换为八进制
`list()`	将数据转换为列表
`tuple()`	将数据转换为元组

函数	描述
`set()`	将数据转换为集合
`dict()`	将数据转换为字典

运算符

算术运算符

取模：`%`
幂：`**`
取整除`//`（商的整数部分）

位运算符

描述	示例表达式	结果
按位与：相应位都为1时结果为1	`0b0101 & 0b0011`	`0b0001`
按位或：相应位只要有1时结果为1	`0b0101 | 0b0011`	`0b0111`
按位异或：相应位相异时结果为1	`0b0101 ^ 0b0011`	`0b0110`
按位取反：每个位取反	`~0b0101`	`-0b0110`
左移：将二进制位左移指定的位数	`0b0101 << 1`	`0b1010`
右移：将二进制位右移指定的位数	`0b0101 >> 1`	`0b0010`

逻辑运算符

运算符	描述
`and`	逻辑与
`or`	逻辑或
`not`	逻辑非

成员运算符

`value in / not in container`可以判断某个元素是否存在于列表/元组/字符串/字典/集合中，其中字典判断的是键而不是值

身份运算符

`a is / is not b`可以判读两个标识符是否引用自同一个对象，即指向的内存空间是否相同。
等于运算符`==`判断的是值，身份运算符`is / is not`判断的是内存空间

```
# 示例 1: 两个变量引用同一个对象
a = [1, 2, 3]
b = a

print(a is b)        # 输出: True (a 和 b 指向同一块内存)
print(a == b)        # 输出: True (a 和 b 的值相等)

# 示例 2: 两个变量具有相同的值，但是不同的对象
c = [1, 2, 3]
print(a is c)        # 输出: False (a 和 c 是不同的对象)
print(a == c)        # 输出: True (a 和 c 的值相等)

# 示例 3: 数字和字符串的对象池
x = 1000
y = 1000
print(x is y)        # 输出: False (在大数时，Python不会重用对象)

x = 10
y = 10
print(x is y)        # 输出: True (小数字在Python中可能会指向相同的内存对象)
```

条件和循环语句

```
if 条件1:
    语句
elif 条件2:
    语句
else:
    语句

# python不支持switch语句！！！
```

```
while 条件:
    语句
else:
    # while-else中的语句在正常跳出循环（不通过break）后执行
    语句


# 使用for循环来遍历可迭代对象（列表、元组、字符串、集合、字典、文件）
for 变量 in 集合:
    语句
else:
```

```
    语句

# 使用break直接跳出当前循环，使用continue跳过当前循环，直接进行下一次循环
```

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

word = "Python"
for letter in word:
    print(letter)

my_dict = {"a": 1, "b": 2, "c": 3}
for key in my_dict:
    print(key)
for value in my_dict.values():
    print(value)
for key, value in my_dict.items():
    print(key, value)
```

使用`range()`函数可以生成数字序列，常搭配`for i in range(value)`使用

- range(stop)：整数序列[0, stop-1]
- range(start, stop)：整数序列[start, stop-1]
- range(start, stop, step)：整数序列[start, stop-1]，步长为step

`range()`返回的是`range`对象，可以用`list(range())`转换为列表

```
for i in range(10, 0, -1):
    print(i)
```

常用模块

math

函数语法	解释
`math.sqrt(x)`	x的平方根
`math.pow(x, y)`	x的y次幂 (x^y)
`math.factorial(x)`	x的阶乘
`math.floor(x)`	小于等于x的最大整数

函数语法	解释
`math.ceil(x)`	大于等于x的最小整数
`math.log(x)`	x的自然对数，即ln(x)
`math.log10(x)`	x的常用对数，即log10(x)
`math.log(x, base)`	返回以base为底的x对数
`math.exp(x)`	返回e的x次幂 (e^x)
`math.sin(x)` / `cos(x)`	返回x的正弦/余弦值 (弧度制)
`math.tan(x)`	返回x的正切值
`math.asin(x)` / `acos(x)`	返回x的反正弦/反余弦值
`math.atan(x)`	返回x的反正切值
`math.degrees(x)`	将x从弧度转换为角度
`math.radians(x)`	将x从角度转换为弧度

random

函数语法	解释
`random.random()`	返回`[0.0, 1.0)`之间的随机数
`random.randint(a, b)`	返回[a, b]之间的随机整数
`random.choice(seq)`	从序列中随机选择一个元素
`random.shuffle(lst)`	将序列中的元素随机排序
`random.sample(seq, k)`	从序列中随机抽取k个元素

os

函数语法	解释
`os.getcwd()`	返回当前工作目录
`os.makedirs(path)`	创建多级目录
`os.remove(path)`	删除指定文件
`os.path.exists(path)`	判断路径是否存在
`os.rename(当前文件名, 新文件名)`	重命名文件或目录
`os.rmdir(path)`	删除空目录
`os.path.abspath(path)`	返回绝对路径

函数语法	解释
`os.path.join(path1, path2)`	拼接两个路径，自动处理分隔符

time

函数	描述
`time.sleep(seconds)`	暂停执行指定的秒数。
`time.time()`	返回当前时间的时间戳（自1970年1月1日起的秒数）。

函数

函数定义格式：

```
def 函数名(参数):
    函数体
    return result  # 可选
```

不可变对象与可变对象

python中，类型是对象的属性，变量没有类型，如`a = [1, 2, 3]`，[1, 2, 3]是List类型，变量a没有类型，他只是一个这个list的引用，即一个指针，指针可以指向不同类型的对象。

不可变对象：整数、浮点数、字符串、元组
可变对象：列表、字典、集合，如：

```
a = 5
a = 10
# int是不可变对象，实际上5被抛弃，然后生成了一个int型的10，再让a指向他
# 这个操作中没有改变a的值，而是重新生成了a

b = [1, 2, 4]
b[2] = 3
# 列表是可变对象，修改了b中索引为2的元素的值，b本身不变，只是其内部的一部分值被修改了
```

python中一切皆对象，因此传参时应说传可变对象或不可变对象：

- 在不可变对象只是传递了对象的值，在函数内部对形参的修改不影响原对象的值
- 可变对象相当于传递了一个指针，在函数内部对形参修改会影响原对象的值

函数的参数

```python
def example(a, b, c):
    return (a + b + c)

# 默认参数
# 在函数定义时给出参数的默认值，调用的若果没有给出参数值，则使用默认值
def example2(a, b, c=0):
    return (a + b + c)

# 位置参数
# 函数调用时，依据函数的定义依次传递参数
print(example(1, 2, 3))  # 6

# 关键字参数
# 函数调用时，通过参数名指定参数值，而不需要有据固定的顺序
print(example(c=3, a=1, b=2))  # 6

# 可变参数
# 向函数传递任意数量的参数，保存到一个元组中，用*标识
def example3(*args):
    for item in args:
        print(item)
example3(1, 2, 3, 4, 5)

# 像函数传递任意数量的关键字参数，保存到一个字典中，用**标识
def example4(**kwargs):
    for key, value in kwargs.items():
        print(f"{key} = {value}")
example4(a=1, b=2, c=3)
```

global 标识符

在函数内部使用global关键字，可以声明一个全局变量，全局变量在函数内部和外部都可以访问，如：

```python
a = 10
b = 10

def modify():
    a = 20 # 局部变量
    global b # 声明全局变量
    global c # 创建全局变量
    b = 30
    c = 40
    # 不可以在声明全局变量的同时为其赋值！！！
    # 比如这样：global d = 50

modify()
print(f'a = {a}, b = {b}, c = {c}')  # a = 10, b = 30, c = 40
```

lambda 表达式

lambda表达式适用于创建简单、一次性的函数，可传入多个参数，返回表达式的值，语法如下：

```
lambda 参数: 表达式

# 示例
add = lambda x, y: x + y
print(add(5, 3))  # 输出: 8

# 根据坐标的y值排序
points = [(2, 3), (1, 2), (4, 1)]
sorted_points = sorted(points, key=lambda point: point[1])
print(sorted_points)  # 输出: [(4, 1), (1, 2), (2, 3)]
'''
在这个过程中，sorted先取出points中的每个元组

然后对每个元组调用lambda表达式，把point=元组传递给表达式

表达式会返回元组中索引为1的元素，也就是把points中每个元组索引为1的元素分别作为sorted的第二个参数key的值

最后sorted会根据每个元组索引为1的元素的大小进行排序
'''
```

模块

import 语句

- import x.y：仅导入x中的对象y，需要使用x.y访问模块中的对象
- import x：导入x中的所有内容，需要使用x.y访问模块中的对象
- from x import y：仅导入x中的对象y，可以直接使用y访问模块中的对象
- from x import *：导入x中的所有内容，可以直接访问x中的对象，如y

导入模块时，解释器会按照固定的顺序从不同的目录中查找模块：

1. 当前文件目录
2. 环境变量 PYTHONPATH
3. 标准库目录
4. 安装的第三方库目录 site-package
5. python内置模块

dir 函数

使用`dir(模块名)`可以查看模块中包含的模块、函数、变量，如：

```
import math

print(dir(math))
```

```
# 输出: ['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan', 'atan2',
'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod', 'frexp',
'hypot', 'ldexp', 'log', 'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

文件IO

open 函数

使用open()函数打开一个文件，为其创建一个file对象，然后对其进行读写，语法如下：

```
file object = open(file, mode='r', encoding=None)

file = open('example.txt', 'r')
# 以只读模式打开当前目录下的example.txt文件
content = file.read()
print(content)
file.close()

# open()常搭配with语句使用，用于执行完对文件的操作后自动关闭文件，节约资源
with open('example.txt', 'r') as file:
    # 读取文件内容并输出
    content = file.read()
    print(content)
```

打开模式类型：

打开模式	模式名称	读取	写入	追加写入	文件存在	文件不存在
'r'	只读	√			√	
'w'	写入		√			√
'a'	追加			√	√	√
'x'	创建		√			√
'r+'	读写	√	√		√	
'w+'	读写	√	√			√
'a+'	读写追加	√		√	√	√

读取: 允许读取文件
写入: 允许写入文件（覆盖现有内容）
追加写入: 在文件末尾追加内容
文件存在: 文件必须存在才能操作
文件不存在: 文件可以不存在并创建

read 读取

`文件名.read(size)`：读取指定大小的字节或字符，不指定则读取整个文件
`文件名.readline()`：每次调用读取一行
`文件名.readlines()`：读取所有行，保存为一个列表，文件中每行为列表中的一个元素

write 写入

`文件名.write(str)`：写入字符串到文件中
`文件名.writelines(lines)`：将字符串列表写入文件中，需要手动添加换行符`\n`

tell seek 定位

每个文件对象有一个内部指针，记录当前读写的位置

`文件名.tell()`：返回当前指针的位置，即距离文件开头的字节数
`文件名.seek(offset, from)`：移动指针到指定位置，offset为相对于from的偏移量，from为偏移的起始位置，0表示文件开头，1表示当前位置，2表示文件末尾

```
f.seek(0)  # 将指针移动到文件开头
f.seek(5)  # 将指针移动到文件第5个字节
f.seek(0, 2)  # 将指针移动到文件末尾
```

其他

`文件名.close()`：关闭文件，释放资源
`文件名.name()`：返回文件名

异常处理

异常处理机制用于捕获并处理运行时的错误，通过`try-except`实现：

```
try:
    # 可能会抛出异常的代码
except (ExceptionType1,(ExceptionType2,...)) as e:
    # 可以一次捕捉多个异常
except ExceptionType3 as e:
    # 可以有多个except语句
except Exception as e:
    # 使用Exception捕捉所有异常
else:
else:
    # 没有异常时执行的代码
finally:
    # 无论是否抛出异常都执行的代码
```

常见异常

异常名称	说明
AttributeError	尝试访问对象中不存在的属性
IndexError	当尝试访问超出序列索引范围的元素
KeyError	访问字典中不存在的键
ValueError	函数接收到的参数类型正确，但值不合适
ZeroDivisionError	进行除零运算
TypeError	执行操作或函数时，应用于不适当的类型对象
FileNotFoundError	尝试打开不存在的文件
ImportError	模块导入失败
OverflowError	数值运算结果超出限制
KeyboardInterrupt	用户中断执行（例如按下Ctrl+C）
RuntimeError	一般运行时错误，没有特定的错误类型
NameError	尝试访问未定义的变量
SyntaxError	语法错误

自定义异常

1. 创建一个异常类：创建一个继承自 `Exception` 类的自定义异常类；
2. 初始化异常类：在自定义异常类中定义 `__init__` 方法，用于初始化异常的详细信息；
3. 抛出异常：使用 raise 关键字来手动抛出自定义异常；
4. 捕获异常：通过 try-except 语句捕获自定义异常并处理。

```
# 定义自定义异常类
class CustomError(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)

# 使用 raise 抛出异常
def check_positive_number(value):
    if value < 0:
        raise CustomError(f"值 {value} 不能为负数！")

try:
    check_positive_number(-10)
except CustomError as e:
    print(f"捕获到自定义异常：{e}")
```

面向对象

类和对象

类是用来描述具有相同的属性和方法的对象的集合，定义了集合中每个对象所共有的属性和方法。对象是类的实例。

```
class ClassName:
   '类的帮助信息'    # 类文档字符串
   class_suite  # 类体
```

```
# -*- coding: UTF-8 -*-
# 定义一个名为Employee的类
class Employee:
   '所有员工的基类'

   # 类变量，在整个类的所有实例间共享，可以使用Employee.empCount访问
   empCount = 0

   def __init__(self, name, salary):
        # 构造方法，用于初始化新创建的对象
        # self表示这个类的实例，定义方法时必须有，但是不需要为其传递参数
        self.name = name
        self.salary = salary
        Employee.empCount += 1

   def displayCount(self):
        print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
        print "Name : ", self.name,  ", Salary: ", self.salary

# 类的实例化的语法和函数类似，使用类名后跟括号来创建对象
# 创建对象时自动调用构造方法 __init__()并传入参数
emp1 = Employee("Zara", 2000)
emp2 = Employee("Manni", 5000)

# 使用 类名.元素 访问类中的变量和方法
emp1.displayEmployee()
emp2.displayEmployee()
print("Total Employee %d" % Employee.empCount)

# 对类的元素进行操作
hasattr(emp1, 'age')    # 如果存在 'age' 属性返回 True
setattr(emp1, 'age', 8) # 添加属性 'age' 值为 8
getattr(emp1, 'age')    # 返回 'age' 属性的值
delattr(emp1, 'age')    # 删除属性 'age'
```

对象的销毁

对象在被创建的同时创建了一个引用计数，当一个对象的引用计数变为0（这个对象不再需要）时，会由解释器在合适的时机被自动调用的析构函数`__del__`进行垃圾回收。

`__del__`不需要手动定义，默认情况下会自动清理对象中的属性，释放资源，当然也可以手动定义他来执行额外的文件清理操作。

```
class MyClass:
    def __init__(self):
        print("对象被创建")

    def __del__(self):
        print("对象被销毁")
```

继承

子类可以通过`class 子类名(父类吗)`继承父类的所有属性和方法，若子类中没有定义某个属性或方法则会直接调用父类的实现。

子类中可以通过方法的重写来改变父类的同名方法，若要在重写的方法中调用父类的原方法需要使用`super().方法名`来实现。

若子类有自己的构造函数，则不会调用父类的构造函数。则需要在子类中调用父类的构造函数来对父类的属性进行初始化，需要使用`super().__init__`。

```
# 父类
class Animal:
    def __init__(self, name):  # 父类的构造函数
        self.name = name  # 父类的属性
        print(f"{self.name} is an animal.")

    def sound(self):  # 父类的方法
        print("Animals make sounds.")

# 子类继承父类
class Dog(Animal):
    def __init__(self, name, breed):  # 子类有自己的构造函数
        super().__init__(name)  # 调用父类的构造函数，初始化父类属性
        self.breed = breed  # 子类特有的属性
        print(f"{self.name} is a {self.breed}.")

    # 子类重写父类的方法
    def sound(self):
        super().sound()  # 调用父类的方法
        print(f"{self.name} barks.")  # 子类的扩展功能

# 使用子类
my_dog = Dog("Buddy", "Golden Retriever")
my_dog.sound()

# 输出:
```

```
# Buddy is an animal.
# Buddy is a Golden Retriever.
# Animals make sounds.
# Buddy barks.
```

类属性与方法

`__private_attrs`：使用双下划线开头表示这是一个私有属性，只能在类的内部通过`self.__private_attrs`访问，不可以从类的外部访问。

`def`：使用def定义类的一个方法，其必须有第一个参数`self`，