■一些常用的零散的东西

* dir函数的用法

# dir 显示传入对象的所有属性和方法  
a = []  
  
print(dir(a))  
# 输出：['\_\_add\_\_', '\_\_class\_\_',......'\_\_subclasshook\_\_', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

●id函数 的用法

# id 返回传入对象的十进制内存地址  
class Cat:  
 def say(self):  
 print("我是猫。")  
  
  
tom = Cat()  
tom.say()  
# 输出：我是猫。  
print(tom)  
"""  
输出：  
<\_\_main\_\_.Cat object at 0x00000266AE3D7CD0>  
Cat： 所属类  
object： tom是一个对象  
0x00000266AE3D7CD0： 十六进制内存地址  
"""  
addr = id(tom)  
print(addr)  
# 输出：2603557682448（十进制的内存地址，id函数默认输出十进制的）  
print(hex(addr))  
# 输出：0x25e30217d10（十六进制内存地址，通过hex函数转换的）

●del关键字 的用法

class Cat:  
 def \_\_init\_\_(self):  
 print("我开始了。创建函数时我自动执行。")  
  
 def \_\_del\_\_(self):  
 print("我结束了。销毁函数前我自动执行。")  
  
  
tom = Cat()  
print("-" \* 50)  
  
"""  
输出：  
我开始了。创建函数时我自动执行。  
--------------------------------------------------  
我结束了。销毁函数前我自动执行。  
注：  
我结束了，在分割线下方输出。因为执行完程序所有命令后，对象被消除时，del内置方法被执行。  
"""

class Cat:  
 def \_\_init\_\_(self):  
 print("我开始了。创建函数时我自动执行。")  
  
 def \_\_del\_\_(self):  
 print("我结束了。销毁函数前我自动执行。")  
  
  
tom = Cat()  
del tom  
print("-" \* 50)  
  
"""  
输出：  
我开始了。创建函数时我自动执行。  
我结束了。销毁函数前我自动执行。  
--------------------------------------------------  
注：  
我结束了，在分割线上方输出。因为执行del关键字，手动将函数删除。所以删除时立即执行了del内置方法。  
"""

●用小括号来换行

def a(x, y):  
 # 将要换行的内容用括号括起来，然后换行，python会自动默认是一行内容  
 return ("这里是函数a的返回值，"  
 "其中一个参数是%s，"  
 "它是第一个参数。"  
 "另一个参数是%s，"  
 "它是第二个参数。"  
 % (x, y))  
  
  
print(a("aaa", "bbb"))  
# 输出：这里是函数a的返回值，其中一个参数是aaa，它是第一个参数。另一个参数是bbb，它是第二个参数。

●身份运算符 is 或 isnot

# 身份运算符is 或 isnot 用来判断两个对象的内存地址是否一致  
  
a = [1, 2, 3]  
b = a  
  
if a is b:  
 print(f"a 的内存地址是{id(a)}，b 的内存地址是{id(b)}，它们的内存地址相同")

# 针对None比较时，python建议使用is来判断  
  
c = None  
  
if c is None:  
 print("我是None。")  
# 输出：我是None。  
  
d = None  
  
if d is c:  
 print("我们都是None。")  
# 输出：我们都是None。（可见，None都被保存到同一个内存地址）

●关于类的属性的定义

直接给类的实例化对象定义属性虽然不报错，但是不推荐。

class Cat:  
 pass  
  
  
tom = Cat()  
tom.name = "汤姆" # 这样给属性赋值虽然不报错，但不推荐。  
print(tom.name)  
# 输出：汤姆

标准方法是类里定义好属性，实例化对象给属性赋值

# 标准的方法是，在类里定义好属性。具体值可暂时用None  
class Cat:  
 def \_\_init\_\_(self):  
 self.name = None  
  
  
tom = Cat()  
tom.name = "汤姆" # 给类里定义好的name属性赋值  
print(tom.name)  
# 输出：汤姆

常见的还有构造的方法

# 比较常见是下面这种构造方法  
class Cat:  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
  
tom = Cat("汤姆")  
print(tom.name)  
# 输出：汤姆

●在外部访问私有属性的方法

# 比较常见是下面这种构造方法  
class Cat:  
 \_\_age = 2  
  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
 def sayage(self):  
 print(self.\_\_age)  
  
  
tom = Cat("汤姆")  
print(tom.\_Cat\_\_age) #使用 类的实例对象加点加单下划线加类名加私有属性 可以在外部访问私有属性，但不建议这样做

■面向对象编程

封装（根据职责将属性和方法封装到一个抽象的类中）

继承（实现代码的重用，相同的代码不需要重复的编写）

多态（不同的对象调用相同的方法，产生不同的执行结果，增加代码的灵活度。）

●内置属性mro

内置属性mro用来记录继承链的关系。如下土，多继承，访问一个属性先找哪个，可以通过mro来查询

class Aa:  
 name = 'jojo'  
 age = 22  
  
  
class Bb(Aa):  
 age = 18  
 name = 'alice'  
  
  
class Cc(Bb, Aa):  
 pass  
  
  
d = Cc()  
  
print(d.name)  
# 输出：alice  
print(d.age)  
# 输出：18  
print(Cc.\_\_mro\_\_)  
# 输出：(<class '\_\_main\_\_.Cc'>, <class '\_\_main\_\_.Bb'>, <class '\_\_main\_\_.Aa'>, <class 'object'>)  
# 例如找d.name，先从Cc找，找不到找Bb, 找不到找Aa，最后找object(所有类的基类)

●类的属性和方法（不同种类和访问方法）

类的属性包括：类属性、实例属性

类的方法包括：类方法、实例方法、静态方法

class Cat:  
  
 # 定义类属性  
 sort = 'animal'  
  
 # 定义实例属性  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
 # 定义实例方法（可访问实例属性，也可访问类属性）  
 def eat(self):  
 print(f"{self.name}属于{self.sort}, 它吃生食物。")  
  
 # 定义类方法（只访问类属性）  
 @classmethod  
 def like(cls):  
 print(f"{cls.sort}都喜欢运动。")  
  
 # 定义静态方法（既不需要访问类属性，也不需要访问实例属性）  
 @staticmethod  
 def say():  
 print("我什么都不想说。")  
  
  
# 将Cat类实例化为一个实例对象  
tom = Cat("汤姆")  
  
# 访问实例属性  
print(tom.name)  
# 输出：汤姆  
  
# 访问实例方法  
tom.eat()  
# 输出：汤姆属于animal, 它吃生食物。  
  
# 访问类属性（类名.属性名）  
print(Cat.sort)  
# 输出：animal  
  
# 访问类方法（类名.方法名）  
Cat.like()  
# 输出：animal都喜欢运动。  
  
# 访问静态方法（类名.方法名）  
Cat.say()  
# 输出：我什么都不想说。

■拆包

●问题描述

# \*args代表任何多个无名参数，返回一个元组  
# \*\*kwargs代表任何多个键值对，返回一个字典  
def a(\*args, \*\*kwargs):  
 print(args)  
 print(kwargs)  
  
  
b = (1, 2, 3)  
c = {"name": "jojo", "age": 22}  
  
# 问题：直接传入参数b 和 c ，那么解释器会理解成都传入了\*args里  
a(b, c)  
  
"""  
输出：  
((1, 2, 3), {'name': 'jojo', 'age': 22})  
{}  
"""

●解决方案1

def a(\*args, \*\*kwargs):  
 print(args)  
 print(kwargs)  
  
  
b = (1, 2, 3)  
c = {"name": "jojo", "age": 22}  
  
# 解决方法1：使用如下这种入参方式进行拆包  
a(\*b, \*\*c)  
  
"""  
输出：  
(1, 2, 3)  
{'name': 'jojo', 'age': 22}  
"""

●解决方案2

def a(\*args, \*\*kwargs):  
 print(args)  
 print(kwargs)  
  
  
b = (1, 2, 3)  
c = {"name": "jojo", "age": 22}  
  
# 解决方法2：使用如下这种入参方式进行拆包  
a(1, 2, 3, name="jojo", age=22)  
  
"""  
输出：  
(1, 2, 3)  
{'name': 'jojo', 'age': 22}  
"""