6.3总结：文件读写、封装、继承、多态

文件读写

文本文件写

```
# 写文件时若路径没有该文件会新建文件
with open("D:\\wtest.txt", mode='w', encoding='UTF-8') as f :
    f.write("ccccccccccccccc\n")  #不会自动换行，需要自行加\n换行
    f.write("ccccccccccccccc\n")
# with语句会自动管理文件的生命周期，在代码块结束后自动关闭文件，无需手动调用f.close()

#如果直接通过open()打开文件且不使用with语句，必须手动调用f.close()关闭文件
f=open("file.txt", "w")
f.write("Hello, World!")
f.close() # 必须手动关闭
```

文本文件读

```
with open("/root/python/test.txt", mode='r', encoding='UTF-8') as f:
    res1 = f.read()  #将文件的全部内容读取为一个字符串
    res2 = f.readlines()  #每行作为一个元素返回列表
    print(res1)
```

mode：
w 覆盖写
a 追加写
r 读

python 库

标准库，扩展库，自定义库
用 import 或者 from ... import 来导入相应的库

```
import csv  # 导入CSV处理模块，用于读写CSV格式文件

data = [{"id": 1, "passwd": "123456", "bat": 1000},  # 示例数据：包含两个用户信息的字典列表
        {"id": 2, "passwd": "abcdef", "bat": 2000}]
```

```python
# 写入CSV文件
with open("/root/python/user.csv", mode="w", encoding="UTF-8") as f:
    # 创建DictWriter对象，指定表头字段顺序
    writer = csv.DictWriter(f, fieldnames=["id", "passwd", "bat"])
    writer.writeheader()  # 写入表头行（id,passwd,bat）
    writer.writerows(data)  # 将data中的所有字典按表头格式写入文件

# 读取CSV文件
with open("/root/python/user.csv", mode="r", encoding="UTF-8") as f:
    reader = csv.DictReader(f)  # 创建DictReader对象，自动将首行作为表头
    # 因为文件是字典列表的格式，因此需要遍历文件中的每一行以取出其中的每一个字典
    for i in reader:
        print(i)  # 输出每一个字典,可通过键名访问字段值：i['id'] 返回 '1'
        data.append(i)  # 将字典逐个添加到列表中给之后使用
        # 若data有原始数据需要覆盖写可于遍历前先清空：data.clear()
```

异常

```python
try:
    x = int(input("请输入一个数字: "))
    z=100/x
    if x<0 :
        raise Exception("x不能小于0")  # 自定义异常
    if x>100:
        raise Exception("x不能大于100")  # 可设置多个异常情况，后面的e会根据情况输出信息
    print("ok")

except ValueError:
    print("您输入的不是数字,请再次尝试输入！")
except ZeroDivisionError:
    print("除数不能为零")
except Exception as e:
    print("其他错误:",e)  # e显示为异常信息
```

封装、继承、多态

封装

类及对象包含属性和方法
属性：静态特征 全局变量 成员
方法：动态特征 函数 功能

魔法方法：不需要调用就可以自动执行。
作用：初始化对象的成员（给对象添加属性）

```
#类定义
class People:
    name=""
    age=0
    def __init__(self,xingming,nianling):
        self.name=xingming
        self.age=nianling
    def show(self):
        print(f"姓名是{self.name},年龄是{self.age}")

#调用
if __name__=="__main__":
    ldh=People("刘德华",50)
    ldh.show()
```

继承

class 子类名（父类名）:
子类直接具备父类的属性和方法
解决代码重用问题，提高开发效率

```
class Student(People):
    grade=""
    def __init__(self, xingming, nianling,nianji):
        super().__init__(xingming,nianling)
        self.grade=nianji
    def test(self):
        print(f"年级是{self.grade}")

    # 方法重写
    def show(self):
        print(f"姓名是{self.name},年龄是{self.age},年级是{self.grade}")
```

多态

多态从字面上理解就是一个事物可以呈现多种状态。
没有继承就没有多态。
多态是能自己进行判断该去执行什么，创建一个列表来体现，面向对象的列表。

```
list1=[ldh,zjl]
# 通过遍历以一个触发使用各子类重写方法的过程就称作多态
for i in l1:
    print(i.name)
    i.show()
```

作业练习

```
# 1.将内容 “ 诚挚邀请您来参加本次宴会 ” 追加到 “ 邀请函 .txt” 文件末尾。
with open("/root/python/邀请函.txt",mode='a',encoding='UTF-8') as f :
    f.write("诚挚邀请您来参加本次宴会")
    # f.close()

# 2.读取邀请函的内容。
with open("/root/python/邀请函.txt",mode="r",encoding="UTF-8") as f:
    result=f.read()
    print(result)
    # f.close()

# 3.修改 atm 取款机数据为持久化永久存储， csv 读取实现（分别使用函数实现读和取） , 使用异常处理 atm 取款机读取 csv 时判断文件是否存在，如果文件不存在提示 " 没有文件使用原数据 "
def init():
    try:
        with open("/root/python/user.csv",mode="r") as f:
            result=csv.DictReader(f)
            data.clear()
            for i in result:
                data.append(i)
            # f.close()
    except FileNotFoundError:
        print("没有文件使用原数据")


# 4.使用异常在 atm 取款机的项目中存款函数中： 如果输入的存款金额不是整数则提示：请输入整数； 如果输入的存款金额不是 100 的倍数或者是负数则提示：请输入 100 的倍数的正整数

# 存款
def CunKuan(userdata):
    money=input("请输入存款金额:")
    money=checkMoney(money)
    userdata["bat"]=str(float(userdata["bat"])+money)
    print(f"您已成功存款{money}，您当前的余额为：{userdata['bat']}")
    inputToReturn()
```

```python
# 检测金额输入
def checkMoney(m):
    try:
        m=float(m)
        if m <=0 or m%100!=0:
            raise Exception("金额必须为100倍数的正整数!")
        return m
    except ValueError:
        print("请输入整数! ")
        inputToReturn()
    except Exception as e:
        print(e)
        inputToReturn()

# 上机练习8
# 1.定义一个水果类，定义属性（名称和颜色），使用魔法方法，然后通过水果类，创建苹果对象、橘子对象、西瓜对象并分别添加上颜色属性，定义一个方法分别输出如：
#红色的苹果真好吃
#橙色的橘子真好吃
#绿色的西瓜真好吃
class Fruit:
    name=""
    color=""

    def __init__(self,name,color):
        self.name=name
        self.color=color

    def show(self):
        print(f"{self.color}的{self.name}真好吃")

    #调用
if __name__=="__main__":
    apple=Fruit("苹果","红色")
    orange=Fruit("橘子","橙色")
    watermelon=Fruit("西瓜","绿色")
    apple.show()
    orange.show()
    watermelon.show()


# 猫类Cat。属性:毛的颜色color，品种breed，亲和度love。方法:吃饭eat()
# 狗类Dog。属性:毛的颜色color，品种breed，忠诚度loyal。方法:吃饭eat()
# 要求 ： 使用封装、继承和多态，根据以上要求抽取父类为Animal，重写eat方法，输出打印如下：
# 有一只亲和度是10级的花色的波斯猫正在吃鱼.....
# 有一只忠诚度是9级的黑色的藏獒正在啃骨头.....
```

```
# 有一只亲和度是8级的白色的加菲猫正在吃鱼.....
# 有一只忠诚度是 6 级的棕色的茶杯犬正在啃骨头.....
class Animal:
    color=""
    breed=""

    def __init__(self,color,breed):
        self.color=color
        self.breed=breed

    def eat(self):
        print(f"有一只{self.color}的{self.breed}正在吃饭....")

class Cat(Animal):
    love=""

    def __init__(self, color, breed,love):
        super().__init__(color, breed)
        self.love=love

    def eat(self):
        print(f"有一只亲和度是{self.love}级{self.color}的{self.breed}正在吃
鱼....")

class Dog(Animal):
    loyal=""

    def __init__(self, color, breed,loyal):
        super().__init__(color, breed)
        self.loyal=loyal

    def eat(self):
        print(f"有一只忠诚度是{self.loyal}级{self.color}的{self.breed}正在啃骨头")

#调用
if __name__=="__main__":
    cat1=Cat("花色","波斯猫","10")
    dog1=Dog("黑色","藏獒","9")
    cat2=Cat("白色","加菲猫","8")
    dog2=Dog("棕色色","茶杯犬","6")
    list1=[cat1,dog1,cat2,dog2]
    for i in list1:
         i.eat()
```