面向对象设计和异常处理

学习目标

- 了解面向对象的内涵。
- 了解面向对象编程和面向过程编程的区别。
- 掌握类的创建和特性。
- 掌握方法的特性。
- 掌握异常处理机制。

对象

面向对象编程（Object Oriented Programming，OOP）是一种软件设计方法。随着科技水平的发展，软件编程的代码量日益增加，面向对象编程应运而生。面向对象编程主要针对大型软件设计而提出，它使得软件设计更加灵活，相对于面向过程编程来讲，能够更好地支持代码复用和设计复用，并且使得代码具有很好的可读性和扩展性。

- **封装性**：实现对象信息隐蔽，利用接口隐蔽对象内部的工作细节。
- **多态性**：不同类的对象使用相同的操作可以得到不同的执行结果。
- **继承性**：实质上就是通过建立专门的类来实现数据共享。

类

类（Class）是面向对象编程（OOP，Object-Oriented Programming）实现数据封装的基础。例如，“鸟”就是一个类，燕子、喜鹊、老鹰等都是属于“鸟”这个类当中的具体实例（具体某件事物），可以把“鸟”看作是所有鸟的集合，同样的“鸟”也属于“动物”这个行列，还可以把“鸟”看作是“动物”的子集，而“动物”是“鸟”的超类。

创建类

Python使用“class”关键字定义类，“class”关键字之后是一个空格，然后是类的名称，接着是一个冒号，最后换行并定义类的语句块实现。

```
>>> class Name:
        def init(self):
            print("My name is Li Hua!")
>>> name=Name()
>>> name.init()
My name is Li Hua!
```

可以使用内置方法“isinstance( )”来测试一个对象是否为某个类的实例。

```
>>> isinstance(name,Name)
True
```

创建类

Python使用“class”关键字定义类，“class”关键字之后是一个空格，然后是类的名称，接着是一个冒号，最后换行并定义类的语句块实现。

类名的首字母一般要大写，要达到顾名思义的效果，并在整个系统的设计和实现中保持风格一致，这有助于团队协作工作。

类的所有方法都必须至少有一个“self”参数，并且必须是方法的第一个形参（假设有多个形参），“self”参数代表创建的对象本身。当然“self”命名只是一个习惯，在实际操作中类的方法中第一个参数的名称是可以自定义的，非必需使用“self”这个名称，但是建议编写代码时仍以“self”命名。

```
>>> class Name:
        def init(self):
            print("My name is Li Hua!")
>>> name=Name()
>>> name.init()
My name is Li Hua!
```

可以使用内置方法“isinstance( )”来测试一个对象是否为某个类的实例。

```
>>> isinstance(name,Name)
True
```

类的特性和方法

“self”参数可以调用类中的变量和方法。

```
>>> class Students:
        def __init__(self,id_number):
            self.id=id_number
        def show(self):
            print(self.id)
>>> stu1=Students(9527)
>>> stu1.show()
9527
```

类的特性和方法

- **公有方法**：自定义的普通成员方法，如同公有成员一样可以通过对象名直接调用，可以访问属于类和对象的成员。
- **私有方法**：私有方法的名字以两个下划线“__”开始，如同私有变量一样不能通过对象名直接调用，可以访问属于类和对象的成员。
- **静态方法**：静态方法可以没有参数，可以通过类名和对象名调用，但不能直接访问属于对象的成员，只能访问属于类的成员。
- **类方法**：一般将“cls”作为类方法的第一个参数名称，但也可以使用其他命名的参数，在调用类方法时不需要为该参数传递值。可以通过类名和对象名直接调用，但不能直接访问属于对象的成员，只能访问属于类的成员。

类的特性和方法

公有方法、私有方法、静态方法和类方法之间的区别：

```
>>> class Root:
    __total = 0                                          #私有成员变量
    def __init__(self, v):    #构造方法/私有方法
        self.__value = v
        Root.__total += 1
    def show(self):           #公有方法
        print('self.__value:', self.__value)
        print('Root.__total:', Root.__total)
    @classmethod              #修饰器，声明类方法
    def classShowTotal(cls):      #类方法，只能访问类成员
        print(cls.__total)
    @staticmethod             #修饰器，声明静态方法
    def staticShowTotal():        #静态方法，只能访问类成员
        print(Root.__total)
>>> r = Root(3)
>>> r.classShowTotal()            #通过对象来调用类方法
1
>>> r.staticShowTotal()           #通过对象来调用静态方法
1
>>> r.show()
self.__value: 3
Root.__total: 1
>>> rr = Root(5)
>>> Root.classShowTotal()         #通过类名调用类方法
2
>>> Root.staticShowTotal()        #通过类名调用静态方法
2
```

类的继承

继承性是面向对象编程的重要特性之一，是为代码复用和设计复用而设计的。自定义一个新类时可以继承一个已有或设计好的类然后进行二次开发，这会大幅度减少开发的工作量。在继承关系中，已有的、设计好的类称为父类、基类或超类，自定义的新类称为子类或派生类。派生类可以继承父类的公有成员，但是不能继承其私有成员。

```
>>> class A:                          #父类
        def init(self):
            self.str=[]
        def ft(self,buf):
            return [n for n in buf if n not in self.str]
>>> class B(A):                       #子类
        def init(self):                #重写覆盖父类中的“init”方法
            self.str=[″11″]
>>> s=B()
>>> s.init()
>>> s.ft([″11″,″22″,″33″,″44″,″55″])
['22', '33', '44', '55']
```

异常

在运行程序的过程中难免会遇到各种非正常情况。例如，分母为零、下标越界、文件不存在、网络异常、类型错误、名字错误或磁盘空间不足等。如果这些错误得不到正确的处理将会导致程序终止运行，所以本结需要学习Python强大的异常处理机制，合理地使用异常处理可以使程序更加健壮，具有更强的容错性，不会因为错误地输入或运行时遇到的问题而造成程序终止。