1. 面向过程和面向对象

1.1. 面向过程

一种较早的编程思想，顾名思义该思想是站在过程的角度思考问题，强调的就是功能行为,功能的执行过程,即先干啥,后干啥。而每一个功能我们都使用函数(类似于方法)把这些步骤一步一步实现，使用的时候依次调用函数就可以了。

最小的程序单元是函数,每个函数负责完成某一个功能,用以接受输入数据,函数对输入数据进行处理,然后输出结果数据.

整个软件系统由一个个的函数组成,其中作为程序入口的函数称之为主函数,主函数依次调用其他函数,普通函数之间可以相互调用,从而实现整个系统功能.

1.2. 面向过程的不足

面向过程的设计,是采用置顶而下的设计方式,在设计阶段就需要考虑每一个模块应该分解成哪些子模块,每一个子模块有细分为更小的子模块,如此类推,直到将模块细化为一个个函数.

1):设计不够直观,与人类的习惯思维不一致.

2):系统软件适应性差,可拓展性差,维护性低.

面向过程最大的问题在于随着系统的膨胀，面向过程将无法应付，最终导致系统的崩溃。为了解决这一种软件危机，我们提出面向对象思想。

1.3. 面向对象

一种基于面向过程的新的编程思想，顾名思义该思想是站在对象的角度思考问题，我们把多个功能合理的放到不同对象里，强调的是具备某些功能的对象。

具备某种功能的实体，称为对象.

面向对象最小的程序单元是:类.

面向对象更加符合我们常规的思维方式，稳定性好，可重用性强，易于开发大型软件产品，有良好的可维护性。

在软件工程上，面向对象可以使工程更加模块化，实现更低的耦合和更高的内聚

三大特征：

1:封装(Encapsulation)；

2:继承(Inheritance)；

3:多态(Polymorphism)；

封装是指将对象的实现细节隐藏起来，然后通过公共的方法来向外暴露该对象的功能。

继承是面向对象实现软件复用的重要手段，当子类继承父类后，子类是一种特殊的父类，能直接或间接获

得父类里的成员。

多态是可以直接把子类对象赋给父类变量，但是运行时依然表现出子类的行为特征，这意味着同一类型的对象在运行时可能表现出不同的行为特征。

2. 类和对象

2.1. 对象和类

类是抽象的,通常情况下类是对一类事物的描述,包括行为和状态, 比如说人类(Person)

对象是一个具体事物的存在,通常情况下对于对象是看得见,摸得着的东西,是属于具体某一个类

类就是创建对象的模板,一个类可以创建多个对象

请分析一下下面的是对象还是类

手机(大小,价格,颜色,拍照,打电话)

我的那个华为手机(5.5 寸,2999,黑色,拍照,打电话)

电风扇(品牌,功率,价格,运转)

水杯(容量,材质,装水)

美女(身高,三围,唱歌,跳舞,姓名,年龄)

前天你在图书馆看见的那个高高的美女(170cm,90,90,90,唱歌,跳舞,)

凤姐(152cm)

2.2. 类的定义和对象的创建

定义一个类,其中语法格式是

```
class 类名(首字母大写):
    #定义类的相关方法
```

定义一个 Person 类

```
class Person:
   pass #pass 代表占位符,不会执行任何操作
```

创建一个 Person 对象

```
#创建对象(类的实例)的语法格式 变量名=类名([参数列表])
p=Person()
print(p)
```

2.3. 类的属性和方法

需求:给 Person 类添加两个属性 name,age,并且添加一个睡觉 sleep 的方法

```
#定义类
```

```
class Person:
    #__init__ 方法,用于给创建的对象进行初始化操作,其中 self 是代表当前所创建的对象,
    #在定义方法的时候需要 self 这个参数,但是调用的时候不需要
    def __init__(self,name,age):
        self.name=name#给当前对象添加属性 name 并赋值为 zhangsan
        self.age=age#给当前对象添加属性 age 并赋值为 18
    def sleep(self):#定义一个方法 sleep
        print("%s is sleeping"%self.name) #打印当前对象的 name
#创建对象 p
p=Person("jack",46);
#调用对象 p 的睡觉方法
p.sleep()
```

2.4. 老贾造车

需求: 电动汽车具有型号(name),最高时速(speed)等基本属性,具有行驶(run)和导航的功能(navigate)等功能

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def make(self):
        print('%s 开始造车工作' % self.name)
        c1 = ElectricCar("FF",200)
        print('%s 造车完毕' % self.name)
        return c1

    def drive(self, car, dest):
        print("%s 正要开%s 去%s" % (self.name,car.name, dest))


class ElectricCar:
    def __init__(self, name, speed):
        self.name = name
        self.speed = speed

    def run(self):
        print("小汽车正在呜呜呜的行驶")

    def navigate(self):
        print("导航可以带您走遍天下")
```

```
def main():
    laojia = Person("laojia", 30)
    c1 = laojia.make()
    print("车的型号", c1)
    laojia.drive(c1, "北京")
main()
```

3. 封装

封装:把对象的方法和数据看成一个整体,将两者存放到一个独立的模块中

信息封装: 对象中的数据隐藏起来,不能直接修改对象的数据,而是通过提供对应的方法完成数据的修改

在修改数据的时候我们可以在方法中进行判断,只有合法的数据才可以进行修改

属性

使用_开头,表示该属性是私有属性,在外面不要随便访问

```
class Person(object):
    def __init__(self,name,age):
        self._name=name
        self._age=age
p1=Person("jack",25)
p1._name # 还是可以正常输出 jack
```

使用__开头(双下划线)开头的私有化更加彻底,在类的外面是无法直接访问这种变量的,只能通过对应的方法进行访问

```
class Person(object):
   def __init__(self,name,age):
        self.__name=name
        self.__age=age
p1=Person("jack",25)
p1.__name
```

私有方法:

```
class Person(object):#括号里面的 object 代表的是继承 object 类
    def __init__(self,name,age):
        self.__name=name
        self.__age=age
    def __inner(self):#定义私有方法
        print(self.__name)
    def getName(self):#对私有属性提供 getter 方法
        return self.__name
    def setName(self,name):#对私有属性提供 setter 方法
```

```
        self.__name=name
p1=Person("jack",25)
print(p1.getName())
```

4. 继承

4.1. 继承的基本概念

继承表示的是类和类之间的所属关系,体现了代码的复用

子类创建的对象同样满足父类的行为特征

在定义类的时候,使用()来描述继承关系 ()中编写父类的名字即可

父类的属性,方法都会继承给子类

私有的属性,方法不会被子类继承,也不能被直接访问

```
class Person(object): #定义一个 Person 类
    def __init__(self,name,age):
        self.__name=name
        self.__age=age
    def getName(self):
        return self.__name
    def setName(self,name):
        self.__name=name
    def getAge(self):
        return self.__age
    def setAge(self,age):
        self.__age=age
class Student(Person):#表示 Student 类继承 Person 类
    pass
s1=Student("jack",26)
print(s1.getAge())
s1.setName("张三")
print(s1.getName())
```

4.2. 调用父类的方法

在子类访问父类的属性/方法的时候有两种方式

方式一: 父类名.方法名(self,[参数列表])

方式二: super().方法名([参数列表])

```
class Student(Person):#表示 Student 类继承 Person 类
    def study(self):
        print("好好学习,天天向上",Person.getName(self))#方式一
        print("学习使人进步",super().getName())#方式二
```

4.3. 方法的覆盖

如果在子类中定义的方法和父类的方法名称一样,那么就会覆盖父类的方法

```
class Person(object): #定义一个 Person 类
    def eat(self):
        print("我们需要吃东西才可以补充能量")
class Student(Person):#表示 Student 类继承 Person 类
    def study(self):#自定义的方法
        print("好好学习,天天向上")
    def eat(self):#覆盖父类的方法
        print("作为学生,我们应该多吃粗粮")
s1=Student()
s1.eat()
```

```
class Person(object):#括号里面的 object 代表的是继承 object 类
    def __init__(self,name,age):
        print("11111")
        self.__name=name
        self.__age=age
    def getName(self):#对私有属性提供 getter 方法
        return self.__name
    def setName(self,name):#对私有属性提供 setter 方法
        self.__name=name
class Student(Person):
    def __init__(self,name,age,sn):
        super().__init__(name,age)#先调用父类的初始化操作
        self.__sn=sn
    def getSn(self):
        return self.sn
    def setSn(self,sn):
        self.__sn=sn
s1=Student("hesj",18,"001")
s1.getName()
```

4.4. 多继承

在 Python 中,一个类可以继承多个父类,多个父类之间使用逗号隔开

```
class Base(object):
    def test(self):
        print("------Base-----test---------")
class A(Base):
    def test1(self):
        print("------A-------test1----------")
```

```
class B(Base):
    def test2(self):
        print("------B-------test2----------")
class C(B,A):
    pass
c=C()
c.test()
c.test2()
c.test1()
```

如果在两个父类中都有同一个方法,那么其执行顺序按照记录的先后顺序执行(C.__mro__),现在采用的是 C3 算法计算其中的执行信息

```
class Base(object):
    def test(self):
        print("------Base-----test---------")
    def test3(self):
        print("------Base-------test3----------")
class A(Base):
    def test1(self):
        print("------A-------test1----------")
    def test3(self):
        print("------A-------test3----------")
class B(Base):
    def test2(self):
        print("------B-------test2----------")
    def test3(self):
        print("------B-------test3----------")
class C(B,A):
    pass
c=C()
print(C.__mro__)#方法解析顺序
c.test3()
```

5. 多态

定义时的类型和运行时的类型不一样,称之为多态,

鸭子类型:是动态类型的一种风格。在这种风格中，一个对象有效的语义，不是由继承自特定的类或实现特定的接口，而是由"当前方法和属性的集合"决定'

```
class Person:
    def introduce(self):
        self.speak()
    def speak(self):
        print("我简单介绍一下我自己")
class Student(Person):
```

```
    def speak(self):
        print("我是一个三好学生,爱学习,爱劳动")
p1=Person()
p1.introduce()
s1=Student()
s1.introduce()
```

6. 类属性

实例属性: 对于不同的对象(实例),其属性值是不一样的.也就是说每个对象都有自己的属性值

类属性:对于所有的对象(实例),其属性值都是一样的,换句话说,类属性是所有对象所共享的

```
class Person:
    num=0 #直接在类上面定义的变量,是属于类的变量,称之为类变量
    def __init__(self):
        Person.num+=1#访问方式是类名.变量名称
p1=Person()
p2=Person()
p3=Person()
print("对象数量:",Person.num)
```

7. 类方法和静态方法

类方法:在类中定义,使用一个@classmethod 注解修饰的方法,我们称之为类方法

静态方法:在类中定义,使用一个@staticmethod 注解修饰的方法,我们称之为静态方法

```
class Base:
    count=0 #定义一个类变量
    def test1(self):#定义一个普通的实例方法
        print("---Base -----test1 ---")
    @classmethod
    def test2(cls):#定义一个类方法
        print("---Base -----test2 ---")
        cls.count+=1
        print(cls.count)
    @staticmethod
    def test3():#定义一个静态方法
        print("----Base-----test3----")
b=Base()
Base.test2()#类方法调用
Base.test3()#静态方法调用
```

8. 构造和析构

构造函数:通常指的是用来创建对象和给对象进行初始化操作的函数

析构函数:在创建的对象进行销毁的时候需要执行的函数,用户做相关的清理工作和资源的释放,他与构造函数的功能正好相反

__new__:创建对象的函数,接受一个参数 cls,字节码对象,用于创建对象,并且需要返回当前所创建的对象

__init__:对创建的对象进行初始化赋值的操作

__del__:当一个对象在内存中被销毁的时候会执行该方法

```
#把下面这些代码保存到文件 Person.py 中去
class Person(object):#括号里面的 object 代表的是继承 object 类
    #创建一个对象,cls 代表的是当前字节码对象,在调用的时候会自动传递
    def __new__(cls,name,age):
        print("Person new")
        return object.__new__(cls)#调用 object 的__new__方法进行创建对象,并且返回
    def __init__(self,name,age):
        print("Person init...")
        self.name=name
        self.age=age
    def __del__(self):#在对象销毁的时候会自动调用__del__方法
        print("Person destroy...")

p1=Person("jack",25)
print(p1.name)
print(123)
```

9. 魔法方法和魔法属性

在 python 中 ,有一些内置好的特定的方法 ,这些方法在进行特定的操作时会自动被调用 ,称之为魔法方法 ,下面介绍几种常见的魔法方法。

对于魔法方式 总是以双下划线开始,双下划线结束

__str__:在将对象转换成字符串 str(对象) 测试的时候，打印对象的信息 __str__方法必须要 return 一个字符串类型的返回值

```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def __str__(self):
        return "My name is {0.name}, My age is {0.age}".format(self)
p1=Person("zhangsan",18)
```

```
print(p1)
```

__repr__:和__str__相同,也是打印字符串的信息,但是对于开发人员使用比较方便,不过在交互界面直接输入对象名的时候会调用__repr__方法输出对象的信息

```
    def __repr__(self):
        return "My name is {0.name}, My age is {0.age}".format(self)
```

__class__:获取到对象实例的字节码对象

```
p1.__class__ #输出__main__.Person
```

__mro__:显示指定类的所有继承脉络和继承顺序,在多继承的时候对于方法和属性寻找的先后顺序

```
print(C.__mro__)
```

__len__:定义当被 len() 调用时的行为,一般对于容器才会实现该方法

__call__:允许一个类的实例像函数一样被调用,可以通过类名() 直接调用 __call__ 方法