```python
# 面向对象
# 类：具有相同特征行为的某一类事物的总称    抽象出来
# 对象：某个类中具体的一个实例(属性、方法)

# 人类
# 张三  18  身高  体重(特征)      跑步、睡觉、游泳(行为)
# 李四  20  身高  体重(特征)      跑步、睡觉、游泳(行为)

class people:  # class 类名:
    name = '无名氏'  # 类属性

    # 构造方法：完成对象的初始化,在创建对象的时候系统会默认调用
    def __init__(self,name,age,address):
        self.name = name
        self.age = age  # 实例属性
        self.address = address

    def say(self): # 类中的方法都会自带 self 参数,表示当前类的所有属性和方法
        print('我是一个正直的人。')

# 创建对象 : 对象名称 = 类名()
zhangsan = people('张三',20,'长沙')
print(zhangsan.name)  # 调用对象的属性和方法:   对象名.属性或者方法名称
print(zhangsan.age)
print(zhangsan.address)
zhangsan.say()
print(people.name)


# lisi = people('李四',28,'上海')
# lisi.age = 21
# print(lisi.name)
# print(lisi.age)
# print(lisi.address)
# lisi.say()


# 内置属性：python 中系统自带的类属性
class people:
    '''我是 people 类的说明,我只能放在第一行'''
    name = '无名氏'
    def __init__(self,name,age):
        self.name = name
        self.age = age
```

```python
# 访问类属性：类名.内置属性
print(people.__dict__)  # 打印出类的所有属性
print(people.__doc__)   # 类的文档字符串，只能放在第一行
print(people.__name__)  # 类名
print(people.__module__) # 类定义所在的模块
print(people.__bases__)  # 类的所有父类构成元素



# 属性(类属性、实例属性、内置属性)
# 方法(实例方法、类方法、静态方法)
class cat:
    '''我是cat类'''
    name = '无名氏'  # 类属性

    def __init__(self,cat_name,cat_sex,cat_age):
        self.cat_name = cat_name  # 实例属性
        self.cat_sex = cat_sex
        self.cat_age = cat_age

    # 实例方法(类当中的普通方法)
    def say(self):  # 第一个参数默认是self
        print('我是一只喵喵,我是实例方法')

    @classmethod  # 修饰器：他下面的方法就是类方法
    def run(cls):  # 第一个参数默认是cls
        print('我是一个类方法')

    @staticmethod  # 他下面的方法就是静态方法
    def sleep():  # 静态方法没有默认的参数
        print('我是一个静态方法')

tom = cat('汤姆','公',2)
# 1、通过 对象、类名 访问 类属性
# print(tom.name)
# print(cat.name)
# 2、只能通过对象访问实例属性
# print(tom.cat_name)
# 3、通过 对象、类名 访问内置属性
# print(tom.__doc__)
# print(cat.__doc__)

# 4、通过 对象、类名 访问实例方法
```

```
# tom.say()
# cat.say('汤姆') # 注意：此时 self 需要传值,会被当作普通的方法
# 5、通过 对象、类名 访问类方法
# tom.run()
# cat.run()
# 6、通过 对象、类名 访问静态方法
tom.sleep()
cat.sleep()



# 内置方法
class people:
    # 1、构造方法：有构造方法系统会默认调用,支持重载,但是系统会调用最后一个
    def __init__(self,name,age,sex):
        self.name = name
        self.age = age
        self.sex = sex

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

    # 2、析构方法：作用是释放资源,系统默认调用
    def __del__(self):
        print('我是析构方法,我是在程序运行完之后自动释放资源')
    # 3、自定义实例输出方法
    def __str__(self):
        return '我是实例输出方法'
    # 4、两个实例的加法操作(实现对象属性值的加法操作)
    def __add__(self, other):
        return self.age + other.age

zhangsan = people('张三',20)
lisi = people('李四',17)
# del zhangsan  # 手动释放资源
# print('---------')
print(zhangsan)
print(zhangsan.age + lisi.age)
print(zhangsan + lisi)
```

```python
# 封装：数据封装、方法封装
# 封装的作用：封装数据的主要原因是保护隐私；封装方法的主要有因是隔离复杂度。

class people:
    def __init__(self,name,id,address):
        self.name = name
        # 数据的封装：私有的属性,只能在类的内部使用,在类的外部是访问不了的
        self.__id = id
        self.address = address
    # 方法的封装：通过共有的方法来访问私有的属性
    def get_id(self): # 获取到 id
        return self.__id
    def set_id(self,id):  # 设置值
        self.__id = id
    def __info(self):
        print('我是私有方法')

zhangsan = people('张三','123456789','深圳')
# print(zhangsan.name)
# print(zhangsan.get_id())
# zhangsan.set_id('888')
# print(zhangsan.get_id())

# 在类的外部访问私有属性、方法   实例._类名__方法名()
zhangsan._people__info()
print(zhangsan._people__id)




# 练习：
# 1、创建一个银行卡类
#    属性：卡号、密码(私有)、姓名(私有)、钱(私有)、身份证(私有)
#    方法：存钱、取钱、查询余额、修改密码(判断原有密码是否正确)
# 2、创建对象进行测试
class bank_card:
    def __init__(self,card_id,passwd,name,money,id):
        self.card_id = card_id
        self.__passwd = passwd
        self.__name = name
        self.__money = money
        self.__id = id
    def save_money(self,num):
```

```
        self.__money = self.__money + num
    def get_money(self,num):
        self.__money = self.__money - num
    def money_info(self):
        print('当前余额是：{}'.format(self.__money))
    def change_passwd(self,old_passwd):
        if self.__passwd == old_passwd:
            self.__passwd = int(input('请重新输入密码：'))
            print('密码修改成功！')
        else:
            print('密码不正确。')

# 测试类
test_card = bank_card(666888,123,'汤姆',100,456789)
test_card.money_info()
test_card.save_money(50)
test_card.money_info()
test_card.get_money(20)
test_card.money_info()
test_card.change_passwd(789)




# 继承：一个类获取另外一个类的属性和方法的过程
#       子类获取父类的属性或者方法
# 人类：属性 name、age、sex  方法：say()、sleep()
# 老师：属性 name、age、sex、职称、科目  方法：say()、sleep()、教学生...
# 学生：属性 name、age、sex、班级      方法：say()、sleep()、喜好...
class people:  # 父类
    def __init__(self,name,age,sex):
        self.name = name
        self.age = age
        self.sex = sex
    def say(self):
        print('我是一个刚正不阿的人。')
    def sleep(self):
        print('我每天都需要睡觉。')
    # 私有的属性、方法是不能被继承的
    def __DriverCar(self):
        print('我有驾照，我会开车。')

class student(people):  # 子类 继承于people类
    # 当子类没有构造方法时可以调用父类的构造方法
```

```
    # 当子类有构造方法时,子类必须手动调用父类的构造方法,一般都需要实现父类的构造方法
    def __init__(self,name,age,sex,class_name):
        # 方法1:类名.__init__(self,...)
        # people.__init__(self,name,age,sex)
        # 方法2:super().__init__(参数列表)   不需要self
        # super().__init__(name,age,sex)
        # 方法3:super(子类名称,self).__init__(参数列表)
        super(student,self).__init__(name,age,sex)
        self.class_name = class_name
    def study(self):
        print('我是一个很会学习的学生。')
    def like(self):
        print('我爱好看四大名著。')
    # 子类重写了父类的方法,执行方法时先会在自身类中查找,没找到则去父类中查找
    def sleep(self):
        print('我一放假我就忍不住要睡个饱。')

make = student('马克',19,'男','测试7班')
print(make.name)
print(make.age)
print(make.class_name)
make.say()
make.study()
make.sleep()




# 多继承:一个子类有多个父类
# 人类、老师--司机
class people:
    def __init__(self,name):
        self.name = name
        print('我是people的构造方法')
    def say(self):
        print('我是people类')

class teacher:
    def __init__(self,CourseName):
        self.CourseName = CourseName
        print('我是teacher的构造方法')
    def say(self):
        print('我是一个老师')
```

```python
class driver(people,teacher):  # 多继承,有多个父类
    # 当父类中有同名的方法,那么就按照继承的先后顺序执行
    # 调用多个父类的构造方法
    def __init__(self,name,CourseName,driver_id):
        people.__init__(self,name)
        teacher.__init__(self,CourseName)
        self.driver_id = driver_id
    def say(self):
        print('我是一个兼职的滴滴司机。')
    def chaoche(self):
        print('我车技很溜,别人只能看到我的尾灯。')

zhangsan = driver('张三','语文',3421)
print(zhangsan.name)
print(zhangsan.CourseName)
print(zhangsan.driver_id)
zhangsan.say()
```

```python
from abc import ABCMeta,abstractmethod
# 抽象类
# 1、包含抽象方法
# 2、抽象只定义不实现
# 3、要有定义抽象类的定义语句
# 4、抽象类是用来被继承的(子类必须实现父类的抽象方法),如果没有被继承那么毫无意义
class animal(metaclass=ABCMeta): # 此写法是必须实现抽象方法的
    # 定义抽象类
    # __metaclass__ = ABCMeta  # 此写法不是强制的必须实现抽象方法 (不推荐)
    def __init__(self,name):
        self.name = name
    # 抽象方法
    @abstractmethod
    def say(self):  # 只声明不实现
        pass

class dog(animal):
    def say(self): # 实现抽象方法
        print('我是 dog 类重写 animal 类的 say 方法')
    def sleep(self):
        print('我也要睡觉。')
```

```
xiaohei = dog('小黑')
print(xiaohei.name)
xiaohei.sleep()
xiaohei.say()

# 人类---抽象类---抽象方法：goto_class_home
# 老师---去教室教书
# 学生---去教室读书
```