Day-6

- Video-001-面向对象
  - 基本概念
    - 面向过程：类似于函数，只能执行，但是没有返回值
      - **侧重怎么做**
        - 把完成某一个需求的**所有步骤从头到位逐步**实现
        - 根据开发需求，将某些功能独立的代码封装成一个又一个函数
        - 最后完成的代码，就是**顺序地调用不同的函数**
      - 特点
        - 1.注重步骤和过程，不注重职责分工
        - 2.如果需求复杂，代码会变得很复杂
        - 3.开发复杂项目，没有固定的套路，开发难度很大
    - 面向对象：更大的封装，**根据职责在一个对象中封装多个方法**
      - **侧重谁来做**
        - 在完成某一个需求前，首先确定职责---要做的事情（方法）
        - **根据职责确定不同的对象，在对象内部封装不同的方法（多个）**
        - 最后完成的代码，就是**顺序地让不同的对象调用不同的方法**
      - 特点
        - 1.**注重对象和职责**，不同的对象承担不同的职责
        - 2.更加**适合应对复杂的需求变化**，是专门应对复杂项目开发，提供的固定套路
        - 3.需要在面向过程基础上，再学习一些面向对象的语法
    - 类 & 对象
      - 类：对一群具有相同特征或者行为的事物的一个统称，**抽象的，不能直接使用，**负责创建对象
        - 特征——属性
        - 行为——方法
      - 对象：由类创建出来的一个**具体存在**，可以直接使用
        - 由哪个类创建出来的对象，就拥有在哪一个类中定义的属性和方法
      - **先有类，再有对象**
      - 关系
        - **类是模版，对象是根据类这个模版创建出来的**
        - 不同对象之间属性可能不同
        - **类中定义的属性和方法，创建出来的对象就有什么属性和方法**

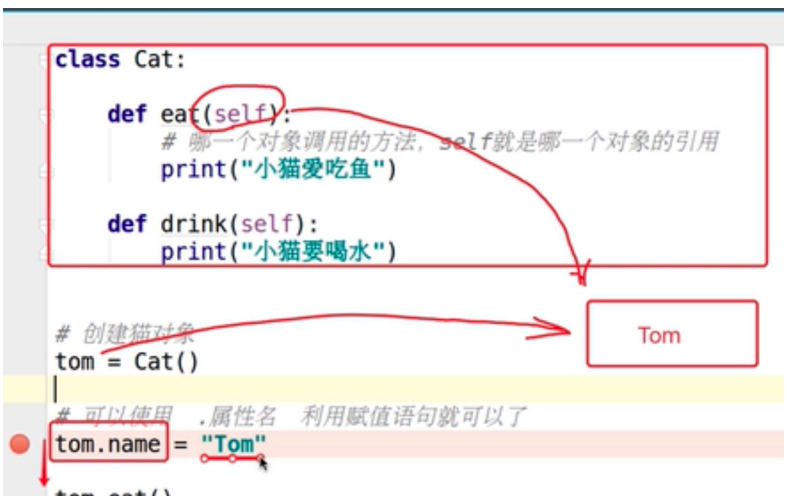
- 类的设计
  - 首先应分析需求，确定程序中需要包含哪些类
  - 满足三要素
    - **类名** 这类事物的名字，**满足大驼峰命名法**
      - **名词提炼法**，分析整个业务流程，出现的 名词 通常就是找到的类
    - **属性** 这类事物具有什么样的特征
      - **对象的特征描述**
    - **方法** 这类事物具有什么样的行为
      - **对象具有的行为**
- 面向对象基础语法
  - 查看对象方法&属性的方法——dir(标识符/数据)函数
    - 在 **标识符 / 数据** 后输入一个 .，然后按下 TAB 键，iPython 会提示该对象能够调用的 **方法列表**
    - 使用内置函数 dir 传入 **标识符 / 数据**，可以查看对象内的 **所有属性及方法**
      - **__方法名__，该格式的方法是Python内置的方法和属性**
  - 定义简单的类
    - 1.封装方法，class 类名，第一个参数必须是self
      - 在 `Python` 中要定义一个只包含方法的类，语法格式如下：

```
class 类名:

    def 方法1(self, 参数列表):
        pass

    def 方法2(self, 参数列表):
        pass
```

      - **方法** 的定义格式和之前学习过的**函数** 几乎一样
      - 区别在于第一个参数必须是 `self`，大家暂时先记住，稍后介绍 `self`
    - 2.创建对象
      - 语法：对象变量 = 类名()
    - **self：哪一个对象调用的方法，self就是哪一个对象的引用**



        - 使用self在方法内部输出每一只猫的名字
            - 在类封装的方法内部，**self就表示当前调用方法的对象自己**
            - **调用方法时，程序员不需要传递self参数**
            - 在方法内部
                - 可以通过self.访问对象的属性
                - 也可以通过self.调用其他的对象方法
        - **不推荐在类的外部给对象增加属性**
- 对象初始化方法
    - 当使用 **类名()** 创建对象时，会**自动执行**以下操作
        - 1.为对象在内存中分配空间——创建对象
        - 2.为对象的属性设置初始值——初始化方法（init）
            - **__init__方法是专门用来定义一个类具有哪些属性的方法**
    - 在初始化方法内部定义属性
        - 在__init__方法内部使用**self.属性名 = 属性的初始值** 就可以定义属性
        - 定义属性之后，在使用类创建的对象，都会拥有该属性
    - 初始化的同时设置初始值：可以对__init__方法进行改造
        - 把希望设置的属性值，定义成 __init__ 方法的参数
        - 在方法内部使用 self.属性 = 形参 接收外部传递的参数
        - 在创建对象时，使用 类名(属性1, 属性2...) 调用
- 内置方法
    - __del__方法
        - 当使用 类名() 创建对象时，为对象 **分配完空间**后，**自动** 调用 __init__ 方法
        - 当一个 **对象被从内存中销毁** 前，会 **自动** 调用 __del__ 方法
    - 生命周期
        - 一个对象从调用 类名() 创建，生命周期开始
        - **一个对象的 __del__ 方法一旦被调用，生命周期结束**
        - 在对象的生命周期内，可以访问对象属性，或者让对象调用方法
    - __str__方法
        - 在 Python 中，使用 print 输出 **对象变量**，默认情况下，会输出这个变量 **引用的对象** 是 **由哪一个类创建的对象**，以及 **在内存中的地址（十六进制表示）**
        - 如果在开发中，**希望使用 print 输出 对象变量 时，能够打印 自定义的内容，就可以利用 __str__ 这个内置方法了**
        - **必须返回一个字符串**
- 封装
    - **封装** 是面向对象编程的一大特点

        - 面向对象编程的 **第一步** —— 将 **属性** 和 **方法** **封装** 到一个抽象的 **类** 中
        - **外界** 使用 **类** 创建 **对象**，然后 **让对象调用方法**
        - **对象方法的细节** 都被 **封装** 在 **类的内部**
        - 注意⚠️
            - **在对象的方法内部，可以直接访问对象的属性**
            - **同一个类创建的多个对象之间，属性互不干扰**
            - **一个对象的属性可以是另一个类创建的对象**
            - **定义属性没有初始值，可以设置为None，表示一个空对象，没有属性和方法**
    - 身份运算符：**用于比较两个对象的内存地址是否 一致**——是否是对同一个对象的使用
        - 在 `Python` 中针对 `None` 比较时，建议使用 `is` 判断

运算符	描述	实例
is	is 是判断两个标识符是不是引用同一个对象	x is y，类似 id(x) == id(y)
is not	is not 是判断两个标识符是不是引用不同对象	x is not y，类似 id(a) != id(b)

is 与 == 区别：

`is` 用于判断 **两个变量** **引用对象是否为同一个** `==` 用于判断 **引用变量的值** 是否相等

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> b is a
False
>>> b == a
True
```

- 引用概念
    - 在面向对象开发中，**引用**的概念是同样适用的！
    - 在 Python 中使用类 **创建对象之后**，tom 变量中 仍然记录的是 **对象在内存中的地址**
    - 也就是 tom 变量 **引用** 了 **新建的猫对象**
    - 使用 print 输出 **对象变量**，默认情况下，是能够输出这个变量 **引用的对象** 是 **由哪一个类创建的对象**，以及 **在内存中的地址（十六进制表示）**
    - 提示：在计算机中，通常使用 **十六进制** 表示 **内存地址**
    - **十进制** 和 **十六进制** 都是用来表达数字的，只是表示的方式不一样
    - **十进制** 和 **十六进制** 的数字之间可以来回转换
    - %d 可以以 **10 进制** 输出数字
    - %x 可以以 **16 进制** 输出数字
- 私有属性 & 私有方法：只希望在对象的内部被使用，而不希望在外部被访问到
    - 对象不希望公开的属性 & 方法
    - 定义方式：在 **定义属性或方法时**，在 **属性名或者方法名前** 增加 **两个下划线**，定义的就是 **私有** 属性或方法
    - 伪私有属性和私有方法：**Python中并没有真正的私有**

        - 在给 **属性**、**方法** 命名时，实际是对 **名称** 做了一些特殊处理，使得外界无法访问到
        - **处理方式**：在 **名称** 前面加上 _类名 => _类名__名称
        - **但不要使用这种方法访问私有属性和私有方法**
- 三大特性
    - **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
        - 概念：子类拥有父类的所有方法和属性
        - 语法

```
class 类名(父类名):

    pass
```

        - 作用
            - **子类** 继承自 **父类**，可以**直接 享受 父类中已经封装好的方法**，不需要再次开发
            - **子类** 中应该根据 **职责**，封装 **子类特有的 属性和方法**
        - **传递性：子类拥有父类以及父类的父类中封装的所有属性和方法**
        - 方法的重写：**当父类当方法实现不能满足子类当需求，可以对方法进行重写**
            - 1.覆盖父类的方法
                - 如果在开发中，**父类的方法实现** 和 **子类的方法实现**，**完全不同**
                - 就可以使用 **覆盖** 的方式，**在子类中 重新编写** 父类的方法实现
            - 2.对父类方法进行拓展
                - 如果在开发中，**子类的方法实现** 中 **包含 父类的方法实现**
                    - **父类原本封装的方法实现** 是 **子类方法的一部分**
                - 就可以使用 **扩展** 的方式
                    - **在子类中 重写** 父类的方法
                    - 在需要的位置**使用 super().父类方法** 来调用父类方法的执行
                    - 代码其他的位置针对子类的需求，编写 **子类特有的代码实现**
            - 3.父类的私有属性 & 私有方法
                - **子类对象 不能** 在自己的方法内部，**直接** 访问 父类的 **私有属性** 或 **私有方法**
                - **子类对象 可以通过 父类 的 公有方法 间接 访问到 私有属性 或 私有方法**
                    - **私有属性、方法** 是对象的隐私，不对外公开，**外界** 以及 **子类** 都不能直接访问
                    - **私有属性、方法** 通常用于做一些内部的事情
        - 多继承：子类可以拥有多个父类，并且具有所有父类的属性和方法

- 语法

```
class 子类名(父类名1, 父类名2...)
    pass
```

- 同名的方法，调用顺序，应该避免父类之间存在同名的属性或者方法
  - MRO搜索方法
    - Python 中针对 **类** 提供了一个 **内置属性** __mro__ 可以查看 **方法** 搜索顺序
    - MRO 是 method resolution order，主要用于 **在多继承时判断 方法、属性 的调用 路径**
  - 程序执行过程
    - 在搜索方法时，是按照 __mro__ 的输出结果 **从左至右** 的顺序查找的
    - 如果在当前类中 **找到方法，就直接执行，不再搜索**
    - 如果 **没有找到，就查找下一个类** 中是否有对应的方法，**如果找到，就直接执行，不再搜索**
    - 如果找到最后一个类，还没有找到方法，程序报错
- 新式类：以object为基类的类，**推荐使用**
  - python3.0中会默认使用object作为该类的基类
  - python2.x中如果没有指定父类，则不会以 object 作为 **基类**
  - **新式类和经典类在多继承时，方法的搜索顺序有变化**

- **多态** 不同的子类对象调用相同的父类方法，产生不同的执行结果，**增加代码的灵活度**
  - 多态可以增加代码的灵活度
  - **以继承和重写父类方法为前提**
  - 是调用方法的技巧，不会影响到类的内部设计
- **面向对象三大特性总结**
  - **封装** 根据 **职责** 将 **属性** 和 **方法** **封装** 到一个抽象的 **类** 中
    - 定义类的准则
  - **继承** **实现代码的重用**，相同的代码不需要重复的编写
    - 设计类的技巧
    - 子类针对自己特有的需求，编写特定的代码
  - **多态** 不同的 **子类对象** 调用相同的 **父类方法**，产生不同的执行结果
    - **多态** 可以 **增加代码的灵活度**
    - 以 **继承** 和 **重写父类方法** 为前提
    - 是调用方法的技巧，**不会影响到类的内部设计**

- **程序执行时传入不同的对象就会产生不同的结果**

以上内容整理于 幕布文档