day06 课堂笔记

课程之前

复习和反馈

列表中的内容都是字典，根据字典的某个键 进行排序

列表.sort(key=lambda x: x['键']) # sort 函数内部做的，会自动调用,会将 列表中的数据作为参数进行传递

bool 类型: True 就是 1, False 0
数字作为 bool 类型: 0 是 False, 非 0 是 True

```
a = [1, True, 2]
a.count(1)
2
if a[-1]:
...     print(True)
... else:
...     print(False)
...
True
```

作业

```
# 循环 .... else  语法
for 变量 in xxx:
    if xx:
        break  # 如果循环执行了 break,就不再执行 else 中的代码
else:
    # 循环不是被 break 终止的,可以执行
```

今日内容

```
面向对象 (class)
- 概念(类/对象)
- 封装(定义类, 调用里边的方法(函数))
```

面向对象 介绍

面向对象(oop) 是一种编程方法，编程思想(即指导如何写代码)，适用于 中大型项目
面向过程也是一种编程思想，适用于小型项目

面向过程 和 面向对象 都可以实现某个编程的目的.

面向过程 考虑的是 实现的细节
面向对象 考虑的是 结果(谁能做这件事)

类 和 对象

类和对象是面对对象编程中 最重要的两个概念

- **类**: 是对具有相同特征或者行为的事物的一个统称, 是抽象的,不能直接使用
  - 指代 多个事物
  - 代码中 类是由关键字 `class` 定义
- **对象**:是由类创建出来的一个具体存在的事物, 可以 直接使用
  - 指代 一个具体事物
  - 代码中 使用 类 去创建(实例化)

```
苹果   ----->  类
红苹果   ----> 类
小明手里的那个红苹果     ----> 对象

狗     ----> 类
大黄狗   -----> 类
李四家的那条 大黄狗   -----> 对象
```

类的构成

```
类的构成，类的三要素
1，类名（多个事物 起一个名字,标识符规则，见名知意，类名用例满足大驼峰命名法(所有单词的首字母大写)）
2，属性 （事物的特征）
3，方法 （事物的行为）
```

面向对象代码的步骤

```
1，设计类（找类的三要素）
2，定义类
3，创建对象(实例化对象)
4，由对象调用类中的方法
```

类的设计

类的设计 就是找三要素. 属性和方法 可能会有很多,我们只需要找到关注的即可

类名的提取:使用名词提炼法 分析整个业务流程，得出的名词，通 常就是类名

需求:

- **小明** 今年 **18 岁**，**身高 1.75**，每天早上 **跑** 完步，会去 **吃** 东西
- **小美** 今年 **17 岁**，**身高 1.65**，小美不跑步，小美喜欢 **吃** 东西

```
类名: 人类 Person, People , Human
属性: 姓名(name), 年龄(age), 身高(height)
方法: 跑(run)/ 吃(eat)
```

需求:

- 一只 黄颜色 的 狗狗 叫 大黄
- 看见生人 汪汪叫
- 看见家人 摇尾巴

```
类名: 狗类(Dog)
属性: 颜色(color), 名字(name)
方法: 叫(bark),  摇尾巴(shake)
```

需求：

- 进入某 Web 项目登录页面，输入用户名、密码、验证码之后，点击登录按钮可以登录系统

```
类名: LoginPage
属性: 用户名(username), 密码(password), 验证
码(verify_code), 登录按钮(login_btn)
方法: 登录方法(login)
```

面向对象基本语法

类的基本使用

1. 定义类

```
在Python 中定义类使用关键字 class.
```

```
class 类名:
    # 在 class 的缩进中定义类的属性和方法,
    def 方法名(self):  # 方法的本质是函数
        pass
```

2. 创建对象(实例化对象)

```
在代码中，对象是由类对象.
类名()  # 就是创建对象
# 一般使用变量将创建的对象保存起来
变量 = 类名()  # 一般将这个变量 称为是对象，本质，变量中保存的是对象的引用地址
```

3. 调用类中的方法

```
由类创建的对象，可以调用类中的方法
- 语法
对象.方法名()
```

案例

```
需求:小猫 爱 吃鱼，小猫 要 喝水
类的设计:
类名: 猫(Cat)
属性: 暂无
方法: 吃鱼(eat), 喝水(drink)
```

```
class Cat:
    def eat(self):  # self  暂时不管
        """吃鱼的方法"""
        print('小猫爱吃鱼...')

    def drink(self):
```

```
        """喝水的方法"""
        print('小猫要喝水')


# 创建对象
tom = Cat()
# 通过对象 调用类中的方法
tom.eat()
tom.drink()
```

self 参数

1，参函数的语法上来看，self 是形参，名字可以任意的变量名，只是我们习惯性叫 self
2，特殊点：self 是一个普通的参数，按照函数的语法,在调用的时候,必须传递实参值，原因，是 Python 解释器自动的将 调用这个方法的对象作为参数传递给 self

所以 self 就是调用这个方法对象

```
class Cat:
    def eat(self):  # self 是调用这个方法的对象
        """吃鱼的方法"""
        print(f'self:{id(self)}')
        print('小猫爱吃鱼...')
```

```
# 创建对象
tom = Cat()
# 通过对象 调用类中的方法
print(f"tom :{id(tom)}")
tom.eat()  # tom  调用 ,self 就是 Tom

blue_cat = Cat()
print(f'blue:{id(blue_cat)}')
blue_cat.eat()  # blue_cat 调用, self 就是 blue_cat
```

属性

```
属性表示事物的特征.
可以给对象添加属性 或者获取对象的属性值.

给对象添加属性:
对象.属性名 = 属性值  # 添加或者修改

获取对象的属性值:
对象.属性名

在方法中操作属性(self 是对象):
self.属性名 = 属性值
self.属性名
```

```python
class Cat:
    def eat(self):  # self 是调用这个方法的对象
        """吃鱼的方法"""
        print(f'self:{id(self)}')
        print(f'小猫{self.name}爱吃鱼...')


# 创建对象
tom = Cat()
# 通过对象 调用类中的方法
print(f"tom :{id(tom)}")
# 给 Tom 对象添加 name 属性
tom.name = '汤姆'
print(tom.name)
tom.eat()

blue_cat = Cat()
print(f'blue:{id(blue_cat)}')
blue_cat.name = '蓝猫'
blue_cat.eat()
```

- 练习

  Cat 添加属性 姓名，年龄
  eat 方法中打印输出 小猫 xx，xx 岁 爱吃鱼

魔法方法

在Python 中存在一类方法，以两个下划线开头，两个下划线结尾，在满足某个条件的情况下,会自动调用， 这一类方法 称为是魔法方法

怎么学习:
1，什么情况下会自动调用(自动调用的时机)
2，应用场景
3，注意事项

初始化方法 `__init__`

1，调用时机
    在创建对象之后,会自动调用.
2，应用场景
    初始化对象，给对象添加属性
3，注意事项
    - 不要写错
    - 如果 属性是会变化的，则可以将这个属性的值作为参数传递，在创建对象的时候,必须传递实参值

```
class Cat:
    def __init__(self, name):
        print('我是 init 方法，我被调用了')  # 验证使用,正式代码不要
```

```
        self.name = name

    def eat(self):
        print(f"小猫 {self.name} 爱吃鱼")


# init 方法 创建对象之后 会自动调用
# 1 会  2 不会
# Cat  # 2 不是创建对象
# Cat()   # 1 因为是创建对象

# tom = Cat   # 2  不是创建对象，即 tom  也是类
#
# blue = Cat()  # 1 创建对象
# b = blue  # 2 不是创建对象，只是引用的传递
#
# t = tom()   # 1，tom  已经是类，类名()  就是创建对象

blue_cat = Cat('蓝猫')
blue_cat.eat()

black_cat = Cat('黑猫')
black_cat.eat()
```

__str__ 方法

1，调用时机

使用 print(对象) 打印对象的时候，会自动调用

1，如果没有定义 __str__ 方法，默认打印的是 对象的引用地址

2，如果定义 __str__ 方法,打印的是 方法的返回值

2，应用场景

使用 print(对象) 打印输出对象的属性信息

3，注意事项

必须返回一个 字符串

```
"""
定义 Cat 类，包含属性 name 和 age，打印对象的时候,可以输出对象的姓名和年龄
类名: Cat
属性: name, age
方法: __str__ ,  __init__
"""


class Cat:
    def __init__(self, name, age):
        self.name = name  # 添加 name 属性
        self.age = age  # 添加 age 属性
```

```
    def __str__(self):  # 一般不使用 print,直接返回
        return f"姓名: {self.name}, 年龄: {self.age}"


# 创建对象
tom = Cat('汤姆', 3)
print(tom)
```

封装案例

封装: 根据要求将属性和方法 定义到类中(定义类)

案例一 小明爱跑步

需求:

- 小明 体重 75.0 公斤
- 小明每次 跑步 会减肥 0.5 公斤
- 小明每次 吃东西 体重增加 1 公斤

```
类名: 人类(Person)
属性: 体重(weight)  姓名(name)
方法: 跑步(run)  ---> 修改属性值
      吃东西(eat)  ---->  修改属性值
    __init__  定义属性
    __str__  打印属性信息使用
```

```python
class Person:
    def __init__(self, name, weight):
        self.name = name  # 姓名
        self.weight = weight  # 体重

    def __str__(self):
        return f"{self.name} 目前的体重为
{self.weight} kg"

    def run(self):
        """跑步的方法"""
        # 体重减少 0.5kg
        # self.weight = self.weight - 0.5
        self.weight -= 0.5
        print(f'{self.name} 跑步了, 体重减少 0.5 kg')

    def eat(self):
        """吃东西的方法"""
        # 体重增加 1kg
        self.weight += 1
```

```
        print(f"{self.name} 大餐一顿，体重增加了 1kg")


if __name__ == '__main__':
    # 创建对象
    xming = Person('小明', 75.0)
    print(xming)
    xming.run()
    print(xming)
    xming.eat()
    print(xming)
    xmei = Person('小美', 45.0)
    print(xmei)
    xmei.run()
    xmei.run()
    print(xmei)
    xmei.eat()
    print(xmei)
```

案例二 登录

- 需求

进入某 Web 项目登录页面，输入用户名、密码、验证码之后登录系统

- 类的设计

```
类名: LoginPage
属性: 用户名(username)、密码(password)、验证
码(verify_code)
方法: login
```

- 代码实现

```python
class LoginPage:
    """登录页面"""
    def __init__(self, username, password, code):
        self.username = username  # 用户名
        self.password = password  # 密码
        self.verify_code = code   # 验证码

    def login(self):
        print(f"1. 输入用户名: {self.username}")
        print(f"2. 输入密码: {self.password}")
        print(f"3. 输入验证码: {self.verify_code}")
        print(f"4. 点击登录")


if __name__ == '__main__':
    admin = LoginPage('admin', '123456', '8888')
    admin.login()
```

案例三 摆放家具

- 需求

```
需求:
1. 房子(House) 有 户型、总面积 和 家具名称列表 - 新房子没有任何的家具
2. 家具(HouseItem) 有 名字 和 占地面积，其中
- 席梦思(bed) 占地 4 平米
- 衣柜(chest) 占地 2 平米
- 餐桌(table) 占地 1.5 平米
3. 将以上三件家具添加到房子中
4. 打印房子时，要求输出:户型、总面积、剩余面积、家具名称列表

剩余 面积
1.在创建房子对象时，定义一个 剩余面积的属性，初始值和总面积相等
2.当调用 add_item 方法，向房间 添加家具 时，让 剩余面积 -= 家具面积
```

- 类的设计

```
类名: 房子类(House)
属性: 户型(h_type),
```

```
        总面积 total_area
        剩余面积 free_area = total_area
        家具名称列表 item_list = []
方法: __init__,  __str__
添加家具的方法 add_item  (1, 修改剩余面积 2, 判断剩余面积和家具面积的关系 3, 想家具列表中添加 家具的名字)
--------------------------
类名: 家具类(HouseItem)
属性: 名字 name
    占地面积 area
方法: __str__ , __init__

三个家具对象
- 席梦思(bed) 占地 4 平米
- 衣柜(chest) 占地 2 平米
- 餐桌(table) 占地 1.5 平米
```

```python
class HouseItem:
    """家具类"""
    def __init__(self, name, area):
        self.name = name   # 名字
        self.area = area   # 占地面积

    def __str__(self):
        return f"{self.name} 占地面积 {self.area} 平米"
```

```python
class House:
    """房子类"""
    def __init__(self, h_type, area):
        self.h_type = h_type    # 户型
        self.total_area = area  # 总面积
        self.free_area = area   # 剩余面积和总面积一样
        self.item_list = []     # 刚开始没有家具

    def __str__(self):
        return f"户型:{self.h_type}、总面积:{self.total_area} 平米、剩余面积:{self.free_area}平米、家具名称列表:{self.item_list}"

    def add_item(self, item):  # 1. 房子对象(self)  2. 家具对象(传参)
        """添加家具, item 家具对象"""
        # 1. 先判断房子的剩余面积和家具的占地面积的关系
        if self.free_area > item.area:  # 对象.属性  获取属性值
            print(f'添加家具: {item.name}')
            self.item_list.append(item.name)
            # 修改剩余面积
            self.free_area -= item.area
        else:
```

```
            print(f"房子剩余面积不足,换个大房子
吧.....")


if __name__ == '__main__':
    # 创建家具对象
    bed = HouseItem('席梦思', 4)
    chest = HouseItem('衣柜', 2)
    table = HouseItem('餐桌', 1.5)
    print(bed)
    print(chest)
    print(table)
    # 创建房子
    house = House('三室一厅', 100)
    print(house)
    house.add_item(bed)
    print(house)
    house.add_item(chest)
    print(house)
    house.add_item(table)
    print(house)
```