# 面向对象_04_面向对象封装案例

## 面向对象封装案例

## — 黑马程序员《Python 入门教程完整版》笔记

## 目标

- 封装
- 小明爱跑步
- 存放家具

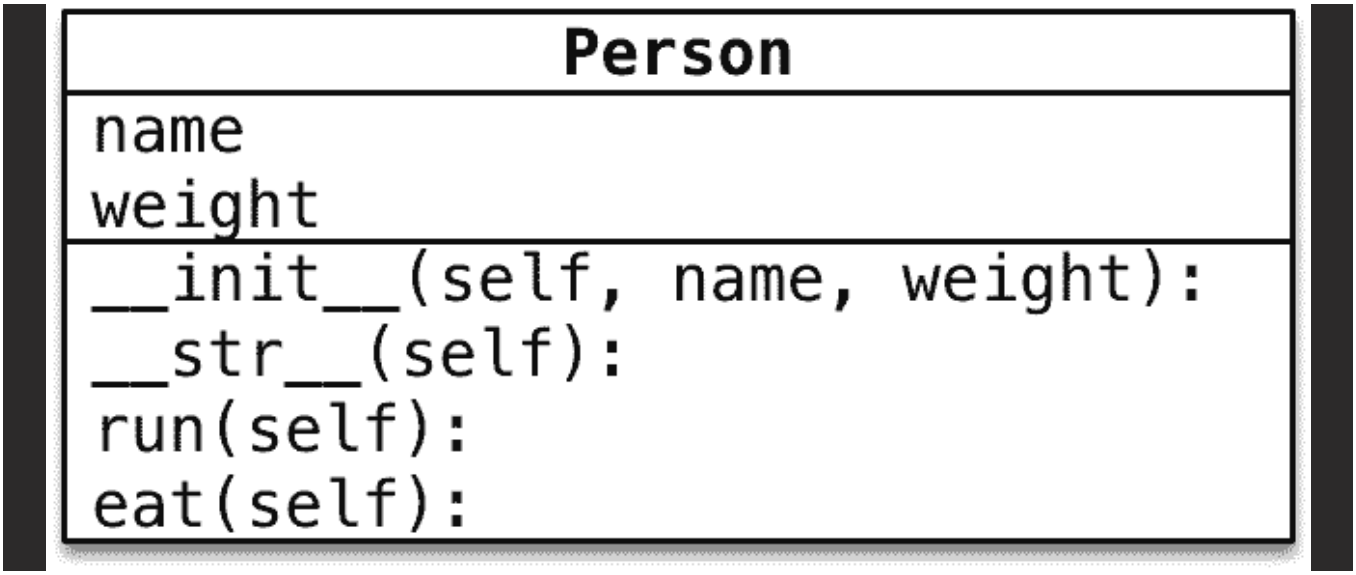
## 01. 封装

1. **封装** 是面向对象编程的一大特点
2. 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
3. **外界** 使用 **类** 创建 **对象**，然后 **让对象调用方法**
4. **对象方法的细节** 都被 **封装** 在 **类的内部**

## 02. 小明爱跑步

### 需求

1. **小明** **体重** `75.0` 公斤
2. 小明每次 **跑步** 会减肥 `0.5` 公斤
3. 小明每次 **吃东西** 体重增加 `1` 公斤




> 提示：在 **对象的方法内部**，是可以 **直接访问对象的属性** 的！

- 代码实现：

```
class Person:
    """人类"""

    def __init__(self, name, weight):

        self.name = name
        self.weight = weight

    def __str__(self):

        return "我的名字叫 %s 体重 %.2f 公斤" % (self.name, self.weight)

    def run(self):
        """跑步"""

        print("%s 爱跑步，跑步锻炼身体" % self.name)
        self.weight -= 0.5

    def eat(self):
        """吃东西"""

        print("%s 是吃货，吃完这顿再减肥" % self.name)
        self.weight += 1


xiaoming = Person("小明", 75)

xiaoming.run()
xiaoming.eat()
xiaoming.eat()

print(xiaoming)
```

## 2.1 小明爱跑步扩展 —— 小美也爱跑步

### 需求

1. **小明** 和 **小美** 都爱跑步
2. **小明** **体重** `75.0` 公斤
3. **小美** **体重** `45.0` 公斤
4. 每次 **跑步** 都会减少 `0.5` 公斤
5. 每次 **吃东西** 都会增加 `1` 公斤

### 提示

1. 在 **对象的方法内部**，是可以 **直接访问对象的属性** 的
2. **同一个类** 创建的 **多个对象** 之间，**属性** 互不干扰！


## 03. 摆放家具

### 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**

| HouseItem |
| --- |
| name<br>area |
| __init__(self, name, area):<br>__str__(self): |

| House |
| --- |
| house_type<br>area<br>free_area<br>item_list |
| __init__(self, house_type, area):<br>__str__(self):<br>add_item(self, item): |

**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**

**思考**：应该先开发哪一个类？

**答案 —— 家具类**

1. 家具简单
2. 房子要使用到家具，**被使用的类**，通常应该先开发

## 3.1 创建家具

```python
class HouseItem:

    def __init__(self, name, area):
        """

        :param name: 家具名称
        :param area: 占地面积
        """
        self.name = name
        self.area = area

    def __str__(self):
        return "[%s] 占地面积 %.2f" % (self.name, self.area)


# 1. 创建家具
bed = HouseItem("席梦思", 4)
chest = HouseItem("衣柜", 2)
table = HouseItem("餐桌", 1.5)

print(bed)
```

```
print(chest)
print(table)
```

### 小结

1. 创建了一个 **家具类**，使用到 `__init__` 和 `__str__` 两个内置方法
2. 使用 **家具类** 创建了 **三个家具对象**，并且 **输出家具信息**

## 3.2 创建房间

```
class House:

    def __init__(self, house_type, area):
        """

        :param house_type: 户型
        :param area: 总面积
        """
        self.house_type = house_type
        self.area = area

        # 剩余面积默认和总面积一致
        self.free_area = area
        # 默认没有任何的家具
        self.item_list = []

    def __str__(self):

        # Python 能够自动的将一对括号内部的代码连接在一起
        return ("户型：%s\n总面积：%.2f[剩余：%.2f]\n家具：%s"
                % (self.house_type, self.area,
                   self.free_area, self.item_list))

    def add_item(self, item):

        print("要添加 %s" % item)

...

# 2. 创建房子对象
my_home = House("两室一厅", 60)

my_home.add_item(bed)
my_home.add_item(chest)
my_home.add_item(table)

print(my_home)
```

### 小结

1. 创建了一个 **房子类**，使用到 `__init__` 和 `__str__` 两个内置方法
2. 准备了一个 `add_item` 方法 **准备添加家具**
3. 使用 **房子类** 创建了 **一个房子对象**
4. 让 **房子对象** 调用了三次 `add_item` 方法，将 **三件家具** 以实参传递到 `add_item` 内部

## 3.3 添加家具

### 需求

- 1> **判断 家具的面积** 是否 **超过剩余面积**，**如果超过**，提示不能添加这件家具
- 2> 将 **家具的名称** 追加到 **家具名称列表** 中
- 3> 用 **房子的剩余面积 - 家具面积**

```
    def add_item(self, item):

        print("要添加 %s" % item)
        # 1. 判断家具面积是否大于剩余面积
        if item.area > self.free_area:
            print("%s 的面积太大，不能添加到房子中" % item.name)

            return

        # 2. 将家具的名称追加到名称列表中
        self.item_list.append(item.name)

        # 3. 计算剩余面积
        self.free_area -= item.area
```

## 3.4 小结

- 主程序只负责创建 **房子** 对象和 **家具** 对象
- 让 **房子** 对象调用 `add_item` 方法 **将家具添加到房子中**
- **面积计算**、**剩余面积**、**家具列表** 等处理都被 **封装** 到 **房子类的内部**