类和对象

> 万物皆对象

面向对象的思想

对象：一切客观存在的事物都是对象

对象

> 1. 有什么：属性
> 2. 能做什么（功能）：方法
>
> 对象=属性+方法

- 属性

> 1. 属性可以也是一个对象（具象）
> 2. 属性也可以是抽象的

- 方法

1．方法可以被其他对象使用
2．方法可以被自己使用
3．方法可以被自己或其他对象调用

方法：对象和对象之间彼此相互调用

- 现实生活中用面向对象的思想解决问题

*********\
目标：去大理
1．怎么去： 找对象
飞机 火车 汽车 巴士 jio 自行车 船 马
选： 飞机 最合适
2．如何使用对象：调方法
飞机的 交通运输方法

- 小结：

解决现实生活的问题：
1．找合适的对象
2．调用合适的方法

- 计算机世界中用面向对象的思想解决问题

1. 计算机世界的问题，来自于现实世界

聊天：微信，QQ，飞秋，电话，邮箱，抖音，微博，快手

玩游戏：QQ游戏平台，4399，游迅，steam，LOL，PUBG，WOW，DNF，CF，CSGO，传奇

买东西：淘宝，京东，拼夕夕，美团外卖，饿了么，携程，猫途鹰，飞猪，12306，南方航空，东方航空，中国航空，链家，房天下，我爱我家，瓜子，懂车帝，汽车之家，太平洋，苏宁，中关村，国美，大众电器

2. 计算机模拟现实世界，从而解决现实世界的问题

计算机和现实世界高度统一

3. 解题思路：

设计通讯录

现实生活：1.找对象：本和笔　　2.调方法：笔向本写

计算机：　1.找对象：本对象，笔对象　2.调方法，调用笔的方法向本的方法传值

- 面向对象的特点

1. 模拟现实世界：
    1. 找对象
    2. 调方法
2. 各司其职：
    每个对象应该做独立的事情，同一类事情，对象要足够简单
3. 弱耦合性：
    对象和对象之间，关系尽量弱化
4. 可重用性：
    对象可以多次使用
5. 可扩展性：
    对象可以扩展更多的功能

- 补充：

面向对象编程： OOP Object oriented Programming（基于对象的一种编程）

OOD：面向对象设计
OOA：面向对象分析
AOP：面向切面编程

类

类--对象

1. 类是对象的模板
2. 类是对象共性的抽象
3. 类是客观对象在人脑中的主观反映

补充：类是抽象的　对象是具象的

- 编写一个类

```
1. 关键字：
        class

2. 语法：
        class 类名：
            属性
            方法
3. 属性：
        变量=值
4. 方法：
        def 方法名（self，参数）：
            方法的实现（方法体）
```

```python
class Dog:
    #  毛
    fur='金色'  # 属性
    # 叫
    def shut(self):  # 方法
        print('汪汪叫')
```

- 创建对象1

```
1. 语法：
        类名（ ）
```

- 类属性

```
1. 类属性
        1. 定义的位置： 类的内部，任何方法的外部
           定义形式：类.属性名=值
           1. 类内部，方法外部：属性名=值
           2. 方法内部：类名.属性名=值
           3. 类外部：类名.属性名=值
        2. 类属性所有对象共用，所有的实例对象共用类属性
        3. 类属性是属于类的
        4. 类属性的使用，可以不创建实例对象
        5. 类属性作用范围：全类共有（和顺序无关）
        6. 如果类属性修改，所有的对象都会改变（前一发而动全局）
        7. 访问形式：
           1. 类.类属性
           2. 实例.类属性
注：类可以创建对象，类创建出的对象也称之为：实例对象，简称：实例或对象
```

- 方法

1. 方法：（普通方法）
    1. 方法定义的位置：
        同类属性
    2. 方法不能直接被类使用，必须先创建对象
        1. 如果用类调用，必须手动传入一个实例对象
        2. 如果用实例对象调用，Python会自动传入当前对象

```
class Dog:
    def shut(self,a,b):  # self--当前实例对象
        print('汪汪叫',a,b)

# Dog.shut(1)
# dog=Dog()
# dog.shut()  # dog.shut(dog)
#
# dog2=Dog()
# dog2.shut() # dog2.shut(dog2)
dog3=Dog()
dog3.shut(1,2)
```

- 实例属性

1. 实例属性

1. 定义的形式：

实例对象.属性名=值

如果在类的外部：

实例对象.属性名=值

如果在方法内部：

self.属性名=值

self代表当前对象，本质上该创建方式和在类的外部创建没有任何区别

2. 实例属性，每个对象各自拥有，互不干扰

3. 访问形式：

实例对象.实例属性

4. 如果实例属性和类属性同名：

会创建一个同名的实例属性，并遮蔽住类属性

```
# class Dog:
#     def hehe(self,n):  #self= dog1
#         self.fur=n  # dog1.fur='金色'
#
# # dog1=Dog()
# # # dog1.fur='金色'  # 实例对象.属性名=值    创建实例属性
# # # print(dog1.fur)
# #
# # dog1.hehe() # dog1.hehe(dog1)   hehe中的self参数 被赋值为 dog1    self 参数相当于dog1
# # print(dog1.fur)
#
```

```
# dog1=Dog()
# dog2=Dog()
#
# dog1.hehe('金色')
# print(dog1.fur)
#
# dog2.hehe('红色')
# print(dog2.fur)



class Dog:
    age=2


dog1=Dog()
dog2=Dog()
dog3=Dog()

print(dog1.age)
print(dog2.age)
print(dog3.age)

Dog.age=3 # 修改类属性

print(dog1.age)
print(dog2.age)
print(dog3.age)
```

```
dog1.fur='gold'   #创建实例属性
dog2.fur='red'
dog3.fur='black'

print(dog1.fur)
print(dog2.fur)
print(dog3.fur)

Dog.sex=True   # 模板改变，所有对象跟着改变
print(dog1.sex)
print(dog2.sex)
print(dog3.sex)

# 类属性，可以怎么访问？
print(Dog.age)     # 类.类属性
print(Dog().age)   # 实例对象.类属性
print(dog1.age)    # 实例对象.类属性

# 实例属性的访问
print(dog1.fur)   # 实例对象.实例属性

print(Dog.fur)
class Dog:
    age=2

dog1=Dog()
```

```
dog2=Dog()
dog3=Dog()

print(dog1.age)
print(dog2.age)
print(dog3.age)

dog1.age=4  # 修改实例属性  实例属性
print(dog1.age)
print(dog2.age)
print(dog3.age)

Dog.age=3
print(dog1.age)
print(dog2.age)
print(dog3.age)

# dog1: age(实例属性，属于自己)  dog2：age（类属性，属于类）  dog3：age（类属性，属于类）

del dog1.age
print(dog1.age)
print(dog2.age)
print(dog3.age)

del dog1.age
print(dog1.age)
print(dog2.age)
```

```
print(dog3.age)
```

- 初始化方法

```
1. 形式：
        def __init__ (self):
            内容
2. 是一个魔法方法，不需要手工调用，在创建对象时自动调用
3. 可以传递参数：
        def __init__ (self，p1,p2...):
            内容
        创建对象时：
            类名（构造参数）
4. 返回值必须是None
5. 如果一个类没有任何初始化方法，创建对象时，则会调用其父类的初始化方法
        初始化方法是必需的（直接，或间接调用一个初始化方法）
        object是所有类的父类
6. 初始化方法的应用：
        一般用于实例属性的初始化
        构造参数名和属性名一致
```

```
# class Dog:
#     age=10  # 类属性
#     def __init__(self,fur,age,sex):
#         # print('我执行了',fur,age,sex)
#         # self.fur=fur
```

```
#         # self.age=age
#         # self.sex=sex
#         # age=20 # 局部变量
#         # self.age=30  # 创建实例属性
#         # print(age,self.age,Dog.age)
#         age=20
#         print(age,self.age,Dog.age)  # 实例对象.
类属性
#
#
# dog1=Dog('金色',2,True)
# # dog2=Dog('红色',4,False)

# print(dog1.age,dog1.fur,dog1.sex)
# print(dog2.age,dog2.fur,dog2.sex)


# class Dog:
#     def __init__(self,fur,age,sex):
#         # 初始化实例属性
#         self.age=age
#         self.fur=fur
#         self.sex=sex
#         return 100
#
# d1=Dog('red',2,True)
# d2=Dog('yellow',3,True)
#
```

```
# print(d1.age,d1.sex,d1.fur)
# print(d2.age,d2.sex,d2.fur)


class Dog(object):
    pass

d=Dog()
```

- 补充：

```
self
1. self 指代当前对象
2. self仅仅是个变量名，可以替换成其他名字
3. 当前对象自动传递给第一个参数，习惯上使用self表示
```

- 私有化

```
1. 成员前加两条下划线，该成员则为：私有成员
2. 类的内部可以访问私有成员，类的外部不能访问
3. 成员私有化，提供匹配的get/set方法
4. 如果方法私有化，不用提供get/set方法
5. Python没有真的私有化，是伪私有化
        使用了名字重构的方式
        重构为：  _类名__属性名

成员：属性和方法统称为成员，属性：成员属性    方法：成员方法
```

```
# class Dog:
#     __age=2
#     def get_age(self):
#         print(self.__age) # 调用类属性
#     def set_age(self,newAge):
#         Dog.__age=newAge
#
#     def __hehe(self):
#         print('dog can speak')
#
# d=Dog()
# d.get_age()
# d.__hehe()

class Dog:
    def __init__(self):
        self.__age=2
    def get_age(self):
        return self.__age
    def set_age(self,newAge):
        self.__age=newAge

d=Dog()
# print(d.get_age())
# d.set_age(10)
# print(d.get_age())
print(d._Dog__age)
```

- 组合

一个对象的属性可以另一个对象
1. 组合：一个对象作为当前对象的属性

```
class Student:
    def __init__(self,name,age,sex):
        self.name=name
        self.age=age
        self.sex=sex
class Teacher:
    def __init__(self,name,age,sex):
        self.name = name
        self.age = age
        self.sex = sex
# 类属性
# class School:
#     student=Student('xiaobo',18,True)
#     teacher=Teacher('feige',19,True)
#
# school=School()
# print(school.teacher.name)
# print(school.student.age)

# 实例属性1
# class School:
#     def __init__(self):
```

```
#         self.student = Student('xiaobo', 18,
True)
#         self.teacher = Teacher('feige', 19,
True)
#
# school=School()
# print(school.teacher.name)

# # 实例属性： 所有实例属性的参数 从形参中来
# class School:
#     def
__init__(self,name1,age1,sex1,name2,age2,sex2):
#         self.student = Student(name1,age1,sex1)
#         self.teacher = Teacher(name2,age2,sex2)
#
# school=School('xb',19,True,'fg',19,True)
# print(school.teacher.name)



# 实例属性： 所有实例属性的参数 从形参中来
class School:
    def __init__(self,student,teacher):
        self.student = student
        self.teacher = teacher

school=School(Student('xiaobo',19,True),Teacher('
feige',18,True))
```

```
print(school.teacher.name)
```