学习目标

1. 掌握在类与对象的概念;
2. 掌握 class 关键字的使用语法;
3. 掌握什么是方法,什么是属性;
4. 掌握 init 方法的使用;
5. 掌握 del 方法的使用;

目录

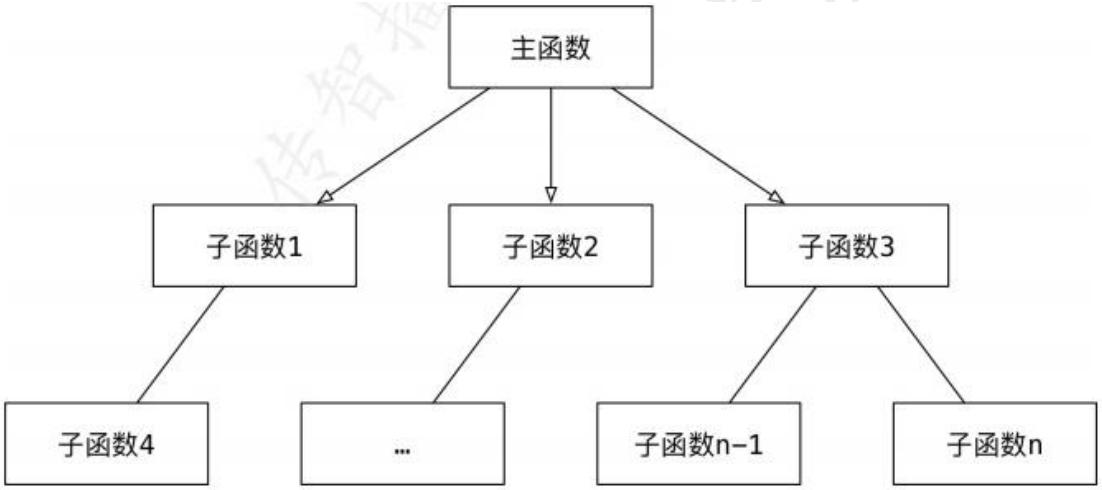
第 1 章 面向对象基础------概念

一、面向函数的编程方式

1. 把完成某一个需求的所有步骤从头到尾逐步实现；

2. 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；

3. 最后完成的代码，就是顺序地调用不同的函数。



二、面向对象的编程方式

相比较函数，面向对象是更大的封装，根据职责在一个对象中封装多个方法

1. 在完成某一个需求前，首先确定职责 —— 要做的事情（方法）；

2. 根据职责确定不同的对象，在对象内部封装不同的方法；

3. 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。

三、类和对象的概念

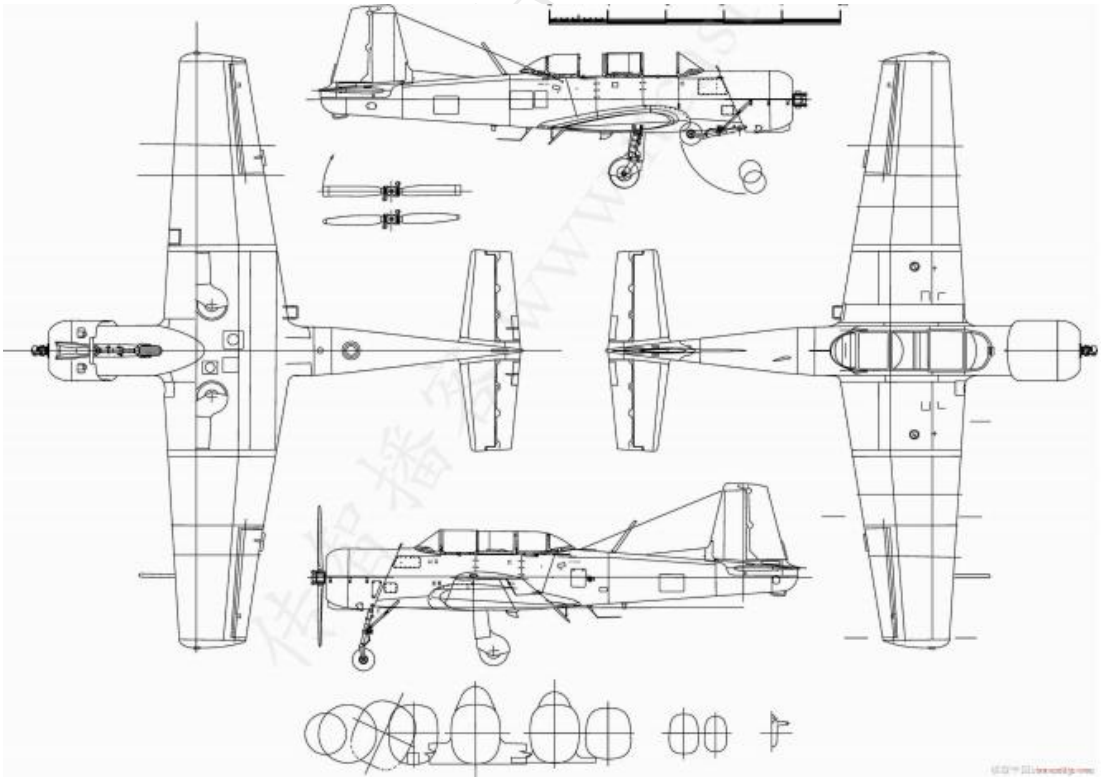
类和对象是面向对象编程的两个核心概念。

1. 类

类是对一群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 。

- 特征被称为属性 ；
- 行为被称为方法 。

类就相当于制造飞机时的图纸，是一个模板



2. 对象

对象是由类创建出来的一个具体存在，可以直接使用。

由哪一个类创建出来的对象，就拥有在哪一个类中定义的:

- 属性 ；
- 方法 。

对象就相当于用图纸制造的飞机。

3. 类和对象的关系

- 类是模板，对象是根据类这个模板创建出来的，应该先有类，再有对象 ；
- 类只有一个，而对象可以有很多个 ；
- 不同的对象之间属性可能会各不相同 ；
- 类中定义了什么属性和方法，对象中就有什么属性和方法，不可能多，也不

可能少 。

四、类的设计

在程序开发中，要设计一个类，通常需要满足一下三个要素：

1. 类名 这类事物的名字 ；

2. 属性 这类事物具有什么样的特征 ；

3. 方法 这类事物具有什么样的行为。

1. 属性和方法的确定

- 对于对象的特征描述，通常可以定义成属性 ，属性的具体实现可以是一个变量；
- 对象具有的行为（动词），通常可以定义成方法，方法的具体实现可以是一个类里面的函数 ；

提示：需求中没有涉及的属性或者方法在设计类时，不需要考虑。

2. 一个小猫类 cat 的描述

- 属性：
  - name(姓名)
- 方法：
  - eat（吃）
  - drink（喝）

第 2 章 面向对象基础------语法

一、class 关键字

class 关键字用于创建一个类，语法如下：

```
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 。

二、类的代码实现

```
class cat:
    def eat(self):
        print("汤姆爱吃鱼")
    def drink(self):
        print("汤姆爱喝水")
```

三、 创建一个对象

名词解释

1. **实例**------通过类创建出来的对象叫做类的实例;

2. **实例化**------创建对象的动作叫做实例化 。

创建对象语法:

对象名 = 类名(参数列表)

类是静态的，只有创建为对象后，才能成为动态运行的程序。

```
# 定义cat类
class cat:
    def eat(self):
```

```
        print("汤姆爱吃鱼")
    def drink(self):
        print("汤姆爱喝水")

# 创建 lazy_cat 对象
lazy_cat = cat()
# 调用对象的 eat 方法
lazy_cat.eat()
# 调用对象的 drink 方法
lazy_cat.drink()
```

四、方法中的 self 参数

- 在类封装的方法内部， self 就表示调用方法的对象自己 ；
- 调用方法时，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；
  - 通过在方法内部使用 **self.属性名 = 值**，为类添加属性
- 在方法内部可以通过 self. 调用对象的方法；
- 在类的外部，通过 对象名. 访问对象的属性和方法 。

五、cat 类添加 name 属性，同时创建两个对象

```
# 定义 cat 类
class cat:
    def set_name(self, name):
        # 给 cat 类添加一个属性 name
        self.name = name

    def eat(self):
        print("%s 爱吃鱼" % self.name)
    def drink(self):
        print("%s 爱喝水" % self.name)


# 创建 lazy_cat 对象
```

```
lazy_cat = cat()
lazy_cat.set_name("懒猫")
lazy_cat.eat()
lazy_cat.drink()

# 创建 tom_cat 对象
tom_cat = cat()
tom_cat.set_name("tom 猫")
tom_cat.eat()
tom_cat.drink()
```

1. 课堂练习---

- 定义一个小狗类 dog

方法名	说明
set_name(self, name)	方法内部为 dog 类添加一个属性 name
show_name(self)	显示 name 属性的值

- 创建小狗类的对象,设置对象的 name 为”旺财”;
- 调用 show_name 显示对象的 name。

第 3 章 面向对象基础------__init__方法

一、__init__初始化方法

__init__就是对象的初始化方法， __init__ 是对象的内置方法。

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. 系统自动调用方法(__ init__) .

二、cat 类增加__init__方法

```
# 定义 cat 类
class cat:
    # 初始化方法
    def __init__(self):

        print("初始化方法")

    def eat(self):

        print("小猫爱吃鱼")

    def drink(self):

        print("小猫爱喝水")


# 创建对象的同时，初始化方法被自动调用
lazy_cat = cat()
```

三、在初始化方法内部为类添加属性

```
# 定义 cat 类
class cat:

    # 初始化方法
    def __init__(self):

        self.name = "猫"

        print("%s 的初始化方法" % self.name)

    def eat(self):

        print("%s 爱吃鱼" % self.name)

    def drink(self):

        print("%s 爱喝水" % self.name)


# 创建对象的同时，初始化方法被自动调用
lazy_cat = cat()
```

四、带有参数的初始化方法

```
# 定义 cat 类
class cat:
    # 带有参数初始化方法
    def __init__(self, name):
        self.name = name

        print("%s 的带有参数的初始化方法" % self.name)

    def eat(self):

        print("%s 爱吃鱼" % self.name)

    def drink(self):

        print("%s 爱喝水" % self.name)


lazy_cat = cat("tom")
```

五、初始化方法的缺省参数

```
# 定义 cat 类
class cat:
    # 带有缺省参数初始化方法

    def __init__(self, name = "猫"):

        self.name = name

        print("%s 的带有参数的初始化方法" % self.name)

    def eat(self):

        print("%s 爱吃鱼" % self.name)

    def drink(self):

        print("%s 爱喝水" % self.name)


# 没有指定 name 的值，此时 name 等于缺省值
lazy_cat = cat()
```

1. 课堂练习---

- 修改小狗类 dog

方法	说明
__init__(self, name)	把属性 name 的定义放入__init__方法中. name 有缺省值为”狗”。

- 创建小狗类的对象,并显示小狗类的 name

第 4 章 面向对象基础------__del__方法

一、__del__方法说明

- __del__方式只能有一个参数 self;
- 当对象在内存中被销毁的时候，__del__方法被系统自动调用;
- 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 ;
- 一个对象的 __del__ 方法一旦被调用，对象的生命周期结束 。

```
# 定义 cat 类
class cat:
    def __init__(self, name = "猫"):
        self.name = name
        print("%s 的带有参数的初始化方法" % self.name)
    def __del__(self):
        print("%s 被销毁了" % self.name)
    def eat(self):
        print("%s 爱吃鱼" % self.name)
    def drink(self):
```

```
        print("%s 爱喝水" % self.name)

lazy_cat = cat()
```

- 在函数内定义的变量，函数执行完毕，变量就被销毁了；
- 在函数外部定义的变量，程序执行完毕，变量就被销毁了；
- 可以通过 del 关键字，显式的销毁一个变量。

二、__del__案例

- 案例一：调用 del 函数在程序完成前把对象销毁

```
# 定义 cat 类
class cat:
    def __init__(self, name = "猫"):
        self.name = name
        print("%s 的带有参数的初始化方法" % self.name)
    def __del__(self):
        print("%s 被销毁了" % self.name)
    def eat(self):
        print("%s 爱吃鱼" % self.name)
    def drink(self):
        print("%s 爱喝水" % self.name)

lazy_cat = cat()
del lazy_cat
print("程序终止")
```

- 案例二：函数执行完成，对象销毁

```
# 定义 cat 类
```

```
class cat:
    def __init__(self, name = "猫"):
        self.name = name
        print("%s 的带有参数的初始化方法" % self.name)
    def __del__(self):
        print("%s 被销毁了" % self.name)
    def eat(self):
        print("%s 爱吃鱼" % self.name)
    def drink(self):
        print("%s 爱喝水" % self.name)

def test_cat():
    lazy_cat = cat()

test_cat()
print("程序终止")
```

第 5 章 面向对象基础------__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）

```
# 定义 cat 类
class cat:

    def __init__(self, name = "猫"):

        self.name = name

        print("%s 的带有参数的初始化方法" % self.name)

    def __del__(self):

        print("%s 被销毁了" % self.name)

    def eat(self):

        print("%s 爱吃鱼" % self.name)

    def drink(self):

        print("%s 爱喝水" % self.name)
```

```
lazy_cat = cat()
print(lazy_cat)
```

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

注意： __str__ 方法必须返回一个字符串

```
# 定义 cat 类
class cat:
    def __init__(self, name = "猫"):
        self.name = name
        print("%s 的带有参数的初始化方法" % self.name)
    def __del__(self):
        print("%s 被销毁了" % self.name)
    def __str__(self):
        return "我是一只%s" % self.name
    def eat(self):
        print("%s 爱吃鱼" % self.name)
    def drink(self):
        print("%s 爱喝水" % self.name)

lazy_cat = cat('aaaa')
# 这个时候 print 将显示__str__函数返回的字符串
print(lazy_cat)
```

1. 课堂练习---

- 修改小狗类 dog;
- 创建小狗类的对象，调用 print(小狗类对象),显示如下字符: ”这是一个小狗类对象”。

第 6 章 面向对象基础------类设计练习

一、calc 类

1. 课堂练习---

设计一个类 calc，实现计算器

功能属性与方法说明如下：

属性	说明
oper	运算符

方法	说明
def calc(self, a, b)	属性 oper 值为”+”，返回 a 加 b 的和; 属性 oper 值为”-”，返回 a 减 b 的差; 属性 oper 值为”*”，返回 a 乘 b 的积; 属性 oper 值为”/”，返回 a 除 b 的结果;