编程思想不同会导致怎样的代码差异?

目录

面向过程

- 以过程为核心，强调解决问题的流程
- 先 xxx 再 xxx 然后 xxx
- 特点：符合时间顺序，易于理解程序逻辑

Python 中的对象

Python 的面向过程也使用了对象，比如：

- 变量指向的基础数据类型
- 通过 id() 函数可以返回对象的整数标识
- 通过 is 可以判断是否为同一对象

Python 中的对象

对象间的比较：

```
list1 = [ 1, 2, 3 ]

list2 = [ 1, 2, 3 ]

list1 == list2 # True

list1 is list2 # False
```

面向对象编程

- 和面向过程编程比起来，面向对象编程更关注一个对象的定义
- 优点是更容易抽象和复用

面向对象编程

- 以列表对象为例，列表数据类型是一类列表
- 列表有着共同的功能，如：添加、删除、切片等
- 当你为自己的程序定义了一个列表实体之后，它也具有了列表的全部功能
- 当你想为列表增加一个新的能力时，只要编写列表数据类型的功能，所有的列表都具备了该能力

面向对象和面向过程编程的区别

- 面向对象比面向过程更适合复杂度高的编码需求
- 面向对象适合有更严格的封装需求的程序
- 面向对象可设计的编程模式更丰富
- 面向对象涉及的技术概念更多

总结

1. 面向对象和面向过程是两种不同的编程思想
2. 不同的编程思想影响着编码风格
3. 面向对象更适合复杂的需求，但也有更高的编码要求

课后作业

有一首经典的童谣：“故事讲的是从前有座山，山上有座庙，庙里有个老和尚，老和尚在给小和尚讲故事，故事讲的是从前有座山，山上有座庙……”

请你使用面向过程和面向对象的伪代码，分别尝试描述一下如何编写此程序。

类与实例：如何使用面向对象的思想编写程序?

目录

定义一个类

```
class Coffee(object):
```

- class 定义类的关键字
- Coffee 类的名称，一般首字母大写
- object 父类，可不写，默认继承自 object
- 还可以使用 type() 函数创建类（一般用于动态创建类）

类的实例化

```
mocha = Coffee()
```

- mocha 类 Coffee 实例化的对象

类的属性

```
class Coffee():

    water = 0

    milk = 0

    liquid = 0
```

类的方法

```
class Coffee(object):

    def add_water(self):

        self.water = 1
```

总结

1. 类的实例化操作可以创建一个该类的对象
2. 对象拥有类的属性和方法
3. 要能够根据上下文区分开类的实例化和函数调用

课后作业

- ✓ 定义一个类，用于表示汉堡。
- ✓ 在该类中包含了增加、减少一片蔬菜，增加、减少一片肉饼以及显示当前蔬菜和肉饼数量的多个方法。
- ✓ 实例化一个汉堡后，通过该实例的方法，为汉堡修改蔬菜和肉饼数量并显示当前汉堡的层数。

类的继承：如何解决代码重用的问题?

目录

为什么需要继承

- 继承可以让新手快速上手大牛设计的代码
- 比如，你希望自己的类能够实现字典 + 额外功能，那么你可以使用如下代码实现：

```
from collections import UserDict

class my_dict(UserDict):

    ...
```

父类与子类

- 在继承中，被继承的类称作父类，继承的类称作子类
- 类的继承是指子类继承了父类的属性、方法

父类与子类

子类继承父类的方法：增加（或减少）功能时可以使用 super() 方法

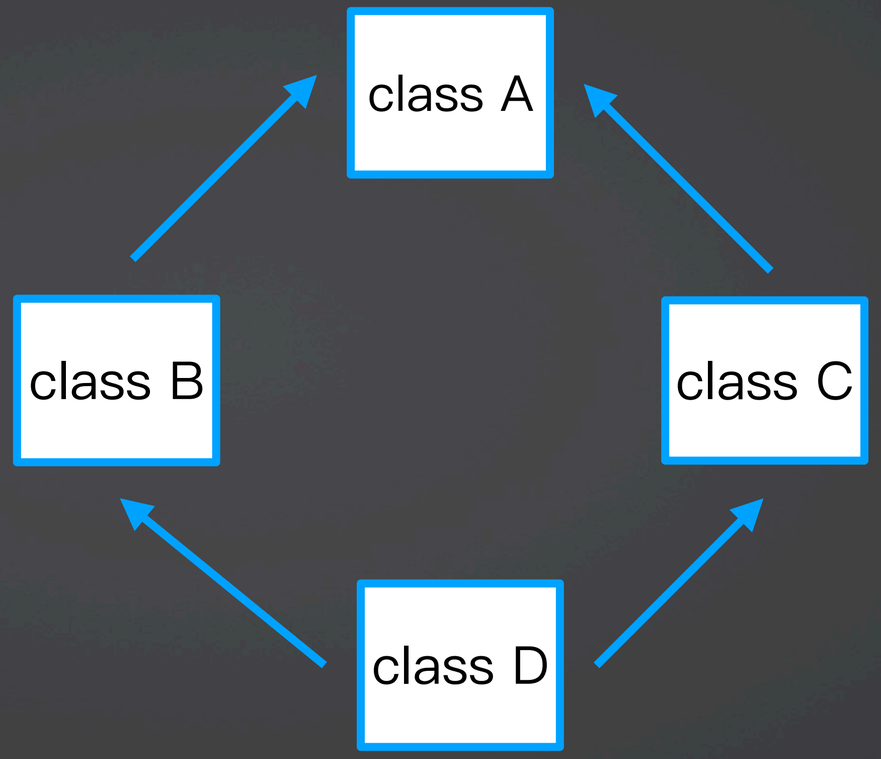
```
class Father(object):
    def run(self):
        pass

class Son(Father):
    def run(self):
        super().run()
        print("run")
```

* 思考：不使用 super() 会怎样?

类的多继承

```
class A:
    ...
class B(A):
    ...
class C(A):
    ...
class D(B, C):
    ...
```



类的多继承

- 菱形继承时，Python 会按照 C3 算法（有向无环路图）按顺序遍历继承图
- 通过类对象名称.__mro__ 可以查看继承顺序
- 多重继承增加了继承的复杂度，应当减少多重继承的使用

混入

- 混入 Mix–In 是指借用多继承的语法，为现有类增加新的方法
- 混入不定义新的属性，只包含方法
- 混入便于重用，但绝不能实例化
- 混入类一般在类名称后增加 Mixin

混入

例如： 披萨可以按层数分为单层、双层，也可以按照形状分为圆形、方形，

可以将它们定义为：单层Mixin、双层Mixin、圆形Mixin、方形Mixin

- 定义一个披萨类：

```
class 披萨(主要材料, 单层Mixin, 圆形Mixin)
```

总结

1. 类可以通过继承实现父类的方法
2. 类可以通过多继承实现更复杂的方法
3. 使用 Mixin 有效减少多继承的复杂度

课后作业

请结合类的继承，为童谣定义类和方法：

小老鼠，上灯台，偷油吃，下不来，喵喵喵，猫来了，叽里咕噜滚下来。

类的装饰器：如何改变类方法的功能?

目录

类的装饰器

- 类的装饰器是类方法的装饰器的缩写
- 你可以通过装饰器改变方法的调用方式和行为

classmethod 装饰器

- classmethod 可以实例的方法定义为类的方法，用于类直接调用

- ```
  class Klass:
      @classmethod
      def func(cls):
          pass
  ```

- cls 表示当前操作的类，可以使用 Klass.func() 调用
  ```

staticmethod 装饰器

- 不需要类的任何信息但又和类相关的一些方法，为了方便维护代码并保持代码工整，可以将该函数定义到类中并使用 staticmethod 修饰
- class Klass():

  @staticmethod

  def func():

  pass
- staticmethod 修饰的方法，不需要使用 self 或 cls

property

```
class Klass:
    @property
    def func(self):
        return self.__varName

    @func.setter
    def func(self, varValue):
        self.__varName = varValue
```

总结

1. 类的装饰器改变了调用的方法和行为
2. 类的装饰器让类的使用更加灵活，但也给新手增加了学习难度
3. 非必要，不要使用类的方法的装饰器

课后作业

定义一个类，类中的属性 userData 可以将属性的值保存到文件中，该属性的行为如下：

1. 对属性赋值时，该值自动保存到文件 userData.txt 中；

2. 读取属性值时，如果该属性为空，那么就从文件 userData 中读取并在终端显示该属性值。

THANKS

小试牛刀：如何开发自动咖啡机？

目录

案例解析

- 调配各种类型的咖啡需要咖啡原液、水、牛奶三种原料
- 三种原料根据不同比例、属性叠加形成新的咖啡品类
- 为了支持前端程序展示，需在开发时交付后端接口

定义类

- 每一杯咖啡成品是一个对象，因此需要定义咖啡类
- 调配咖啡需要水、咖啡液、牛奶三种原料，需要分别定义类

定义属性和方法

- 咖啡成品有冷、热的差别，因此水、牛奶需要有冷、热属性
- 三种原料都需要添加，因此需要添加方法
- 三种原料的单位不同，咖啡液以份数为单位，水和牛奶以毫升为单位

测试

- 实例化后进行测试，并优化输出结果

总结

1. 定义类的属性和方法，是完善面向对象编程的主要方法
2. 实例的行为即方法，需要在编程过程中不断调整
3. 程序迭代过程需要不断调整继承关系，抽象对象的类

避坑指南：类的常见错误

目录

语法错误

- 忘记使用 self 关键字
- 错误使用 self 与 cls 关键字

设计错误

- 哪些对象应该被抽象为类?
- 哪些功能应该被定义为属性，哪些应该被定义为方法?
- 如何解决类之间的依赖?

设计错误

SOLID 原则：

- S 单一职责原则
- O 开闭原则
- L 里氏替换原则
- I 接口隔离原则
- D 依赖倒置原则

设计错误

单一职责原则：

- 类只负责做一件事，即只有一个职责。即：越小越好

设计错误

开闭原则：

- 类应当对扩展开放，对修改关闭，使其有更好的可维护性

设计错误

里氏替换原则：

- 某个对象使用类的子类时，应当和使用父类有相同的行为。
- 即：对于任何类，客户端都能应该能无差别的使用它的子类，并且不会影响运行时的预期行为。

设计错误

接口隔离原则：

- Python 使用“鸭子类型”实现接口，接口越小越好

设计错误

依赖倒置原则：

- 高层模块不应该依赖低层模块，而是应该让二者依赖抽象

总结

1. self 是刚开始学习面向对象编程时，最容易忽略的语法
2. 编写多个类时，解决依赖关系是初学者最难把握的部分，使用 SOLID 指导原则，可以有效拆分类

作业

请你设计一款网络音乐播放器，并基于本讲学习的SOLID原则，试着将网络音乐播放器的各功能拆分成函数。

* 只需设计函数之间依赖关系，不需要实现函数功能

init方法：如何为对象传递参数?

目录

魔术方法

- 有一类方法，以“__”开始和结束，实现了除一般方法外的特殊功能，被称作魔术方法

init方法

- __init__() 称作初始化方法，用于属性和方法初始化方法
- 在类实例化时自动进行初始化
- 初始化方法还可以让类实例化时接收参数

init 方法

```
class Klass:

    def __init__(self, 接收参数)

        self.var = 参数


实例 = Klass( 参数 )
```

其他的魔术方法

- Python 中的魔术方法非常多，按照分类可以参考数据模型：

  https://docs.python.org/zh-cn/3.10/reference/datamodel.html#special-method-names

- 也可以通过 dir（数据类型）得到该类型的魔术方法，按数据类型分类学习

总结

1. 魔术方法是扩展现有数据类型的最佳实践
2. __init__() 函数在类的编写中经常用于初始化和参数处理
3. 扩展数据类型默认的功能时，应首先考虑魔术方法

课后作业

请参考字典数据类型，实现自己的字典数据类型，要求当有重复的 key 被修改时，提示用户该 key 重复，不允许修改。如：

```
dict1 = MyDict()

dict1[ 'a' ] = 100

dict1[ 'a' ] = 200 # 报错，已有名为 'a' 的 key
```

THANKS