作业情况

- 共性问题
  - 装饰器实现时未使用functools.wraps
  - 装饰器需要参数时的如何实现
  - 建议大家在时间截止后再开放github等公开代码

School of Economics and Management, Beihang University

现代程序设计技术

赵吉昌

本周内容

- 面向对象编程
  - 抽象类
  - 接口
  - 泛函数
  - 适配器模式

抽象类

- type()函数
  - 并非仅仅返回对象的类型
  - Python使用type()函数创建类对象
    - 函数和类不是编译时定义的，而是在运行时动态创建的
  - type()函数依次传入3个参数
    - 类名称
    - 继承的父类集合(tuple)
    - 属性（数据或方法）字典
  - Demo: mcl.py

抽象类

- 元类
  - metaclass
  - 控制类的创建行为
    - 先定义metaclass，然后创建类，最后创建实例
    - 类可以看成metaclass创建的“实例”
  - 元类的定义
    - metaclass的类名总是以Metaclass结尾
    - metaclass是类的模板，所以必须从type类型派生
  - 元类的使用
    - 通过关键字参数metaclass来指定元类来创建类
    - Demo: listm.py

抽象类

- 元类
  - 为什么要动态修改？
    - 可以直接在继承`list`的类里添加`add`方法
  - 但有时需要动态定义类
    - `Object Relational Mapping(ORM)`
    - 关系数据库的一行映射为一个实例对象，也就是一个类对应一个表
    - “透明化”数据库相关的操作
    - 类需要动态定义
    - `Demo: orm.py`

抽象类

- 抽象类
  - abstract class
  - 特殊的类，只能被继承，不能被实例化
  - 从不同的类中抽取相同的属性和行为
- 抽象类与普通类的区别
  - 抽象类中有抽象方法
  - 不能被实例化，只能被继承
  - 子类必须“实现”抽象方法
    - 语法上要求必须覆盖

抽象类

- 抽象类的实现
  - 借助abc模块实现
  - import abc
  - class Fruit(metaclass=abc.ABCMeta):
  - @abc.abstractmethod
  - def harvest(self):
  - pass
  - @abc.abstractmethod
  - def grow(self):
  - pass
  - 也可以写成：class Fruit(abc.ABC)

抽象类

- 抽象类的实现
  - 继承抽象类
    - `class Watermelon(Fruit):`
    - `    def harvest(self):`
    - `        print("从地里摘")`
    - `    def grow(self):`
    - `        print("用种子种")`
  - 注册抽象类
    - `@Fruit.register #Fruit.register(Orange)`
    - `class Orange:`
    - `    def harvest(self):`
    - `        print("从树上摘")`
    - `    def grow(self):`
    - `        print("种橘子树")`

抽象类

- 抽象类的实现
  - Demo: ac.py
  - 继承与注册的差别
  - w=Watermelon()
  - o=Orange()
  - isinstance(w,Fruit)
  - isintance(o,Fruit)
  - issubclass(Watermelon,Fruit)
  - issubclass(Orange,Fruit)
  - print([sc.__name__ for sc in Fruit.__subclasses__()])
    - 不会包含注册的子类

抽象类

- 虚拟子类
  - 对于用户自定义的类型，即使不注册，抽象基类也能把一个类识别为虚拟子类
  - 实现一个名为 `__subclasshook__` 的类专有方法
    - `isinstance`
    - `issubclass`
  - Demo: `vsub.py`

接口

- 接口
  - Interface
  - Python中没有专门的支持
  - 但可以约定任何方法都只是一种规范，具体的功能需要子类实现
  - 与抽象类的区别
    - 抽象类中可以对一些抽象方法做出基础实现
    - 接口中所有方法只是规范，不做任何实现
  - 使用原则
    - 继承类应该尽量避免多继承
    - 继承（实现）接口鼓励多继承

接口

- 补充：鸭子类型(duck typing)
  - 一个对象有效的语义，不是由继承自特定的类或实现特定的接口决定的，而是由"当前方法和属性的集合"决定
  - “如果一只鸟走起来像鸭子、游泳起来像鸭子、叫起来也像鸭子，那么它就可以被称为鸭子。”
  - 在鸭子类型中，关注点在于对象的行为，即提供的方法，而不在于对象所属的类型
  - 在 Python 中创建功能完善的类型无需继承，只需实现符合对应类型协议的方法
    - 协议是非正式的接口，只在文档中定义，不在代码中定义，可以看作是约定俗成的惯例
    - 鸭子类型通常得益于"不"测试方法和函数中参数的类型，而是依赖文档、清晰的代码和测试来确保正确使用
  - 对类功能的运行进扩充，更好的动态性，但影响执行效率
  - `Demo: duckt.py`

泛函数

- 泛函数
  - 函数对不同的参数类型会提供不同的处理方式
  - 通过装饰器来实现
  - 类似于重载机制
    - `from functools import singledispatch`
    - `@singledispatch`
    - `def func():`
      - `pass`
    - `@func.register(int)#注册到int的处理`
    - `def _(num):`
      - `pass`
    - `Demo: olf.py`

适配器模式

- 适配器模式(Adapter)
  - 将某个类的接口转换成客户端期望的另一个接口表示，目的是消除由于接口不匹配所造成的类的兼容性问题
  - 目标类(Target)
    - 定义客户所需的接口，可以是一个抽象类或接口，也可以是具体类
  - 适配器类(Adapter)
    - 转换器，通过调用另一个接口对Adaptee和Target进行适配
  - 适配者类(Adaptee)
    - 被适配类，包括了客户希望的业务方法

适配器模式

- Demo
  - ad.py
- 适用场景
  - 没有现成的代码
  - 利用既有组件可能成本更低
  - 版本升级与兼容性
    - 新版本：Adaptee，旧版本：Target，Adapter类：实现旧版本类与新版本类的兼容

本周作业

- 在使用python时，我们经常会用到许多工具库，它们提供了较为方便的函数调用。但是仍然会有一些情况，例如数据类型或格式不符合函数要求，参数存在差异等，使得调用前需要对数据进行额外处理。本次作业要求基于matplotlib，wordcloud，PIL, imageio等绘图库的绘制函数，设计并实现适配器抽象类和不同的适配类，以实现不同类型数据的多样化可视。具体要求如下：
- 1. 要求设计抽象类Plotter，至少包含抽象方法plot(data, *args, **kwargs)方法，以期通过不同子类的具体实现来支持多类型数据的绘制，至少包括数值型数据，文本，图片等。
- 2. 实现类PointPlotter, 实现数据点型数据的绘制，即输入数据为[(x,y)...]型，每个元素为一个Point类的实例。
- 3. 实现类ArrayPlotter, 实现多维数组型数据的绘制，即输入数据可能是[[x1,x2...],[y1,y2...]]或者[[x1,x2...],[y1,y2...],[z1,z2...]]。
- 4. 实现类TextPlotter，实现文本型数据的绘制，即输入数据为一段或多段文本，应进行切词，关键词选择（根据频率或tf-idf)，继而生成词云。
- 5. 实现类ImagePlotter，实现图片型数据的绘制，即输入数据为图片的路径或者图片内容（可以是多张图片），呈现图片并按某种布局组织（如2x2等)。
- 6. 实现类GifPlotter, 支持一组图片序列的可视化（通过文件路径或图片内容输入），但输出是gif格式的动态图。
- 附加7：在3中，如果多维数组超过3维，可否支持pca等降维并绘制，实现类KeyFeaturePlooter？（了解pca或者TSNE）
- 附件8：如果输入是一段音频（比如mp3文件），如何进行绘制，实现类MusickPlotter？（了解librosa里的display模块）
- 附加9：在6中，如果输入是一段落视频的话，能否通过帧采样，将视频绘制为gif并输出为微信表情包，实现类VideoPlotter？（了解cv2)