# 第三周作业总结

- 超时有没有关系？
  - 有
  - 延长作业时间从来没有效果（student syndrome）
- 做不完有没有关系？
  - 没关系
  - 及时答疑，实在不会的可以空着，附加题更不用太勉强
- 做“不好”有没有关系？
  - 有
  - 要尽自己的全力
- 能不能抱怨？
  - 可以
  - 不要“骂”助教，朋友圈“骂”我就可以了
- 要不要回头看？
  - 一定要看示范作业
  - 学习并提高

# 第三周作业总结

- 共性问题
  - pdf全名要规范，便于助教迅速知道是谁
  - 数据清洗中应当包括对表情符的处理，这里比较建议仅仅去掉表情符号两端的方括号，将对表情的描述保留，有助于测量情绪。
  - 根据数据的类型和分析的目的来选择可视化的方式，可视化一定要有意义，能够方便数据洞察

School of Economics ang Management, Beihang University

# 现代程序设计技术

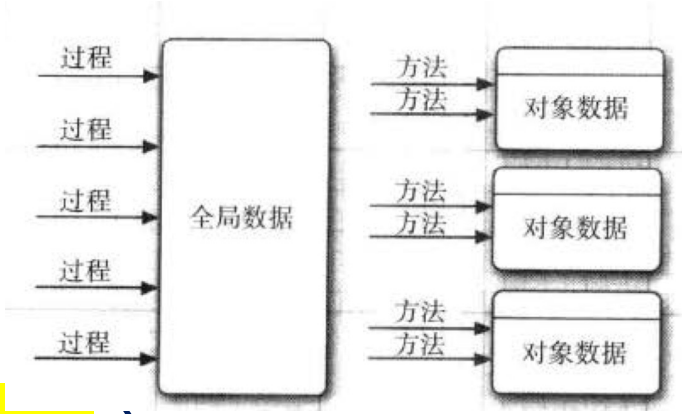
赵吉昌

# 本周内容

- Python基础
  - 面向对象程序设计
  - 自定义类

# 面向对象程序设计

- OO
  - 一种程序设计范式
  - 程序由对象组成，每个对象包含对用户公开的特定功能和隐藏的实现部分
  - 对象是数据与相关行为的集合
  - 不必关心对象的具体实现，只要能满足用户的需求即可
- 与结构化程序设计的差异
  - 将数据搁在第一位
  - 更加适用于规模大的问题
  - Shape的例子（第一周在线没讲清楚）



# 面向对象程序设计

- 类
  - 对象的类型，用来描述对象
  - 构造对象的模板
  - 定义了该集合中每个对象所共有的属性和方法
  - 由类构造对象的过程称之为实例化，或创建类的实例
- 一些重要概念
  - 多态
    - 可以对不同类型的对象执行相同的操作
  - 封装
    - 将数据和行为组合，并对外隐藏数据及行为的实现方式
    - 对象的状态：当下实例域的值的特定组合
  - 继承
    - 通过扩展一个类来建立另外一个类

# 面向对象程序设计

- 对象的特性
  - 行为
    - 通过可调用的方法定义
  - 状态
    - 对象属性当前的取值的组合
    - 对象状态的改变必须通过调用方法实现
  - 标识
    - 每个对象实例的标识应该是不同的

# 面向对象程序设计

表 4-1 表达类关系的 UML 符号

| 关　系 | UML 连接符 |
| --- | --- |
| 继承 |  |
| 接口实现 |  |
| 依赖 |  |
| 聚合 |  |
| 关联 |  |
| 直接关联 |  |

- 类与类之间的关系
  - 依赖
    - 一个类的方法操纵另一个类的实例
    - 耦合度及其最小化
  - 聚合
    - 类A包含类B的实例对象
  - 继承
    - 类A由类B扩展而来
    - 如果类A扩展类B，类A不但包含从类B继承的方法，还会拥有一些额外的功能
  - UML

# 面向对象程序设计

- 使用预定义类
  - 标准库
  - 第三方库
  - 文档
- 使用自定义类
  - 识别类
  - 名词对应类
  - 动词对应方法
  - 类间的关系
  - 个人的经验

# 自定义类

- 创建类
  - 使用 class 语句来创建一个新类，class 之后为类的名称并以冒号结尾：
  - class ClassName:
  - """类的信息"""
  - <statement-1>
  - .
  - .
  - <statement-N>
  - 补充：class ClassName(object)的写法

# 自定义类

- 创建类
  - 类的文档信息可以通过`ClassName.__doc__`查看
  - 类的定义要先执行才能生效
    - 可以在函数中或`if`等中进行类定义
  - 类会创建一个新的命名空间

# 自定义类

- 辨析
  - 类对象
    - 类对象支持属性引用和实例化
    - 属性引用可以是类变量，也可以是类方法
  - 实例对象
    - 实例对象仅能进行属性引用（数据或方法）

# 自定义类

- 辨析
  - 类变量
    - 类变量在所有实例化的对象中是公用的
    - 类变量定义在类中且在类方法之外
    - 类变量通常不作为实例变量使用
    - 类比于Java中的静态属性
  - 实例变量
    - 每个实例独有
    - 第一次使用时自动生成
      - 与类变量重名时仍作为实例变量

# 自定义类

- 类的数据属性
  - 只要能避免冲突，可以向一个实例对象添加自定义的数据属性，而不会影响方法的正确性
    - 内置类的实例对象并不支持
    - Python类不能用来实现纯净的数据类型
  - 可以通过del删除
    - `del c.name`
  - 应该谨慎地使用数据属性
    - 与方法重名或与类变量重名？
  - 从方法内部引用数据属性（或其他方法）并没有快捷方式
    - 往往需要通过实例对象的引用来间接访问

# 自定义类

- 补充：类的数据属性
  - 默认情况下通过字典__dict__维护数据属性
  - 能否像内置类一样不允许增加属性？
  - 定义__slots__，元组，类数据属性的描述
  - 类实例只能拥有slots中定义的数据属性，不能再增加新的数据属性
  - __dict__不能再使用

# 自定义类

- 类的方法
  - 在类的内部，使用 `def` 关键字来定义一个方法
  - 与一般函数定义不同，类方法必须包含参数 `self`，且为第一个参数
    - `self`并非`python`关键字，可写其他名称
    - 但应该按惯例写作`self`
    - `self`代表类的实例
    - 在调用时不必传入相应的参数
  - 同名的数据属性会覆盖方法

# 自定义类

- 类的方法
  - 通过实例调用一个方法就相当于将该实例插入到参数列表的最前面，并通过类对象来调用相应的函数
  - class D:
    - def f(self,a,b):
      - self.a=a
      - self.b=b
      - pass
  - d=D()
  - d.f(a,b) 等价于 D.f(d,a,b)吗？

# 自定义类

- 类的方法
  - 对于类对象，为函数(function)
  - 对于实例对象，为方法(method)
  - ```
    class T:
        def f(self):
            pass
    ```
  - ```
    print(type(T.f)) #function
    ```
  - ```
    t=T()
    ```
  - ```
    print(type(t.f)) #method
    ```

# 自定义类

- 类的方法
  - 方法定义不一定只在类中，也可以将一个函数对象赋值给类中的一个局部变量

```
def f1(self, x, y):
    return min(x, y)
class C:
    f = f1
c=C()
print(c.f(2,3))
print(type(c.f))#是method还是function?
```

# 自定义类

- 类的方法
  - 构造方法
    - `__new__()`为构造方法，参数为将要构造的类，无`self`参数，返回新创建的实例对象
    - 极少使用（后续内容中会涉及到）
  - 初始化方法
    - `__init__()`为初始化方法，在类实例化时会自动调用
    - 有`self`参数
    - 也可以有除`self`外的其他参数
    - 只能返回`None`
    - 较为常用

# 自定义类

- 类的私有性
  - 严格来讲，python类的属性和方法都是对外公开的
    - 通过类对象或实例对象可以访问
  - 通过一些“约定”来约束外部的访问
    - 在属性或方法前加_（protected)
    - 在属性或方法前加__（private）
    - private时会发生命名改装（name mangling)，外部访问时需要加上_<类名>的前缀
    - 但仅为约定
      - __dict__可以查看

# 自定义类

- 类的私有性
  - 属性一般应该protected私有
    - 封装性的体现：不宜通过实例直接对属性进行访问或修改，而应该通过类所提供的方法进行可控的访问
    - 子类可以继承
    - _属性名 这种写法更常见
  - 方法也可以私有
    - 避免对实例状态的破坏：一些方法可能只在类内才被使用，其使用容易对类的相关状态产生影响，需要在可控的情形下调用，且并非该类对外提供的功能

# 本周作业

- 实现文本处理的Tokenizer类
  - 具体要求见作业的附件
  - 提供的代码结构仅为示例，可以自主调整，增加数据属性或私有方法等
  - 附加作业设计到文本的预训练模型，如果太麻烦可以暂不考虑