

- 共性问题
  - 没有对文本长度进行统计
    - 选用特定的句子长度的原因
  - 没有讨论和one-hot的区别
  - 很多同学都尝试了BERT预训练模型
    - 值得鼓励
    - 有些同学直接用tokenizer出来的结果算欧几里得距离
    - 但是有些同学好像花费了很多时间但没有成功



School of Economics and Management, Beihang University

# 现代程序设计技术

赵吉昌

- 面向对象编程
  - 工厂模式(Factory)
  - 异常处理

- 对象的职责
  - 对象本身所具有的职责
  - 创建对象的职责
  - 使用对象的职责
- 创建对象的方式
  - 直接创建对象
  - 通过复制创建对象
    - 注意浅、深的区别 ( Demo: `sd.py`)
    - `pickle`
  - 通过工厂类创建对象

- 创建对象和使用对象的职责耦合在一起
  - 耦合太强
  - 可能影响程序的扩展
- 解决的场景
  - 两个类A和B之间的关系应该仅仅是A创建B或是A使用B，而不能两种关系都有
  - 将对象的创建和使用分离，也使得系统更加符合“单一职责原则”，有利于对功能的复用和系统的维护
  - 防止用来实例化一个类的数据和代码在多个类中到处都是
  - 一些类实例的构建过程可能十分复杂，需要进行封装
  - 为类实例的构建提供更好的多态支持（不确定具体对象）

- 工厂模式的好处
  - 解耦
  - 降低代码重复
  - 减少了使用者因为创建逻辑导致的错误
  - Demo : 水果工厂
    - 简单工厂，工厂方法，抽象工厂
    - `Fruit, Apple, Orange, FruitFactory`
    - `f.py`

- 异常与处理

- 输入数据或设备状态不会一直理想并正确
- 应避免程序的错误或外部环境的影响给用户造成不佳的使用检验
  - 向用户通告可能的错误
  - 保存所有的已有结果
  - 允许用户以妥善的形式退出程序

- 常见异常
  - 用户输入错误
    - 格式、语法
  - 设备错误
    - 硬件故障、临时性下线
  - 物理限制
    - 空间不足
  - 代码错误
    - 对象为空、无效数据、除0



- python Exception hierarchy

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
        +-- FloatingPointError
        +-- OverflowError
        +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
    +-- ImportError
        +-- ModuleNotFoundError
    +-- LookupError
        +-- IndexError
        +-- KeyError
    +-- MemoryError
    +-- NameError
        +-- UnboundLocalError
    +-- OSError
        +-- BlockingIOError
        +-- ChildProcessError
        +-- ConnectionError
            +-- BrokenPipeError
            +-- ConnectionAbortedError
            +-- ConnectionRefusedError
            +-- ConnectionResetError
        +-- FileExistsError
        +-- FileNotFoundError
        +-- InterruptedError
```

```
+-- IsADirectoryError
+-- NotADirectoryError
+-- PermissionError
+-- ProcessLookupError
+-- TimeoutError
+-- ReferenceError
+-- RuntimeError
    +-- NotImplementedError
    +-- RecursionError
+-- SyntaxError
    +-- IndentationError
        +-- TabError
+-- SystemError
+-- TypeError
+-- ValueError
    +-- UnicodeError
        +-- UnicodeDecodeError
        +-- UnicodeEncodeError
        +-- UnicodeTranslateError
+-- Warning
    +-- DeprecationWarning
    +-- PendingDeprecationWarning
    +-- RuntimeWarning
    +-- SyntaxWarning
    +-- UserWarning
    +-- FutureWarning
    +-- ImportWarning
    +-- UnicodeWarning
    +-- BytesWarning
    +-- EncodingWarning
    +-- ResourceWarning
```

- 异常捕获

- try...except

- try:
      - <statements>
      - ...
    - except 异常:
      - <statements>
    - except 异常2:
      - <statements>
    - except 异常3:
      - <statements>
    - ...

- 异常捕获
  - 执行try子句
    - 在try和except关键字之间的部分
  - 无异常发生时except子句被忽略
  - 如try子句在执行过程中发生异常，则该子句其余部分会被忽略
  - 如异常匹配except指定的异常类型，则执行对应的except子句
  - 如发生异常在except子句中没有与之匹配的分支，则传递到上一级 try 语句
  - 如最终仍找不到对应的处理语句，则作为未处理异常，终止程序运行并显示提示信息

- 异常捕获

- 包含多个 `except` 子句时最多只有一个分支会  
被执行
- 处理程序应只针对对应的 `try` 子句中的异常进  
行处理，而不是其他的 `try` 的处理程序中的  
异常
- 一个 `except` 子句也可以同时处理多个异常
  - 这些异常将被放在一个括号里成为一个元组
  - `except (RuntimeError, TypeError,  
NameError) :`
    - `pass`

## • 异常捕获

- 最后一个 `except` 子句可以省略异常名称以作为通配符使用
  - 慎用，会隐藏一个实际的程序错误信息
- 可以使用这种方法打印一条错误信息,然后重新抛出异常 (允许调用者处理这个异常)

```
try:
    pass
except OSError as err:
    print("OS error: {0}".format(err))
except: #也可以写成except BaseException:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

- `sys.exc_info` 返回元组: `exc_type` 是异常的对象类型, `exc_value` 是异常的值, `exc_tb` 是一个 `traceback` 对象, 其中包含出错的行数、位置等

## • 异常捕获

- `try ... except` 语句可以带有一个 `else` 子句，该子句只能出现在所有 `except` 子句之后
- 当 `try` 子句没有抛出异常时，需要执行一些代码，可以使用 `else` 子句
- 使用 `else` 子句比在 `try` 子句中附加代码要好，因为这样可以避免 `try ... except` 意外地截获本来不属于它们保护的那些代码抛出的异常

```
- try:
    • f = open(arg, 'r')
- except IOError:
    • print('cannot open', arg)
- else:
    • print(arg, 'has', len(f.readlines()), 'lines')
    •
```

- 处理捕获

- 异常处理并不仅仅处理那些直接发生在`try`子句中的异常，还能处理子句中调用的函数（甚至间接调用的函数）里抛出的异常

- 异常抛出

- `raise`语句允许程序员强制抛出一个指定的异常

- `raise NameError('Oh Error Happens')`

- 要抛出的异常必须是一个异常实例或异常类（继承自`Exception`的类）

- `raise`语句也能重新抛出异常

- 不想处理或无法处理

- `raise`



- 用户自定义异常
  - 直接或间接的从Exception类派生
    - 注意不是从BaseException派生
  - 异常类的命名多以 “Error” 结尾
  - 通常为了保持简单，异常类只包含属性信息，以供异常处理
  - 如果一个新建的模块中需要抛出几种不同的错误，通常的作法是为该模块定义一个异常基类，然后针对不同的错误类型派生出对应的异常子类
  - Demo
    - account.py

- 用户自定义异常类

- 发生异常时一般应通过参数提供更细致、具体的异常信息，可作为异常类的属性存在
- 在异常名（列表）之后可以为 `except` 子句指定一个变量
  - 该变量将绑定于一个异常实例
  - 附属的参数存储在 `args` 属性中
- 异常实例一般会定义 `__str__()`
  - 可以直接访问并打印参数而不必引用参数
  - 并不鼓励仅这么做
- 更好的做法是给异常传递参数
  - 如果要传递多个参数，可以传递一个元组
  - 一旦异常发生，能够在抛出前绑定所有指定的属性
    - 比如构建一个 `message` 属性，包含异常的详细信息

- 用户自定义异常类

- try:

- `raise Exception('arg1', 'arg2')`

- except Exception as inst:

- `print(type(inst))`

- `print(inst.args)`

- `print(inst)`

- 定义清理行为
  - 通过`finally`子句实现
  - 定义了无论在任何情况下都会执行的清理行为
    - 如资源释放等操作
  - 无论`try`子句有无发生异常, `finally`子句都会执行
    - 即使`try`进行了`return`
  - `try:`
    - `result = x / y`
  - `except ZeroDivisionError:`
    - `print("division by zero!")`
  - `else:`
    - `print("result is", result)`
  - `finally:`
    - `print("executing finally clause")`

- 预定义的清理行为
  - 可以为类定义标准的清理行为
    - `__enter__()` 方法：进入时调用，注意要有返回值
    - `__exit__()` (四个参数) 方法：离开时调用
  - `with ... as`
  - Demo
    - `withdb.py`
- 在 `finally` 子句中应只做打印错误信息或者关闭资源等操作
- 避免在 `finally` 语句块中再次抛出异常

- `return`语句的位置
  - 不要在`try`, `except`, `else`子句里写返回值
    - 尤其是有`else`子句时, `try`中的`return`会导致`else`不执行
  - 如果`finally`有返回值, 其将会覆盖原始的返回值
  - Demo: `tryr.py`

- 异常处理的原则
  - 异常处理不能代替简单的测试
    - 耗时，成本高
    - 只在异常情况下使用异常处理
  - 不要过分地细化异常
    - 代码量膨胀
  - 利用异常层次结构
    - 自定义类
  - 不要压制异常
    - `except`里什么也不做
  - 早抛出、晚捕获

- 断言

- 用于判断一个表达式，并在表达式`False`的时候触发异常

- `assert expression`

- 等价于

- `if not expression:`

- `raise AssertionError`

- `assert` 后面也可以紧跟参数

- `assert expression [, arguments]`

- `if not expression:`

- `raise AssertionError(arguments)`

- Demo

- `assert 1==2, '1等于2'`



# 本周作业



- 经济管理中通常有大量的数据以csv等结构化格式存在，如本次作业要用的空气质量数据。数据见在线平台，格式说明如  
<https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data>。
  - 1. 至少实现一个数据分析类，以提供数据的读取及基本的时间（如某区域某类型污染物随时间的变化）和空间分析（某时间点或时间段北京空气质量的空间分布态势）方法。
  - 2. 至少实现一个数据可视化类，以提供上述时空分析结果的可视化，如以曲线、饼、地图等形式对结果进行呈现。
  - 3. 如果数据中包含空值等异常值（可人工注入错误数据以测试程序），在进行数据分析以及可视化前需要检查数据。因此需要实现NotNumError类，继承ValueError，并加入新属性region, year, month, day, hour, pollutant，对数据进行检测，若取到的一系列数据中包含空值等明显错误，则抛出该异常，并提供异常信息。在此基础上，利用try except捕获该异常，打印异常信息，并对对应位置的数据进行适当的填充。
  - 4. （附加）污染物含量与气象状态本身是否有相关性？请丰富数据分析类和数据可视化类，增加关于这些相关性探索的方法。
  - 5. （附加）思考不同区域时间变化的趋势及差异的管理意义。
  - 提示：了解并使用csv（标准库），**openpyxl**等来读取csv等结构化文件，或直接视为一般文本文件读取。