School of Economics ang Management, Beihang University

现代程序设计技术

赵吉昌
jichang@buaa.edu.cn

关于作业的回头看

- 对上次作业的一些释疑
  - 自己主要需要实现哪些方法
    - 根据提供的文档和窗口长度生成网络
    - 根据提供的权重进行边的删除
  - 如何分析网络
    - 找关键词：用度、pagerank等排序
    - 找话题：社团分析
  - 如何可视化网络
    - 规模不能太大
    - 可以用Gephi工具

本周内容

- 面向对象编程
  - 工厂模式(Factory)
  - 异常处理

工厂模式

- 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（sd.py)
    - pickle
  - 通过工厂类创建对象

工厂模式

- 创建对象和使用对象的职责耦合在一起
  - 耦合太强
  - 导致严重的问题
- 解决的途径
  - 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

工厂模式的好处

- 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
    - Fruit, Apple, Orange, FruitFactory
    - f.py

异常处理

- 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

异常处理

- 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

异常处理

- python内置异常

```
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
```

```
      |    +-- 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
           +-- ResourceWarning
```

异常处理

- 异常捕获
  - try...except
    - try:
      - <statements>
      - …
    - except 异常:
      - <statements>
    - except 异常2:
      - <statements>
    - except 异常3:
      - <statements>
    - …

异常处理

- 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

异常处理

- 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`

异常处理

- 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

异常处理

- 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')
```

异常处理

- 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

异常处理

- 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`

异常处理

- 用户自定义异常
  - 直接或间接的从Exception类派生
  - 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

异常处理

- 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

异常处理

- 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)

异常处理

- 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
  - try:
    - result = x / y
  - except ZeroDivisionError:
    - print("division by zero!")
  - else:
    - print("result is", result)
  - finally:
    - print("executing finally clause")

异常处理

- 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
    - `wtest.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

异常处理

- return语句的位置
  - 不要在try，except，else子句里写返回值
  - 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

- 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

异常处理

- 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

- 无。