作业总结

- 代表性评论
  - 数据量有限，特征词过多
  - 可采用蒋晓桐等同学的方法，即计算重心时，过滤特征词数量过少的样本
- 函数封装
  - 部分同学在jupyter-notebook里完成，没有使用函数
  - 建议在jupyter中也采用函数对功能进行封装，同时减少不同代码块之间的依赖（参考示例作业）
  - 从这周起，对封装的要求不断提高，请务必注意
  - 关于jupyter使用的取舍
- 提交形式
  - 按作业要求整理处理思路和相关结果到文档并提供实现代码（参照示例作业）

School of Economics ang Management, Beihang University

现代程序设计技术

赵吉昌
jichang@buaa.edu.cn

本周内容

- Python基础
  - 模块
  - 输入输出

模块

- Module
  - 一个Python文件，以.py结尾，包含了对象定义和可执行语句
    - 模块能定义函数，类和变量，模块里也能包含可执行的代码
    - 更有逻辑和层次地组织Python代码
    - 相关的代码进行单独的组织会使代码更容易理解并被复用
  - 模块的模块名（字符串）可以由全局变量__name__得到
  - python module.py时，其__name__被设置为"__main__"

import语句

- import
  - import module1[, module2[,…]]
  - import语句应该放在模块的初始位置
  - 当解释器遇到import语句时，如果模块在当前的搜索路径就会被导入
    - 搜索路径为当前目录或存储在sys.path的路径
  - 可通过模块名称来访问模块成员
    - 如果频繁使用一个函数，则可以将之赋给本地变量
  - 模块在解释器会话中只导入一次
    - 如果更新，需要重启解释器或者重新加载
      - import imp
      - imp.reload(modulename)#交互式情况下会用到

from ... import语句

- from...import
  - 从模块导入指定的部分到当前的命名空间
  - 既可以是子模块（甚至子包），也可以是函数、类或变量等
  - from ... import *
    - 导入所有部分
    - 除下划线(_)开头的命名
    - 不建议过多使用
      - 低效繁杂

可执行语句

- 可执行语句
  - 模块可以包含可执行语句
  - 这些语句一般用来初始化模块，其仅在第一次被导入的地方执行一次

全局变量

- 全局变量
  - 模块私有符号表，并被模块内所有的函数作为全局符号表使用
  - 模块内部可以使用全局变量，而无需担心与其他模块全局变量冲突
  - 可通过模块访问其全局变量
    - `modname.itemname`

模块加载

- 加速
  - 在__pycache__目录下以module.version.pyc 名字缓存模块编译后的版本
  - 编译后的模块是跨平台的
  - 解释器会检查源文件与编译版本的修改日期以确定是否过期并需要重新编译
    - 如果没有源文件则不会检查缓存
  - 若支持没有源文件（只有编译版）的程序发布，则编译后的模块必须在源目录下

模块加载

- 优化
  - 为了减少编译模块的大小，可在Python命令行中使用-O或者-OO“优化”
    - -O参数删除了断言语句
    - -OO参数删除了断言语句和 __doc__字符串
  - 缓存文件名会发生变化
    - 包括opt等优化标记
  - 仅在确定无误的场合使用这一选项
    - 并非加速代码的执行速度
    - 程序逻辑可能依赖__doc__等
  - compileall为指定目录的所有模块创建缓存
    - python -m compileall 目录

dir()函数

- dir()
  - 按模块名搜索模块定义并返回一个字符串类型的命名列表
  - 无参数调用时，dir()函数返回当前模块定义的命名

包

- 包
  - 模块集，结构化的模块命名空间
    - 圆点.
    - A.B的模块表示了名为A的包中名为B的子模块
  - 避免模块之间的命名冲突
  - 通过分层的文件体系来组织
    - 为了让Python将目录当做内容包，必须包含__init__.py文件
    - 一般空白即可
    - 可以执行包的初始化代码
    - __all__变量

包

- 包文件组织的示例

```
sound/                          顶层包
      __init__.py               初始化 sound 包
      formats/                  文件格式转换子包
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  声音效果子包
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  filters 子包
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

包

- 包的导入
  - 可只导入包里的特定模块
  - `import a.b.c`
    - 通过完整的名称来引用
    - `a.b.c.fun()`
  - `from a.b import c`
    - `c.fun()`
  - `from a.b.c import fun`
    - `fun()`

包

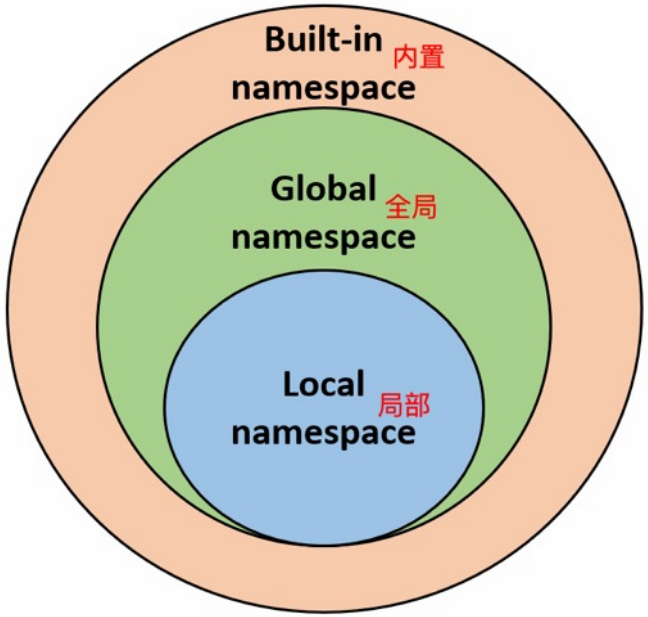
- 包的导入
  - `from package import *`
  - 如果包中的`__init__.py`代码定义了一个名为`__all__`的列表，就会按照列表中给出的模块名进行导入
  - 如果没有定义`__all__`，则不会导入所有子模块

包

- 包内引用
  - `from module import name` 来显式地从相对位置导入
  - 用点号标明关联导入当前和上级包
  - `from . import b`
  - `from .. import d`
  - `from ... import e`

命名空间

- namespace
  - 从名称到对象的映射
  - 避免名字冲突的一种方法
    - 各个命名空间是独立的
    - 虽然一个命名空间中不能有重名，但不同的命名空间中可以重名
  - 三种类型



三种命名空间

命名空间

- 三种类型
  - 内置名称(built-in names)
    - Python语言内置的名称，比如函数名abs和异常名称Exception等
  - 全局名称(global names)
    - 模块中定义的名称，如函数、类、其它导入的模块、模块级的变量和常量等
  - 局部名称(local names)
    - 函数中定义的名称，如函数的参数和局部变量等，或类中定义的函数及变量等
  - 查找顺序
    - 局部的命名空间->全局命名空间->内置命名空间

命名空间

- 命名空间的生命周期
  - 命名空间的生命周期取决于对象的作用域，如果对象执行完成，则该命名空间的生命周期就结束
  - Python 中只有模块(module)，类(class)以及函数(def、lambda)才会引入新的作用域，其它的代码块（如 if/elif/else/、try/except、for/while等）并不会引入新的作用域
    - 在这些语句内定义的变量，在代码块外也可访问
    - 与c语言是否一样？

作用域

- 四种作用域
  - L(Local)
    - 最内层，包含局部变量，比如一个函数或方法内部
  - E(Enclosing)
    - 包含了非局部(non-local)也非全局(non-global)的变量，比如嵌套函数等，一个函数A 中包含了另一个函数B，那么对于B，A中的作用域就为 nonlocal
  - G(Global)
    - 当前模块的最外层，比如当前模块的全局变量
  - B(Built-in)
    - 包含了内建的变量/关键字等
  - 搜索规则顺序
    - L -> E -> G ->B

作用域

- 一个例子
  - a = 10
  - def fa():
  - a= a + 1
  - print(a)
  - fa()
  - 这段代码能否正常运行？

虚拟环境和包

- 虚拟环境
  - Python应用程序经常会使用一些不属于标准库的包和模块，或者应用程序有时候需要某个特定版本的库
  - 导致无法安装一个Python来满足每个应用程序的要求
  - 创建一个虚拟环境（通常简称为“virtualenv”），包含一个特定版本的Python，以及一些附加的包的独立的目录树

虚拟环境和包

- pyvenv
  - 指定安装一个特定的版本的Python
  - 或选择一个指定的Python版本
  - pyvenv 虚拟环境目录
    - 会创建一个目录，并且也在目录里面创建一个包含Python解释器，标准库，以及各种配套文件的 Python “副本”
  - 激活环境
    - win: Scripts/activate
    - linux/unix: source /bin/activate
    - 激活虚拟环境会改变shell提示符，显示正在使用的虚拟环境
  - 可以使用pip进行对应版本的包管理

输入输出

- format()函数
  - 通过位置
    - print('a1 = {} a2= {} a3= {}'.format('first','second','third'))
    - print('a1 = {1} a2= {0} a3= {2}'.format('first','second','third'))
  - 通过关键字参数
    - print('your name is {name} , age is {age}'.format(age=87,name='jack'))
  - 通过对象属性
    - print('name = {p.name} age = {p.age}'.format(p=p))

输入输出

- format()函数
  - 通过对象属性
    - dt = {'k1': 1, 'k2': 2, 'k3': 3}
    - print('k1: {0[k1]}; k2: {0[k2]}; k3: {0[k3]}'.format(dt))
    - print('k1: {k1}; k2: {k2}; k3: {k3}'.format(**dt))

输入输出

- format()函数
  - 通过下标
    - print('{0[1]} {0[2]} {1[2]} {1[0]}'.format(s1,s2))
  - 格式化输出
    - 对齐与填充
      - print('{0:10} ==> {1:10d}'.format(name, number))
    - 浮点小数输出
      - print('常量 PI 的值近似为 {0:.3f}。'.format(math.pi))

输入输出

- format()函数
  - 格式化输出
    - 进制及其他显示
      - b ： 二进制
      - d ：十进制
      - o ：八进制
      - x ：十六进制
      - !s ：将对象格式化转换成字符串
      - !a ：将对象格式化转换成ASCII
        - » print('{!a}'.format('天津tianjin'))
      - !r ：将对象格式化转换成repr

输入输出

- f-strings
  - 语法更间洁，速度也更快

```
year = 2016
event = 'Ceremony'
print(f'Results of the {year} {event}')

yes_votes = 42_572_654
no_votes = 43_132_495
percentage = yes_votes / (yes_votes + no_votes)
print(f'{yes_votes:-9} YES votes  {percentage:2.2%}')

animals='eels'
print(f'My hovercraft is full of {animals!r}.')
```

输入输出

- 序列化与反序列化
  - 序列化（serialization)
    - 将程序中的对象保存到文件中以永久存储
  - 反序列化（deserialization）
    - 从文件中创建上一次程序保存的对象
  - In computer science, in the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment.
  - When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object.

输入输出

- pickle
  - import pickle
  - #文件
  - pickle.dump(obj, file, [,protocol])
  - pickle.load(file)
  - #字符串
  - pickle.dumps(obj[, protocol])
  - pickle.loads(string)

输入输出

- json
  - import json
  - #文件
  - json.dump(obj,f)
  - json.load(f)
  - #字符串
  - json.dumps(obj)
  - json.loads(str)

Python	JSON
dict	object
list, tuple	array
str	string
int, float, int- & float-derived Enums	number
True	true
False	false
None	null

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

输入输出

- 文件操作
  - os模块
    - 目录遍历
    - 文件是否存在
  - shutil模块
    - 文件移动
    - 文件删除
    - 文件复制
  - zipfile模块
    - 压缩文件读取或存储

本周作业

- 图是非常重要的一种数据结构。本次作业提供了newmovies数据集，希望基于该数据，实现一个简单的图分析包，以在相应的模块中读取并存储用户节点信息，建立无向图结构，并进一步实现相关统计分析甚至可视化功能。
  - 其他细节见课程中心的要求（这里搁不下了）
  - 注意：主要强调对包结构的设计和组织
  - 附加：了解并学习networkx库，尤其是对社交网络等相关方向感兴趣的同学。