项目总体规划：总体是实现Python/Flask/Git/Virtualnev一个博客系统核心模块的开发、扩容，从一个简单的项目开始，最后实现一个复杂的可用、可以在阿里云/aws/自己电脑上部署的博客系统。

我们会提供主体脚手架代码，然后主要实现：我们增加的小功能、测试代码、部署自动化会分批次拆分。

主要考察：

1. 提供相关tutorial，然后去实现我们想要的新feature的程序设计/程序实现
2. 对我们的feature的需求进行时间评估，可以采取敏捷方式拆分feature为更细的颗粒度为1，2，3，5个点（每个点代表自己2小时的工作量），进一步评估开发时间和文档、测试代码的维护时间（可能涉及到重构、测试代码变更）
3. 新feature对应的文档编写，新feature引入的函数的注释，新feature对应的用户操作手册也需要维护（可以手写，可以通过pydoc之类根据注释生成然后修改，但是需要根据feature增加/修改而变化）
4. 每个阶段必须保证项目可运行，如果新加入的feature引起代码重构/软件重组时，重构代码也需要相应的注释和文档，和相应的接口测试（每个阶段都需要运维人员知道怎么去部署，运行）
5. 项目的打包、虚拟环境隔离、本地维护环境和实际生产环境部署的区别，考察系统提交
6. 如果是小组合作性质，共同完成（将来我们可以考察维护人员各种不同的角色和分工，代码提交的工作流是否符合git flow的工作方式等）
7. 如果是小组合作性质，共同完成（将来我们可以考察代码风格和格式的统一性，是否维护一套代码规范，减少软件维护的成本）
8. 将来相关新feature的对应的模块单元测试/系统集成测试功能

—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-—-

背景资料、操作指南和必读须知

一.

阅读python3 相关文档，了解相关语法结构

操作指南：<https://docs.python.org/zh-cn/3/tutorial/index.html>

二.

对代码的可读性和编码标准经常就忽略了（甚至从一开始就没有建立起这些标准。）但是，在代码库中保持一致的风格和测试标准是减少软件维护负担的重要因素，它能确保未来的开发人员能够快速了解新的项目情况 ，并保持项目的良好运行.

因为我们下面要开发、维护的是基于python/flask的工程，所以请先了解，安装flake8,检查某些Python代码的风格和质量:

操作指南：https://flake8.pycqa.org/en/latest/

三.



软件开发和软件维护在人员和时间上存在差异。如果软件维护工作是由该软件的开发人员完成，则维护工作相对比较容易，因为这些人员熟悉软件的功能和结构等。但是，通常开发人员和维护人员是不同的，况且维护阶段持续时间很长，可能是10～20年的时间，原来的开发工具、方法和技术与当前有很大的差异，这也造成了维护的困难。container技术则让开发人员维护容器，运维和开发人员变得更紧密。

了解docker相关部署方法，复习上次作业的相关阅读材料（主要是Q4～Q6）

使用docker安装mysql镜像：

操作指南：<https://dev.mysql.com/doc/mysql-installation-excerpt/8.0/en/docker-mysql-getting-started.html>

四.

Flask是一个使用Python编写的轻量级Web应用框架，

阅读了解Flask相关文档，

了解使用和配置

操作指南：<https://dormousehole.readthedocs.io/en/latest/>

五

了解pip3使用（python3.4之后内置），如果是低于python3.4版本，需要安装python3对应的pip依赖库管理：python3-pip

因为手工安装各种依赖库会降低软件的可维护性，增加不确定性，使用pip3安装和管理第三方库，还会管理库之间的依赖(dependency)关系

六（可选，可以作为附加分，或者必选，老师自行决定）

使用virtualenv 用来创建隔离的Python环境，降低工程维护成本，提供移植性。

处理python环境的多版本和模块依赖，以及相应的权限是一个很常见的问题。保证了每个模块的独立性。比如需要同时部署A和B两个项目，A项目要用到a这个库，而a这个库又依赖于c这个库的1.0版本。同时呢，B项目需要用到b这个库，b又依赖于c的2.0版本，这时候如果直接都安装在系统上的话， 就不一定是哪个坏掉了。所以使用这个虚拟环境还是十分有必要的

操作指南：https://virtualenv.pypa.io/en/latest/

具体项目须知

**Day1~Day2**

day1\_2start 文件夹是2个人日工作量的初始代码（提供给学生）

**day1\_2end** 文件夹是**2**个人日工作量工程结束后的参考代码（学生可以不用**100%**实现得一模一样，但是大致会是这样子，提供给老师做参考，不要提供给学生）

项目要求：

**1**

启动mysql镜像，在mysql数据库中创建blog\_db数据库

**2**

通过SQLAlchemy 使用ORM来创建用户表：User，提供享用CURD操作（增删改查）

**3**

通过SQLAlchemy 使用ORM来创建博客表：Post，提供享用CURD操作（增删改查）

Post需要和User表维护创建者关联、创建时间关系，通过SQLAlchemy的外键实现

**4**

通过SQLAlchemy 使用ORM来创建博客评论表：Comment，提供享用CURD操作（增删改查）

Comment需要和Post表关联（通过post.id），通过SQLAlchemy的外键实现

**5**

通过SQLAlchemy 使用ORM来创建博客标签表：Tag，提供享用CURD操作（增删改查）

**6**

评估以上所有1～5需求的实现时间，对我们的feature的需求进行时间评估，可以采取敏捷方式拆分feature为更细的颗粒度为1，2，3，5个点（每个点代表自己2小时的工作量），进一步评估开发时间和文档、测试代码的维护时间（可能涉及到重构、测试代码变更）

**7**

需要对Flask构建的类进行文档描述，说明自己创建的列的属性、名字和意义。主要的方法也需要在文档中注释出主要的功能，参数类型和用法

**8**

软件项目部署的维护：完成ORM之后，为了提供软件维护的便捷性，使用Alembic维护所有的数据库建表、数据库结构修改操作，阅读相关文档：https://flask-migrate.readthedocs.io/

Python3 manage.py db init

Python3 manager.py db imgrate -m“inital migrate”

需要在migrate时候添加有意义的注释和生成名字，提高代码可理解性，从而降低软件维护的难度。如果无意义的名字和注释，会酌情扣分

**9**

完成1~5所有功能性的feature代码编写，完成6～8的软件维护类工作，完成之后需要使用flake8检查全部代码文件的代码风格和规范一致性，减少后续软件维护的难度，提高代码可阅读性，运行：flake8 \*.py 检查所有python文件，检查和处理所有的语法和规范的报警信息。

**Day3~Day5**

day3\_5static 是提供素材：css样式库、字体库, 相关js库，可以在Day3~5开始前提供给学生

day3\_5start 文件夹是学生在结束Day1、Day2之后工程的结果（是否在第**3**天开始阶段提供**day1\_2end**代码工程样例给学生，由老师自行决定，我觉得如果作业分**2**次是可以在第一次做完提交完之后，提供出来**day1\_2end**的样例）

**day3\_5end** 文件夹是**2**个人日工作量工程结束后的参考代码（学生可以不用**100%**实现得一模一样，但是大致会是这样子，提供给老师做参考，不要提供给学生）

项目要求：

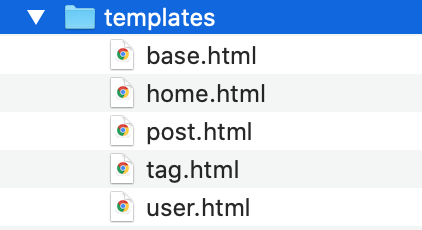
**10**

完成了数据组织后，需要在浏览器中显示信息。

我们使用Flask Jinja 内置模版语言, 学习相关文档：

<https://dormousehole.readthedocs.io/en/latest/templating.html>

**11**



请使用模版语言，在项目根目录创建template的新目录 ，创建5个页面：

base.html, 其他home.html, post.html, tag.html, user.html 都继承自base.html, 根据DRY原则进行页面架构。为了统一，请使用Bootstrap样式库编写css，使用至少 v3及以上版本，具体UI 布局可以自由选择。

根据DRY原则（Don’t repeat myself)使用模版集成，DRY原则，有人称之“一次且仅此一次”原则，其实就是“不要重复”。这句话也是著名的《程序员修炼之道》中的核心原则之一。软件整体架构简化从而易于理解和软件维护，方便扩展。目标就是“高内聚，低耦合”。计算机科学家Craig Larman说过：“最好维护的代码就是没有代码。好的程序员的代码产量是负的。他通过减少代码来增加功能”

**12**

Home实现html页面和后端main.py中具体功能如下：

Home.html主要显示：最近的5条Post数据，最流行的Tags，

同时后端py代码中实现后端对应数据封装功能，通过类似

return render\_template(

'home.html',

posts=posts,

recent=recent,

top\_tags=top\_tags

)

方式提供给前端html网页，url地址为：@app.route(‘/') 或者 @app.route('/<int:page>')

**13**

Post.html主要提供：添加post数据，和现实post数据的功能，根据GET/POST方法不同后端提供不同数据给前台。

请使用 WTForms实现提交功能

url地址为：@app.route(‘/post/<int:post\_id>', methods=('GET', ‘POST’))

**14**

tag.html 提供：对应tag\_name的tag信息，该tag对应的post数据列表

url地址：@app.route(‘/tag/<string:tag\_name>')

**15**

user.html提供：用户名对应的用户信息介绍，该用户发表的所有post的数据

url地址：@app.route(‘/user/<string:username>')

**16**

完成11～15的功能，你应该可以得到一个具有完整功能的博客程序，并且可以实际部署到自己电脑/阿里云/aws上实际运行了。同时你需要对需求11～15完成需求6和需求7的项目工作量评估，和工程文档编写