1. 第一节课
   1. 使用flask创建app对象
   2. 3个模式：
      1. debug
      2. 修改port
      3. 修改host
2. 第二节课
   1. 理解url/根路由
   2. 定义url
      1. 不带参的url
      2. 带参数的url
         1. 参数固定到path中
         2. 查询字符串传参数
3. 第三节课
   1. 如何给前端网页做渲染(render\_template)
      1. 静态
         1. 单纯Html内设置文字
      2. 动态
         1. 在url中提取内容传给html，在html中渲染
4. 第四节课
   1. 如何把app.py中的对象传入html
      1. 创建instance
      2. 创建字典
5. 第五节课
   1. 如何使用过滤器？
      1. 在html中输入{{param| filter}}而不是{{param}}
   2. 如何diy过滤器？
      1. 在app中创建diy的函数，并加入到template中
      2. 在html中使用过滤器 e.g. {{param| diy\_filter}}
6. 第六节课
   1. If
      1. {%if (condition)%}
      2. <div>…</div>
      3. {%elif (condition) %}
      4. <div>…</div>
      5. {%endif%}
   2. For
      1. {%for … in … %}
      2. <div>…</div>
      3. {%endfor%}
7. 第七节课
   1. 模板继承
      1. 为什么需要？因为子页面和首页有很多重复，但子页面也有自己专属的东西
      2. {%extends “base.html”%}
   2. 子页面继承后如何加自己独有的东西？ — block
      1. 继承title
         1. {%block title%}{%endblock%}
      2. 继承body
         1. {%block body%}{%endblock%}
8. 第八节课
   1. 在html中加载jpg, css, js文件
      1. Jpg
         1. 放在body里
         2. <img src = {{url\_for(“static”, filename= “images/…”)}}>
      2. Css
         1. 放在title里
         2. <link rel =“stylesheet”, href = {{url\_for(…)}}>
      3. js
         1. 放在title里
         2. <script src = {{url\_for(…)}}>
9. 第九节课
   1. mysql8安装
   2. navicat安装
   3. 用flask连接mysql数据库和app
10. 第十节课
    1. Orm模型 — 创建table
       1. 创建User class并将其作为一个table
       2. 加入id, username, password作为key
       3. db.create\_all()将表同步到数据库
11. 第十一节课
    1. orm模型— CRUD操作
       1. Add — user = User(….), db.session.add(user), db.session.commit()
       2. Query — users = User.query.filter\_by(username =…), for user in users: print…
       3. Update — user = User.query.get(1), user.password = …,db.session.commit()
       4. Delete — user = User.query.get(1), db.session.delete(user), db.session.commit()
12. 第十二节课
    1. ORM模型外键
       1. 使用 author = db.relationship(**"User"**,backref =**"articles"**)
          1. 让articles成为User class的一个属性
          2. 我们可以通过user读取它所有文章
13. 第十三节课
    1. 使用flask\_migrate让表实时更新到数据库
    2. Flask db init — flask db migrate — flask db upgrade
14. 实战第一节课
    1. 问答平台结构搭建: extension, models, app, blueprints
       1. Extension
          1. 创建了一个空的db
       2. Models
          1. 包含所有的ORM model
       3. Blueprints
          1. Qa — 问答界面 bp = Blueprint(**"qa"**,\_\_name\_\_, url\_prefix=**“/“**)
          2. Auth — 用户注册
15. 实战第二节课
    1. User模型的创建
       1. Id, username, password, email, join\_time
16. 实战第三节课
    1. 使用jinja2模版对静态html改造
       1. 把register.html继承自base.html，从而一个父模板可重复利用
17. 实战第四节课
    1. Flask邮件发送功能实现
       1. 在config设置邮箱配置
       2. 在exts引入Mail,创建mail object
       3. 在app里init app
       4. 在auth里创建测试函数
18. 实战第五节课
    1. 设计邮箱验证码
       1. 引入string.digits, 0123456789，这样的有4排
       2. 从这4排中随机抽取4个数字, random.sample()，得到一个list
       3. 用””.join()连起来并发送
19. 实战第六节课
    1. 用数据库储存验证码，之后需要和用户输入进行比对
       1. 在数据库创建EmailCaptchaModel class，存储验证码和邮箱
       2. 在auth中创建EmailCaptchaModel的instance，将验证码存入数据库
    2. 后端
       1. 这一步是后端的操作，我们还遗漏了什么？
    3. 前端
       1. 自始至终我们都是在用自己的邮箱提取验证码，i.e. 192.168.3.6:8000/auth/captcha/email?email=xxxx@qq.com
       2. 我们需要在网页中提取用户的邮箱 —- 这一步是前端操作
20. 实战第七节课
    1. 修改前端使得验证码发送
21. 实战第八课
    1. 修改前端使得收到验证码有一个倒计时
    2. 现在有什么问题？
       1. 验证码和邮箱匹配吗？
       2. 密码和确认密码是否一致呢？
22. 实战第九课
    1. 数据验证 — flask-wtf
       1. 验证对象：email, username, password, 验证码
       2. 固定格式验证 （email格式，长度）
       3. 验证邮箱是否已经注册
       4. 验证码是否匹配
23. 实战第十节课
    1. 完成注册页面
       1. 在auth中用post请求将用户的输入全部读取包括username, email…
       2. 创建一个RegisterForm对象将数据存储: form = RegisterForm(request.form)
       3. 用form.validate()将所有数据进行验证
       4. 把form里的数据存到数据库: username = form.username.data
       5. 注册成功应将页面转到登录：redirect(url\_for(“auth.login”))
24. 实战第十一节课
    1. 渲染注册模板: 改成extends “base.html”
25. 实战第十二节课
    1. 登录功能后端实现
       1. 用request.form读取数据并创建LoginForm存储
       2. 用form.validate()进行验证
       3. 读取email, 在user表中寻找它是否存在
       4. 如果存在，检查密码是否一致： check\_password\_hash(哈希过的密码，用户输入的密码)
    2. 把userid 放在session中，使得下一次登录浏览器就认识用户，就不需要取db里核对密码了，让程序更快
26. 实战第十三节课
    1. 实现hook
       1. 在login的request发出前，我们需要提取seesion中的user id，然后在我们的数据库中找到那个user,这样就不需要在login里找了，加快进程
       2. user = User.query.get(user\_id). —> setter(g, “user”,user)，g是一个全局变量，我们把user存在里面之后在别的地方就可以用了
27. 实战第十四节课
    1. 登录和非登录状态切换 — 修改base.html
       1. 如果登录，右上角是 用户名 和 退出登录
       2. 如果未登录， 右上角是 登录 和 注册
    2. 如何更新前端和后端达到这个呢？
       1. 前端：在html中放入logout的路径
       2. 之前： <a class “nav-link” href=“#”>退出登录</a>
       3. 之后：<a class “nav-link” href=“{{ url\_for(‘auth.logout’)}}”>退出登录</a>
       4. 后端：写logout函数
       5. 只要把session清空就行 session.clear()
28. 实战第十五节课
    1. 发布问答页面渲染
29. 实战第十六节课
    1. 发布问答后端实现
       1. 创建QuestionModel,包含 title, content, author\_id, author, create\_time
       2. 在form中创建QuestionForm以便之后做验证
       3. 完成public\_qa,读取form中的title，content，author，存到数据库中
    2. 问题：因为我们的author是用g.user提取的，但是如果用户未登录，g.user=None,那么发布问题就会报错，如何解决呢？
30. 实战第十七节课
    1. 登录装饰器的实现
       1. 为什么需要？如果不用装饰器，每个程序都要写一个if g.user,这样很没有效率
       2. 装饰器就是外面多叠了一个函数
       3. 如何做？创建decorator.py，然后写if g.user…
31. 实战第十八节课
    1. 首页问答列表渲染
       1. 在qa的index()中读取所有的question，并以降序排列，使得最新的问题放在最前面，并把questions传入前端
       2. 在index.html中建立一个for loop读取所有的question, 放到对应的位置上，比如说question title会导入question.tile
32. 实战第十九课
    1. 详情页渲染（detail）
       1. 现在的问题是什么？点击问题但是无法跳转到详情页
       2. （后端）在qa中创建一个函数qa\_detail，url以question id为变量，**’/qa/detail/<qa\_id>’，它会返回详情页面**
       3. （前端）用url\_for将index.html里点击详情页设置为跳转到qa.qa\_detail，从而展示详情页
33. 实战第二十课
    1. 回答模型创建
       1. 一个回答包含：content, create\_time; author, question
       2. Author是外键，需要创建关系。也就是说需要从一个question拿到它所有的回答，这个回答需要用降序排列
          1. Question = db.relationship(QuestionModel, backref= db.backref(“answer”, order\_by =create\_time.desc()))
34. 实战第二十一课
    1. 完成发布回答功能
       1. qa中创建public\_answer(), 创建AnserForm (后端)
          1. 如果 form validate —> 返回content, question\_id, 创建AnswerModel，存入db
          2. 如果不validate，转到qa\_detail界面
       2. 评论后跳转到public\_answer(前端）
          1. 因为AnswerForm里要检查question\_id,我们把question\_id作为hidden传进去
35. 实战倒数第二课
    1. 回答列表完善
       1. 实时更新
       2. 显示评论个数
36. 实战倒数第一课
    1. 实现搜索功能