labibi Documentation

Release 1.0

C. Titus Brown

April 24, 2014

Contents

Lecture/lab: Tu/Th 3-4:20pm, 55 Union Bldg

Instructor: C. Titus Brown, ctb@msu.edu, BPS 2228(c)

TA: Leigh Sheneman, leighs@msu.edu, BPS 2228.

Office hours will be in 3353 Egr at 9pm on Wednesday evenings, unless otherwise stated.

Objectives:

In this course, you will learn how the Web works by working on an HTTP server, a backend Web app (including database and HTML generation), and a front-end JavaScript interface. As part of this we will discuss concepts in client-server and peer-to-peer architectures and how all of this technology works "under the hood" on today's Internet. We'll also discuss issues and approaches to developing software with an eye to maintainability, and learn about the practical separation of concerns in Web application stacks, from browser through server. A key part of this course will be the use of git and github. This course will be programming intensive and you should expect to either know Python or be prepared to learn it fairly well on your own.

Read more in the syllabus

Make sure you're on the mailing list.

Class resources:

CHAPTER 1

Blog post assignments

Up to two students may sign up to write a blog post about each class day, and each student may sign up for two blog posts total. Each blog post will count towards half a project at the end of the class (i.e. 2 blog posts will satisfy 10% of your grade).

To sign up for a blog post, go to https://github.com/ged-lab/msu-cse491-2013/, fork the repo, edit this document, and submit a pull request to me to merge it back in.

1.1 Content

Blog posts should discuss most of the following in ~5-7 paragraphs/400 words minimum:

- An outline of what was discussed in class;
- Relevant links that were brought up, and how/why they're relevant;
- Your thoughts (including agreements or disagreements) on any opinions expressed by the instructor or by other students.

You can also add:

- Other links on related topics that you found interesting;
- Additional thoughts that you wanted to bring;
- Relevant personal experiences.

1.2 Class days

- [leflerja] 4. 1/16
- [ ] 5. 1/21
- [ ] 6. 1/23
- [ ] 7. 1/28
- [ ] 8. 1/30
- [ ] 9. 2/4
- [ ] 10. 2/6
- [ ] 11. 2/11

- [leflerja] 12. 2/13 - Rich Enbody speaks on security
- [ ] 13. 2/18
- [ ] 14. 2/20
- [ ] 15. 2/25
- [ ] 16. 2/27
- [ ] 17. 3/11
- [ ] 18. 3/13
- [ ] 19. 3/18
- [ ] 20. 3/20
- [ ] 21. 3/25
- [ ] 23. 3/27
- [ ] 24. 4/1
- [ ] 25. 4/3
- [ ] 26. 4/8
- [ ] 27. 4/10 -> CTB NOT IN TOWN/CLASS CANCELLED?
- [ ] 28. 4/15
- [ ] 29. 4/17
- [ ] 30. 4/22

CHAPTER 2

Basic Instructions for Git and Github

Links:

- An interactive git tutorial, 'try git'
- Pro Git (the book)
- A tutorial introduction to git
- Top 10 Git tutorials for beginners

See also this video that Titus made about merging

2.1 1. Cloning a repository.

Cloning a repository (from github or anywhere else) makes a local copy of the contents of that repository.

To clone a repository, locate your HTTPS repository URL; it should look something like this:

```
https://github.com/ctb/cse491-serverz.git
```

where 'ctb' is your username instead, and 'cse491-serverz' is whatever repository you're trying to clone locally.

note: it *must* end in .git.

Then do 'git clone $URL', replacing $URL with the repository URL. This will create a directory named after the repository. You can rename this directory to whatever you want, move it around, etc; it's entirely self-contained.

You can now edit files and do whatever you want in this repo.

2.2 2. Committing changes

Do a 'git status' to see what git thinks has been changed.

'git diff' will show the differences between the last commit and the current changes.

The command:

```
git commit -am "my changes"
```

will commit all the changes to the repository. A 'git status' immediately afterwards should show no changes.

'git log' will show you a list of commits.

2.3 3. Pushing changes to github

The command:

```
git push origin master
```

will push all changes in the master branch (the default one) to the remote location called 'origin', which, by default, is wherever you cloned things from.

Here, 'master' is the branch. So if you have a branch, say, 'other', you can do:

```
git push origin other:other
```

You can use 'git remote' to add, remove, edit, and otherwise mess with your various location aliases (e.g. 'origin').

2.4 4. Creating new branches, and switching branches

To create a new branch called 'other', you can do:

```
git checkout -b other
```

This will copy your *current* branch into a *new* branch called 'other'.

You can switch to an existing branch by doing:

```
git checkout other
```

and you can see existing branches with:

```
git branch
```

2.5 5. Pushing changes to github with different branch names.

There's no reason you *have* to use the same branch names in your local repo as in your github repo. For example, if you do:

```
git push origin master:other
```

this will push the contents of your *local* master branch into the *remote* branch named 'other'.

CHAPTER 3

Using virtualenv

We're using virtualenv to manage software installs etc. I'll keep this page up to date with packages that need to be installed for HWs and in-class exercises.

Note: There's no harm in deleting and recreating your virtualenv. But you can't move them around; they contain hard-coded paths.

3.1 Creating a virtualenv

Pick a location (a directory that does not yet exist) and type:

```
python2.7 -m virtualenv $location
```

This will create a new virtual environment in $location. For example,

```
rm -fr ~/cse491.env
python2.7 -m virtualenv ~/cse491.env
```

will create a virtualenv in the directory 'cse491.env' in your home directory.

(You only need to create a virtualenv once.)

3.2 Activating your virtualenv

Every time you log in or open a new shell window, you need to activate the virtualenv so your Python knows about it. To do this in csh (the default shell), type:

```
source $location/bin/activate.csh
```

In bash, do:

```
. $location/bin/activate
```

So, for example, in csh, you would do

```
source ~/cse491.env/bin/activate.csh
```

3.3 Installing software in the virtualenv

You will need nose, requests, and coverage; you only need to install these once for each virtualenv.

```
pip install -U nose
pip install -U requests
pip install -U coverage
pip install -U jinja2
pip install -U twill
pip install http://quixote.ca/releases/Quixote-2.8.tar.gz
source ~/cse491.env/bin/activate.csh
```

Class pages and HWs, by day:

CHAPTER 4

Day 30: Thursday, April 24, 2014

0. Reading: http://www.jeffknupp.com/blog/2014/03/03/what-is-a-web-framework/
1. Quizlet
2. Presentation by CTB
3. Presentation by Grig Gheorghiu, NastyGal

Topics touched on with code/HW:

- version control
- git, github, pull requests, and merging
- code review
- open protocols and distributed systems
- automated testing
- HTTP
- WSGI
- unit testing
- Python modules
- Quixote
- Continuous Integration
- JavaScript and JQuery
- twill and HTTP testing
- AJAX
- Cookies; sqlite
- async vs threading
- transactions & synchronization

Reading:

- Xanadu and the Web
- Deep Web: hidden bits of the Web

- high quality software engineering
- mock objects
- Tor
- technical debt
- prediction markets
- Web 2.0
- Scaling Web sites
- DevOps
- SSL and certificates
- NodeJS
- Bitcoin

CHAPTER 5

Day 29: Tuesday, April 22, 2014

0. Read How the Bitcoin protocol actually works.

   Additional reading: The future of the blockchain; Bitcoin tutorial

1. Quiz.
2. Bitcoin discussion points
   - tries to work well in an imperfect world: attackers, latency
   - notice how synchronization is a hard problem?
   - asymmetry in computing: used for validation, verification
   - random algorithms used to break ties (blockchain): look up skip lists, my PyCon talk last year (video)

   Is Bitcoin a plot by the NSA?

   My theory: is Bitcoin about testing the reliability of certain cryptographic operations?
3. Contention, locking, and ACID.

   See: Atomicity, Consistency, Isolation, Durability (ACID). Note, locking is *expensive*.

   (a) read Transactions, through the 5th paragraph ("Another important property...")

   (b) read Transaction isolation, up to 13.2.1 (through "To set the transaction isolation level...")

   (Presentation)

   (c) for each transaction isolation level, list out one or more applications for which those guarantees are either probably or certainly sufficient for reliable performance (think worst case for "certainly"!)

   In table-sized groups of 4-8, please list out 1 or more applications for each isolation level, annotated as "probably" or "certainly" sufficient. (List them out here)
4. SIRS forms.

CHAPTER 6

Homework 13

Due by noon on Thursday, Apr 24th.

Note: you will be asked to do 50 total points of projects for this class.

0. Merge hw12 into your master. Please don't delete the 'hw12' branch.

   Hand in this homework on branch 'hw13' on your github account, and set up a pull request between hw13 and your master branch. Don't merge it, just set up the PR.

1. Finish implementing 50 points worth of projects total.

   Make sure to explain what you did in the ChangeLog, in detail, including instructions on how to execute things. Note that all previous project features should continue working with the new project features – so, if you implemented logins and cookies and now want to do SQL persistence, you should persist your login and cookie information in SQL as well as the images.

   As per the syllabus, you are allowed to work collaboratively but everything you hand in must have your own name on the commits. The only exception to this is if you are working with someone else's project, but the sum of the project points must be N*10 (if you work in a group with N=2 people, each person must implement 10 pts worth of projects). Note that you are also responsible for making sure the other persons' code doesn't break your code.

   If you run into technical difficulties that can't be resolved, you can ask (via e-mail) for an extension until the following Thursday. In your e-mail write in a few sentences what problem you're running into, what you've tried in terms of debugging it, and what you think the problem is. If I grant you an extension I will find someone to help you via code review/pull request on github. Extensions may not be granted, however, so make sure you're really stuck before asking for one...

6.1 After handing things in:

Do a clean clone of your repo and make sure that all the tests pass and that all your functionality works on the clean clone, on arctic.

That is, do:

```
git clone https://github.com/ctb/cse491-serverz -b hw13 test-hw13
cd test-hw13
(run stuff)
```

Using ChangeLog, please explain your project choice and implementation in sufficient detail to let someone else (me) who is _not_ psychic understand what you've done and run it. Test your code and any instructions using a clean checkout. I mean it.

CHAPTER 7

Day 28: Thursday, April 17, 2014

0. Read http://debuggable.com/posts/understanding-node-js:4bd98440-45e4-4a9a-8ef7-0f7ecbdd56cb. See also http://www.nodebeginner.org/#building-the-application-stack.
1. Quiz on reading
2. Discussion.
3. Next week's schedule: Tuesday (XSS and Selenium; DB; SIRS); Thursday (NastyGal, Grig Gheorghiu will speak via teleconf)
4. This week's homework will be: (a) finish out project points by Th; (b) fix stuff that I find in your homework, by following Th (two weeks).
5. No class or final during final's week!
6. In class exercise on git merging.
7. git merging prez

7.1 In class exercise: git merging

Brief description: check out cse491-mergez (https://github.com/ctb/cse491-mergez) and merge the fix_stddev and listcomp branches into master.

Longer description:

> This code implements the calculation of standard deviation for a discrete random variable (see the Wikipedia page), with values taken from a single column text file.
>
> Two separate improvements have been made to the codebase. Please merge them into a single branch.

Note: Be sure to take the time to look through the project and repository!

Workflow:

1. Clone the repository:

```
git clone https://github.com/ctb/cse491-mergez.git
```

2. Get all of the branches:

```
cd cse491-mergez
git fetch origin
```

3. Make sure you're on master:

```
git checkout master
```

4. Get a list of branches, including remote ones:

   ```
   git branch -r
   ```

5. Make a new merge branch to do the merge on:

   ```
   git checkout -b try_merge
   ```

6. Merge in one of the two feature branches:

   ```
   git merge origin/listcomp
   ```

   and resolve merge conflicts, if any (see below).

7. Merge in the other feature branch:

   ```
   git merge origin/fix_stddev
   ```

   and resolve merge conflicts, if any (see below).

8. Go back to master.

   ```
   git checkout master
   ```

9. Merge the 'merge' branch in:

   ```
   git merge try_merge
   ```

10. (If you have time) Fork https://github.com/ctb/cse491-mergez, push your master branch to your own copy, and set up a pull request from your master to my master.
11. Bask in the warm glow of success.

7.2 Resolving merge conflicts

When git merge encounters syntactically unmergeable code – where one change directly conflicts with the other branch – it will annotate the code with both versions:

```
<<<< HEAD

 (one version of conflicting code)

====

 (other version of conflicting code)

>>>> other
```

To clear this conflict, you need to (a) edit the code so it is syntactically and semantically correct, which means removing the <<<< etc annotations; and (b) do a 'git add filename' for each file you fixed, plus a 'git commit' to commit the changes. Then a 'git status' should report no conflicted files.

CHAPTER 8

Day 27: Tuesday, April 15, 2014

0. Read http://en.wikipedia.org/wiki/Asynchronous_I/O

On branch day27, I've changed image.py (link) to do SQLite saving/loading of images.

Look at the changes to server.py on day27-threading (link) and day27-async (link).

Question – does the 'add_image' function (lines 35-54) in image.py need to be protected from concurrent execution in (a) threading and (b) async?

If (a) threading is YES and (b) async is YES, choose pink.

If (a) threading is YES and (b) async is NO, choose green.

If (a) threading is NO and (b) async is YES, choose yellow.

If (a) threading is NO and (b) async is NO, choose blue.

CHAPTER 9

Homework 12

Due by noon on Thursday, Apr 17th.

Note: you will be asked to do 50 total points of projects for this class.

0. Merge hw11 into your master. Please don't delete the 'hw11' branch.

   Hand in this homework on branch 'hw12' on your github account, and set up a pull request between hw12 and your master branch. Don't merge it, just set up the PR.

1. Pick 10 points worth of projects from `projects` and implement.

   Make sure to explain what you did in the ChangeLog, in detail, including instructions on how to execute things. Note that all previous project features should continue working with the new project features – so, if you implemented logins and cookies and now want to do SQL persistence, you should persist your login and cookie information in SQL as well as the images.

   As per the syllabus, you are allowed to work collaboratively but everything you hand in must have your own name on the commits. The only exception to this is if you are working with someone else's project, but the sum of the project points must be N*10 (if you work in a group with N=2 people, each person must implement 10 pts worth of projects). Note that you are also responsible for making sure the other persons' code doesn't break your code.

   If you run into technical difficulties that can't be resolved, you can ask (via e-mail) for an extension until the following Thursday. In your e-mail write in a few sentences what problem you're running into, what you've tried in terms of debugging it, and what you think the problem is. If I grant you an extension I will find someone to help you via code review/pull request on github. Extensions may not be granted, however, so make sure you're really stuck before asking for one...

9.1 After handing things in:

Do a clean clone of your repo and make sure that all the tests pass and that all your functionality works on the clean clone, on arctic.

That is, do:

```
git clone https://github.com/ctb/cse491-serverz -b hw12 test-hw12
cd test-hw12
(run stuff)
```

Using ChangeLog, please explain your project choice and implementation in sufficient detail to let someone else (me) who is _not_ psychic understand what you've done and run it. Test your code and any instructions using a clean checkout. I mean it.

CHAPTER 10

Day 25: Tuesday, April 8, 2014

0. Read https://httpd.apache.org/docs/2.2/ssl/ssl_intro.html and http://blog.hartleybrody.com/https-certificates/
1. Fill out the quiz
2. Discussion.
3. Git exercises from *Day 24: Thursday, April 3, 2014*.
4. Other stuff

   SQL loading & Python modules.

   (a) See setup() in imageapp/__init__.py (link); this contains stuff that you only run once an app. It's in a function so you can choose to run it or not to run it (e.g. for tests)

   (b) See imageapp/image.py (link). The 'images = {}' will be run on import, no matter what you do. There's no way to disable it, in particular.

   tl;dr? Almost always put code in a function.

   Q: where should you put database creation/data model definition code? (See sqlite/create.py (link on day23 branch)

   More git reading: http://who-t.blogspot.com/2014/03/using-git-next-level.html

   A quest for understanding Web stuff: http://jakearchibald.github.io/request-quest/

CHAPTER 11

Homework 11

Due by noon on Thursday, Apr 10th.

Note: you will be asked to do 50 total points of projects for this class.

0. Merge hw10 into your master. Please don't delete the 'hw10' branch.

   Hand in this homework on branch 'hw11' on your github account, and set up a pull request between hw11 and your master branch. Don't merge it, just set up the PR.

1. Make the cookie app from *Day 23: Tuesday, April 1, 2014* work in your server.py, when run with '-A cookie'. Note in particular that this means you need proper HTTP_COOKIE handling in your WSGI server...

   In particular, try out the twill test on the day23 branch of cse491-serverz works; see the test script here. You can run it like so:

   ```
   twill-sh -u http://hostname:port/ twill-tests/cookieapp.twill
   ```

2. Change imageapp to store its images in a SQL database. Feel free to swipe the sqlite3 code from *Day 23: Tuesday, April 1, 2014*. Make sure to separate database creation code from image loading code, and execute the database creation code only once (or, when needed). Put in instructions on how to do this if it's not automatically figured out...

   Two notes:

   - This is just about the images, nothing else;
   - If you've already done any of the data persistence projects, you're good; just tell me that in the ChangeLog for hw11.

3. Pick 10 points worth of projects from `projects` and implement.

   Make sure to explain what you did in the ChangeLog, in detail, including instructions on how to execute things. Note that all previous project features should continue working with the new project features – so, if you implemented logins and cookies and now want to do SQL persistence, you should persist your login and cookie information in SQL as well as the images.

   As per the syllabus, you are allowed to work collaboratively but everything you hand in must have your own name on the commits. The only exception to this is if you are working with someone else's project, but the sum of the project points must be N*10 (if you work in a group with N=2 people, each person must implement 10 pts worth of projects). Note that you are also responsible for making sure the other persons' code doesn't break your code.

   If you run into technical difficulties that can't be resolved, you can ask (via e-mail) for an extension until the following Thursday. In your e-mail write in a few sentences what problem you're running into, what you've tried in terms of debugging it, and what you think the problem is. If I grant you an extension I will find someone to

help you via code review/pull request on github. Extensions may not be granted, however, so make sure you're really stuck before asking for one...

11.1 After handing things in:

Do a clean clone of your repo and make sure that all the tests pass and that all your functionality works on the clean clone, on arctic.

That is, do:

```
git clone https://github.com/ctb/cse491-serverz -b hw11 test-hw11
cd test-hw11
(check stuff over)
```

Using ChangeLog, please explain your project choice and implementation in sufficient detail to let someone else (me) who is _not_ psychic understand what you've done and run it. Test your code and any instructions using a clean checkout. I mean it.

CHAPTER 12

Day 24: Thursday, April 3, 2014

0. Read http://www.aosabook.org/en/integration.html
1. Fill out the quiz
2. Discussion.
3. In-class exercises (see below).

12.1 In-class exercises

1. In pairs, look at http://jobs.msu.edu. Try searching for two different job postings (4698 and 7617, for example). Can you communicate these to your partner via an e-mailed URL? Try sending each other the URL of the different job.

   What is the mechanism the site is using to keep track of what job you're looking at? What's a better way?

2. Check out branch 'day24-ice' of ctb's cse491-serverz, provided by a student:

   ```
   git clone https://github.com/ctb/cse491-serverz.git day24-ice -b day24-ice
   ```

   and run the imageapp, 'cd day24-ice; python server.py -A image'.

   In pairs, have one person log in to the same server as 'user1'/'user1' and the other as 'user2'/'user2'. Now check to see who you are logged in as.

   What's the mechanism the site is using to keep track of who you're logged in as? What's a better way to do this?

   Note: if you're all alone, you can do this with two different incognito windows opening onto the same server.

3. Check out 'day24-ice' as in #2.

   In imageapp/templates/base.html, check out the two different ways of including style sheets in your HTML – play around with uncommenting, and looking at the effect on the output. Do both work? Which is the "right" way to do this, and why?

4. Check out cse491-textz:

   ```
   git clone https://github.com/ctb/cse491-textz.git
   ```

   and, using 'git diff', figure out at which commit we lost the white rabbit.

   Specifically, you can use 'git log' to see the commit history; 'git checkout <commit prefix>' to check out specific versions of the repo; 'git diff <commit prefix1>..<commit prefix2>' to diff between two repos. 'git checkout master' will get the tip of the master branch back.

Note that the white rabbit is present in the initial git commit, '58f7df':

```
cd cse491-textz
git checkout 58f7df
grep rabbit cities.txt
```

but absent in the tip:

```
git checkout master
grep rabbit cities.txt
```

5. Check out cse491-textz as in #4; use 'git blame' to figure out at what commit the name Defargo was introduced into the text.

CHAPTER 13

Day 23: Tuesday, April 1, 2014

0. Skim DevOps for recruiters and read this article and this article on ChaosMonkey.
1. Fill out the quiz
2. Discussion.
3. Introducing cookies, and "secure" cookies.

   http://en.wikipedia.org/wiki/HTTP_cookie

   See also 'cookieapp' in cse491-server.

   Check out the day23 branch (see bottom of page), then go into day23/ and run:

   ```
   python ref-server.py -A cookie
   ```

4. Introducing SQLite and SQL.

   https://docs.python.org/2/library/sqlite3.html

   http://sebastianraschka.com/Articles/sqlite3_database.html

   http://software-carpentry.org/v4/databases/

   See the sqlite/ subdirectory in cse491-server for some examples. To run the examples, check out the day23 branch (as below), and then go into day23/sqlite/, and run:

   ```
   python create.py
   python insert.py
   python retrieve.py out.png
   ```

   (Then verify that 'out.png' is a valid PNG file :)

Reminder, to look at the day23 serverz repo, put a copy in the directory 'day23' by doing::

```
git clone https://github.com/ctb/cse491-serverz.git day23 -b day23
```

CHAPTER 14

Homework 10

Due by noon on Thursday, Apr 3rd.

Note: you will be asked to do 50 total points of projects for this class.

0. Merge hw9 into your master. Please don't delete the 'hw9' branch.

   Hand in this homework on branch 'hw10' on your github account, and set up a pull request between hw10 and your master branch. Don't merge it, just set up the PR.

1. Pick 10 points worth of projects from `projects` and implement.

   Make sure to explain what you did in the ChangeLog, in detail.

   As per the syllabus, you are allowed to work collaboratively but everything you hand in must have your own name on the commits. The only exception to this is if you are working with someone else's project, but the sum of the project points must be N*5 (if you work in a group with N=2 people, each person must implement 5 pts worth of projects). Note that you are also responsible for making sure the other persons' code doesn't break your code.

   If you run into technical difficulties that can't be resolved, you can ask (via e-mail) for an extension until the following Thursday. In your e-mail write in a few sentences what problem you're running into, what you've tried in terms of debugging it, and what you think the problem is. If I grant you an extension I will find someone to help you via code review/pull request on github. Extensions may not be granted, however, so make sure you're really stuck before asking for one...

14.1 After handing things in:

Do a clean checkout of your repo and make sure that all the tests pass and that all your functionality works on the clean checkout, on arctic.

Using ChangeLog, please explain your project choice and implementation in sufficient detail to let someone else (me) who is _not_ psychic understand what you've done and run it. Test your code and any instructions using a clean checkout. I mean it.

CHAPTER 15

Day 22: Thursday, Mar 27th, 2014

0. Read What's the Open Web?.
1. Fill out quiz
2. Discussion.
3. Discuss: `projects`
3. Code review offer.
4. Future classes; strategy.

   Tuesday classes will start with reading and quizes; then move on to technology vignettes; then I will talk with people about their projects and project benchmarks & goals.

CHAPTER 16

Day 21: Tuesday, Mar 25th, 2014

0. Read Scaling Twitter
1. Fill out quiz.
2. Discussion.
3. Read through the image saving code in the day 21 branch and understand what it does. Talk amongst yourselves and discuss any problems this code might have for high traffic Web sites. Fill out this form when you're done.

   Note: You can also look at the difference between the day20 and day21 branches.

   Reminder, to look at the day21 serverz repo, put a copy in the directory 'day21' by doing::

   ```
   git clone https://github.com/ctb/cse491-serverz.git day21 -b day21
   ```

   and then to run it:

   ```
   cd day21/
   python2.7 run-imageapp.py
   ```

CHAPTER 17

Homework 9

Due by noon on Thursday, Mar 27th.

0. Merge hw8 into your master. Please don't delete the 'hw8' branch.

   Hand in this homework on branch 'hw9' on your github account, and set up a pull request between hw9 and your master branch. Don't merge it, just set up the PR.

1. Integrate the quotes app into server.py, so that it can be run with '-A quotes'. Instead of '/quotes-2.html' have the default ('/') be the page that displays the quotes.

2. Integrate the chats app into server.py, so that it can be run with '-A chat'. Amend the chats app so that the time of each message is displayed in the chat window, too.

3. Pick 5 points worth of projects from `projects` and implement.

   Make sure to explain what you did in the ChangeLog, in detail.

   As per the syllabus, you are allowed to work collaboratively but everything you hand in must have your own name on the commits. The only exception to this is if you are working with someone else's project, but the sum of the project points must be N*5 (if you work in a group with N=2 people, each person must implement 5 pts worth of projects). Note that you are also responsible for making sure the other persons' code doesn't break your code.

17.1 After handing things in:

Do a clean checkout of your repo and make sure that all the tests pass and that all your functionality works on the clean checkout, on arctic.

Using ChangeLog, please explain your project choice and implementation in sufficient detail to let someone else (me) who is _not_ psychic understand what you've done and run it. Test your code and any instructions using a clean checkout. I mean it.

CHAPTER 18

Day 20: Thursday, Mar 20th, 2014

0. Read What is Web 2.0? and, if you feel like it, the Wikipedia page on Web 2.0.
1. Quiz. (May need to log out of your regular google.com account & log into your msu.edu account.)
2. Discussion
3. "AJAX"

   Why AJAX vs simply loading the page?

   Note: no content push options in HTTP!

   Various reasons for using AJAX:

   - Partial load of data.
   - More interactivity.
   - Less page rendering all at once on the client.
4. Quote app query diagram. (See *Day 19: Tuesday, Mar 18th, 2014*.)
5. Write a query diagram for the chat app.

   To get the chats app, do a clean checkout of my serverz repo into the directory 'day20' by doing:

   ```
   git clone https://github.com/ctb/cse491-serverz.git day20 -b day20
   ```

   and then:

   ```
   cd day20/chat/
   python2.7 chat-server <port number>
   ```

   You probably won't need to be in an activate virtualenv to do this.

   Finally, go to:

   ```
   arctic.cse.msu.edu:<port number>/
   ```

   and enter a few messages.

   Draw two query diagrams, the first for what happens when you submit a message and see it in your own browser chat window, and the second for what happens when someone else submits a message and you see it in your chat window.

   Then, write your NetID on the piece of paper and put 'em in a stack on Table 4.

Part of your HW will be to implement the chat and quotes apps in your own server.py, i.e. as WSGI apps. Feel free to get started with that AFTER you do the query diagrams.

CHAPTER 19

Day 19: Tuesday, Mar 18th, 2014

0. Read A conspiracy of hogs and skim the Wikipedia entry on Prediction market.

1. Quizlet. You may need to sign into your MSU account on Google first.
2. Discussion.
3. More query diagrams: redirects, and JavaScript.
4. Quick discussion of twill.

   (Note: is anyone interested in rewriting this using 'requests'?)
5. twill exercise

   ...AND/OR...
6. write a query diagram for the quotes app.

   To get the quotes app, do a clean checkout of my serverz repo into the directory 'day19' by doing:

   ```
   git clone https://github.com/ctb/cse491-serverz.git day19 -b day19
   ```

   and then:

   ```
   cd day19/quotes/
   python2.7 quotes-server <port number>
   ```

   You probably won't need to be in an activate virtualenv to do this.

   Finally, go to:

   ```
   arctic.cse.msu.edu:<port number>/quotes-2.html
   ```

   Draw a query diagram explaining the HTTP serving that's going on here, and then write your NetID on the piece of paper and put 'em in a stack on Table 6.

CHAPTER 20

Homework 8

Due by noon on Thursday, Mar 20th.

0. Merge hw7 into your master. Please don't delete the 'hw7' branch.

   Hand in this homework on branch 'hw8' on your github account, and set up a pull request between hw8 and your master branch. Don't merge it, just set up the PR.

1. Implement command line options in server.py to run the following WSGI apps:

   imageapp

   quixote.demo.altdemo

   your app from hw6.

   Specifically, 'server.py -A image -p 8000' should run imageapp on port 8000. '-A altdemo' and '-A myapp' should do the obvious. If '-p' is not specified then the port should be chosen randomly.

   Use argparse to accomplish this.

2. Run the twill tests for the image and altdemo apps. The 'imageapp' and 'altdemo' tests (below) should run without modification.

   (Twill provides automated testing for Web apps; it's basically a command-line browser.)

   You can get the twill tests here:

   https://github.com/ctb/cse491-serverz/tree/hw8-twill-tests

   To run the twill tests for the imageapp, change into the directory containing your server.py, and run:

   ```
   twill-sh -u http://localhost:8000/ twill-tests/imageapp-1.twill
   ```

   (You will need to have server.py running in another terminal window already.)

   You should see output like this:

   ```
   >> EXECUTING FILE twill-tests/imageapp-1.twill
   ==> at http://localhost:8000
   AT LINE: twill-tests/imageapp-1.twill:0
   ==> at http://localhost:8000/upload
   AT LINE: twill-tests/imageapp-1.twill:1

   Added file "imageapp/dice.png" to file upload field "file"

   AT LINE: twill-tests/imageapp-1.twill:2
   Note: submit is using submit button: name="None", value=""
   ```

```
AT LINE: twill-tests/imageapp-1.twill:3
AT LINE: twill-tests/imageapp-1.twill:5
==> at http://localhost:8000/image
AT LINE: twill-tests/imageapp-1.twill:6
AT LINE: twill-tests/imageapp-1.twill:7
--
1 of 1 files SUCCEEDED.
```

The 'run-qx' and 'run-imageapp' scripts should already work; your challenge is to make sure that your server.py implementation (which should use YOUR WSGI server) works.

Twill documentation is here. To install it in your virtualenv, do 'pip install -U twill'; see *Using virtualenv* for full instructions.

3. Write a twill test for your app that executes all of the URLs and checks the return code (200); put the test in twill-tests/ and name it myapp-1.twill.
4. Pick 5 points worth of projects from `projects` and implement.

   Make sure to explain what you did in the ChangeLog, in detail.

   As per the syllabus, you are allowed to work collaboratively but everything you hand in must have your own name on the commits. The only exception to this is if you are working with someone else's hw8 project, but the sum of the project points must be N*5 (if you work in a group with N=2 people, each person must implement 5 pts worth of projects). Note that you are also responsible for making sure the other persons' code doesn't break your code.

20.1 After handing things in:

Do a clean checkout of your repo and make sure that all the tests pass and that all your functionality works on the clean checkout, on arctic.

Using ChangeLog, please explain your project choice and implementation in sufficient detail to let someone else (me) who is _not_ psychic understand what you've done and run it. Test your code and any instructions using a clean checkout. I mean it.

CHAPTER 21

Day 18: Thursday, Mar 13th, 2014

0. Read the wikipedia entry on Technical Debt.

   Ancillary readings:

   http://martinfowler.com/bliki/TechnicalDebt.html

   http://blog.codinghorror.com/paying-down-your-technical-debt/

   http://www.construx.com/10x_Software_Development/Technical_Debt/

1. Quizlet. Please fill this out even if you haven't done the reading.
2. Discussion of technical debt.
3. Query diagrams intro..
4. Make query diagrams for three queries on the 'run-imageapp.py' server in my repo, under branch day18; specifically, '/image', '/upload', and '/upload2'. Include the process of actually uploading a file via /upload2. (note the 'imageapp' directory contains two png files that you can upload).

   Note: you can do a clean checkout into the directory 'day18' by doing:

   ```
   git clone https://github.com/ctb/cse491-serverz.git day18 -b day18
   ```

   and run it with 'run-imageapp.py'.

   Also note: these are example solutions for the JavaScript stuff in *Day 16: Thursday, Feb 27th, 2014*.

   In class, draw three diagrams on a piece of paper or with a graphics program, put your NetID on the top of one page, and hand them in via e-mail or (*gasp*) by hand. Feel free to work in groups of 2-3; just include all y'all's NetID.

   Last note - you can track requests by looking at your running imageapp server output. (Ignore the favicon.ico request.)

5. Brief going over of *Homework 8*.

CHAPTER 22

Day 17: Tuesday, Mar 11th, 2014

0. Read The Inside Story of Tor.
1. Quizlet
2. Presentation: How the Internet works.
3. Review quizlet answers & discuss.

CHAPTER 23

Day 16: Thursday, Feb 27th, 2014

0. Lecture notes
1. Walk through solution for *Homework 7*, problem 3. (How are images served?)
2. Form submission and redirects:

   See https://github.com/ctb/cse491-serverz/blob/day16/imageapp/root.py, functions 'upload' and 'upload2'.
3. Read through the examples in `javascript` and integrate one JavaScript and one JQuery example each into one or more of your pages from HW 7. (Don't worry too much about which page, and no, they don't have to be on any specific page.) You should be editing your templates files to do this.

Note: you can either pull these files into your hw7 branch by doing

```
git pull https://github.com/ctb/cse491-serverz.git day16
```

or by doing a clean checkout in some other directory and then copying over the ones you care about:

```
git clone https://github.com/ctb/cse491-serverz.git day16 -b day16
```

or you can just download and save the example files through github. Whatever suits.

Also note: if you're having trouble loading the jquery-1.3.2.min.js file through your server, you can use this code snippet in your RootDirectory class (in imageapp/root.py):

```
# make sure to import:
#    from quixote.util import StaticFile
#    import os.path

# 'filename' must be in the _q_exports list
_q_exports = [ ..., 'filename', ...]

filename = StaticFile(os.path.abspath("./filename.txt"))
```

Here, 'filename.txt' will be served from whatever directory you're running your WSGI server from.

CHAPTER 24

Day 15: Tuesday, Feb 25th, 2014

1. Presentation: https://docs.google.com/presentation/d/13v9kfPxIJl2er3WYvnxWR_pvyrLcSsUaGHTnRHCQvHY/edit?usp=shari
2. A brief intro to Continuous Integration
3. Travis and Jenkins

24.1 Continuous Integration

See the presentation.

CHAPTER 25

Homework 7

Due by noon on Thursday, Feb 27th.

0. Merge hw6 into your master. Please don't delete the 'hw6' branch.

   Hand in this homework on branch 'hw7' on your github account, and set up a pull request between hw7 and your master branch. Don't merge it, just set up the PR.

1. Add cookie header handling into your WSGI server, and test it by making sure that the login link in `quixote.demo.altdemo` works.

   See https://github.com/ctb/cse491-serverz/blob/hw5-wsgi/run-qx.py for demo code that shows how to run altdemo; this code uses the WSGI reference server, but your HW 7.1 should use your own WSGI server.

2. Merge in or copy 'run-imageapp' and the 'imageapp/' directory hierarchy from my hw7-imageapp branch (see https://github.com/ctb/cse491-serverz/tree/hw7-imageapp). As with HW 7.1, the 'run-imageapp' code uses the WSGI reference server, and you should make sure that 'imageapp' runs in your own WSGI server.

3. Modify the imageapp to display the latest image on the front (index) page, not just the '/image' page.

4. Do a clean checkout of your repo and make sure that all the tests pass and that all your hw7 functionality works on the clean checkout.

5. Have a good break! There will be no HW over the break.

CHAPTER 26

Homework 6

Due by noon on Thursday, Feb 20th. If you need an extension, ask.

0. Merge hw5 into your master. Please don't delete the 'hw5' branch :)

   Hand in this homework on branch 'hw6' on your github account, and set up a pull request between hw6 and your master branch. Don't merge it, just set up the PR.

1. Write a function that serves a file, together with the appropriate content-type. You can stick with image/jpeg and text/plain for now, e.g. a .jpg file would have content-type image/jpeg and a .txt file would be text/plain.

   Make /image serve an image (JPG), and /file serve a text file of some sort.

   Hint: 'fp = open(filename, "rb"); data = fp.read(); fp.close()'

2. Make sure you WSGI server works with all three of the Quixote demo apps, as in *Day 13: Tuesday, Feb 18th, 2014*. Note that the 'login' functionality in altdemo will not work yet; that's OK.

3. Use the wsgiref validator to evaluate your WSGI app. Apart from cookies, is anything else missing or broken? Fix the obvious things.

4. Do a clean checkout of your repo and make sure that all the tests pass and that all your hw6 functionality works on the clean checkout.

   ("Clean checkout" means make a fresh clone of the repository somewhere else, so that you can be sure you've checked everything into the repo.)

CHAPTER 27

Day 13: Tuesday, Feb 18th, 2014

1. Fill out: quizlet on testing
2. Homework will be posted tonight, due Thursday. If people want an extension, let me know.
2. A brief intro to Python modules.
3. An actual WSGI application.
4. Individual homework questions?

27.1 An actual WSGI application

Activate your virtualenv, or create/recreate it (see *Using virtualenv*).

then, install Quixote into your virtualenv:

```
pip install http://quixote.ca/releases/Quixote-2.8.tar.gz
```

and, finally, try running one of the Quixote demo applications in your own WSGI server; see https://github.com/ctb/cse491-serverz/blob/hw5-wsgi/run-qx.py for code to create the WSGI app.

As an example, I've adapted eunbong's server code to run the Quixote app. See the commit and diff for details on what I did – it didn't take much.

CHAPTER 28

Homework 5

Due by noon on Thursday, Feb 13th.

0. Merge hw4 into your master. Please don't delete the 'hw4' branch :)

   Hand in this homework on branch 'hw5' on your github account, and set up a pull request between hw4 and your master branch. Don't merge it, just set up the PR.

1. Move all of your content-creation code (anything in server.py after reading in the request) into a new file, 'app.py', and refactor it to look like a WSGI application.

   More specifically, follow the WSGI app specification.

   Your app should work in the ref-server.py available here – you should be able to merge this into your repo by doing:

   ```
   git pull https://github.com/ctb/cse491-serverz.git hw5-wsgi
   ```

   This will give you a basic 'app' file and a demonstration of how to run it in ref-server.py.

   Once you've refactored your app.py code, you should get the same Web page results as from hw4, but through the WSGI server running in ref-server.py instead of with your own socket handling code.

2. Refactor the handle_connection function to run WSGI apps; see the WSGI server info for an example, although you'll need to ignore some of the CGI-specific details...

   Basically, what you need to do is separate out the actual HTTP request parsing from the code that generates a response (which, in any case, is now a WSGI app in app.py, right?). A few tips:

   - 'environ' is a dictionary that you create from the HTTP request; see environ variables.
   - you should fill in 'REQUEST_METHOD', 'PATH_INFO' (leave SCRIPT_NAME blank), 'QUERY_STRING', 'CONTENT_TYPE', 'CONTENT_LENGTH', and 'wsgi.input' in the environ dictionary;
   - wsgi.input is that StringIO object you used in hw4 – this contains the POST data, if any; empty otherwise.
   - 'start_response' should probably be defined *within* your handle connection_function (although there are other ways to do it). It is responsible for storing the status code and headers returned by the app, until the time comes to create the HTTP response.

   Your logic could look something like this:

   (a) read in entire request

   (b) parse request headers, build environ dictionary

   (c) define start_response

(d) call WSGI app with start_response and environ

(e) build HTTP response from results of start_response and return value of WSGI app.

The 'simple_app' in app.py on my hw5-wsgi branch (see app.py) is potentially a useful debugging tool; your server should work with it, as well as with the app in your app.py

3. Template inheritance and proper HTML.

   Create a file base.html and use it as a "base template" to return proper-ish HTML, as in the jinja2 example from last year.

   (Each of your HTML files should inherit from this base.html and fill in only their specific content.)

   Please *do* specify an HTML title (<title> tag) for each page.

4. Your tests still pass, right?

CHAPTER 29

Day 11: Tuesday, Feb 11th, 2014

0. Read http://www.artima.com/scalazine/articles/twitter_on_scala.html
1. Quiz (link) and brief discussion.
2. 'git stash' and 'git stash apply' demonstration
3. Code review of a specific chunk o' code.

29.1 Pull requests and doing code reviews

(In-class exercise.)

In brief, everyone should fetch this branch into their local repo,

> https://github.com/ctb/cse491-serverz/tree/day11

make at least one change, push the changed branch to their github repo, and then set up a pull request back to me.

Instructions:

On arctic, in your existing repo OR in a newly cloned copy of YOUR github repo (see *Basic Instructions for Git and Github*):

```
git fetch https://github.com/ctb/cse491-serverz.git day11:day11
```

If you have a bunch of changes in your repo for hw5, please commit them or stash them ('git stash' – you can retrieve with 'git stash apply'). Then check out the day 11 branch:

```
git checkout day11
```

Now, edit ChangeLog and add in an entry saying "in-class exercise."

Then, push this branch back to your own github account:

```
git commit -am "day 11 exercise"
git push -u origin day11:day11
```

Go to to your own github account and set up a pull request between YOUR repository/day11 and MY repository/master branch (ctb). It should look like this.

Now, do a code review. You can comment line by line (go to the files view of the pull request) AND/OR make changes to your day11 branch and push them to your repo.

```
git push origin day11
```

General code review:

0. Does it run?
1. Do tests pass?
2. Spaces rather than tabs.
3. Spaces after #.
4. Properly spelled variable names .
5. Try writing tests to break something. For example, do you believe their POST logic?
6. 80 character line lengths.
7. Test with multiple browsers.
8. Use code coverage to find things that their tests don't test, and see if you can break their code. (See *Day 8: Thursday, January 30th, 2014*)

And/or do a specific code review for *Homework 4*.

Reminder about features:

1. handle multipart/form-data.
2. arbitrary size requests.
3. implement templates.
4. implement "404 not found"

CHAPTER 30

Day 10: Thursday, Feb 6th, 2014

0. Read Testing in Python: using nose & mocks.
1. Fill out the quiz and do some discussion.
2. Why does cgi.py suck so much?
3. More code analysis?
4. Teach me mock tests?
5. Code review of hw4.

30.1 Code review checklist

Basic rules:

0. Does it run.
1. Do tests pass.
2. Spaces rather than tabs.
3. Spaces after #.
4. Properly spelled variable names .
5. Try writing tests to break something. For example, do you believe their POST logic?
6. 80 character line lengths.
7. Test with multiple browsers.
8. Use code coverage to find things that their tests don't test, and see if you can break their code.

30.1.1 List of repositories

massiek: github site - pulls - branches - repo URL for cloning

eunbong: github site - pulls - branches - repo URL for cloning

matheusldaraujo: github site - pulls - branches - repo URL for cloning

ConnorAvery: github site - pulls - branches - repo URL for cloning

jprickles: github site - pulls - branches - repo URL for cloning

jkteuber: github site - pulls - branches - repo URL for cloning

beckhamer: github site - pulls - branches - repo URL for cloning

john3209: github site - pulls - branches - repo URL for cloning

mill1256: github site - pulls - branches - repo URL for cloning

YourBestFriend: github site - pulls - branches - repo URL for cloning

hoffm386: github site - pulls - branches - repo URL for cloning

joshshadik: github site - pulls - branches - repo URL for cloning

msweet18: github site - pulls - branches - repo URL for cloning

juru13: github site - pulls - branches - repo URL for cloning

mcdonaldca: github site - pulls - branches - repo URL for cloning

filajust: github site - pulls - branches - repo URL for cloning

leflerja: github site - pulls - branches - repo URL for cloning

FireSBurnsmuP: github site - pulls - branches - repo URL for cloning

koppmana: github site - pulls - branches - repo URL for cloning

Karmeow: github site - pulls - branches - repo URL for cloning

curljosh: github site - pulls - branches - repo URL for cloning

yispencer: github site - pulls - branches - repo URL for cloning

glisto18: github site - pulls - branches - repo URL for cloning

mannin92: github site - pulls - branches - repo URL for cloning

westjour: github site - pulls - branches - repo URL for cloning

jbull477: github site - pulls - branches - repo URL for cloning

fakestuff: github site - pulls - branches - repo URL for cloning

msu-web-dev: github site - pulls - branches - repo URL for cloning

MaxwellGBrown: github site - pulls - branches - repo URL for cloning

xavierdhjr: github site - pulls - branches - repo URL for cloning

ettemaet: github site - pulls - branches - repo URL for cloning

lieblic2: github site - pulls - branches - repo URL for cloning

bjurgess1: github site - pulls - branches - repo URL for cloning

suhkang: github site - pulls - branches - repo URL for cloning

jonest31: github site - pulls - branches - repo URL for cloning

tsloncz: github site - pulls - branches - repo URL for cloning

zhopping: github site - pulls - branches - repo URL for cloning

MattyAyOh: github site - pulls - branches - repo URL for cloning

o2themar: github site - pulls - branches - repo URL for cloning

phammin1: github site - pulls - branches - repo URL for cloning

Badsauce: github site - pulls - branches - repo URL for cloning

DuncanCYoung: github site - pulls - branches - repo URL for cloning

cameronkeif: github site - pulls - branches - repo URL for cloning

majeedus: github site - pulls - branches - repo URL for cloning

polavar3: github site - pulls - branches - repo URL for cloning

brtaylor92: github site - pulls - branches - repo URL for cloning

labrenzm: github site - pulls - branches - repo URL for cloning

QSSS: github site - pulls - branches - repo URL for cloning

sarteleb: github site - pulls - branches - repo URL for cloning

JRucinski: github site - pulls - branches - repo URL for cloning

fenderic: github site - pulls - branches - repo URL for cloning

CHAPTER 31

Day 9: Tuesday, Feb 4th, 2014

0. Read the following two links: what is WSGI, Wikipedia on WSGI; skim a stack overflow question and the WSGI PEP
1. Quiz: answer some questions
2. What is StackOverflow anyway? Hey, and what's a PEP?
3. WSGI in practice; separation of concerns. (Presentation.)
4. Thinkin' 'bout functions.

31.1 Fun with functions and callables

try1: what does the following code print, if placed in a file and executed?

```
def f(n):
    return 3 * n

def g():
    return f

value = g()
print value(5)
```

try2: what does the following code print, if placed in a file and executed?

```
def g(n):
    def f():
        return 8*n

    return f

value = g(5)
print value()
```

try3: what does the following code print, if placed in a file and executed?

```
class Klassy(object):
    def __init__(self, n):
        self.n = n

    def val(self):
        return 4*self.n
```

```
k = Klassy(5)
print k.val()
```

try4: what does the following code print, if placed in a file and executed?

```
class Klassy(object):
    def __init__(self, n):
        self.n = n

    def val(self):
        def g(m):
            return 3 * self.n + m

        return g

k = Klassy(4)

fn = k.val()
print fn(3)
```

try5: what does the following code print, if placed in a file and executed?

```
class Klassy(object):
    def __init__(self, n):
        self.n = n

    def val(self):
        def g(m):
            return 3 * self.n + m

        return g

k = Klassy(4)

fn = k.val()

k.n = 8
print fn(4)
```

try6: what does the following code print, if placed in a file and executed?

```
class Klassy(object):
    def __init__(self, n):
        self.n = n

    def __call__(self):
        return 5*self.n

k = Klassy(5)
print k()
```

try7: what does the following code print, if placed in a file and executed?

```
def some_function(other_fn, value):
    value = value*5

    value2 = other_fn(value)

    return value2
```

```
def f(n):
    return n + 1

def g(m):
    return m - 1

print some_function(g, 5)
print some_function(f, 4)
```

try8: what does the following code print, if placed in a file and executed?

```
global_value = 6

def some_function(other_fn, value):
    value = value*global_value

    value2 = other_fn(value)

    return value2

def f(n):
    return n + 1

def g(m):
    return m - 1

print some_function(g, 5)
print some_function(f, 4)

global_value = 2

print some_function(g, 5)
print some_function(f, 4)
```

CHAPTER 32

Homework 4

Due by noon on Thursday, Feb 6th.

0. Merge hw3 into your master. Please don't delete the 'hw3' branch :)

   Hand in this homework on branch 'hw4' on your github account, and set up a pull request between hw4 and your master branch. Don't merge it, just set up the PR.

1. Modify your POST code handling code to properly handle requests that have a Content-Type of multipart/form-data.

   (a) Modify send-post-request to send multipart/form-data; see this StackOverflow post.

   (b) Use the 'FieldStorage' class in the built-in 'cgi' module to parse the POST form data; see these docs. Basically, you will want to encapsulate all of the POST request content into a StringIO.StringIO() object and pass that in as 'fp', and put all of the POST request headers into a dictionary, and pass that in as 'headers'.

   (c) Retool your form response code to work with both application/x-www-form-urlencoded and multipart/form-data submission types; this will require writing tests that specify both! Test this manually by first setting the form 'enctype' and verifying that your code still does what you expect it to; then write this into your tests. Your automated tests must test GET and POST, and must test both possible enctypes for POST.

   You might also look at this discussion of GET and POST, and the formal W3C forms documentation.

2. Fix your code to work with arbitrary size requests.

   Right now, everyone is just using 'c.recv(1000)' to read the request in. But requests can actually be of arbitrary size! The size of the content payload is specified in the Content-Length header in the request. For GET requests it's 0 - easy. For POST requests, it's arbitrary.

   Make your code work with requests > 1000 (OR, change 1000 to be smaller, and make your code work with that). You should *not* count on 'c.recv(N)' returning N bytes, as this is not guaranteed; it will return *no more than* N bytes, and is guaranteed to return at least 1; so, plan to be calling c.recv multiple times.

   One trick here is that if you try to read one byte too many, your 'recv' will hang (see: blocking I/O).

   (The best way I've found to get this working is to change 'c.recv(1000)' to 'c.recv(1)', and then make your code work with *that*.)

   And, remember to write tests... that test something...

3. HTML and templating.

   Take a look at https://github.com/ctb/cse491-webz/tree/master/jinja2, and implement basic templating with jinja2.

Briefly, this means:

- install jinja2 in your virtualenv ('pip install -U jinja2');
- create a subdirectory called 'templates';
- in that subdirectory, place files that correspond to your different pages, including your form handling; basically, anything that generates HTML;
- call 'render' to generate HTML and return that HTML from your Web app; see 'render.py' in the cse491-webz repo, above, for example code.

Please also feel free to modify your HTML tests to just check the response code (200) and some key words in your HTML response.

4. Modify your code to return a "404 Not Found" status, with an (in)appropriate error message, when a URL that you *don't* handle is requested.
5. Check your code coverage and make sure that everything in your handle_connection function is covered. Everything.

CHAPTER 33

Day 8: Thursday, January 30th, 2014

0. Read the links from *Day 7: Tuesday, January 28th, 2014*: http://www.fastcompany.com/28121/they-write-right-stuff and http://www.infoworld.com/print/15243; skim http://calleam.com/WTPF/?page_id=2086
1. Quiz and discussion.
2. Lecture on code coverage and code paths: presentation
3. Code review and code coverage of hw3

I'm happy to help with any git problems you have, also.

33.1 Computing and displaying code coverage stats

First, activate your virtualenv (see *Using virtualenv*). Then install coverage:

```
pip install -U coverage
```

(You may want to reactivate your virtualenv after installing this; I've had some path problems.)

Next, grab some code to run it on – let's use the 'hw2-solutions' branch from https://github.com/ctb/cse491-serverz.

```
mkdir ~/cse491
cd ~/cse491
git clone https://github.com/ctb/cse491-serverz.git -b hw2-solutions day8
```

Run the tests with code coverage enabled:

```
cd day8
nosetests --with-coverage
```

Generate HTML output of your code coverage:

```
coverage html
```

which will put it in the directory 'htmlcov', and then post it to your arctic Web site:

```
mkdir ~/web/
cp -r htmlcov ~/web/day8-htmlcov
chmod -R a+rx ~/web/
```

Now go to http://www.cse.msu.edu/~$username/day8-htmlcov/ to look at the code coverage report – mine is here:

> http://www.cse.msu.edu/~ctb/day8-htmlcov/

If you look at 'server.py', you'll see some covered stuff (green) and some uncovered stuff (red):

http://www.cse.msu.edu/~ctb/day8-htmlcov/server.html

Why? Can you think of any way to "cover" the red stuff in the tests?

33.1.1 Things to try

Run code coverage on your own hw3. What did you fail to test in server.py?

Deactivate some of your tests by putting a 'return' right after the start of the function. Does the code coverage change in the expected way?

Do a review of someone else's hw3, and either make comments on their pull request or on their code directly (by checking it out locally). Note, you can grab someone's code by doing something like this, in an existing repo:

```
git fetch https://github.com/ctb/cse491-serverz hw3 ctb-hw3
git checkout ctb-hw3
```

If you make changes, comments, etc. then you can share them by first committing them, and then doing:

```
git push origin ctb-hw3:ctb-hw3
```

and setting up a pull request between YOUR copy of their branch (here, ctb-hw3 in your github repo) and THEIR hw3 branch.

If you use coverage analysis during your review, you might be able to find untested logic – and there might be bugs in there, too. Try writing a test that breaks their code (or at least covers it).

Code review checklist

Basic rules:

0. Does it run.
1. Do tests pass.
2. Spaces rather than tabs.
3. Spaces after #.
4. Properly spelled variable names .
5. Try writing tests to break something. For example, do you believe their POST logic?
6. 80 character line lengths.
7. Test with multiple browsers.
8. Use code coverage to find things that their tests don't test, and see if you can break their code.

33.2 List of repositories

massiek: github site - pulls - branches - repo URL for cloning

eunbong: github site - pulls - branches - repo URL for cloning

matheusldaraujo: github site - pulls - branches - repo URL for cloning

ConnorAvery: github site - pulls - branches - repo URL for cloning

jprickles: github site - pulls - branches - repo URL for cloning

jkteuber: github site - pulls - branches - repo URL for cloning
beckhamer: github site - pulls - branches - repo URL for cloning
john3209: github site - pulls - branches - repo URL for cloning
mill1256: github site - pulls - branches - repo URL for cloning
YourBestFriend: github site - pulls - branches - repo URL for cloning
hoffm386: github site - pulls - branches - repo URL for cloning
joshshadik: github site - pulls - branches - repo URL for cloning
msweet18: github site - pulls - branches - repo URL for cloning
juru13: github site - pulls - branches - repo URL for cloning
mcdonaldca: github site - pulls - branches - repo URL for cloning
filajust: github site - pulls - branches - repo URL for cloning
leflerja: github site - pulls - branches - repo URL for cloning
FireSBurnsmuP: github site - pulls - branches - repo URL for cloning
koppmana: github site - pulls - branches - repo URL for cloning
Karmeow: github site - pulls - branches - repo URL for cloning
curljosh: github site - pulls - branches - repo URL for cloning
yispencer: github site - pulls - branches - repo URL for cloning
glisto18: github site - pulls - branches - repo URL for cloning
mannin92: github site - pulls - branches - repo URL for cloning
westjour: github site - pulls - branches - repo URL for cloning
jbull477: github site - pulls - branches - repo URL for cloning
fakestuff: github site - pulls - branches - repo URL for cloning
msu-web-dev: github site - pulls - branches - repo URL for cloning
MaxwellGBrown: github site - pulls - branches - repo URL for cloning
xavierdhjr: github site - pulls - branches - repo URL for cloning
ettemaet: github site - pulls - branches - repo URL for cloning
lieblic2: github site - pulls - branches - repo URL for cloning
bjurgess1: github site - pulls - branches - repo URL for cloning
suhkang: github site - pulls - branches - repo URL for cloning
jonest31: github site - pulls - branches - repo URL for cloning
tsloncz: github site - pulls - branches - repo URL for cloning
zhopping: github site - pulls - branches - repo URL for cloning
MattyAyOh: github site - pulls - branches - repo URL for cloning
o2themar: github site - pulls - branches - repo URL for cloning
phammin1: github site - pulls - branches - repo URL for cloning
Badsauce: github site - pulls - branches - repo URL for cloning

DuncanCYoung: github site - pulls - branches - repo URL for cloning

cameronkeif: github site - pulls - branches - repo URL for cloning

majeedus: github site - pulls - branches - repo URL for cloning

polavar3: github site - pulls - branches - repo URL for cloning

brtaylor92: github site - pulls - branches - repo URL for cloning

labrenzm: github site - pulls - branches - repo URL for cloning

QSSS: github site - pulls - branches - repo URL for cloning

sarteleb: github site - pulls - branches - repo URL for cloning

JRucinski: github site - pulls - branches - repo URL for cloning

fenderic: github site - pulls - branches - repo URL for cloning

CHAPTER 34

Day 7: Tuesday, January 28th, 2014

0. Read: http://www.fastcompany.com/28121/they-write-right-stuff and http://www.infoworld.com/print/15243; skim http://calleam.com/WTPF/?page_id=2086
2. Lecture and demo on abstraction mechanisms in Python; ipynb.
1. More discussion of "advanced" Python, and WSGI standard.
2. dicts
3. Refactoring

CHAPTER 35

Homework 3

Due by noon on Thursday, Jan 30th.

0. Branches, etc.

   At your leisure, merge hw2 into your master. (This can be done by merging the pull request). If this has been done correctly, visiting your default github page for cse491-serverz should show you the code from hw2. You may have trouble merging; I'll produce a video over the weekend on how to do this. Until then, just work on the hw2 branch.)

   Your master branch should now contain hw2.

   To hand in this homework, put everything on a new branch, hw3 ('git checkout -b hw3'), and push that to github. Set up a pull request between hw3 and your master branch (again, matching to hw2) and *don't* merge the pull request – just leave it there.

1. Form parsing/GET.

   Use the 'urlparse' library (included with Python 2.7) to parse the 'path' component of HTTP requests so that GET form data can be handled. You should end up using both the 'urlparse' and 'parse_qs' functions.

   See the GET footnotes for more information on the basics.

   Here is an HTML form that will generate an HTTP GET with form values:

   ```
   <form action='/submit' method='GET'>
   <input type='text' name='firstname'>
   <input type='text' name='lastname'>

   </form>
   ```

   Serve this form via your Web server (e.g. as '/form') and then implement code to take '/submit' requests and return a page containing the string, "Hello Mr. $firstname $lastname." (You might look up Python string interpolation to figure out how to substitute variables, although if you want to be a bad programmer you can just use '+' to concatenate strings.)

   In addition, create an automated test verifying the results of the form parsing. (i.e. create a fake GET request with firstname/lastname set, and assert that the page contains the correct response containing that firstname/lastname).

2. Form parsing/POST/content-type.

   Edit or copy the form to also do this via a POST, for request content-type of application/x-www-form-urlencoded. Note that for POST requests the content will be submitted as part of the content body, which for GET requests is empty; so you need to parse that.

   Also see: http://www.cs.tut.fi/~jkorpela/forms/methods.html

You only need one form (so you can choose to do either a GET or a POST) but please make sure that both GET and POSTs *can work* in your HTTP server implementation. The best way to do this is to get both GET and POST working, and then make sure that you have automated tests for both; I expect to see tests for both.

3. Refactor your main function.

   Split the different pages up into different functions, so that '/' goes to (for example) 'index(...)'. Try to make the functions all take the same set of parameters, for future purposes. You can still have a bunch of if/else statements in the main handle_connection function ;).

4. Reminder: submit as a pull request, unmerged, from branch 'hw3' to branch 'master' (which contains hw2) on your github repo.

CHAPTER 36

Day 6: Thursday, January 23rd, 2014

0. Read: https://www.thc.org/root/phun/unmaintain.html. You might also like BOFH.
1. Quiz and discussion.
2. Presentation: HTTP, HTML, and links; ~/web/ on arctic. Presentation link
3. Presentation: git and merging.
4. Options for today!
   (a) Fix/ask for help with your hw2/pull request. (Ask your neighbor(s) first; then ask Titus.)
   (b) Code review! (See below.)
   (c) Play with (static) HTML on arctic.
   (d) Work through github tutorials: http://try.github.io/ and http://pcottle.github.io/learnGitBranching/

36.1 Code review HOWTO v2

1. Pick a name from the list of repositories below; go to the repository.
2. See if they have a pull request. If they do, go to step 5.
3. If they don't have a pull request, see if they have a hw2 branch. If they do, set up a pull request for them :)
4. If they don't have a hw2 branch, go to step 1.
5. Clone their repository (git clone https://github.com/USERNAME/cse491-servers.git) and then do a 'git checkout hw2'.
6. Perform your code review; place comments on the lines of code in the pull request.
7. Once done, go back and look at someone else's code, or do something else. (If you have questions on your code review, please feel free to e-mail me: ctb@msu.edu)

36.1.1 What to look at in code review

Basic rules:

0. Does it run.
1. Do tests pass.
2. Spaces rather than tabs.

3. Spaces after #.
4. Properly spelled variable names .
5. Try writing tests to break something. For example, do you believe their POST logic?
6. 80 character line lengths.
7. Test with multiple browsers.

36.2 Play with static HTML on arctic

Reminder: HTML reference – see through #4, linking pages.

On arctic, do:

```
mkdir ~/web/
mkdir ~/web/day6

echo "<a href='hello.html'>Hello, world</a>" > ~/web/day6/index.html
echo "Hello, world. <a href='./'>Go back</a>" > ~/web/day6/hello.html

chmod -R a+rx ~/web/
```

Now, in a Web browser, go to:

```
http://www.cse.msu.edu/~USERNAME/day6/
```

(make sure you have the '~' in there.)

This is how you create static pages on arctic. It is also a simple way to play with HTML to make sure that your basic HTML works right.

Tasks:

- create an absolute link to somewhere else on the index.html page
- create an absolute internal link to another page on CSE
- create a relative link to another page within your repository or elsewhere on CSE.

36.3 List of repositories

massiek: github site - pulls - branches - repo URL for cloning

eunbong: github site - pulls - branches - repo URL for cloning

matheusldaraujo: github site - pulls - branches - repo URL for cloning

ConnorAvery: github site - pulls - branches - repo URL for cloning

jprickles: github site - pulls - branches - repo URL for cloning

jkteuber: github site - pulls - branches - repo URL for cloning

beckhamer: github site - pulls - branches - repo URL for cloning

john3209: github site - pulls - branches - repo URL for cloning

mill1256: github site - pulls - branches - repo URL for cloning

YourBestFriend: github site - pulls - branches - repo URL for cloning
hoffm386: github site - pulls - branches - repo URL for cloning
joshshadik: github site - pulls - branches - repo URL for cloning
msweet18: github site - pulls - branches - repo URL for cloning
juru13: github site - pulls - branches - repo URL for cloning
mcdonaldca: github site - pulls - branches - repo URL for cloning
filajust: github site - pulls - branches - repo URL for cloning
leflerja: github site - pulls - branches - repo URL for cloning
FireSBurnsmuP: github site - pulls - branches - repo URL for cloning
koppmana: github site - pulls - branches - repo URL for cloning
Karmeow: github site - pulls - branches - repo URL for cloning
curljosh: github site - pulls - branches - repo URL for cloning
yispencer: github site - pulls - branches - repo URL for cloning
glisto18: github site - pulls - branches - repo URL for cloning
mannin92: github site - pulls - branches - repo URL for cloning
westjour: github site - pulls - branches - repo URL for cloning
jbull477: github site - pulls - branches - repo URL for cloning
fakestuff: github site - pulls - branches - repo URL for cloning
msu-web-dev: github site - pulls - branches - repo URL for cloning
MaxwellGBrown: github site - pulls - branches - repo URL for cloning
xavierdhjr: github site - pulls - branches - repo URL for cloning
ettemaet: github site - pulls - branches - repo URL for cloning
lieblic2: github site - pulls - branches - repo URL for cloning
bjurgess1: github site - pulls - branches - repo URL for cloning
suhkang: github site - pulls - branches - repo URL for cloning
jonest31: github site - pulls - branches - repo URL for cloning
tsloncz: github site - pulls - branches - repo URL for cloning
zhopping: github site - pulls - branches - repo URL for cloning
MattyAyOh: github site - pulls - branches - repo URL for cloning
o2themar: github site - pulls - branches - repo URL for cloning
phammin1: github site - pulls - branches - repo URL for cloning
Badsauce: github site - pulls - branches - repo URL for cloning
DuncanCYoung: github site - pulls - branches - repo URL for cloning
cameronkeif: github site - pulls - branches - repo URL for cloning
majeedus: github site - pulls - branches - repo URL for cloning
polavar3: github site - pulls - branches - repo URL for cloning

brtaylor92: github site - pulls - branches - repo URL for cloning

labrenzm: github site - pulls - branches - repo URL for cloning

QSSS: github site - pulls - branches - repo URL for cloning

sarteleb: github site - pulls - branches - repo URL for cloning

JRucinski: github site - pulls - branches - repo URL for cloning

fenderic: github site - pulls - branches - repo URL for cloning

CHAPTER 37

Day 5: Tuesday, January 21st, 2014

0. For class, read: http://ivory.idyll.org/blog/software-quality-death-spiral.html
1. Quiz and discussion.
2. Roadmap for next few weeks.

   Week 4: forms - submitting basic user input data to the web server; HTML.

   Week 5: WSGI - building a fully functional Web server component; templating.

   Week 6: More interesting Web apps; header processing & cookies.
3. Structure of HTTP, revisited. See presentation.

   Payload of request, abstractly

   Payload of response, abstractly
4. String whacking.

   Read `strings` and try to solve these problems generically, using only those string manipulation commands:

   (a) Pick out the 3rd value, e.g.

   ```
   f("a,b,c,d,e,f") == "c"
   ```

   (b) Extract everything after the 4th comma in a string, e.g.

   ```
   f("a,b,c,d,e,f,g") == "e,f,g"
   ```

   (c) Return the fourth and fifth lines of a multiline string, e.g.

   ```
   f("a\nb\nc\nd\ne\nf\n") = ["d", "e"]
   ```

   (d) Pick out the third and fourth values, removing leading underscores, e.g.:

   ```
   f("_a,_b,_c,_d,_e,_f") = ["d", "e"]
   ```

   See also String Methods, and Strings: Part I, Part II, and Part III.
8. Testing.

   Create a new directory & download two files to arctic by doing:

```
mkdir cse491-day5
cd cse491-day5
wget https://github.com/ged-lab/msu-cse491-2013/raw/master/day5.py
wget https://github.com/ged-lab/msu-cse491-2013/raw/master/tests_day5.py
```

Activate your virtualenv:

```
source ~/cse491.env/bin/activate.csh
```

and then run nosetests:

```
nosetests
```

You should see 8 errors from the code in 'day5.py'. Fix the code in 'day5.py' so that the tests all pass!

Solutions here: https://github.com/ged-lab/msu-cse491-2013/blob/master/day5-solved.py

CHAPTER 38

Homework 2

Due by noon on Thursday, Jan 23rd. Reminder, per the syllabus you may use The Google, friends, family, etc. to complete the homework.

0. Do all of the below on a new branch, 'hw2'. (See *Basic Instructions for Git and Github* for info on how to create a new branch.)

1. Fix any HW1 problems; see my solution here

   (a) All lines in the header must end in rn.

   (b) ChangeLog must be updated.

2. Refactor 'server.py' to be testable, and run some tests.

   a. Change server.py to have two functions, main() and handle_connection(conn). The 'main' function should have all of the current code (through the 'accept' statement in the loop) in it, while the actual connection handling (all the c.send stuff, and c.close) should be done in 'handle_connection(conn)'.

   At the end of server.py, then put:

   ```
   if __name__ == '__main__':
      main()
   ```

   'python server.py' should then still run the server, but 'python -c "import server"' should do nothing.

   (a) Make sure your code passes the test in test_server.py.

   Merge in my 'hw2' code, to your hw2 branch:

   ```
   git pull https://github.com/ctb/cse491-serverz.git hw2
   ```

   This will give you a new file, 'test_server.py'.

   Next, make sure you're in an activated virtualenv, and run 'pip install nose'.

   Finally, run 'nosetests':

   ```
   nosetests -s
   ```

   and (when you are done fixing your code :) you should see no errors.

   For more information on nose, read this introduction.

2. Update your server.py code to grab the request data (use 'c.recv(1000)'), extract the 'path' component, and return different HTML content for the following request URLs:

```
/
/content
/file
/image
```

Extend the tests to test each one of these; you should end up with at least four test functions in 'test_server', one for each of the request URLs.

3. Modify '/' to return HTML that contains links to /content, /file, and /image. Make sure the tests still pass (i.e. fix 'em).
4. Mofify the handle_connection function to handle POST requests separately from GET; use the script 'send-post-request' to test this. For now, just return 'hello world' or some such.

   Write a test for this behavior, too.

   Note, to use 'send-post-request', you'll need to run

   ```
   pip install requests
   ```

   inside your virtualenv.
5. Update ChangeLog with whatever it is you've changed; commit, push everything to your repository as part of the 'hw2' branch on your github repo. See *Basic Instructions for Git and Github*; you should be doing something like:

   ```
   git push origin hw2:hw2
   ```
6. On github, set up a pull request between your two branches, 'master' and 'hw2' (FROM hw2 INTO base master). Make sure your diff in the pull request contains all of the changes you wanted to have.

Done!

—

Links:

- Nose and testing tutorial
- HTML reference – see through #4, linking pages.
- The requests documentation

CHAPTER 39

Day 4: Thursday, January 16th, 2014

0. For class, read http://en.wikipedia.org/wiki/Deep_Web. Maybe also check out http://arxiv.org/pdf/1312.6122v1.pdf.

1. Quiz and discussion.

2. Code review HOWTO:

   To do a code review, today:

   (a) Check out and run/review; add comments and/or fix;

   (b) Commit your comments.

   (c) Push to branch 'hw1-review' on your own fork, and submit pull requests to original repository.

   To do all of this with git, let's do this in a new repo.

   ```
   git clone <repository under review> cse491-hw1-review
   cd cse491-hw-review
   # ... make changes ...
   git commit -am "review of hw1"
   git push <your home repository> master:hw1-review
   # now, go set up a pull request.
   ```

   The last 'git push' command is the only truly new one; you're just copying your default branch, 'master', to your home repository, with the new name 'hw1-review'.

   Your code review should focus on the following issues:

   (a) Does it run

   (b) Spaces rather than tabs

   (c) Spaces after # in comments

   (d) Properly spelled variable names

   See http://www.python.org/dev/peps/pep-0008/ for general Python code thoughts ;).

0. Do code review!

CHAPTER 40

Day 3: Tuesday, January 14th, 2014

0. For class, read http://www.codinghorror.com/blog/2009/10/the-xanadu-dream.html
1. Quiz and discussion.
2. Any homework questions?
   (a) What is ChangeLog, and how do I modify it? Do I use 'git log'? (No; just edit it with a text editor.)
   (b) How and where should I ask questions? At the Piazza site.
3. Demonstration: git and github; editing online and pull requests.

   (Note: this is how we will be doing code review and HW hand in. Also see GitHub Flow.)

   I will need a volunteer! Basic mousing skills needed, and bravery in the face of the unknown.
   (a) Log in to github.
   (b) Go to https://github.com/ged-lab/msu-cse491-2013
   (c) Click on 'fork'.
   (d) Go find the 'blog-posts.txt' file and click on it.
   (e) Click on 'edit'.
   (f) Add 'Hello, world!' just below the first line of equals
   (g) Scroll down and push 'commit'.
   (h) Click on 'pull requests' (upper right)
   (i) Click on 'set up new pull request'.
   (j) ...
4. Blog post discussion and signup; see *Blog post assignments*
5. Looking at the HTTP protocol.

   Concepts:

   - statelessness: Web browsers send a request, Web servers send a response.
   - permission/authentication-less: by default, the expectation is that you can link to any open page; by default, there is no authentication.
   - open: the HTTP protocol is an open standard that can be implemented by anyone, on either side – no licensing, no copyrights, no patents.
   - coupling: how interconnected things are. tightly coupled vs loosely coupled.

6. Code examples! (See *Day 2: Thursday, January 9th, 2014*)

CHAPTER 41

Day 2: Thursday, January 9th, 2014

0. For class, read http://en.wikipedia.org/wiki/History_of_the_World_Wide_Web
1. For class, fill out this survey.
2. See *Homework 1*. Any questions?
3. In class, fill out the reading quiz.
4. Discussion: why was the Web so successful?
5. How the class will work, approximately.
6. If we have time: what do the following code examples do?

Example 1, placed in 'example.py':

```
#!/usr/bin/env python
print 'hello, world'
```

Example 2:

```
import socket
```

Example 3:

```
while "foo":
   print 'hello, world'
```

Example 4:

```
while "":
   print 'hello, world'
```

CHAPTER 42

Homework 1

Due by noon on Thursday, Jan 16th. Reminder, per the syllabus you may use The Google, friends, family, etc. to complete the homework.

0. Sign up for the class mailing list.

1. Sign up for a github account (it's free!) at http://github.com/. (You can use whatever account name you want, but it will be public and may be connected to your real name by The Google, so 'luzer420' is probably a bad idea for your future.)

2. Go to the cse491-serverz repository and fork it into your own github account; then, clone it into your own CSE cluster account on arctic and follow the instructions in README.txt to get 'server.py' running. (See *Basic Instructions for Git and Github* for clone instructions. Please feel free to use your own computer, too; just make sure you're using python2.7 and a recent version of git.)

   Once you have server.py running, modify the code to return an HTTP 1.0 response, containing '200 OK' response line, a 'Content-type' of text/html, and a message body saying '<h1>Hello, world<h1> this is ctb's Web server.' (Replace 'ctb' with your own NetID or github username.)

   For more information on HTTP, please see this link:

   http://www.jmarshall.com/easy/http/#structure

   especially 'Sample HTTP exchange'. Note, you can emit a CRLF in Python with "\r\n".

   Verify that when you point your Web browser at your network server, you see the message you wrote, with 'Hello, world' in bigger letters than the rest. (The <h1> and </h1> should be invisible.)

   Once complete, commit your changes with a sensible commit message, and push them to your repository (again, see *Basic Instructions for Git and Github*). **Make sure your changed code is present in your repository by visiting the Web site for your repository and viewing the code. Deviation will not be tolerated.**

   OPTIONAL: immediately after the 'accept' call, put in a line 'print c.recv(1000)'. What does this do, and what is the format of the output?

3. Update ChangeLog with whatever it is you've changed; commit, push. (Yes, this is part of your homework.)

3. Fill out this form so that I know what repository is connected to what NetID!

Links:

- *Basic Instructions for Git and Github*
- The socket module in Python

CHAPTER 43

Indices and tables

- *genindex*
- *modindex*
- *search*