

# Udacity Full Stack Web Developer Nanodegree

---

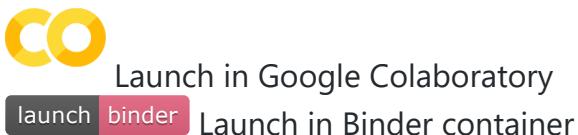


Udacity Full Stack Developer Nanodegree program

Brendon Smith ([br3ndonland](#))

[CODE LICENSE](#) [MIT](#)

Code in this repository is provided under the terms of the [MIT license](#).



## Table of contents

---

- Introduction:
  - [0.1 Readme](#)
  - [0.2 Syllabus](#)
  - [0.3 Markdown guide](#)
  - [0.4 Program feedback](#)
- Foundations:
  - [1.1 Shell Workshop](#)
  - [1.2 Git](#)
  - [1.3 GitHub](#)
  - [1.4.1,2,3 Python](#)
  - [1.4.4 Python intro](#)
  - [1.4.5 Functions: Break timer and secret message](#)
  - [1.4.6 Classes: Turtle graphics](#)
  - [1.4.7 Classes: send texts](#)
  - [1.4.8 Classes: Profanity checker](#)
  - [1.4.9 Classes: Movie website](#)
  - [1.4.10 Make Classes: Advanced topics](#)
  - [1.5.1 HTTP requests and responses](#)
  - [1.5.2 The Web from Python](#)
  - [1.5.3 HTTP in the real world](#)
- Front-end:
  - [2.1.1 HTML](#)

- 2.1.2 CSS
- 2.1.3 Sizing
- 2.1.4 Positioning
- 2.1.5 Floats
- 2.2 Responsive Design
- Back-end:
  - 3.1 Full Stack Web Developer Nanodegree program virtual machine
  - 3.2 Data and tables
- Web applications:
  - 4.1.1 APIs
  - 4.1.2 APIs repo pull requests
  - 4.2.1 CRUD
  - 4.2.2 Python web servers
  - 4.2.3 Flask
  - 4.2.4 Agile iterative development
  - 4.3 FSND JavaScript, AJAX, and APIs lessons
  - 4.4 Authentication and Authorization
- Servers:
  - 5.1 Linux server configuration

# README

---



Udacity Full Stack Developer Nanodegree program

Brendon Smith ([br3ndonland](#))

[CODE LICENSE](#) [MIT](#)

Code in this repository is provided under the terms of the [MIT license](#).



Launch in Google Colaboratory

[launch](#) [binder](#) Launch in Binder container

## Table of Contents

---

- [Description](#)
- [Computing environment](#)
  - [Keymapping](#)
  - [Package management](#)
  - [Shell](#)
  - [Version control](#)
  - [Text editor](#)
  - [Browsers](#)
- [Code syntax](#)
  - [JavaScript](#)
  - [Python](#)
- [Strategies](#)
  - [Projects](#)
  - [Lessons](#)

## Description

---

This is a repository for [Udacity Full Stack Web Developer Nanodegree program](#) (FSND) course notes and materials. The full program description and syllabus can be found on the [Udacity website](#), in [this repo](#), and via [PDF download](#).

Full stack web developers work on all aspects of websites and apps, from front end (features that users see) to back end (servers and databases). In this program, I built skills including:

- Developing webpages based on mockup images from designers
- Querying and manipulating large databases with SQL

- Creating functional multi-page web apps with databases and sign-ins
- Fetching data from Application Programming Interfaces (APIs)
- Deploying apps to Linux servers

The Full Stack Web Developer Nanodegree program is focused on projects, in which students can independently implement what they have learned in the lessons. I stored each project in its own repository.

1. [SQL database logs analysis](#)
2. [Python Flask catalog app](#)
3. [Linux server deployment](#)

The FSND program was recently reorganized. When I went through the program, it was longer, and I did three additional projects that are no longer included:

1. [Python web server movie trailer site](#)
2. [Portfolio website](#)
3. [Neighborhood map](#)

## Computing environment

---

Here are some suggestions for your computing environment. I use macOS, but these suggestions are easy to adapt for Linux or Windows. For full details on my personal setup, see my [setup.md](#) file on GitHub.

### Keymapping

- I use a [Microsoft Sculpt ergonomic keyboard](#).
- I remap the caps lock key to escape. This is built in to macOS now (System Preferences -> Keyboard -> Keyboard -> Modifier Keys)



- For more elaborate keymapping, check out [Karabiner](#).

## › Package management

- I use [Homebrew](#) on macOS.
  - Homebrew includes [Homebrew-Cask](#) to manage other macOS applications.
  - Note that [Homebrew 2.0.0](#) now runs on Linux and Windows.
  - Linux and Windows can also work with another package manager, such as [apton](#) Linux or [Chocolatey](#) on Windows.
- Install Homebrew from the command line:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master
```

- Install Homebrew packages with `brew install`:

```
# Homebrew packages
brew install git node python3 pipenv zsh zsh-completions

# Homebrew casks
brew cask install 1password backblaze docker figma firefox-developer-edition google-chro
```

- Update with `brew update` and `brew upgrade`.
- Check health of installation with `brew doctor`.
- Searching with `brew search <name>` now also searches casks.
- View info with `brew info <name>`.
- See the Homebrew [docs](#) for further info.

## › Shell

- Zsh
  - Like Bash with more features. See the Wes Bos [Command Line Power User course](#) for a tutorial. There is a version included with macOS, but it may be out of date.
  - Install via Homebrew:

```
brew install zsh zsh-completions
```

- [oh-my-zsh](#)
  - Install via `curl`:

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"
```

- Configure in `~/.zshrc`.

- [Pure prompt](#)

- Install from npm:

```
npm install --global pure-prompt
```

- Add the prompt to `~/.zshrc`:

```
# .zshrc continues above
plugins=(
    git node npm
)

source $ZSH/oh-my-zsh.sh

# User configuration
# Pure prompt: https://github.com/sindresorhus/pure
autoload -U promptinit; promptinit
prompt pure
# .zshrc continues below
```

- Try it out:

```
brew --version
echo 'stay udacious'
```

- [zsh-syntax-highlighting](#)

- Clone the repo into `~/.oh-my-zsh/plugins`:

```
git clone https://github.com/zsh-users/zsh-syntax-highlighting.git ${ZSH_CUSTOM:-~/}
```

- Activate the plugin in `~/.zshrc`:

- Open `~/.zshrc` from the command line with `code ~/.zshrc`.
- Add the plugin:

```
# .zshrc continues above
plugins=(
    git node npm zsh-syntax-highlighting
```

```
)  
  
source $ZSH/oh-my-zsh.sh  
  
# User configuration  
# Pure prompt: https://github.com/sindresorhus/pure  
autoload -U promptinit; promptinit  
prompt pure  
# .zshrc continues below
```

- [trash-cli](#): Moves files to the trash instead of permanently deleting with `rm`.

- Try it out:

```
touch file.txt  
trash file.txt
```

- For my terminal applications, I use:

- [iTerm2](#) and the [iterm2-snazzy](#) theme
  - [Hyper](#) and the [hyper-snazzy](#) theme
  - [VSCode integrated terminal](#)

## Version control

- [Git](#) is recommended for version control.
- I use the following general Git commit practices.
  - Separate subject from body with a blank line
  - Limit the subject line to 50 characters
  - Capitalize the subject line
  - Do not end the subject line with a period
  - Use the imperative mood in the subject line
    - A properly formed Git commit subject line should always be able to complete the following sentence: If applied, this commit will *[your subject line here]*
  - Wrap the body at 72 characters
  - Use the body to explain what and why vs. how
- Here's how the commit message might look:

```
Imperative commit title limited to 50 characters  
# Blank line  
- More detailed commit message body  
- List of key points and updates that the commit provides  
- Lines need to be manually wrapped at 72 characters
```

- See [how to make a Git commit message](#) and the [Udacity Git Commit Message Style Guide](#). Udacity recommends specifying the type of commit, like `feat: commit title`.
- Branching may not be needed in these projects, but in more complicated projects:
  - The `master` and `dev` branches are generally long-running branches.
  - Short-lived feature branches are merged to `dev`, then deleted.
  - The only commits to `master` are production-ready merges from `dev`.
  - See [Atlassian's comparing workflows page](#) for more.
- Undoing commits
  - [GitHub Blog: How to undo almost anything with Git](#)
  - [Atlassian: Rewriting history](#)
- I [connect to GitHub with SSH](#).

## Text editor

- I use [Microsoft Visual Studio Code](#) (VSCode). I have provided some suggested settings in [.vscode/settings.json](#). My full VSCode config is available in [this public GitHub Gist](#), and can be pulled into VSCode with the [Settings Sync extension](#).
- Theme: [Material Theme](#) Palenight
- Font: [Dank Mono](#) is my favorite. I particularly like the ligatures. I also like [IBM Plex Mono](#) and [Ubuntu Mono](#).
- See the [VSCode docs](#) to get started.
- Key features:
  - [Command palette](#): Cmd+shift+P opens a menu with available commands, based on your current context.
  - [Quick open](#): Cmd+p to quickly open files.
  - [Breadcrumbs](#): Cmd+shift+. opens up a navigation menu. Enables rapid navigation of files.
- Useful extensions:
  - [GitLens](#)
  - [Markdown All in One](#)
  - [markdownlint](#)
  - [Material Theme](#)
  - [Prettier](#)
  - [Python](#)
  - [Settings Sync](#)
  - [StandardJS](#)

## Browsers

- [Firefox Quantum Developer Edition](#)
  - Helpful dev tools, especially for CSS Grid
- [Google Chrome](#)
  - Progressive Web Apps (PWAs) dev tools

- Developer edition and canary may not be stable enough for regular use

[\(Back to top\)](#)

## ’ Code syntax

---

Decisive autoformatters save time and prevent [bikeshedding](#). Try changing the formatting of [fizzbuzz.js](#) or [fizzbuzz.py](#), then formatting with Standard (for JavaScript) or Black (for Python) as described below, to see what I mean.

### ’ JavaScript

- I recommend [JavaScript Standard Style](#) or [Prettier](#).
- [JavaScript Standard Style](#):
  - Two-space indentations, no semicolons.
  - It is important to understand automatic semicolon insertion (ASI), but you do not need to write semicolons in your JavaScript files. For more info, see the helpful blog posts by [Isaac Schlueter](#) (npm) and [Feross Aboukhadijeh](#)(JavaScript Standard Style).
  - Standard Style is different from the style recommended in the [Udacity Frontend Nanodegree Style Guide](#). If you use Standard Style, just make a note of it in your README and project submission. I submitted my projects formatted with Standard Style, and the reviewers complemented me on how clean and readable it was.
  - There is a StandardJS [vscode extension](#) available for linting and autoformatting. I have noticed I sometimes need to save files more than once for all formatting to be applied.
- [Prettier](#):
  - Another useful autoformatter.
  - Prettier supports more languages than Standard Style, including TypeScript, Markdown, HTML, and CSS (and maybe even Python, eventually).
  - [Configuration options](#) are specified in a `.prettierrc` file. There is minimal configuration needed. I add `"semi": false`. Some users may also want to add `"trailingComma": "es5"`, or extend the default 80 character line length to 100.
  - There is a [Prettier VSCode extension](#).

### ’ Python

- Helpful VSCode resources:
  - [Python in Visual Studio Code](#)
  - [Getting Started with Python in VS Code](#)
  - [Editing Python in Visual Studio Code](#).
- [Use Python 3 \(modern Python\)](#). Python 2 (legacy Python) is nearing its [end of life](#).
- I recommend installing `python3` and `pipenv` with Homebrew. After installing `python3`, `pip` can be updated with `pip3 install --upgrade pip setuptools wheel`.
- [I format Python code with Black](#).
  - VSCode provides built-in support for Black. I set VSCode to autoformat on save.
  - If you use Black to format your Python code, just make a note of it in your README and project submission.

- Black is still considered a pre-release, and Pipenv may throw some errors. There are two key steps to using Black within a Pipenv:
  - a. Installing Black with a `--dev` flag: `pipenv install black --dev`
  - b. Allowing pre-releases into the lock file: `pipenv lock --pre`
- If you prefer the less-decisive PEP 8 format, I recommend [autopep8](#) for autoformatting. VSCode also has built-in [Python formatting](#) support for autopep8.
- Jupyter
  - I [install JupyterLab](#) with Pipenv.
  - Install Homebrew from the command line as described on the [Homebrew website](#).
  - After installing Homebrew, install the necessary Homebrew packages from the command line.
  - Once installation is complete, navigate to your project's directory, install dependencies, and run JupyterLab.
  - Here are the necessary command line arguments:

```

brew install python3
brew install pipenv
brew install jupyter
cd path/where/you/want/jupyterlab
# On first install: pipenv install jupyterlab requests numpy statistics twilio
# After Pipfile is generated
pipenv install
pipenv shell
# Install any JupyterLab extensions at this point
(pipenv) $ jupyter labextension install @jupyterlab/toc
(pipenv) $ jupyter lab

```

- I previously used [Anaconda](#) to manage my Python and R distributions, and now use Homebrew. I switched because Anaconda is a very large installation, not as flexible or general as Homebrew, and not as important for virtual environments now that we have Pipenv.
- There are two options for running the Jupyter Notebooks in the cloud: [Google Colaboratory](#) and [Binder](#).

[\(Back to top\)](#)

## › Strategies

---

## › Projects

- **Focus on projects.** The Nanodegree is awarded for projects, not lessons and quizzes. As you advance past the fundamentals into the project topics, you may want to just skip directly to the

projects, and go back through the lessons as needed. [Other students](#) have succeeded with this project-focused approach.

- **Master Markdown.**
  - Markdown is a simplified HTML syntax that is very useful for notes and documentation.
  - Practicing Markdown will enable you to become fluent in a simple syntax.
  - Check out my Markdown guide at [docs/markdown-guide.md](#).
  - Helpful VSCode resources:
    - [Markdown and Visual Studio Code](#)
- **Keep computational narratives.**
  - When coding projects, I keep computational narratives describing what I do at each step, like journals or lab notebooks. I learned how to keep computational narratives from scientific computing in Jupyter Notebook/JupyterLab and RMarkdown.
  - Computational narratives capture my train of thought, so I can retrace my steps, retain what I have learned, easily generate documentation, and teach others.
  - Here are some examples of my computational narratives:
    - [SQL logs analysis project narrative](#)
    - [Flask catalog app narrative](#)
    - [Flask catalog app server deployment narrative](#)
- **Break projects down into actionable steps.**
  - This is a key skill. As a developer, you will be given large tasks, like the projects in this Nanodegree program, without a clear step-by-step plan. You will need to plan out the project and make progress on each step.
  - I break the project down into steps, and make the steps headers in my computational narratives. I generate a Table of Contents from the headers, using the "Create Table of Contents" feature of the Markdown All In One VSCode extension.
  - As I work, I include code in Markdown fenced code blocks.
  - If you get stuck or blocked:
    - **Take a break.** Go do something else you enjoy. It's normal to get confused and frustrated. Don't let it get to you. When you are learning, expect to experience more [stretch](#) than [flow](#).
    - **Talk it out.** This is called "rubber duck debugging," because some people put a rubber duck on their desk and pretend they are talking to the duck. See the [rubber duck debugging website](#) and [CS50 2018 - Lecture 4 - Data Structures: ddb50](#).
    - **Write it down.** In addition to verbal rubber duck debugging, write the situation down in your computational narrative. Explain what you're stuck on.
    - **Break it down.** Break the problem down into smaller steps, and continue working through each step. Make sure you document the steps you take to solve the problem.

## ^ Lessons

- **Limit lesson time.** Speed up videos to 1.5x or 2x, and set a timer when working through the lessons. I used the [Pomodoro technique](#), and limited myself to 25 minutes max per lesson section. I would often complete 2-3 lesson sections per 25 minute interval.
- **Limit quiz attempts.** When I was getting started, I aimed to complete 100% of the lesson material, and I took the quizzes too seriously. I pushed myself to answer quiz questions correctly without checking solutions. This led me to hit sticking points, and I would sometimes

take 1-2 days just to complete a quiz. As I went on, I set a limit of three quiz attempts. If I didn't get it in three attempts, I would check the solution and move on.

- **Take notes on the lessons.** See [info/markdown-methods.md](#) for more information on Markdown, and [info/udacity-lesson-notes-sample.md](#) for an example.
  - When beginning a Udacity lesson, create a new Markdown file in your text editor.
  - Reserve `H1` for the title at the top, like `# Lesson 1. Shell workshop`.
  - Reserve `H2` ( `##` ) for breaking the lesson into different sections. If the lesson just has one section, I would recommend calling it `## Concepts` , like Udacity does.
  - Paste in the sections of the lesson from the Udacity interface, and set each one to `H3` ( `###` ).



- As with projects, use the lesson notes file as a computational narrative while you work through each part of the lesson.
- At the end of the lesson, I generate a Table of Contents from the headers, using the "Create Table of Contents" feature of the Markdown All In One VSCode extension.

See my [program feedback](#) for more comments.

# Syllabus

---



Udacity Full Stack Developer Nanodegree program

[Download Syllabus](#)

## Table of Contents

---

- [Full Stack Curriculum](#)
  - [Prerequisite Knowledge](#)
  - [Developer Fundamentals](#)
  - [Databases with SQL and Python](#)
  - [Servers, Authorization, and CRUD](#)
  - [Deploying to Linux Servers](#)
- [Need to prepare?](#)

## Full Stack Curriculum

---

In this Nanodegree program, you'll learn how to build and manage relational databases to store and handle application data, and build powerful server-side applications to serve that data to any type of client-side application.

You'll learn to build applications that can support any front-end, and scale to support hundreds of thousands of users.

**4 Months to complete, 160 hours total (10 hours/week x 16 weeks)**

## Prerequisite Knowledge

To enroll, you should have experience with Python Programming (or another object-oriented programming language), Programming with JavaScript, Git/GitHub, HTML basics, and Data Structures including Lists, Arrays, Dictionaries. See detailed requirements.

## Developer Fundamentals

Brush up on your knowledge of essential developers' tools such as the Unix shell, Git, and Github; then apply your skills to investigate HTTP, the web's fundamental protocol.

## Databases with SQL and Python

Master relational databases with the power of SQL, and leverage Python to incorporate database logic into your programs.

## › Project: Logs Analysis

Analyze data from the logs of a web service to answer questions such as "What is the most popular page?" and "When was the error rate high?" using advanced SQL queries.

## › Servers, Authorization, and CRUD

Build multi-user web applications using the Flask framework, SQLAlchemy, and authentication providers such as Google.

## › Project: Item Catalog

Develop an application that provides a list of items within a variety of categories, as well as provide a user registration and authentication system. Registered users will have the ability to post, edit, and delete their own items.

## › Deploying to Linux Servers

Deploy your web applications onto Linux servers. Learn the essentials of securing and configuring Linux web servers, then deploy a full stack application to a live server with a database and routing.

## › Project: Linux Server Configuration

Take a baseline installation of a Linux distribution on a virtual machine and prepare it to host your web applications; this will include installing updates, thereby securing it from a number of attack vectors and installing/configuring web and database servers.

## › Need to prepare?

---

We offer a number of Nanodegree programs and free courses that can help you prepare:

- [Intro to Programming](#)
- [Front End Web Developer](#)
- [Introduction to Python Programming](#)
- [Introduction to Algorithms](#)
- [Version Control with Git](#)
- [Intro to HTML and CSS](#)

# Markdown guide

Brendon Smith ([br3ndonland](#))

## Table of Contents

- Markdown syntax
    - Syntactic suggestions
    - General Markdown resources
  - Markdown apps
    - Text editors
    - IDEs
    - Note apps
    - In-browser editors
    - Social apps

**Markdown** is a simplified HTML syntax. It has most of the functionality of HTML while being much easier to read, and is very widely used (for example, READMEs on GitHub).

Here's a comparison of the same code written in Markdown and HTML:

```
udacity-google-07.md x udacity-google-07.html x udacity-google

## 7.10. Defaults and Destructuring

#### Defaults and destructuring arrays

* You can combine default function parameters with destructuring to create some pretty powerful functions!

```js
function createGrid([width = 5, height = 5]) {
  return `Generates a ${width} x ${height} grid`;
}

createGrid(); // Generates a 5 x 5 grid
createGrid([2]); // Generates a 2 x 5 grid
createGrid([2, 3]); // Generates a 2 x 3 grid
createGrid([undefined, 3]); // Generates a 5 x 3 grid
```

| Returns:
| Generates a 5 x 5 grid
| Generates a 2 x 5 grid
| Generates a 2 x 3 grid
| Generates a 5 x 3 grid
| 

The `createGrid()` function expects an array to be passed to it. It uses destructuring to set the first item in the array to the `width` and the second item to be the `height`. If the array is empty or if it has only one item in
```
1296 <h3 id="710-defaults-and-destructuring">
1297   <a id="user-content-710-defaults-and-destructuring" href="#710-defaults-and-destructuring" class="anchor" aria-hidden="true"><span aria-hidden="true" class="octicon octicon-link-external">/</span></a> 7.10. Defaults and Destructuring:</h3>
1298 <h4 id="defaults-and-destructuring-arrays">
1299   <a id="user-content-defaults-and-destructuring-arrays" href="#defaults-and-destructuring-arrays" class="anchor" aria-hidden="true"><span aria-hidden="true" class="octicon octicon-link-external">/</span></a> Defaults and Destructuring arrays:</h4>
1300 <p>You can combine default function parameters with destructuring to create some pretty powerful functions!</p>
1301 <div class="highlight highlight-source-js"><pre><span class="pl-k">function</span> <span class="pl-en">(</span> <span class="pl-k">[</span> <span class="pl-s1">[</span> <span class="pl-s1">width</span>, <span class="pl-s1">height</span> <span class="pl-s1">]=</span> ]</span>)</span> {<span class="pl-k">return</span> `Generates a ${width} x ${height} grid`;</span>}</pre></div>
1302 <span class="pl-c" style="color: #808080;">// Generates a 5 x 5 grid</span>
1303 <span class="pl-c" style="color: #808080;">// Generates a 2 x 5 grid</span>
1304 <span class="pl-c" style="color: #808080;">// Generates a 2 x 3 grid</span>
1305 <span class="pl-c" style="color: #808080;">// Generates a 5 x 3 grid</span>
1306 <span class="pl-c" style="color: #808080;">// The `createGrid()` function expects an array to be passed to it. It uses</span>
1307 <span class="pl-c" style="color: #808080;">// destructuring to set the first item in the array to the `width` and the second</span>
1308 <span class="pl-c" style="color: #808080;">// item to be the `height`. If the array is empty or if it has only one item in</span>
```

```

## Markdown syntax

## ‣ Syntactic suggestions

Suggestions for standardized Markdown formatting have been provided by [markdownlint](#) and [Markdown Style Guide](#). Here are a few personal pointers:

## › File extensions

- Several different extensions can be used, including .md, .mdown, and .markdown.
- I prefer to use .md for brevity and consistency.

## › Headers

- Create headers with `#`. Each `#` increases header level (`##` is outline level two), up to six levels.
- For organization, I reserve H1 (`#`) for the title of the file at the top. Major headers begin with H2 (`##`).
- I use headers to create a **Table of Contents (TOC)** at the beginning of the file.
  - I add `## Table of Contents` before the TOC for navigation. I also include `<!-- omit in toc -->`, which tells the VSCode Markdown All In One extension not to put the `## Table of Contents` header itself into the TOC.
  - I include [\(Back to top\)](#) links after each section for easy navigation back to the top of the page. Simply write `[(Back to top)](#top)`.
  - I add and auto-update TOCs in vscode with the [Markdown All in One](#) extension.
  - [Markdown All in One](#), JupyterLab and RStudio provide inline TOC displays ([see below](#)).
  - Prior to vscode, I was adding and updating TOCs with [DocToc](#) from the command line.

## › Text

- **Bold text:** use double star at beginning and end of text to bold
- **Italics:** Single star with no space before and after. Note that underscores also work.
- I prefer to indent Markdown text with two spaces. Four spaces can be read by some systems as code blocks.

## › Lists

- Lists should be preceded by a blank line.
  - Single `*`, `-`, or `+` at beginning of line, followed by tab or space.
    - Indent with tab for next outline level
      - Like this
1. Ordered lists
  2. Like this
    - And you can add in unordered lists within ordered lists like this.
    - Adding an unordered list within an ordered list requires two levels of indentation.

## › Code

You can include `inline code inside single backticks`

`Fenced code blocks inside triple backticks`

- Code blocks can be indented to match your lists

like this

- In GitHub-Flavored Markdown, you can specify the language next to the first set of triple backticks for syntax highlighting. Each language has a full name (like `python`), and an abbreviation (like `py`). The full list of supported languages can be found in [GitHub's Linguist repo](#), which is used to detect languages on GitHub. The `languages.yml` contains a list of the available abbreviations (called "extensions" in the YAML) for each language.

- Shell: `shell` or `sh`

```
git status  
git commit
```

- JavaScript ([fizzbuzz.js](#), ES6, formatted with [Standard](#)): `javascript` or `js`

```
const fizzBuzz = () => {  
  for (let i = 1; i <= 100; i++) {  
    let out = ''  
    if (i % 3 === 0) out += 'Fizz'  
    if (i % 5 === 0) out += 'Buzz'  
    console.log(out || i)  
  }  
}
```

- Python ([fizzbuzz.py](#), Python 3, formatted with [Black](#)): `python` or `py`

```
def fizzbuzz():  
    """Print 1-100  
    Multiples of 3: Fizz  
    Multiples of 5: Buzz  
    Multiples of 3 and 5: FizzBuzz  
    """  
  
    for i in range(1, 101):  
        out = ""  
        if i % 3 == 0:  
            out += "Fizz"  
        if i % 5 == 0:  
            out += "Buzz"  
        print(out or i)
```

## › Images

```
![Alt text that appears below the image in the output](/path/to/img.jpg "Optional title  
that will show up when you hover over the image in the output")`
```

I still prefer to use HTML image tags, because they allow for more customization. In particular, it's useful to set width on SVG.

```

```



([Back to top](#))

## › General Markdown resources

- [MarkdownGuide](#)
- [markdownlint syntax suggestions](#)
- [Markdown Style Guide](#)
- [GitHub-Flavored Markdown](#)
- [Dillinger](#) is a helpful online Markdown editor with live preview.
- [Turndown](#) is an HTML to Markdown converter.
- [Udacity README course](#)

([Back to top](#))

## › Markdown apps

---

### › Text editors

Most code editors have extensions for Markdown.

#### › Atom

[Atom](#) has good Markdown support. See the [Flight Manual](#) for instructions.

#### › Sublime Text

Here's how to set up Sublime Text for Markdown:

- Install [Sublime Text](#)
  - I like the Mariana color scheme and the Adaptive theme.
- Install [Package Control](#)
- Use Package Control from within Sublime Text to install:
  - [MarkdownEditing](#)
  - [Markdown Preview](#)
  - [MarkdownLivePreview](#): Has some issues with lack of wrapping in the previews. See [GitHub Issue tracker](#).

#### › Visual Studio Code (vscode)

In my opinion, vscode is currently the best editor for working with Markdown. Here's why:

- [Markdown All in One](#)
  - TOC auto-generation and update
  - Live TOC in explorer panel
  - Easier keyboard shortcuts than Sublime Text (with the exception of hyperlink insertion, which I added with a [keybinding](#))
- [markdownlint](#)
  - Lints Markdown files based on style recommendations for standardizing code.
- Built in live preview
  - The [Markdown preview can be extended](#) with custom CSS. The CSS must be placed within the current workspace folder. There isn't yet support for absolute file paths to the CSS, but this feature has been [requested](#). I have also tried using an HTTPS link to a GitHub Gist, but haven't had success with that yet.

Full vscode configuration is available via [Settings Sync](#) with [this public GitHub gist](#).

See [Markdown and Visual Studio Code](#) for more info.

[\(Back to top\)](#)

## › IDEs

IDE = Integrated Development Environment

## › JupyterLab

[JupyterLab](#) is produced by [Project Jupyter](#). It is most widely used for scientific computing with Python, but supports many programming languages. It allows you to create "reproducible computational narratives," containing Markdown text interspersed with code chunks that you can run. JupyterLab has [some awesome features](#) and was previously Jupyter notebook.

I would suggest using [JupyterLab](#) within a Pipenv virtual environment. I use the [Homebrew](#) package manager on macOS to install Python 3, the [Pipenv](#) virtual environment tool, and [Jupyter](#). Here are some setup instructions:

- Install Homebrew from the command line as described on the [Homebrew website](#).
- After installing Homebrew, install the necessary Homebrew packages from the command line:

```
brew install python3
brew install pipenv
brew install jupyter
```

- Once installation of Homebrew and its packages is complete, navigate to the desired directory, or [clone a repository from GitHub](#).

```
cd path/to/a/directory
git clone
```

- Finally, install the virtual environment with Pipenv, which includes [JupyterLab](#) and the other necessary packages, and launch JupyterLab to run the Jupyter Notebook.

```
cd tdi-proposal
pipenv install
pipenv shell
# Install any JupyterLab extensions at this point
(pipenv) $ jupyter labextension install @jupyterlab/toc
# Launch JupyterLab
(pipenv) $ jupyter lab
```

I previously used [Anaconda](#) to manage my Python and R distributions, and now use Homebrew. I switched because Anaconda is not as flexible or general as Homebrew, not as important for virtual environments now that we have Pipenv, and is a very large installation that is difficult to manage and uninstall.

For examples of how to use Jupyter Notebook/JupyterLab, you can check out my [Udacity Full Stack Web Developer Nanodegree program repo](#).

## › RStudio

- [RStudio](#) is an IDE for the R programming language, used mostly for statistics and data science.
- Like JupyterLab, R Markdown documents contain Markdown text with functional R code chunks.
- I have provided an example of scientific data analysis with R Markdown on [GitHub](#).

[\(Back to top\)](#)

## › Note apps

Also see [Notable's comparison table](#).

### › Bear

#### › Bear pros

- This is one of the best Markdown note apps.
- Supports Markdown. Uses a modified syntax called Polar Bear.
- Tags and subtags (nested tags)
- Untagged notes easily identified
- Themes
- Syntax highlighting
- Supports internal relative links
- Evernote migration and import (though not perfect-see cons below)
- Writing tools, like word counts and read time
- Reasonably-priced subscription plan

## › Bear cons

- Apple only (macOS, iOS, iCloud), with [no plans to support Android](#).
- Not encrypted
- Collaboration features could be better. No shared notebooks.
- Sidebar should be more condensed.
- Web clipper needs some work. Doesn't properly capture text on all sites.
- Evernote import doesn't convert Evernote internal note links to Bear note links. Joplin also has this same issue, and it's a major barrier for switching from Evernote.

## › Day One

### › Day One pros

- Day One [supports Markdown](#).

### › Day One cons

- Subscription service
- Google Drive sync requires Google login, so you can't use it if you have [Advanced Protection](#) enabled.

## › Dropbox Paper

### › Dropbox Paper pros

- Markdown export
- Collaboration
- Sync
- Embedding works well

### › Dropbox Paper cons

- Paper files don't show up in your regular Dropbox file structure
- No tags
- No themes
- No internal linking
- Only has H1-H2. H3 shows up as bold when downloading Markdown files.
- PDFs embedded in Dropbox Paper documents don't lead to the actual PDF when clicked, in the mobile apps.
- PDF thumbnails are too large when inserted as single PDFs. When two or more files are inserted, the thumbnails are next to each other and the size is more reasonable. Needs a "view as attachment" option like Evernote.

## › iA Writer

iA = information Architects

## › iA Writer pros

- Cross-platform
- Light/dark themes
- Currently just a one-time payment model, though that will probably change.

## › iA Writer cons

- Lacks some of the features of Bear
- No syntax highlighting
- Multimedia?
- Evernote import?

## › Inkdrop

### › Inkdrop pros

- [Features:](#)
  - Markdown
  - Themes
  - Encryption
  - Cross-platform. Seems to be like Bear, but cross-platform.
  - Check out the developer's [Medium blog](#).

### › Inkdrop cons

- Subscription plan
- Android app has poor reviews (~3.3 rating)

## › Joplin

### › Joplin pros

- Open-source
- Desktop is Electron, mobile is React Native.
- Flexible cloud sync
- Markdown format
- Evernote import

### › Joplin cons

- Evernote to Joplin migration not ideal (see below).
- Documentation on the website is okay, but [contributing guidelines](#) are not well delineated. The code and stack should be clearly explained so people can easily contribute. I should know roughly where I need to go in the codebase to add a feature.
- Laurent Cozic only makes \$60/month on Patreon. Not sustainable.

### › Evernote to Joplin migration

- Tried this on 201809
- Export Evernote notebook to .enex.
- Joplin -> Import -> ENEX
  - Images came through
  - Formatting generally came through well.
- Updates needed
  - Critical
    - Internal note links import as Evernote links, not Joplin links. This is a critical issue for me.
    - Example:
      - Joplin internal note link: `:/cddf8681528148b9aad5077198e58a4a`
      - Evernote internal note link: `evernote:///view/6168869/s55/426ec3ae-6cad-48c7-aab3-b300845063ef/426ec3ae-6cad-48c7-aab3-b300845063ef/`
  - Links sometimes also just lead to the top note result in the notebook, rather than the specific note needed.
  - Note URL not yet available (issue #427). This will be important for clipped news articles.
  - Bold text comes through as big section breaks (see issue #767)
    - I had to un-bold all the headers I put into my notes.
- Enhancements
  - Evernote code blocks not transferring in as Markdown code blocks.
  - Markdown TOC (issue #478)
  - Searching in notes: The search bar only searches notes, but doesn't reveal results within the note. See #382.

## › Jottings

iOS-only note app with Markdown, tagging, and Dropbox sync.

## › Journey

### › Journey pros

- Encrypted
- Cross-platform
- Google Drive sync
- Import from Day One, Evernote, etc

### › Journey cons

- Forces Google login before allowing access to app on Mac. This is a problem because Google Drive sync is not allowed when [Google Advanced Protection Program](#) is enabled.
- Subscription options
- Vague Evernote import capabilities

## › Laverna

## › Laverna pros

- Encrypted
- Markdown
- Multimedia
- Internal linking
- Make tags with #tag
- Open source
- Built with Electron

## › Laverna cons

- Evernote import?
- No Android app yet
- No dark themes yet
- No Markdown TOC
- Development coming along slowly

## › Notable

Heard about Notable via the [Changelog weekly email #238](#).

## › Notable pros

From the [README](#):

The markdown-based note-taking app that doesn't suck.

I couldn't find a note-taking app that ticked all the boxes I'm interested in: notes are written and rendered in GitHub-flavored Markdown, no WYSIWYG, no proprietary formats, I can run a search & replace across all notes, notes support attachments, the app isn't bloated, the app has a pretty interface, tags are indefinitely nestable and can import Evernote notes (because that's what I was using before).

So I built my own.

The developer has made extensive comparisons with other note apps. See [Notable's comparison table](#).

## › Notable cons

- No mobile app yet

## › Simplenote

From WordPress

## › Simplenote pros

- Markdown support

## › Simplenote cons

- Evernote import?
- Multimedia?

## › Standard Notes

### › Standard Notes pros

- Simple, dependable text note app
- Note tagging
- Themes like solarized and dark
- Extensions to add features like Markdown
- Encrypted
- Backup to Dropbox and Google Drive
- "Built to last"

### › Standard Notes cons

- User interface is generally not intuitive.
  - The Account -> Encryption section says "8/8 notes and tags encrypted." Why would I want to combine the number of notes with the number of tags?
- Very difficult to even figure out how to log in to my account online. Apparently there are separate accounts for the web app and premium? I think it's the [dashboard](#).
- Extensions make it way too confusing.
  - I have to select different editors? I don't want a different editor for every task, I want to use one editor for all tasks.
  - Extensions used to only work on desktop and web (they now work on mobile apparently).
- Attachments
  - Can't attach files from mobile devices.
  - Not great with multimedia. Tried to drag and drop a movie, and it just displayed the movie instead of my notes.
- [Evernote import](#):
  - Formatting, images, and attachments will not be copied over.
  - Have to break up .enex into 250 MB segments.
  - Import is not intuitive. Importing anything goes through import backup, but most imports are not backups.
  - **Doesn't do a good job of Evernote HTML to Markdown conversion.** Loses file attachments and lists, doesn't recognize headers. The options are either retain HTML, or strip all formatting.
  - My response to email survey:

*Have you ever tried Standard Notes Extended? No. If I became a regular user, I would happily pay for Extended. I'm not regularly using Standard Notes because I'm heavily invested in Evernote. I've been using it for six years, and have a 4 GB database with ~4400 tagged notes. I would love to migrate from Evernote to another service, converting from rich text to Markdown and encrypting my data, while keeping my multimedia attachments. Bear is one example of this type of migration, but I don't use Bear because it's Apple-only.*

## › Trilium Notes

Heard about Trilium via the [Changelog weekly email #237](#).

### › Trilium pros

- Evernote import: I haven't tried it yet, so I can't comment on import of formatting, internal links, etc.
- Code editing with syntax highlighting
- Also supports mind mapping
- Encryption

### › Trilium cons

- Work in progress
- No mobile app yet

## › Turtl

[Turtl blog on Tumblr](#)

### › Turtl pros

- Promising encrypted Evernote alternative
- Evernote import coming in 0.6.5
- Markdown
- Sharing
- Some multimedia support
- Android app

### › Turtl cons

- Still needs more development, and development has been very slow.
- Not sure how dependable this app will be. The developers don't even own a MacBook.
- No dark themes yet

## › Typora

### › Typora pros

- File list panel, allowing you to use any cloud service to sync.

- CSS configurable
- Support for images
- Document export
- Footnote feature is cool

#### › Typora cons

- Feature-poor
- Not for android
- If it's not for mobile and not multimedia, what's the point beyond Sublime Text? Or Dillinger?

### › [Ulysses](#)

#### › Ulysses pros

- Nice interface
- Markdown
- Dark themes
- Can also manage journal articles and research

#### › Ulysses cons

- Apple only (macOS/iOS)
- Encryption?
- Moved to subscription model
- Maintains article tags as "keywords," a fatal flaw shared by other apps like [Papers](#). See my notes on citation managers in [this public GitHub Gist](#)

## ³ In-browser editors

- [Dillinger](#): In-browser Markdown editor with live preview.
- [GitBook](#): Online platform for writing documentation. It combines and converts Markdown files into multimedia-enabled notebooks, like book chapters.
- [Gnotes](#): Write Markdown, save to Dropbox, see in Evernote. One-way only.
- [Markdown Here](#): Allows Markdown formatting for in-browser web apps like Gmail. See [GitHub](#).
- [Marxico](#):
  - In-browser Markdown editor based on Dillinger.
  - Works with images.
  - Has live TOC.
  - Renders well on mobile browsers, but only selection capability available is "select all".
  - May not have seamless integration with Evernote. See [discussion on Evernote forums](#).
- [StackEdit](#): In-browser Markdown editor with live preview. Uses PageDown, the engine powering the Markdown capabilities of the Stack Exchange forums.
- Udacity discussion forums: You can use Markdown formatting in your forum posts.

([Back to top](#))

## › Social apps

### › Gitter

[Gitter](#) uses Markdown formatting for chats. See their [Markdown basics](#) support article.

### › Slack

Slack uses a simplified pseudo-Markdown to format messages. I call it "slackdown." On the [Slack "Format your messages" page](#), Slack states that it will not be building in full Markdown capabilities.

# Program feedback

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

[br3ndonland](#)

## Table of Contents

---

- [Highlights](#)
- [Lessons](#)
  - [Code and tech](#)
  - [Organization](#)
  - [Time](#)
- [Mentor](#)
- [Networking](#)
  - [Forums](#)
  - [Slack](#)
  - [In-classroom chat](#)
  - [Knowledge](#)
- [Exit survey](#)
- [Program developments](#)

## Highlights

---

- **Code review was wonderful.** I was really impressed with how thoughtful and thorough the reviews were. I learned a great deal from the suggestions in the reviews. The turnaround time was very fast, sometimes only a few hours.
- The website and Android app were generally responsive and powerful.

[\(Back to top\)](#)

## Lessons

---

### Code and tech

- It was good that we used Python instead of Ruby. Python is more flexible. For example, I also have training in data science, so I can use my Python skills for data analysis, which I couldn't really do with Ruby.
- Some frameworks, like Flask, were current, others, like Knockout, were outdated.

- The Vagrant virtual machine was helpful, but a little bit complicated. Udacity could **consider containerizing the code** for further control over the environment, and easier setup by students.
- **The code in the lessons is of inconsistent quality. We are learning, and we need to see well-structured, consistent, and current code.** Udacity is focused on releasing new Nanodegree programs, without maintaining the programs they already have. We need to learn based on current best practices, which requires frequent updates. Compare Udacity's lack of updates with someone like [Wes Bos](#), for example, who has already re-recorded his React course three times. And that's just one guy. Udacity has an entire company full of people, and they can't even maintain their lessons and code.
  - Let's look at an example, Lessons 14-18 on APIs (from free course [Designing RESTful APIs](#))
    - Python 2 instead of 3
    - HTTP requests use `httpplib2` instead of `requests`
    - Authentication is done with `oauth2client`, which [has been deprecated](#).
    - Code is not PEP 8 compliant, yet we are required to submit PEP 8 compliant code for the item catalog project.
    - Code mixes spaces and tabs.
    - JavaScript camelCase used instead of underscores for function names.
    - Doesn't even have a proper Markdown-formatted README
    - No license. How can you distribute this material to students without a license?
    - Code exists, confusingly, in two repos: [OAuth2.0](#) and [ud330](#). The code in ud330 is formatted a little better than the OAuth repo.
- **Udacity's GitHub repos are not maintained. Pull requests are ignored or not merged.** The GitHub repos are an opportunity to enrich the student experience by teaching good open source practices, but Udacity is not doing this.
- Many of the repos are not licensed.
- In addition to videos, the lessons should have Markdown-formatted **written lesson outlines**.

[\(Back to top\)](#)

## ◦ Organization

The program could benefit from reorganization.

### ◦ Part 1

- I didn't find the first project, the Python movie trailer web server (after the Python programming foundations), to be very effective. I completed the project before learning about front-end and HTTP, so I didn't really know what I was doing. The project would make more sense after part 2 instead of after part 1.
- This also wasn't a really effective way to implement a web server. The server would be more effectively implemented as a Flask app, as we did for the item catalog in part 3.

### ◦ Part 2

- Part 2 was designated as "optional," but really, all the lessons are optional, so this distinction was not useful.

- It would have been helpful to know more about the shell, Git, and GitHub before doing the first project.
- I would suggest moving the Python HTTP requests lessons (Part 02, Lessons 11-13) to Part 01 between the foundational Python programming (Part 01, Lessons 04-11) and the front-end web lessons (Part 01, lessons 12-19). This would improve the continuity of Python programming, and also teach foundations of how the web works, before moving in-depth into front-end.

### › Part 3

- Why did we go through APIs in part 3 (from course [Designing RESTful APIs](#)), when we weren't going to use them until the neighborhood map project in part 4?
- We also have more API lessons in part 4. The lessons could be consolidated.

### › Part 4

- Why is part 4 called "The Frontend" when we have already been working on front-end (HTML, CSS, portfolio website)? It should be called "Web applications."
- Part 4 should include the [ES6 - JavaScript Improved course](#) and the [Asynchronous JavaScript Requests course](#).

### › Part 5

- Part 5 should provide more assistance for setting up a server in the cloud, not just in a Vagrant virtual machine.
- Container technologies should be presented as an alternative method.

### › Incorporating free courses into the Nanodegree program

There are sometimes differences in the organization between the free and Nanodegree program-bundled versions of the courses that can be confusing.

- For example, in the Python programming foundations work, the instructor Kunal says there are four lessons, but his "lesson 1" material starts on lesson 5 based on the Full Stack Web Developer Nanodegree program course navigation.
- As another example, in part 3 on databases, the four part course by Lorenzo is actually lessons 6-9.
- Having subdirectories within each program part would help with this. For example:
  - Part 1, programming foundations
    - Programming foundations with Python
      - Lesson 1
      - Lesson 2
      - Lesson 3
      - Lesson 4
- In this way, you could keep the course numbering and organization the same for the free courses and the courses within the Nanodegree program.

### › Current FSND program organization

Here is the program organization from when I entered the program in August 2017:

## 1. Programming foundations

- [Programming Foundations with Python](#)
- Project 1: Python web server (movie trailer site)
- [Intro to HTML and CSS](#)
- [Responsive Web Design Fundamentals by Google](#)
- Project 2: Portfolio website

## 2. Developer tools

- [Shell Workshop](#)
- [Version Control with Git](#)
- [GitHub & Collaboration](#)
- [HTTP & Web Servers: Python web servers](#)

## 3. The back-end

- [Intro to Relational Databases](#)
- Project 3: SQL database logs analysis
- [Full Stack Foundations: CRUD, web servers, Flask, agile](#)
- [Authentication & Authorization: OAuth](#)
- [Designing RESTful APIs](#)
- Project 04: Flask item catalog app

## 4. The front-end

- JavaScript
- jQuery
- [Intro to AJAX](#)
- APIs
- Project 05: Neighborhood map

## 5. Servers

- Linux server deployment
- Project 06: Linux server

## › Suggested FSND program organization

Here is how I would suggest progressing through the program:

## 1. Programming foundations

- [Shell Workshop](#)
- [Version Control with Git](#)
- [GitHub & Collaboration](#)
- [Programming Foundations with Python](#)
- [HTTP & Web Servers: Python web servers](#)
- Project 1: Python web server (movie trailer site)

## 2. The front-end

- [Intro to HTML and CSS](#)
- [Responsive Web Design Fundamentals by Google](#)

- Project 2: Portfolio website

### 3. The back-end

- [Intro to Relational Databases](#)
- Project 3: SQL database logs analysis

### 4. Web applications

- [Full Stack Foundations](#): CRUD, web servers, Flask, agile
- [Authentication & Authorization: OAuth](#)
- [Designing RESTful APIs](#)
- Project 4: Flask item catalog app
- [Intro to AJAX](#)
- [Google Maps APIs](#)
- [JavaScript Design Patterns](#)
- I highly recommend additional introductory JavaScript coursework.
  - [cs50 JavaScript lecture](#)
  - [cs50 CSCI E-33a JavaScript lecture](#)
  - [Wes Bos JavaScript30](#)
  - [Syntax podcast](#)
  - [Udacity ES6 - JavaScript Improved course](#)
  - [Udacity Asynchronous JavaScript Requests course](#)
  - Mozilla Developer Network ([MDN](#))
    - [MDN JavaScript learning pathway](#)
    - [Grammar and types](#)
      - [Variable scope](#)
      - [Prototypal inheritance](#)
- Project 5: JavaScript neighborhood map

### 5. Servers

- [Configuring Linux Web Servers](#)
- Project 6: Linux server configuration and app deployment

[\(Back to top\)](#)

## Time

- The lessons took much longer than the time estimations show. For the first two parts, each lesson took me 6-12 hours over 1-2 days. I'm not sure if the number of hours is a useful metric. Maybe showing the number of lesson parts would be more useful.
- I took careful notes to effectively retain the material, which takes some extra time.
- I realized the **quizzes were frequently sticking points for me**, especially in Karl's lessons on Python HTTP and SQL. I pushed myself to get the quizzes correct without checking the solutions, because I wanted to make sure I was learning and thinking correctly. During the quizzes, I often felt like I wasn't getting it, or didn't have the information I needed to answer the questions. These sticking points sometimes took me a day or more to break through, and reduced my energy and motivation. With help from my mentor, I decided to speed up the lessons by limiting my quiz answer attempts, and pushing myself to complete the lessons in the time suggested by Udacity.

[\(Back to top\)](#)

## ^ Mentor

---

- My mentors were generally supportive and encouraging.
- I think the feedback could be more substantive and specific. I didn't find mentors helpful for specific coding challenges. The response wasn't fast enough, and the mentor usually didn't provide specific assistance with code.
- It would also help if the mentor relationship was more clearly defined. What are the boundaries? What can and can't they do for us?
- **Independence is important for success in this program.** The mentors are not involved enough to provide formative guidance, and the lesson materials don't provide enough instruction. Students that are less motivated or independent could benefit from an in-person code school instead.

[\(Back to top\)](#)

## ^ Networking

---

### ^ Forums

- Forums are a helpful way to troubleshoot code, because they are organized and searchable like Stack Overflow.
- However, the responses will be slower, which is why chat is helpful.

### ^ Slack

- When I started, the Slack workspace was just a jumble of information from people at all stages of the program. Slack was reorganized by @alex-udacity in October 2017 to have more channels, which was helpful.
- It's unclear how the changes to the forums and new in-classroom chat affect Slack.
- Someone from the forums also started up the [Udaciouspeople Slack workspace](#) for students from all Nanodegree programs. Why do we need this when the FSND already has a channel?

### ^ In-classroom chat

- It appears that Udacity expanded the previous mentor chat feature. In general, I liked this feature for communicating with my mentor. It was strange that Markdown entered in the web app showed up as plain text in the Android app.
- It's a little easier to connect with classmates, because the chat is right there, instead of in a separate tab or app.
- Markdown formatting is nice, but should be noted like it is on GitHub (Underneath the comments box, it reads "[Styling with Markdown is supported](#)").
- It needs to be more organized (@mentions, threads, notifications, and more clear date stamping in addition to time), and needs search, so that students can find previous answers like they can in the forums.

- There is also only a small field of view, so it's really difficult to read more than a few messages.
- Basically, I'm not sure what this adds beyond Slack.

## › Knowledge

- There is a new Stack Overflow-like [knowledge base](#) feature, as of 20180526. I posted an [answer](#). This may be a new version of the discussion forum.
- This should be useful in the future, because knowledge is retained, and useful answers will rise to the top.
- Frustratingly, the knowledge forum doesn't use Markdown. I have to re-format everything in HTML rich text.
- Can't sort by votes.

## › Exit survey

---

| Which parts of the Nanodegree program, if any, were most valuable in helping you feel prepared to begin a job search or seek a promotion in this field?

Code review and resume review.

| What, if anything, would help you feel more prepared to begin a job search or seek a promotion in this field?

The code needs to be updated for Python 3, JavaScript ES6, React, Docker, and other current technologies.

| Do you believe that the Nanodegree program was worth your time?

4/5

| How likely is it that you would recommend a Udacity Nanodegree program to a friend or colleague?

6/10

| Why would you recommend or not recommend this program?

- Code and lessons
  - It was good that we used Python instead of Ruby. Python is more broadly useful.
  - Some frameworks, like Flask, were current, others, like Knockout, were outdated.
  - **The code in the lessons is of inconsistent quality. We are learning, and we need to see well-structured, consistent, and current code.**
  - **Udacity's GitHub repos are not maintained. Pull requests are ignored or not merged.**
  - The lesson content could be more thorough and informative. Videos could be longer and could go more in-depth into the content.
  - The lessons are poorly organized.
- Mentors
  - **Independence is important for success in this program.** The mentors are not involved enough to provide formative guidance, and the lesson materials don't provide enough

instruction. Students that are less motivated or independent could benefit from an in-person code school instead.

- Networking
  - New Knowledge feature is useful.
  - In-classroom chat is not useful.
  - Slack is too cluttered.

## ’ Program developments

---

- They frequently change the Nanodegree programs and platform, but don't have a clear changelog.
- Around June 2018, Udacity abbreviated the FSND. There are now three projects, it costs \$999, and you only have four months. They probably thought there was too much overlap with the Front-End Nanodegree program (FEND). The FSND basically had most of the FEND. It was also a long program, so they probably wanted to shorten it to increase completion rates.
- They are also trying to maintain separation between FEND and the React Nanodegree program.
- They moved the career platform outside the Nanodegrees.

# Shell workshop

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Shell Workshop](#)

## Lesson

---

- We used the Bash shell, the most popular UNIX-style shell.
- `$ echo $COLUMNS x $LINES` prints the terminal window size. 80x24 is common.
- `ls` commands
  - *All of these are correct!*
    - `$ ls Pictures` is the most direct way to do it.
    - `$ cd Pictures ; ls` changes directory to the Pictures subdirectory, then runs ls in that directory.
    - `$ ls Pictures/../Pictures` is unnecessarily verbose, but it does work.
- `pwd` print working directory
  - `.` this directory
  - `..` parent directory
  - `~` tilde for home directory
  - *Why are shell commands so short?*
    - It makes for less typing. The Unix system was originally designed in an era when the connections between computers and terminals were very slow, so making commands really short made it much faster to use. This is true not only for the shell, but also for other parts of the Unix system, such as the C programming language.
- `ls -l` parameters and options
  - *If you wanted to list all the files whose names start with the word bear, how would you do it?*
    - The pattern `bear*` matches any file whose name starts with the word bear.
- `cd` and `mv` moving files
  - `mv` takes two arguments, separated by a space: origin destination
  - *I want to move the epub files back from Documents/Books to Documents. How can I do this? My current working directory is my home directory, and Documents is inside that directory.*
    - `$ mv 'Documents/Books'/- Documents`
    - `$ cd Documents; mv 'Books'/*.epub .`
    - `$ cd 'Documents/Books'; mv - ..`
  - By the way, when you quote something in the shell, you always use straight quotes. This is what you'll get if you type into a terminal window. However, if you copy and paste from a web page or document, you should be really careful to make sure that it hasn't accidentally been written with "curly quotes". Curly quotes will not work in the shell. Single

quotes and double quotes do slightly different things in the shell. If you're unsure which to use, go for single quotes.

- Downloading with `curl`
    - command (`curl`)
    - options, such as how to save the file. Use `-L` to follow redirects
    - object to operate on
    - Enter a shell command to download <https://tinyurl.com/zeyq9vc> and save it as the file `dictionary.txt`. Remember to use the option to follow web redirects.
      - `$ cd downloads`
      - `$ curl -o dictionary.txt -L 'https://tinyurl.com/zeyq9vc'`
    - By the way, a lot of URLs have special characters in them, such as the & sign, which have unusual meanings to the shell. That's why I'm always putting these URLs in quotes ... even though these particular examples would work without them, it's a good practice to get into.
    - Also see cs50 Lecture 06 HTTP
  - Removing files with `rm`
    - You have four files named Good File, Bad Name Good File, Bad File, Good Name Bad Face. You want to remove files 3 and 4, while leaving 1 and 2 intact. There are two commands below that will accomplish this goal. Choose them:
      - `$ rm 'Bad File' 'Good Name Bad Face'`
      - `$ rm *Bad F*`
  - Searching and pipes with `grep` and `wc`
    - How many words are there in `dictionary.txt` that match the pattern `ibo`?
      - `$ grep ibo dictionary.txt | wc -l` runs the `grep` command, searches for the string `ibo` in `dictionary.txt`, and invokes the `wc` word count function to count the number of lines with the `-l` argument.
    - \*What are grep patterns called?- Research question! You can use grep for more than just matching words. There's a specific term for the patterns that grep lets you use. Use your favorite search engine and do a little research to find out what those patterns are called.
      - *regular expressions*
      - Regular expressions are also known as regexps or regexes. There is actually a whole complex language of patterns that you can use with `grep` ... which is entirely beyond the scope of this course. And regular expressions are used in lots of other programs too, including text editors ... and this quiz. The regular expression used to match correct answers to this quiz is `^[Rr]eg.*[Ee]x.*` which means "any string that starts with `reg` and has `ex` in it, but upper- and lowercase R and E are both OK."
  - Shell and environment variables
    - `echo $PWD` refers to an environment variable. The value of `$PWD` is the current working directory, same as what you'd see if you ran the `pwd` command. Every program you run on a Unix-like system has some working directory. It usually starts out as the directory you were in when you started that program. So the `PWD` variable is an environment variable. It's not just internal to the shell.
  - Startup files
    - Usually located in `/bin`, which stands for binary.
  - Controlling the shell prompt (`$PS1`)

- The instructor customized his shell prompt. I tried it out on <http://ezprompt.net/>, but didn't really find it necessary to change it. Changes are made to the .bash\_profile.
- Aliases
  - You can set command aliases with the `alias` command.
- Shell resources
  - [The Bash Academy](#)
  - [Bash Beginners Guide](#)
  - [Bash Programming HOWTO](#)
  - [Regexpr — Learn Regular Expressions](#)
- Feedback from me
  - Quick, informative and fun. It's easy to ignore the command line, so I'm glad we spent some time strengthening our skills. Thanks!

# Git

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Version control with Git](#)

## Table of Contents

---

- [Table of Contents](#)
- [Lesson 1. Version control intro](#)
  - [Lesson 1 parts 1-3](#)
  - [Lesson 1 part 4. Mac/Linux setup](#)
- [Lesson 2. Create a Git repo](#)
- [Lesson 3. Review a repo's history](#)
- [Lesson 4. Add commits to a repo](#)
  - [Bypass The Editor With The -m Flag](#)
  - [What To Include In A Commit](#)
  - [Good Commit Messages](#)
  - [Globbing Crash Course](#)
- [Lesson 5. Tagging, branching and merging](#)
  - [Tagging](#)
  - [Branching and merging](#)
  - [Feedback on lesson 5](#)
- [Lesson 6. Undoing changes](#)
  - [Reset vs Revert](#)
  - [Relative Commit References](#)
  - [The git reset Command](#)
  - [Reset Recap](#)

## Lesson 1. Version control intro

---

### Lesson 1 parts 1-3

- The instructor Richard Kalehoff introduced version control with an analogy. If playing a board game, snapping a picture during play is like making a save point.
- Most popular VCS: Git, Subversion, Mercurial
- Centralized vs. Distributed: Git is distributed (DVCS)

- [Centralized vs. DVCS from the Atlassian Blog](#)
- Version Control in Daily Use
  - Google Docs' Revision history page is incredibly powerful! I've used it on several occasions to salvage text that I'd written at one point, erased, and then realized I actually did want to keep.
  - But for all its ability, it's not as powerful as we'd like. What's it missing? A few that I can think of are:
    - the ability to label a change
    - the ability to give a detailed explanation of why a change was made
    - the ability to move between different versions of the same document
    - the ability to undo change A, make edit B, then get back change A without affecting edit B
  - The version control tool, Git, can do all of those things - and more!!! (bet you didn't see that coming!) So have I sold you yet on the awesomeness that is Git? I hope so, cause we're about to dive into it in the next section.
- Terms: see ud123-git-keyterms.pdf

## › Lesson 1 part 4. Mac/Linux setup

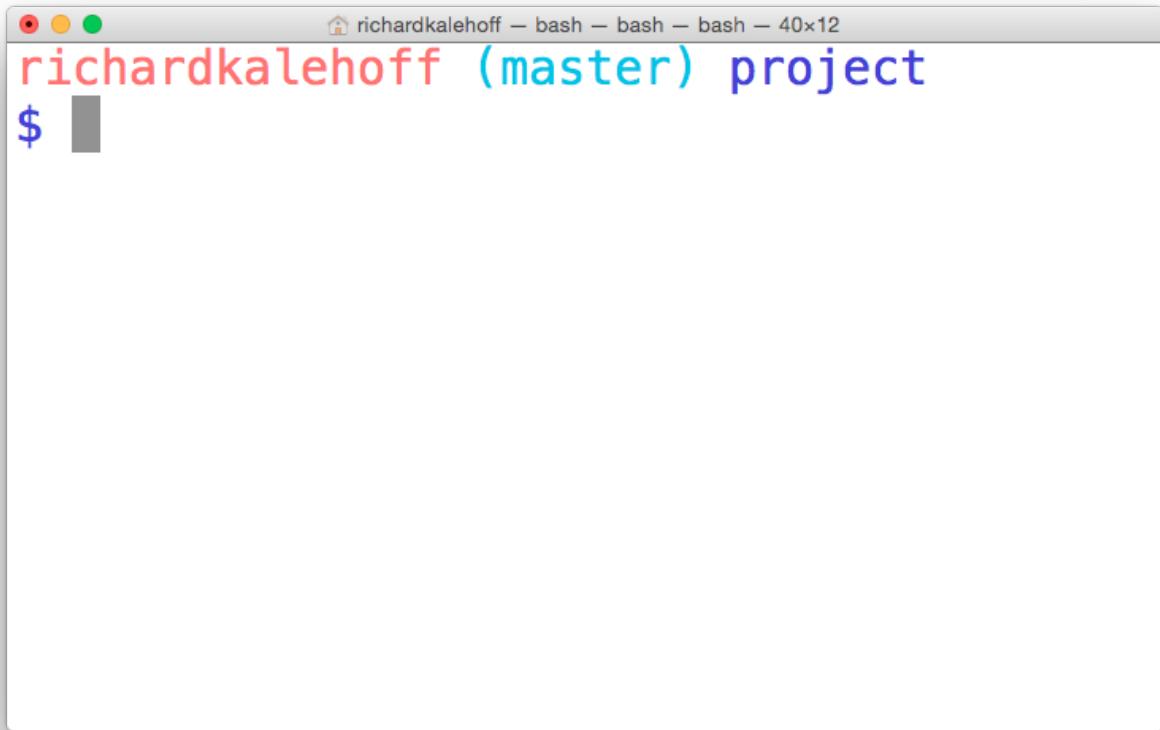
- I installed the bash prompt configuration.
- It took me a few tries to set Sublime Text 3 as the Git editor. Got it to work with `$ git config --global core.editor "'/Applications/Sublime Text.app/Contents/SharedSupport/bin/subl' -nw"`

## › Instructor notes

### ⌚ Configuring Mac's Terminal

We're about to configure the Terminal to display helpful information when in a directory that's under version control. \*This is an optional step!- You do not need to re-configure your terminal for Git to work. You can complete the entire course without reconfiguring it. However, reconfiguring the Terminal makes it significantly easier to use.

If you choose to configure your Terminal, here's what it should look like when you're finished.



The terminal application on MacOS. The terminal has been configured to display version control information.

## Configuration Steps

To configure the terminal, we'll perform the following steps:

1. download the zipped file
2. move the directory `udacity-terminal-config` to your home directory and name it `.udacity-terminal-config` (there's a dot at the front, now!)
3. move the `bash_profile` file to your home directory and name it `.bash_profile` (there's a dot at the front, now!)
  - o if you \*already- have a `.bash_profile` file in your home directory, transfer the content from the downloaded `bash_profile` to your existing `.bash_profile`

Download the zipped file in the Resources pane to get started.

## Installing Git

Git is actually installed on MacOS, but we'll be reinstalling it so that we'll have the newest version:

1. go to <https://git-scm.com/downloads>
2. download the software for Mac
3. install Git choosing all of the default options

Once everything is installed, you should be able to run `git` on the command line. If it displays the usage information, then you're good to go!

If you run into any issues, let us know in the forum.

## First Time Git Configuration

Before you can start using Git, you need to configure it. Run each of the following lines on the command line to make sure everything is set up.

```
# sets up Git with your name  
git config --global user.name "<Your-Full-Name>"  
  
# sets up Git with your email  
git config --global user.email "<your-email-address>"  
  
# makes sure that Git output is colored  
git config --global color.ui auto  
  
# displays the original state in a conflict  
git config --global merge.conflictstyle diff3  
  
git config --list
```

## Git & Code Editor

The last step of configuration is to get Git working with your code editor. Below are three of the most popular code editors. If you use a different editor, then do a quick search on Google for "associate X text editor with Git" (replace the X with the name of your code editor).

### Atom Editor Setup

```
git config --global core.editor "atom --wait"
```

### Sublime Text Setup

```
git config --global core.editor "'/Applications/Sublime Text  
2.app/Contents/SharedSupport/bin/subl' -n -w"
```

### VSCode Setup

```
git config --global core.editor "code --wait"
```

If you have any questions, post them on the forum.

## ⌚ Everything Is All Set Up

### Task List

- I've installed Git
- I've configured Git with my username

- I've configured Git with my email
- I've configured Git to use my chosen editor

## 🔗 Supporting Materials

[udacity-terminal-config.zip](#)

[\(Back to TOC\)](#)

---

## Lesson 2. Create a Git repo

- Flew through this lesson.
- `git init`
  - Further Research
    - [Git Internals - Plumbing and Porcelain](#) (advanced - bookmark this and check it out later)
    - [Customizing Git - Git Hooks](#)
- `git clone`
  - Successfully cloned the blog repo with `git clone https://github.com/udacity/course-git-blog-project blog-project`
- `git status`

[\(Back to TOC\)](#)

---

## Lesson 3. Review a repo's history

- This lesson was helpful. I hadn't worked with `git log` or `git show` much before this.
- `git log`
  - You can search the log by entering a string after `git log`. To search for an entry with SHA containing 8aa6668, type `git log 8aa6668`.
  - Searching by commit message:
    - | Use `git log` to find the commit that has the message Set article timestamp color.
    - | Which commit belongs to that SHA? Provide the first 7 characters of the SHA.
  - I got stuck in a secondary prompt when searching `git log "Set article timestamp color"`. After a while, I realized bash was looking for a closing double quotation character. I just entered `"` and my PS1 command prompt was restored.
  - I had to search the Git documentation to figure this out. I did a text search for "message" on the [git-log page](#). I found the `--grep=` modifier, enclosed the string in single quotes, and... success! `git log --grep='Set article timestamp color'`. First seven characters of SHA: `5de135a`.
- `git log --oneline` is more concise.
- `git log --stat` shows changes. Combine with a search for SHA with `git log 8d3ea36 --stat`

- Add `-w` to ignore whitespace changes
- Combine with `git log -p --stat`
- `git show`

[\(Back to TOC\)](#)

---

## ’ Lesson 4. Add commits to a repo

---

### ’ Bypass The Editor With The `-m` Flag

TIP: If the commit message you're writing is short and you don't want to wait for your code editor to open up to type it out, you can pass your message directly on the command line with the `-m` flag: `$ git commit -m "Initial commit"`

### ’ What To Include In A Commit

I've been telling you what files to create, giving you the content to include, and telling you when you should make commits. But when you're on your own, how do you know what you should include in a commit and when/how often you should make commits?

The goal is that each commit has a single focus. Each commit should record a single-unit change. Now this can be a bit subjective (which is totally fine), but each commit should make a change to just one aspect of the project.

Now this isn't limiting the number of lines of code that are added/removed or the number of files that are added/removed/modified. Let's say you want to change your sidebar to add a new image. You'll probably:

- add a new image to the project files
- alter the HTML
- add/modify CSS to incorporate the new image

A commit that records all of these changes would be totally fine!

Conversely, a commit shouldn't include unrelated changes - changes to the sidebar and rewording content in the footer. These two aren't related to each other and shouldn't be included in the same commit. Work on one change first, commit that, and then change the second one. That way, if it turns out that one change had a bug and you have to undo it, you don't have to undo the other change too.

The best way that I've found to think about what should be in a commit is to think, "What if all changes introduced in this commit were erased?". If a commit were erased, it should only remove one thing.

Don't worry, commits don't get randomly erased.

In a later lesson, we'll look at using Git to undo changes made in commits and how to manually, carefully remove the last commit that was made.

## Good Commit Messages

Let's take a quick stroll down Stickler Lane and ask the question:

How do I write a good commit message? And why should I care?

These are fantastic questions! I can't stress enough how important it is to spend some time writing a good commit message.

Now, what makes a "good" commit message? That's a great question and has been written about a number of times. Here are some important things to think about when crafting a good commit message:

Do

- do keep the message short (less than 60-ish characters)
- do explain what the commit does (not how or why!)

Do not

- do not explain why the changes are made (more on this below)
- do not explain how the changes are made (that's what git log -p is for!)
- do not use the word "and"
  - if you have to use "and", your commit message is probably doing too many changes
    - break the changes into separate commits
  - e.g. "make the background color pink and increase the size of the sidebar"

The best way that I've found to come up with a commit message is to finish this phrase, "This commit will...". However, you finish that phrase, use that as your commit message.

Above all, be consistent in how you write your commit messages!

[Udacity Git Commit Message Style Guide](#)

## Globbing Crash Course

Let's say that you add 50 images to your project, but want Git to ignore all of them. Does this mean you have to list each and every filename in the .gitignore file? Oh gosh no, that would be crazy! Instead, you can use a concept called globbing.

Globbing lets you use special characters to match patterns/characters. In the .gitignore file, you can use the following:

```
blank lines can be used for spacing
# - marks line as a comment
- - matches 0 or more characters
? - matches 1 character
[abc] - matches a, b, or c
** - matches nested directories - a/**/z matches
a/z
```

```
a/b/z  
a/b/c/z
```

So if all of the 50 images are JPEG images in the "samples" folder, we could add the following line to `.gitignore` to have Git ignore all 50 images.

```
samples/*.jpg
```

## Lesson 5. Tagging, branching and merging

This was another helpful lesson.

### Tagging

- Tags are just what you would think, they tag a commit for future reference.
- We used `$ git tag -a v1.0` on the same `new-git-repository`. The `-a` indicates annotated.
- I was able to see the tag simply with `git log --oneline`, even though the tutorial said I shouldn't be able to. There was an addendum in the lesson: `--decorate` Flag Changes in Git 2.13. The lesson should just be updated instead.

```
br3ndonland (master *) new-git-project  
$ git log --oneline  
2e104e2 (HEAD -> master, tag: v1.0) Add header to blog  
63afe9d Initial commit
```

- If you make a mistake, just delete the tag with `$ git tag -d v1.0` and make a new one

### Branching and merging

- Running `$ git log --oneline --graph --all` will show the branching.
- Switch branches with `git checkout`.
- Switch and create a branch in one step with `git checkout -b <new-branch-name> master`.
- Merge conflicts: we basically created a branch ahead of `master`, modified a line of code and committed, checked out back to `master`, modified the same line of code again and committed, checked out back to `master` again, and attempted to merge. It took me a couple of tries, but I got it to work.
- Final status

```
$ git log --oneline --graph --all  
- d8ede3c (HEAD -> master) Merge branch 'heading-mod1'
```

```
\ 
| - 286d83b (heading-mod1) Modify header again to demo merge conflict
- | 98ec7d0 (heading-mod2) Modify header once again to demo merge conflict
|/
- 6cf2cb3 (heading-crusader) Set page heading to Crusade
- b329862 (heading-update) Set page heading to Quest
- 6895eb1 Merge branch 'sidebar'
|\
| - 254f7e4 (sidebar) Add text to sidebar
| - f895dd3 Add sidebar.
- | e385b99 (footer) Add links to social media
- | 4f7968c Improve site heading for SEO
- | d6a086f Set background color for page
|/
- 27f3cf6 Update for Udacity branching lesson fsnd02_06
- 2e104e2 (tag: v1.0) Add header to blog
- 63afe9d Initial commit
```

## › Feedback on lesson 5

I was able to see the tag simply with `git log --oneline`, even though the tutorial said I shouldn't be able to. There was an addendum in the lesson: `--decorate` Flag Changes in Git 2.13. The lesson should just be updated instead.

---

## › Lesson 6. Undoing changes

Major commands used:

- `git commit --amend`
- `git revert`
- `git reset`

## › Reset vs Revert

At first glance, resetting might seem coincidentally close to reverting, but they are actually quite different. Reverting creates a new commit that reverts or undos a previous commit. Resetting, on the other hand, erases commits!

### Resetting Is Dangerous

You've got to be careful with Git's resetting capabilities. This is one of the few commands that lets you erase commits from the repository. If a commit is no longer in the repository, then its content is gone.

To alleviate the stress a bit, Git does keep track of everything for about 30 days before it completely erases anything. To access this content, you'll need to use the `git reflog` command. Check out these links for more info:

- [git-reflog](#)

- Rewriting History
- reflog, your safety net

## Relative Commit References

You already know that you can reference commits by their SHA, by tags, branches, and the special HEAD pointer. Sometimes that's not enough, though. There will be times when you'll want to reference a commit relative to another commit. For example, there will be times where you'll want to tell Git about the commit that's one before the current commit...or two before the current commit. There are special characters called "Ancestry References" that we can use to tell Git about these relative references. Those characters are:

```
^ - indicates the parent commit  
~ - indicates the first parent commit
```

Here's how we can refer to previous commits:

- the parent commit – the following indicate the parent commit of the current commit

```
HEAD^  
HEAD~  
HEAD~1
```

- the grandparent commit – the following indicate the grandparent commit of the current commit

```
HEAD^^  
HEAD~2
```

- the great-grandparent commit – the following indicate the great-grandparent commit of the current commit

```
HEAD^^^  
HEAD~3
```

The main difference between the ^ and the ~ is when a commit is created from a merge. A merge commit has two parents. With a merge commit, the ^ reference is used to indicate the first parent of the commit while ^2 indicates the second parent. The first parent is the branch you were on when you ran git merge while the second parent is the branch that was merged in.

It's easier if we look at an example. This what my git log currently shows:

```
- 9ec05ca (HEAD -> master) Revert "Set page heading to "Quests & Crusades""  
- db7e87a Set page heading to "Quests & Crusades"  
- 796ddb0 Merge branch 'heading-update'  
|\  
| - 4c9749e (heading-update) Set page heading to "Crusade"
```

```
- | 0c5975a Set page heading to "Quest"
|/
|- 1a56a81 Merge branch 'sidebar'
|\
| - f69811c (sidebar) Update sidebar with favorite movie
| - e6c65a6 Add new sidebar content
- | e014d91 (footer) Add links to social media
- | 209752a Improve site heading for SEO
- | 3772ab1 Set background color for page
|/
- 5bfe5e7 Add starting HTML structure
- 6fa5f34 Add .gitignore file
- a879849 Add header to blog
- 94de470 Initial commit
```

Let's look at how we'd refer to some of the previous commits. Since HEAD points to the 9ec05ca commt:

```
HEAD^ is the db7e87a commit
HEAD~1 is also the db7e87a commit
HEAD^^ is the 796ddb0 commit
HEAD~2 is also the 796ddb0 commit
HEAD^^^ is the 0c5975a commit
HEAD~3 is also the 0c5975a commit
HEAD^^^2 is the 4c9749e commit (this is the grandparent's (HEAD^^) second parent (^2))
```

Quiz: which commit is referenced by HEAD~4^2?

f69811c

HEAD~4 references the fourth parent commit of the current one and then the ^2 tells us that it's the second parent of the merge commit (the one that got merged in!).

## ²The git reset Command

The git reset command is used to reset (erase) commits:

```
$ git reset <reference-to-commit>
```

It can be used to:

- move the HEAD and current branch pointer to the referenced commit
- erase commits
- move committed changes to the staging index
- unstage committed changes

### ⌚ Git Reset's Flags

The way that Git determines if it erases, stages previously committed changes, or unstages previously committed changes is by the flag that's used. The flags are:

- `--mixed`
- `--soft`
- `--hard`

It's easier to understand how they work with a little [animation](#).

## 💡 Backup Branch 💡

Remember that using the `git reset` command will erase commits from the current branch. So if you want to follow along with all the resetting stuff that's coming up, you'll need to create a branch on the current commit that you can use as a backup.

Before I do any resetting, I usually create a backup branch on the most-recent commit so that I can get back to the commits if I make a mistake:

```
$ git branch backup
```

## ⚡ Reset Recap

To recap, the `git reset` command is used to erase commits:

```
$ git reset <reference-to-commit>
```

It can be used to:

- move the HEAD and current branch pointer to the referenced commit
- erase commits with the `--hard` flag
- moves committed changes to the staging index with the `--soft` flag
- unstages committed changes `--mixed` flag

Typically, ancestry references are used to indicate previous commits. The ancestry references are:

- `^` – indicates the parent commit
- `~` – indicates the first parent commit

Further Research

- [git-reset](#) from Git docs
- [Reset Demystified](#) from Git Blog
- [Ancestry References](#) from Git Book

# GitHub

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[GitHub & Collaboration](#)

## Table of Contents

---

- [Table of Contents](#)
- [Lesson 1. Intro to GitHub Remotes](#)
- [Lesson 2. Forking](#)
  - [Summary](#)
  - [What To Work On](#)
  - [CONTRIBUTING.md File](#)
  - [Topic Branches](#)
  - [Best Practices](#)
  - [Recap](#)
- [Lesson 3. Staying in sync with remote repositories by pulling and rebasing](#)
  - [Intro](#)
  - [Staying in sync with the source project](#)
  - [Manage an active PR](#)
  - [Squash commits with rebase](#)
  - [Course wrap-up](#)

## Lesson 1. Intro to GitHub Remotes

---

- Made a test repo `github-sample-repo` and created the three intro files in one step with `$ touch README.md index.html app.css`.
- Configured repo to push with ssh: `$ git remote set-url origin git@github.com:br3ndonland/udacity-travel-plans.git`
- Pushed with `$ git push origin master`.  
One very important thing to know is that this `origin/master` tracking branch is not a live representation of where the branch exists on the remote repository. If a change is made to the remote repository not by us but by someone else, the `origin/master` tracking branch in our local repository will not move. We have to tell it to go check for any updates and then it will move. We'll look at how to do this in the next section.

- Made changes on github, then brought the changes to the local repo with `$ git pull origin master`.
  - Pull vs. fetch : `fetch` just retrieves remote changes, but does not directly merge. It's like half of `pull`. The other half of `pull` is a merge with the local repo.
- 

## ’Lesson 2. Forking

### ’Summary

- We forked and cloned repos.
- We cloned the [Google Chrome lighthouse](#) project.
  - `$ git shortlog -s -n` can be used to display the commit authors.
  - `$ git log --author="Paul Lewis"` can be used to display commits by specific authors.
  - `$ git log --grep "css bug"` can be used to search for the commits that mention a css bug.

### ’What To Work On

Let's say you're using some third-party library to help you build a project. What happens if, while you're using this third-party library, you come across a bug or a misspelling? You have the skill to be able to fix it, but you don't have direct access to make modifications to the original library. Well that's not a problem because you know that forking another developer's repository copies it to your account and gives you full access to `git pull` and `git push` to it!

But what are you supposed to do now that you've got full access to a duplicate of the other developer's project. We'll look at this in the next section but if you have forked a project and you have code in your fork that's not in the original project, you can get code into the original project by sending the original project's maintainer a request to include your code changes. This request is known as a "Pull Requests". Again, we'll look at sending and working with pull requests in the next lesson.

So you know it's possible to get your code in the original project and you know you want to help and fix this spelling/code mistake. So you got something to work on! But how do you go about actually contributing to the project in the way that the original project maintainer will be happy with and will end up actually incorporating your changes?

The first thing you should always look for in a project is a file with the name `CONTRIBUTING.md`.

### ’CONTRIBUTING.md File

The name of the `CONTRIBUTING.md` file is typically written in all caps so that it's easily seen. As you could probably tell by its name, this file lists out the information you should follow to contribute to the project. You should look for this file before you start doing development work of any kind.

Let's look at the lighthouse project's contributing file:

You can see that the top line of the file says:

We'd love your help! This doc covers how to become a contributor and submit code to the project.

There are two main sections to this file:

- the "For Contributors" section
- the "For Maintainers" section

Each one of these sections has subsections of its own to instruct readers on how to contribute to and work with this project.

Let's take a look at the section on signing the contributors license. You can see that to be able to contribute to this project you need to sign Google's Contributor License Agreement.

## Topic Branches

The best way to organize the set of commits/changes you want to contribute back to the project is to put them all on a topic branch. Now what do I mean by a topic branch? Unlike the master branch which is the default branch that holds all of the commits for your entire project, a topic branch host commits for just a single concept or single area of change.

For example if there is a problem with the login form for logging into the website, then a branch name to address this specific issue could be called:

- login
- login-bug
- signup-bug
- login-form-bug

etc.

There are plenty of names that can be used for a topic branch's name. You just want to use a clear descriptive name for the branch so that if, for example, you list out all of the branches you can immediately see what changes are supposed to be in a branch just by its name.

## Best Practices

### Write Descriptive Commit Messages

While we're talking about naming branches clearly that describe what changes the branch contains, I need to throw in another reminder about how critical it is to write clear, descriptive, commit messages. The more descriptive your branch name and commit messages are the more likely it is that the project's maintainer will not have to ask you questions about the purpose of your code or have dig into the code themselves. The less work the maintainer has to do, the faster they'll include your changes into the project.

### Create Small, Focused Commits

This has been stressed numerous times before but make sure when you are committing changes to the project that you make smaller commits. Don't make massive commits that record 10+ file changes and changes to hundreds of lines of code. You want to make smaller, more frequent commits that record just a handful of file changes with a smaller number of line changes.

Think about it this way: if the developer does not like a portion of the changes you're adding to a massive commit, there's no way for them to say, "I like commit A, but just not the part where you change the sidebar's background color." A commit can't be broken down into smaller chunks, so make sure your commits are in small enough chunks and that each commit is focused on altering just one thing. This way the maintainer can say I like commits A, B, C, D, and F but not commit E.

## ⌚ Update The README

And lastly if any of the code changes that you're adding drastically changes the project you should update the README file to instruct others about this change.

## ՚ Recap

Before you start doing any work, make sure to look for the project's CONTRIBUTING.md file.

Next, it's a good idea to look at the GitHub issues for the project

- look at the existing issues to see if one is similar to the change you want to contribute
- if necessary create a new issue
- communicate the changes you'd like to make to the project maintainer in the issue

When you start developing, commit all of your work on a topic branch:

- do not work on the master branch
- make sure to give the topic branch clear, descriptive name

As a general best practice for writing commits:

- make frequent, smaller commits
- use clear and descriptive commit messages
- update the README file, if necessary

---

## ՚ Lesson 3. Staying in sync with remote repositories by pulling and rebasing

---

## ՚ Intro

**Create pull requests from topic branches.**

- Forked the course-collaboration-travel-plans repo, then cloned it to desktop.

- Created a topic branch: `$ git checkout -b include-brendons-destinations`
- Made changes and committed to topic branch: `$ git commit -m "Add Brendon's destinations"`
- Configured repo for ssh: `$ git remote set-url origin git@github.com:br3ndonland/course-collaboration-travel-plans.git`
- Pushed the topic branch to GitHub: `$ git push origin include-brendons-destinations`
- At first, I was a little confused as to why you would need to branch if you have already forked. It's for practical reasons. When submitting a pull request, the pull request name is auto-populated from the branch name.
- On the "Open a pull request" screen:
  - the `base fork` is the original repo that the fork was created from.
  - The `head fork` is the branch in your fork that is currently set to `HEAD` on GitHub.
  - When I first got to the pull request screen, it showed `base fork: udacity/course-collaboration-travel-plans`. If I had submitted the pull request this way, it would have gone to Udacity. Instead, I switched to `base fork: br3ndonland/course-collaboration-travel-plans`. This basically becomes a merge. Udacity didn't clearly instruct the students to fork before submitting pull requests, which is why the [repo has thousands of pull requests](#).

## › Instructor notes

A pull request is a request for the source repository to pull in your commits and merge them with their project. To create a pull request, a couple of things need to happen:

- you must fork the source repository
- clone your fork down to your machine
- make some commits (ideally on a topic branch!)
- push the commits back to your fork
- create a new pull request and choose the branch that has your new commits

As you can see, it's actually not too difficult to create a pull request. When I was first learning Git, GitHub, and how to collaborate, I was extremely nervous about making commits, and working with remote repos, but especially submitting a pull request to another developer's project! As long as you following the steps we covered in the previous section on:

- reviewing the project's CONTRIBUTING.md file
- checking out the project's existing issues
- talking with the project maintainer

...your pull request is sure to be included!

## › Staying in sync with the source project

### › Stars and popularity

Starring can be a useful feature to help you keep track of repositories you're interested in. But stars have also turned into a means of measuring a repo's popularity.

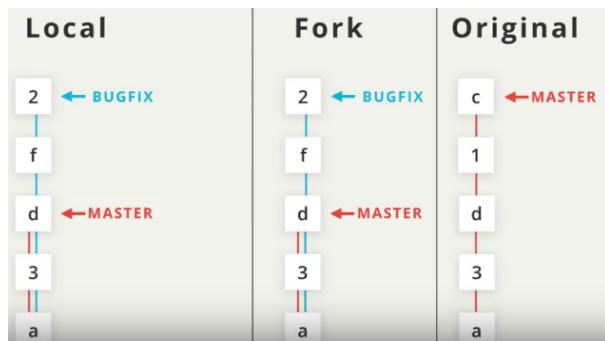
If you'd rather not increase a repository's stars, then check out "watching" a repository.

## › Watching

Watching enables notification of changes to a repo.

## › Including Upstream Changes

We have a clone of a fork. How do we stay in sync with the original master?



- We can either pull from the master into the origin fork and then to the clone, or just pull directly from the origin master and then push to our fork.
- Richard named the master `upstream`. It could be any name. The name `upstream` was already provided by Udacity:

```
$ git remote -v
origin  git@github.com:br3ndonland/course-collaboration-travel-plans.git (fetch)
origin  git@github.com:br3ndonland/course-collaboration-travel-plans.git (push)
upstream  https://github.com/udacity/course-collaboration-travel-plans.git (fetch)
upstream  https://github.com/udacity/course-collaboration-travel-plans.git (push)
```

- Note that `origin` points to the fork.
- Using `git fetch upstream master` pulled in the changes from the master branch on the upstream remote repository. What single command would we use if we want to fetch the upstream/master changes and merge them into the master branch?
  - `git pull upstream master`
  - Remember that `pull` does both a `fetch` and `merge`.

## › Manage an active PR

The project maintainer may decide not to accept your changes right away. They might request you to make some additional changes to your code before accepting your request and merging in your changes. Most likely they will communicate their desired changes through the conversation on the pull requests page.

One thing that I've grown to love about both the Git command line tool and the GitHub interface is how helpful they are with recommendations on what to do next. Near the bottom of the comments, there's a suggestion by GitHub that tells us how to add more commit; we need to add them to the same branch and push to my fork:

Add more commits by pushing to the include-richards-destinations branch on richardkalehoff/course-collaboration-travel-plan.

When you submit a pull request, remember that you're asking another developer to add your code changes to their project. If they ask you to make some minor (even major!) changes to your pull request, that doesn't mean they're rejecting your work! It just means that they would like the code added to their project in a certain way.

The CONTRIBUTING.md file should be used to list out all information that the project's maintainer wants, so make sure to follow the information there. But that doesn't mean there might be times where the project's maintainer will ask you to do a few additional things.

So what should you do? Well, if you want your pull request to be accepted, then you make the change! Remember that the tab in GitHub is called the "Conversation" tab. So feel free to communicate back and forth with the project's maintainer to clarify exactly what they want you to do.

It also wouldn't hurt to thank them for taking the time to look over your pull request. Most of the developers that are working on open source projects are doing it unpaid. So remember to:

- be kind - the project's maintainer is a regular person just like you
- be patient - they will respond as soon as they are able

So Lam is asking that I combine my changes together before she'll merge in my pull request. Combining commits together is a process called squashing. Let's look at how to do that!

Recap

As simple as it may seem, working on an active pull request is mostly about communication!

If the project's maintainer is requesting changes to the pull request, then:

- make any necessary commits on the same branch in your local repository that your pull request is based on
- push the branch to the your fork of the source repository

The commits will then show up on the pull request page.

## ² Squash commits with rebase

```
git rebase -i <base>
```

- To reference commits, `HEAD~3`, SHA, branch name, or tag name can be used.
- Richard recommends making a backup branch before rebasing.

## ² Course wrap-up

- <http://up-for-grabs.net/>
- <http://www.firsttimersonly.com/>
- [first-timers-only label on GitHub](#)
- ["first timers only" blog post](#)
- try tackling some Git and GitHub challenges with the [Git-it app](#)

Wanna see what a developer's very first pull request is? Check out at <http://firstpr.me/>

# Lessons 01-03. Introduction

---

Udacity Full Stack Web Developer Nanodegree program

Part 01. Programming fundamentals and the web

Brendon Smith

br3ndonland

## Lesson 01. Welcome

---

### Intro

---

Mike Wales, curriculum manager

Miriam Swords Kalk, nanodegree student experience

### Readiness assessment

---

*When the going gets tough, what motivates you?*

*Online learning is a journey and we're here to help. Tell us more about how you stay motivated to work through challenges.*

Personal improvement. Growth keeps me going.

Long-term goals. I want to improve science with new technology tools. I need to develop the skills to make this happen.

*-> I have written about this before, but should consolidate my writings on motivation. See Life as a scientist.docx about remembering your purpose etc.*

The readiness assessment suggested the intro to programming nanodegree, but I will keep going. I have an advanced educational background, experience, and I don't have time to complete both the intro to programming and full-stack web developer nanodegrees before I get a job. If I get stuck, I will go through a period of intense focused learning until I have the necessary competency. I will also complement my Udacity training with in-person events whenever possible.

Feedback to Udacity:

Thank you for this introduction, and for offering the readiness assessment! I appreciate your thoughtfulness and consideration.

## Lesson 02: career services available to you

---

This will be very helpful. I can refine my resume and search for jobs while completing my degree.

I have updated my profile with current information.

## Lesson 03: nanodegree orientation

---

### 01. welcome to orientation

---

### 02. projects and progress

---

- Udacity focuses on learning by doing.
- The program is built around projects. Reviewing all the course material isn't a requirement for graduation.

### 03. connecting with your community

---

- I joined the Slack channel.
- Hi everyone! I'm Brendon Smith. I have a PhD in nutrition and have been a postdoc researcher and lab manager at Harvard for two years now. I want to build new technology tools to make science more efficient and reproducible. I have experience with R, Python, Git, and front-end web development. Some of my friends at MIT told me about Udacity. I'm looking forward to learning here with all of you!

### 04. support

---

### 05. project submission

---

### 06. integrity and mindset

---

The Honor Code:

*I hereby confirm that all my project submissions consist of my own work. Accordingly, I will document and cite the origins of any part(s) of my project submissions that were taken from websites, books, forums, GitHub repositories, or any other source and explain why I used them for any part of my submission. I understand that I may be asked to explain my work in a video call with a Udacity Mentor before my Nanodegree graduation is conferred.*

### 07. finding time to study

---

- They recommend  $\geq 10$  hrs / week in  $\geq 2$  blocks. I set it aside, though I will basically be working full-time right now.

## ’08. final tips

---

- "Be relentless in searching for an answer on your own."
- Be an active member of your community. Help other students.

# Lesson 04. Python intro

---

Udacity Full Stack Web Developer Nanodegree program

Part 01. Programming fundamentals and the web

[Programming foundations with Python](<https://www.udacity.com/course/programming-foundations-with-python--ud036>)

Brendon Smith

br3ndonland

## ’ 01. What will we create?

---

Kunal, Udacity instructor

1. project take a break
2. project profanity editor
3. project movie database

*I stopped here for a bit to review basic python and drill loops and if statements. See [Appendix](#) below for some examples.*

## ’ 02. Quiz: comfort level

---

- 

Kunal said that in his computer science classes, students often started out interested, but became frustrated after the difficulty level skyrocketed.

- I chose 3, somewhat comfortable, with writing computer programs.

## ’ 03. What

---

should I know?

*(viewed source HTML on the website, copied here. Remember [Markdown can interpret HTML](#)).*

Need to brush up? Check out these helpful tutorials below:

## ’ If Statements

1. [Tutorialspoint: If-Else Blocks](#)
2. [CS 101: If Statements](#)

## ’ While Loops

1. [Tutorialspoint: Loops](#)
2. [CS 101: While Loops](#)

## ’ Functions

1. [Loyola University: Defining Functions](#)
2. [CS 101: Functions](#)

Have questions? Head to the [forums](#) for discussion with the Udacity Community.

## ’ 04. Quiz: test for loops

---

They had a dance video shaking shoulders to the left and right three times.

I didn't really agree with the answer, so I re-coded it myself:

```
# Python for loops quiz
# I would actually code it like this:

num_1 = 1

while(num_1 <= 3):
    print('shake shoulders to the left')
    print('shake shoulders to the right')
    # omission of the line below may result in an infinite loop.
    num_1 = num_1 + 1
print('Strike a pose')
```

## ’ 05. Quiz: Test for If Statements

---

I drilled if-else statements for a while (see conditionals section below). I understood this immediately! Very happy with my syntax progress.

```
def determine_direction(message):
    if message == 'TAKE THE ROAD LESS TRAVELED':
        print('Turn left')
    elif message == 'COMFORT IS DIVINE':
        print('Turn right')
    else:
        print('Stop here')

determine_direction('TAKE THE ROAD LESS TRAVELED')
```

## ’ 06. What Will We Learn?

---

Somewhat familiar ideas: lesson 0 and 1

- Functions like `print('Hello, World!')`

New ideas: lesson 2 and 3

- Classes
- Object- oriented programming

## Appendix

---

### Selection of jupyter notebook

- This nanodegree will use Python.
- Jupyter allows both markdown and inline code chunks in the same notebook.
- It is easier to write Markdown in Sublime, but jupyter is a better choice because of the coding functionality.
- The ability to run individual cells is useful for both code and markdown. Rendering Markdown is quicker than knitting an entire document with Rstudio, or using Markdown Preview with Sublime.
- Rstudio has some similar features, but jupyter has support for more coding languages.
- 

Jupyter is also useful for research because of checkpoint/version control and collaboration features. It is a good idea to get used to jupyter so I can use it for collaborative, reproducible research in the future.

See [GitHub](#) for more info on my computer setup.

### Resources

#### Python

- [Python 3 documentation](#)
- [Python PEP8 style guide](#)
- [Python PEP8 style checker](#)
- [Python Central](#)
- 

[jupyter/iPython documentation](#)

#### Markdown

- [Markdown Guide](#)
- 

[CommonMark](#)

- [GitHub-Flavored Markdown (GFM) Basic writing and formatting syntax] (<https://help.github.com/articles/basic-writing-and-formatting-syntax/>)
- [Page from the original creator of Markdown](#)
- 

[Dillinger](#), a helpful online Markdown editor with live preview.

## › User input

I worked on this later, out of interest.

```
name = input("Please type your name: ")
# Old syntax
print('Yo yo yo', name + '!', 'My new album just dropped!')
# Python 3 string formatting
print("Yo yo yo {}! My new album just dropped!".format(name))
```

## › Loops

```
# looping examples
## simple example from GA_python101
### this example prints each letter of the reference string `phrase` in turn, as uppercase

phrase = 'Happy Birthday!' # establish a reference

for letter in phrase: # perform the following for each letter in the phrase
    print(letter.upper()) # print the letter as uppercase
```

```
◀ ━━━━━━ ▶
integers_2 = range(1, 16)

for x in integers_2:
    print(x)
```

```
# looping examples
## example from Harvard Computefest Intro to Python 2017 python-cf17-02-language.ipynb

colors = 'red green blue yellow white black pink'.split()
print(colors)
for c in colors:
    print('color is', c)
```

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

```

num_1 = 1

while(num_1 <= 3):
    print('shake shoulders to the left')
    print('shake shoulders to the right')
    num_1 = num_1 + 1 # omission of this line may result in an infinite loop.
else:
    print('Strike a pose.')

```

To stop an infinite loop in jupyter, press `Esc` to enter command mode, and then `i,i` to interrupt the kernel.

## Conditions

```

# Conditionals (if statements)

## simple example from GA_python101

secret = 55
value = 55

if secret == value:
    print('they are equal')
else:
    print('they are not equal')

```

```

## more complicated example
#####
#### not sure how to do this with a range, like:
#### integers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
#### integers = range(1, 15)
integer = 15

if integer == 1:
    print('this number is one')
elif integer == 10:
    print('this number is ten')
elif integer == 15:
    print('Finally! This number is fifteen!')
else:
    print('End of line.')

```

## Functions

```

# Functions

## factorial example from [repl.it](https://repl.it)
### try running it with 0 and 7 as the inputs in the last line.

```

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
factorial(7)
```

```
# Functions
## example from Harvard Computefest Intro to Python 2017
## python-cf17-02-language.ipynb

def say_hi(me):
    print("Yo {}".format(me))

me = "br3ndonland"
say_hi(me)

def f(x):
    "A function f of x that calculates a second order function for y and also z"
    y = 3*x + 2*x**2
    z = 5*x**2 + 4*y**(0.5)
    return y, z # tuple packing

f(4)
```

```
# Functions
## example from Harvard Computefest Intro to Python 2017
## python-cf17-02-language.ipynb
## Function exercise
## * Create a list of 10 numbers, your choice.
## * Use a `for` loop to iterate over it.
## * Think about what you'd need to do to calculate the average
## * Remember the `len()` function.

number_list = [1, -594, 90, 45, 7, 3, 59, 9087, 4, 2]

def myaverage(numlist):
    'calculate the mean from an iterable of numbers'
    print('given numlist:', numlist)
    total = 0
    for x in numlist:
        total += x
    mean = total / float(len(numlist))
    print("Got a mean of", mean)
    return mean
average = myaverage(number_list)
average
```

```
# Functions
## that was a very time-consuming way of calculating an average.
```

```
## (in human time, not computer time)
## how about this
import statistics
statistics.mean(number_list)
```

```
# numpy is also faster for humans, but imports a larger package
# so may take longer for the computer.
import numpy

numpy.mean(number_list)
```

# Lesson 05. Functions: Break timer and secret message

---

Udacity Full Stack Web Developer Nanodegree program

Part 01. Programming fundamentals and the web [Programming foundations with Python]  
(<https://www.udacity.com/course/programming-foundations-with-python--ud036>)

Brendon Smith

br3ndonland

## ’01. Course Map

---

*note that Kunal refers to this lesson in the video as lesson 1, but it is listed in the Udacity course navigation as lesson 5 because there was additional intro material before it.*

Lesson 0. Introduction

Lesson

1. Functions

Lesson 2. Using Classes

Lesson 3. Making Classes

## ’02. Take a Break (story)

---

We want to remind people to take breaks while they are spending long hours on the computer!

This take a break project is a nice example of how to build a computer program by breaking the objective down into a series of actionable steps, then testing each step.

## ’03. Take a Break

---

(output)

## ’04. How would you do this?

---

If you were writing the "Take a Break" program, what steps would you take to get to the output?

1. Track the time
  - import a module that follows the computer's clock
2. After two hours: open browser to play song
  - write a command to open browser, and include link to YouTube video

3. Repeat in two more hours

## ’05. Planning the break

---

Post answer on [discussion forum](#).

Welcome to Udacity Discussion Forum — **thanks for contributing!**

- Does your reply improve the conversation in some way?
- Be kind to your fellow community members.
- Constructive criticism is welcome, but criticize *ideas*, not people.

For more, [see our community guidelines](#). This panel will only appear for your first 2 posts.

## ’06. one way of doing this

---

Create a loop to repeat three times: \* Wait two hours

- Open browser

## ’07. Launching python

---

Already rocking jupyter

## ’08. Quiz: what is the error?

---

We will use the [ `webbrowser` function](<https://docs.python.org/3/library/webbrowser.html?highlight=webbrowser#module-webbrowser>) in the standard library.

I decided to play the song "Hours" by Tycho. Very appropriate for this example.

*I tried the code before playing the instructional video:*

got it to work on the command line first:

```
$  
python -m webbrowser -t "https://www.youtube.com/watch?v=IuG06WHcruU"
```

```
# have to import the webbrowser module for jupyter  
import webbrowser  
webbrowser.open_new_tab("https://www.youtube.com/watch?v=IuG06WHcruU")
```

## ’09. Squashing the bug

---

Kunal gets the same error in his instructional video. Already figured it out!

## 10. Making the program wait

```
import time
import webbrowser
time.sleep(7)
# try 7 seconds.
# For two hours, time.sleep(2*60*60)
webbrowser.open_new_tab("https://www.youtube.com/watch?v=IuG06WHcruU")
```

See [Tutorialspoint](#) for more.

## 11. Break timer loop quiz and programming milestone

My independent completion of this quiz was my first programming milestone.

I tried this before watching the video. I decided to use a `for` loop. See [this Stack Overflow discussion](<https://stackoverflow.com/questions/17647907/for-or-while-loop-to-do-something-n-times>) for a comparison of `for` and `while` loops.

I got the `for` loop version to run correctly on my first try:

```
import
time
import webbrowser

for i in range(3):
    time.sleep(7) # try 7 seconds.
For 2 hours, use 2*60*60.
webbrowser.open_new_tab("https://www.youtube.com/watch?v=IuG06WHcruU")
```

I don't like the suggestion of `while` loops here. It would make more sense to me syntactically to use a `for` loop. When formatting as a `while` loop, I initially got stuck in an infinite loop, which I stopped by pressing `Esc` to enter command mode, and then `i,i` to interrupt the kernel. In order to run this as a `while` loop, you have to create a superfluous reference of integers, then add 1 to the count each time.

Breaks have to be counted each time, as Kunal did in his example. He requires three additional lines of code to perform the same function:

```
import time
import webbrowser

total_breaks = 3
break_count
= 0 # starting point
```

```
while(break_count < total_breaks):
    time.sleep(7) #
    try 7 seconds
    webbrowser.open_new_tab("https://www.youtube.com/watch?v=IuG06WHcruU")
    break_count = break_count + 1 # add 1 each time, to stop at 3 and avoid infinite
    loop
```

I am already learning to think in the most efficient way possible in Python! I'm becoming a programmer!

The current time on the instructor's computer is Wednesday February 12 16:40:40 2014. I picked up here on Wednesday at 16:20!

He also added a timestamp:

```
print('This program started on
' + time.ctime())
```

```
# break timer
import time
import webbrowser

for i in range(3):
    time.sleep(7) # try 7 seconds. For 2 hours, use 2*60*60.
    webbrowser.open_new_tab("https://www.youtube.com/watch?v=IuG06WHcruU")
```

## ’12. Making the Program Wait Longer

---

Already figured this out. the `time.sleep()` function takes in seconds, so `time.sleep(2*60*60)`.

## ’13. Where Does Webbrowser Come From?

---

`webbrowser` and `time` are part of the Python Standard Library. We don't see exactly how they communicate with the operating system to function. This hiding of detail is called **abstraction**. I am familiar with abstraction from David Evans' [cs4414: Operating Systems](#) class.

## ’14. Reading Webbrowser Documentation

---

## ’15. Take a Break Mini-Project

---

## ’18. Secret Message (story)

---

We have to rename files on the computer. The files have location names and numbers in them. The numbers are either at the beginning or end of the file name, and contain 1-3 digits.

They spelled "Edinbrough" incorrectly.

## 19. Secret Message (output)

---

The output is file names without numbers in them.

## 20. Quiz: Planning a Secret Message

---

What steps would you take to remove numbers from a bunch of file names on your computer?

- This is a standard task that they mention in the Python documentation tutorial under "[1. Whetting Your Appetite](#)." Just looking at the documentation and thinking about it:
- Define a function:
- Point Python to the directory where the files are located.
  - Ask Python to remove numbers from the name string of each file, in the `range(9)`. This could maybe be done with a `clear()` or `del()` command. We could potentially use the `os.path()`, `path.rename()`, or `os.rename()` functions.
- Loop the function for each file in the directory.

We had to post our responses in the forum, but the link was old and the thread was locked. I found an open thread, with the title [Planning a secret message](#) (yes, message was spelled incorrectly).

Kunal's steps:

1. Get file names
2. For each file, rename by removing numbers

## 21. Opening a File

---

Resources:

`class="ureact-markdown--markdown--3lhZa ureact-markdown ">`

1) [Zip File with 50 photos](#)

(after downloading, unzip this file to see all the photos)

2) [Stack Overflow Help Page](#)

(how to list all files in a given folder)

3) [How to get the path of a folder on a Mac](#)

(these instructions will help you find the path of the folder which contains photos)

He googled "find file names in a folder in Python" and clicked the Stack Overflow page. They reference the `os.listdir` function.

We started constructing our function:

```
import  
os  
  
def rename_files():  
    # (1) get file names from a folder  
    file_list =  
os.listdir('<path-to-prank-directory>')  
    print(file_list)
```

## 22. Quiz: Changing Filenames

Kunal suggested searching the Python documentation for the `os` module. Already [there!](#) I easily got the correct answer to this quiz.

Great Job

Yes! Reading documentation is a key skill and takes effort to acquire. You are beginning to practice a key attribute of an expert programmer.

## 23. Checking os Documentation

The resources section below recommends using the `<string>.translate()` function. I anticipated this need, because we are removing numbers from the file name, but the numbers are embedded within the string of the file name. We need a function to parse the file name.

Kunal starts typing the `for` loop, then goes to the shell window to test out some commands.

Try it in the shell first with a string to imitate the file name:

```
python  
>>> file_name = '48athens.jpg'  
>>> translation_table =  
str.maketrans('0123456789', ' ', '0123456789')  
>>>  
file_name.translate(translation_table)  
'athens.jpg'
```

Python 3 requires a separate step for construction of the translation table (see Resources section below). The second argument is just an empty string of ten spaces (length equal to the numbers string). I initially tried this with the `range(9)` function to be more concise, but didn't get the correct result.

This works regardless of the position of the numbers in the file name.

Note that we will have to define the variable `file_name` for the list of files in the folder.

Resources:

## Helpful Links

- 1) [How to use for loops in Python](#)
- 2) [How to use the translate function in Python](#)

## More information on the string function translate

Let us understand the documentation for `string.translate`

In the python documentation for string translate, you will see

```
string.translate(s, table[, deletechars])
```

Notice that there are three arguments (or inputs to the function), `s`, `table`, and the optional `deletechars` (we know it's optional because it's in square brackets). But only two are shown in the video: `table` and `deletechars`. Don't worry, this will be explained later. For now, learn this little trick, doing this:

```
import string  
  
string.translate(file_name, None, '0123456789')
```

is equivalent to doing this (putting `file_name` out front, in place of the `string` module):

```
file_name.translate(None, '0123456789')
```

## Python 3 Note

String translation changed in the transition from Python 2.7 to Python 3. The translation table is now separate from the `translate()` function itself. To do this kind of translation in Python 3, you will need to create a translation table, and pass that into the `translate()` function:

```
translation_table = str.maketrans("0123456789", "",  
"0123456789")  
filename.translate(translation_table)
```

You can read more about these changes [in the documentation](#).

## 24. Renaming Files

We can run the `<string>.translate()` function within the `os.rename` function.

## 25. Quiz: What Is the Error?

I thought the error would be that we have not yet defined the variable `file_name` for the list of files in the folder.

## 26. Quiz: Squashing the Bug

The error was actually that he was in the directory above the desired folder.

He wrote some code within the `rename_files()` function to check and save the current directory:

```
saved_path = os.getcwd()
print('Current working directory is ' + saved_path)
```

Then, at the end of the `rename_files()` function, he returned to the directory:

```
os.chdir(saved_path)
```

At first, I thought this wouldn't be necessary, because `os.chdir()` can be used to specify the correct directory before the loop to rename files. The problem is that Python remembers the directory when it is changed. It also follows the directory when it moves, so if you move the folder to the trash, the directory will be pointing to the trash. Saving the path as an object and returning there at the end of the code ensures that Python is looking at the expected directory.

*Why do we need to repeat the number string in the str.maketrans() function?* -> If you delete the second number string, there will be spaces in the file name. If you delete the string of spaces, and leave the two number strings, nothing happens.

*What is `file_name`?* -> It's possible that the string function actually has `file_name` pre-defined.

## Secret message example full code

```
import os

def rename_files():
    """Remove numbers from a list of image files."""
    # Save current working directory
    saved_path = os.getcwd()
    print('Current working directory is ' + saved_path)

    # Get file names from a folder
    os.chdir('img/prank')
    file_list = os.listdir()
    print(file_list)

    # For each file, rename filename
    for file_name in file_list:
        # construct translation table for the translate function
        t_table = str.maketrans('0123456789', ' ', '0123456789')
        # rename file with new name
        os.rename(file_name, file_name.translate(t_table))

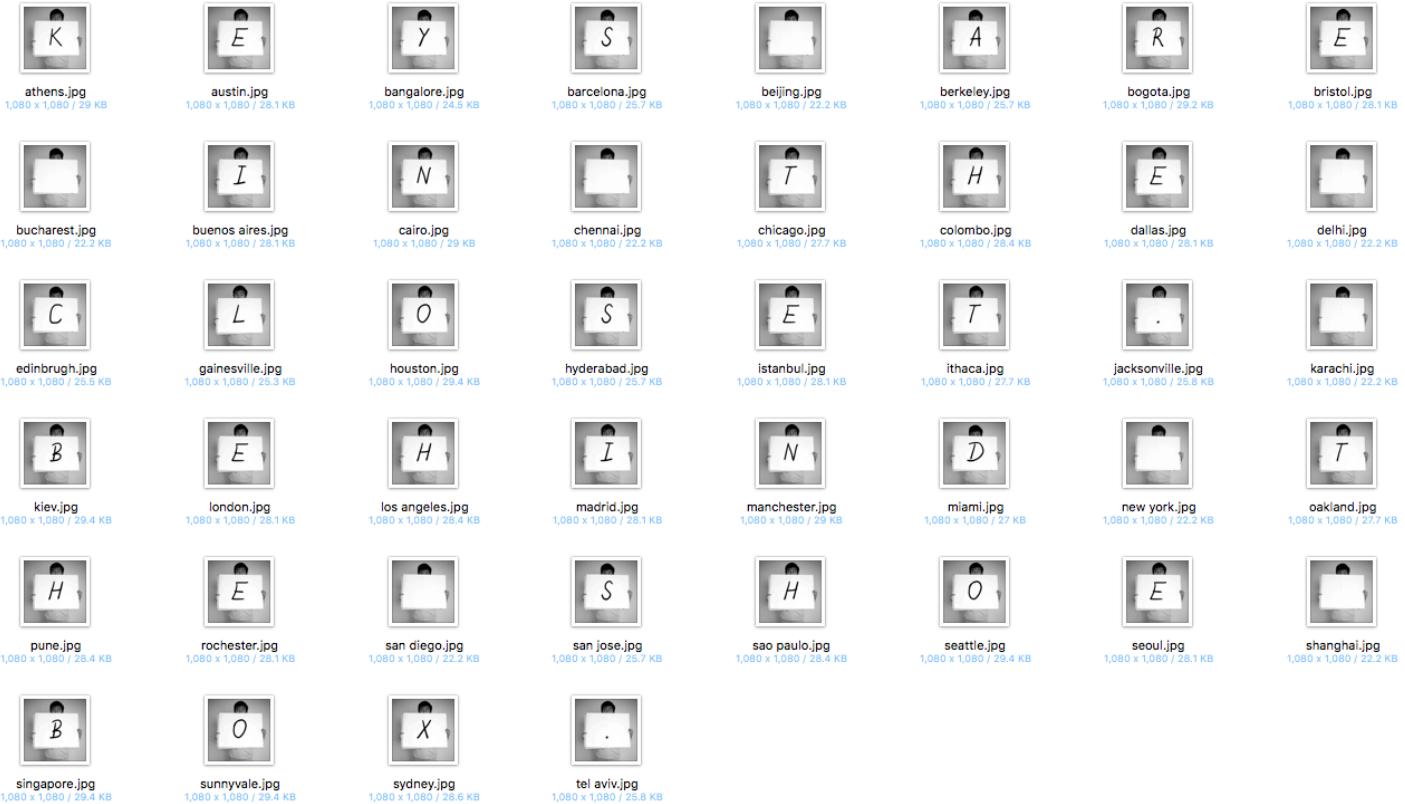
    # Display list of renamed files
    renamed_files = os.listdir()
    print(renamed_files)

    # Return to previous working directory
    os.chdir(saved_path)
    print('Directory changed back to ' + os.getcwd())
```

```
rename_files()
```

Success! Viewing the files in name order reveals the secret message: 'KEYS ARE IN THE CLOSET.  
BEHIND THE SHOEBOX.'

I kept the zipped image directory. Unzip for easy access to the original files when re-running the code.



## 27. Quiz: Rename Troubles

What would happen if: a) we tried to rename a file that doesn't exist? -> code returns an error b) we tried to rename a file to a name that already exists in the folder? -> code runs, but nothing is changed.

Exceptions tell Python what to do if it encounters an error. Resources section on exceptions:

In case you are curious, here is some information on [Exceptions](#). This is an advanced topic in object oriented programming. We will uncover several of these in Lesson 3b. We are still working on creating the best introduction to Exceptions. In the meantime, the link above should help you get started.

## 28. Where Does `os` Come From?

`os` is a module in the Python standard library. This means that it's basically a .py file with code in it that runs when `os` is loaded by Python. I checked the source code out on [GitHub](#).

I noted the reference to POSIX in the comments at the top. What is POSIX?

From Wikipedia:

The Portable Operating System Interface (POSIX)[1] is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. POSIX defines the application programming interface (API), along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems. [2][3]

Unix was selected as the basis for a standard system interface partly because it was "manufacturer-neutral." However, several major versions of Unix existed—so there was a need to develop a common denominator system.

## ’ 29. Reading os Documentation

Kunal just pointed us to the Python documentation, which I have already been reading.

## ’ 30. Quiz: Secret Message Mini-Project

Create your own secret message. Goal: spell out HELLO WORLD

- Udacity provided a folder for download, 'alphabet,' containing one image for each letter of the alphabet in capital letters. Space and period were also provided.
- [optional] write code to create a new directory
- [optional] Copy files: need three 'L' (Chennai) and two 'O' (Dallas).
- Put files in desired order, and save to new directory 'alphabet1' (manually in finder):
  - i. Bristol
  - ii. Beijing
  - iii. Chennai
  - iv.

Chennai 5. Dallas 6. Madrid 7. Jacksonville 8. Dallas 9. Gainesville 10. Chennai 11. Barcelona

- Rename files: instead of removing numbers and keeping the letters, I decided to add numbers to the file names, and then remove the letters.
  - I found the `ascii_lowercase[]` call in the string package on [Stack Overflow] (<https://stackoverflow.com/questions/3190122/python-how-to-print-range-a-z>), and used it to remove all letters. The second argument in the `str.maketrans()` function was just 26 empty spaces.
- It worked! Unfortunately though, my strategy resulted in the removal of the file extension. I included a second `for` loop to add back jpg.
- Note that the print statements and inclusion of the file lists as variables can be helpful for instructional purposes, but aren't necessary. Instead of creating a variable, the file list can be referenced directly with `os.listdir()`. Also see [Stack Overflow] (<https://stackoverflow.com/questions/16736080/change-the-file-extension-for-files-in-a-folder-in-python>).

```
# draft code
# testing beginning of code with new folder
alphabet1_list = os.listdir('img/alphabet1')
print(alphabet1_list)
```

## Secret message mini-project full code

```
import os
import string

def rename_files():
    """Remove lowercase letters from a list of image files."""
    # Save current working directory
    saved_path = os.getcwd()
    print('Current working directory is ' + saved_path)

    # Get file names from a folder
    os.chdir('img/alphabet1')
    alphabet1_list = os.listdir()
    print(alphabet1_list)

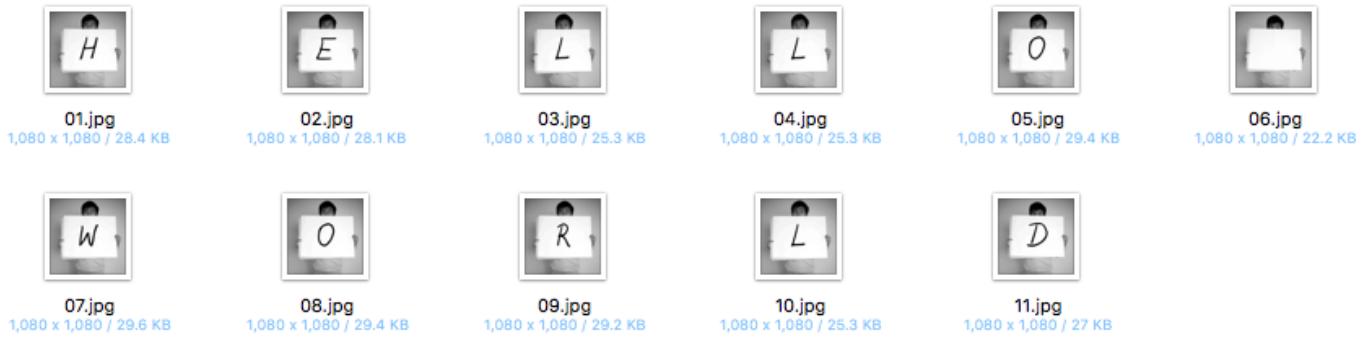
    # For each file, rename filename
    for file_name in alphabet1_list:
        # construct translation table for the translate function
        t_table = str.maketrans(string.ascii_lowercase[:],
                               '',
                               string.ascii_lowercase[:])
        # rename file with new name
        os.rename(file_name, file_name.translate(t_table))

    # Add back extension after deleting letters
    alphabet1_list_new = os.listdir()
    print(alphabet1_list_new)
    for file_name in alphabet1_list_new:
        os.rename(file_name, file_name + '.jpg')

    # Display list of renamed files
    renamed_files = os.listdir()
    print(renamed_files)

    # Return to previous working directory
    os.chdir(saved_path)
    print('Directory changed back to ' + os.getcwd())

rename_files()
```



## 31. Edge Case

---

So far, we have been creating functions, and then writing code line by line within the functions. We will continue to use this for our next programming challenge, creating a movie website.

## 32. When Functions Do Not Suffice

---

- Functions may not be adequate when operations get more complicated.
- If we want to pull up information about a movie, we can't just manually type all the information into the Python function. Instead, we should define a template. We could copy the template for each movie, but what if we need to change it?
- Instead of a function, we are going to use the `class` in Python to create the movie website.

# Lesson 06. Classes: Turtle graphics

---

Udacity Full Stack Web Developer Nanodegree program

Part 01. Programming fundamentals and the web [Programming foundations with Python]  
(<https://www.udacity.com/course/programming-foundations-with-python--ud036>)

Brendon Smith

br3ndonland

## ’ 01. Course Map

---

## ’ 02. Drawing Turtles (Story)

---

Drawing shapes like squares, circles, and fractals is one of the easiest ways to learn about classes.

## ’ 03.

---

Drawing Turtles (Output)

The output will be a series of squares that rotate around to form a circle.

## ’ 04. Quiz: How to Draw a Square

---

*What instructions will you provide if Kunal is walking around on a carpet trying to draw a square with his steps?*

1. walk east 5 paces
2. walk south 5 paces
3. walk west 5 paces
4. walk north 5 paces

## ’ 05. Drawing a Square

---

*Why is the Python function called turtle?*

-> from the [Python documentation](#):

Turtle graphics is a popular way for introducing programming to kids. It was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966.

Imagine a robotic turtle starting at (0, 0) in the x-y plane. After an `import turtle`, give it the command `turtle.forward(15)`, and it moves (on-screen!) 15 pixels in the direction it is facing, drawing a line as it moves. Give it the command `turtle.right(25)`, and it rotates in-place 25 degrees clockwise.

Resources in the instructor notes:

- [Changing Turtle's Shape](#)
- [Changing Turtle's Color](#)
- [Changing Turtle's Speed](#)
- Use this discussion thread to post your response: [Drawing Square Turtles](#)

Other resources I found:

- [Turtle.py | The Pragmatic Procrastinator](#)
- [Stack Overflow Turtle colors](<https://stackoverflow.com/questions/18621285/how-do-you-add-100-colors-using-a-loop-into-a-turtle-graphics-design-code>)
- [matplotlib pastel colors](#)

## 06. Change Turtle Shape, Color, and Speed

Note that in Jupyter notebook I seem to be getting a turtle `Terminator` error every second time. It may still be running in the notebook's Python kernel. Running the code in the shell also works.

```
import turtle

def draw_square():
    """Draw a square with
    turtle graphics."""
    window = turtle.Screen()
    window.bgcolor('cyan')
    brad = turtle.Turtle()
    brad.shape('turtle')
    brad.color('pink')
    brad.speed(3)

    brad.forward(100)
    brad.right(90)
    brad.forward(100)
    brad.right(90)
    brad.forward(100)
    brad.right(90)

    turtle.done()

draw_square()
```

## '07. Where Does Turtle Come From?

turtle (lowercase) is a module in the Python standard library.

Turtle (uppercase) is a class within the turtle module.

A class is "a neatly packaged box, that puts things together really well. Within `turtle.Turtle()` are various functions, like `def __init__`.

---

### 8. Reading Turtle Documentation

Already found [it](#).

## '09. Two Turtles

Adding a second turtle that starts where the first stops:

```
import turtle

def draw_square_and_circle():
    """Draw a square and
circle with turtle graphics."""
    window = turtle.Screen()
window.bgcolor('cyan')

    brad = turtle.Turtle()
    brad.shape('turtle')
brad.color('pink')
    brad.speed(3)

    brad.forward(100)
    brad.right(90)
brad.forward(100)
    brad.right(90)
    brad.forward(100)
    brad.right(90)
brad.forward(100)
    brad.right(90)

    angie = turtle.Turtle()
angie.shape('arrow')
    angie.color('blue')
    angie.circle(100)
turtle.done()

draw_square_and_circle()
```

## 10. What's Wrong With This Code?

My answer: Not efficient - lots of repeats. Should include loops.

Correct. Repetitive lines of code, and the name of the function needs to be changed now that we also draw a circle.

## 11.

Quiz: Improving Code Quality

There is no triangle function, but you can draw two lines and then a diagonal one connecting them by returning to the home position.

```
import turtle

def draw_shapes():
    """Draw shapes with
    turtle graphics."""
    window = turtle.Screen()
    window.bgcolor('cyan')
# Draw square
    brad = turtle.Turtle()
    brad.shape('turtle')
    brad.color('pink')
    brad.speed(3)

    for i in range(0, 4):
        brad.forward(100)
        brad.right(90)

    # Draw circle
    angie =
    turtle.Turtle()
    angie.shape('arrow')
    angie.color('blue')
    angie.circle(100)

    # Draw extra credit triangle
    beavis = turtle.Turtle()
    beavis.color('brown')
    beavis.backward(300)
    beavis.left(90)
    beavis.backward(300)
    beavis.home()

    turtle.done()

draw_shapes()
```

Posted response in "Making Turtle Code Better" Udacity forum.

## 12. Quiz: What is a Class?

A class is like a blueprint. The objects or instances are like examples of the blueprint.

## 13 Quiz: Making a Circle out of Squares

```
import turtle

def draw_square(some_turtle):
    """Draw a square with turtle graphics."""
    for i in range(0, 4):
        some_turtle.forward(100)
        some_turtle.right(90)
def many_squares():
    """Use turtle graphics to form a circle out of many
    squares."""
    window = turtle.Screen()
    window.bgcolor('cyan')

    brad =
    turtle.Turtle()
    brad.shape('turtle')
    brad.color('pink')
    brad.speed(3)
    for i in range(0, 36):
        draw_square(brad)
    brad.right(10)

    turtle.done()

many_squares()
```

## 14. Quiz: Turtle Mini-Project

I created a colorful spiral, transporting the viewer into the Tron game grid. The background image is from [Wikipedia](#).

```
import turtle
import colorsys

def spiral_into_the_grid():
```

```

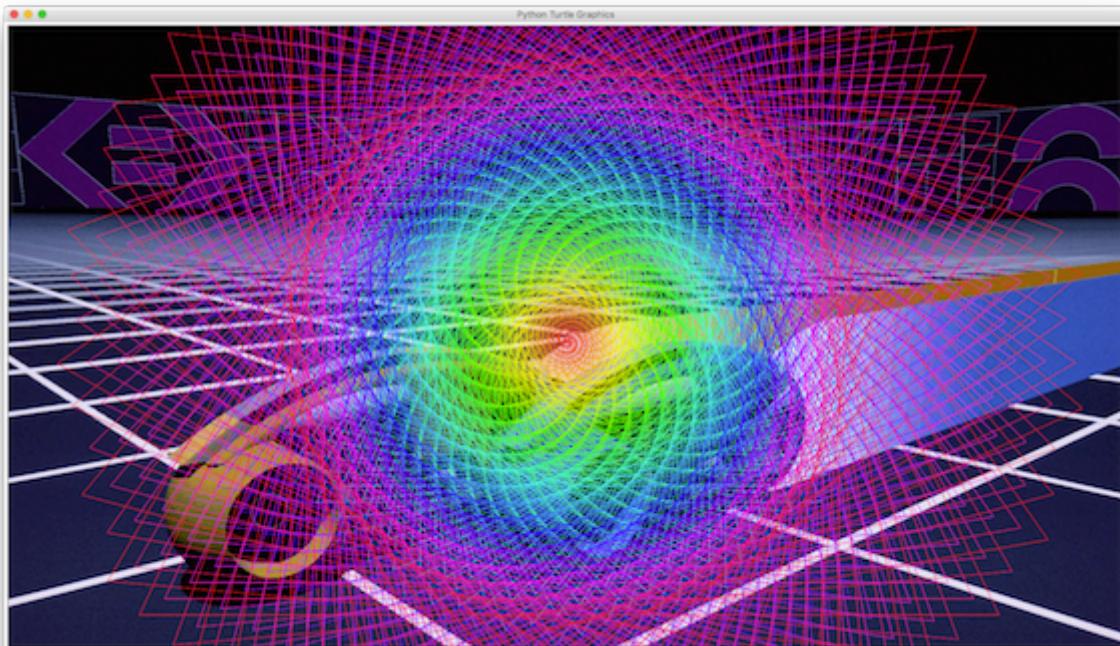
"""Use turtle graphics to create a colorful spiral."""
turtle.setup(width=1600, height=900)
turtle.speed(0)
turtle.hideturtle()
window = turtle.Screen()
window.bgpic('img/TRON.gif')

for i in range(1250):
    colors = colorsys.hsv_to_rgb(i / 1250, 1.0, 1.0)
    turtle.color(colors)
    turtle.forward(i)
    turtle.left(115)

turtle.done()

spiral_into_the_grid()

```



## 15. Quiz: Comfort Level

How comfortable am I with writing computer programs? I said 3 - "somewhat comfortable." This is what I said before beginning the module. I have learned to write some simple programs, but need more training before I can write anything complex or original. Based on the project submissions from other students in the discussion forum, my competency and originality are above-average.

## 16. They Look So Similar

Kunal explained something that was difficult for him to understand in college. Calling classes and calling functions look similar.

`webbrowser.open()` calls a function. `turtle.Turtle()` actually calls an init function within the class `turtle`.

---

## Appendix

I was wondering if Python was used for more advanced graphics applications. On the [Python homepage](#), I came across a [testimonial](<https://www.python.org/success-stories/industrial-light-magic-runs-python/>) by Tim Fortenberry, Industrial Light & Magic, about how they use Python for their image processing workflow:

ILM runs a batch processing environment capable of modeling, rendering and compositing tens of thousands of motion picture frames per day. Thousands of machines running Linux, IRIX, Compaq Tru64, OS X, Solaris, and Windows join together to provide a production pipeline used by ~800 users daily. Speed of development is key, and Python was a faster way to code (and re-code) the programs that control this production pipeline.

# Lesson 07. Classes: send texts

---

Udacity Full Stack Web Developer Nanodegree program

Part 01. Programming fundamentals and the web

[Programming foundations with Python](<https://www.udacity.com/course/programming-foundations-with-python--ud036>)

Brendon Smith

br3ndonland

## ’ 01. Course Map

---

We're now moving on to the second part of Lesson 2: using classes.

## ’ 02. Send Text Messages (Story)

---

We will design a system for automated sending of text messages. Cool!

## ’ 03. Send Text Messages (Output)

---

The output sends automated text messages to his phone.

## ’ 04. Introducing Twilio

---

`twilio` is a Python package that sends text messages with Python code. It does not come with the Python standard library.

The PyPI (Python Package Index) ranking site is down. According to [this reddit](#), the owner has had trouble maintaining the site. As alternatives, check out PyPI directly, or the [Python 3 Wall of Superpowers](#) (formerly Wall of Shame).

## ’ 05. Download Twilio

---

I searched "twilio python" and found the [twilio SMS and MMS python quickstart video](#), and the associated [\[guide\]](#)(<https://www.twilio.com/docs/guides/how-to-send-sms-messages-in-python>). It was helpful. He had his account\_sid, auth\_token, and phone number stored in environment variables with `os.environ[]`. He also made a demo message about the Millenium Falcon in Star Wars!

## ’ [install twilio

helper library with pip](<https://www.twilio.com/docs/libraries/python>)

Twilio is included within the Pipenv. It can be installed with

```
pip install  
twilio
```

```
import twilio  
print(twilio.__version__)
```

## ’06. Quiz:

---

Twilio Download Feedback

Downloaded successfully

## ’07. Setting Up Our Code

---

## ’08. Registering with Twilio

---

Got phone number (415) 903-8062

## ’09. Running Our Code

---

### [Python Helper Library SMS Test](#)

The account\_sid is like a username, and the auth\_token is like a password.

```
from twilio.rest  
import Client  
  
# Your Account SID from twilio.com/console  
account_sid = ""  
#  
Your Auth Token from twilio.com/console  
auth_token = ""  
  
client =  
Client(account_sid, auth_token)  
  
message = client.messages.create(  
    to="+", #  
    my mobile #  
    from_="+14159038062", # twilio #  
    body="Hello from Python!")  
print(message.sid)
```

It worked!

## 10. Quiz: Python Keyword From

What does `from` do in Python? -> accesses part of a Python module.

From Python documentation:

As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module (the collection of variables that you have access to in a script executed at the top level and in calculator mode).

Note that in general the practice of importing `*` from a module or package is frowned upon, since it often causes poorly readable code. However, it is okay to use it to save typing in interactive sessions.

From [Stack Overflow](<https://stackoverflow.com/questions/710551/import-module-or-from-module-import>):

The difference between `import module` and `from module import foo` is mainly subjective. Pick the one you like best and be consistent in your use of it. Here are some points to help you decide.

`import module`

- Pros:

- Less maintenance of your `import` statements. Don't need to add any additional imports to start using another item from the module

- Cons:

- Typing `module.foo` in your code can be tedious and redundant (tedium can be minimized by using `import module as mo` then typing `mo.foo`)

`from module import foo`

- Pros:

- Less typing to use `foo`
- More control over which items of a module can be accessed

- Cons:

- To use a new item from the module you have to update your `import` statement
- You lose context about `foo`. For example, it's less clear what `ceil()` does compared to `math.ceil()`

Either method is acceptable, but **don't** use `from module import *`.

For any reasonable large set of code, if you `import *` you will likely be cementing it into the module, unable to be removed. This is because it is difficult to determine what items used in the code are coming from 'module', making it easy to get to the point where you think you don't use the `import` any more but it's extremely difficult to be sure.

[tutorialspoint]([https://www.tutorialspoint.com/python/python\\_modules.htm](https://www.tutorialspoint.com/python/python_modules.htm)) > The `from...import` Statement > Python's from statement lets you import specific attributes from a module into the current namespace. The from...import has the following syntax – > > `'from modname import name1[, name2[, ... nameN]]'` > For example, to import the function fibonacci from the module fib, use the following statement – > > `'from fib import fibonacci'` > This statement does not import the entire module fib into the current namespace; it just introduces the item fibonacci from the module fib into the global symbol table of the importing module. > > The `from...import \*` Statement: > It is also possible to import all names from a module into the current namespace by using the following import statement – > > `'from modname import *'` > This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.

## ’11. Investigating the Code

---

Resources from Udacity:

- 1) [How does Python Keyword `from` work](#)
- 2) [Read the actual Twilio code on Github](#)

already checked out the GitHub page.

## ’12. Where Does Twilio Come From?

---

`TwilioRestClient` is a class within the `twilio.rest` module.

## ’13. Connecting Turtle, Twilio and Classes

---

Classes can be used as blueprints to build repeated instances, or objects. The activities performed in each instance are defined inside the class.

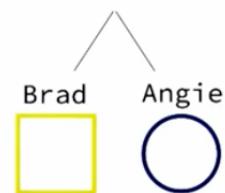
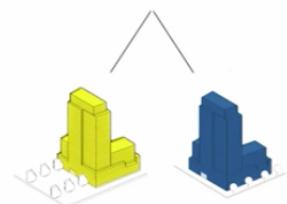
## Turtle graphics and Twilio as examples of classes and objects

Class



class Turtle  
=====

Instance  
or  
Object



## 14. Send Text Messages Mini-Project

---

Answer the following questions on the discussion forum:

1. *What is a class?*
2. *What is an instance of a class?*
- 3.

*Thus far we have compared the class to a blueprint. Can you think of another analogy to explain classes?*

My answers:

1. *What is a class?* -> a class is a collection of functions written in Python code.
2. *What is an instance of a class?* -> an instance is an specific adaptation of the code functions available in a class.
3. *Thus far we have compared the class to a blueprint. Can you think of another analogy to explain classes?* -> it's like a template that you can fill in.

# Lesson 08. Classes: Profanity checker

---

Udacity Full Stack Web Developer Nanodegree program

Part 01. Programming fundamentals and the web [Programming foundations with Python]  
(<https://www.udacity.com/course/programming-foundations-with-python--ud036>)

Brendon Smith

br3ndonland

## ’01. Course Map

---

## ’02. Quiz: Embarrassing story

---

When I heard Kunal's story of accidentally sending a curse word to his boss because he typed the wrong letter, I immediately thought of text auto-correct and looked up the history of its development. Wired had a [good article](#). It was developed by Dean Hachamovitch, head of data science at Microsoft. There is an [autocorrect program on PyPi](#).

## ’03. Quiz: Planning Profanity Editor

---

- Create a list of words, like Microsoft's "Words which should neither be flagged nor suggested" from the [Wired article](#), and replacements.
- Find the words in the file
- Replace words with replacements.

## ’04. Reading from a File

---

Starting to code: read text from a file with `<file>.read()`.

"It just blows my mind that we can read from a text file with only a few lines of code." Kunal

```
def read_text():
    """Read the contents of a text file."""
    quotes = open('movie_quotes.txt')
    contents_of_file = quotes.read()
    print(contents_of_file)
    quotes.close()

read_text()
```

## ’05. Quiz: Place Function `open`

---

Where do you think `open` resides? -> I guessed correctly by looking at the code. It's in the standard library, but separate from `os`.

Functions like `open` are so commonly used, that they are always available. They don't require a separate `import` command.

## ’06.

---

Reading Open Documentation

## ’07. Connecting Turtles and Open

---

`open` is one of the [built-in functions](#). It mentions objects.

| Open file and return a corresponding [file object](#).

Where have we come across this before? -> in the previous lesson on classes. An object is an instance of a class. Open, like Turtle, is calling some `__init__`-like function.

| "Being able to read documentation and experimenting with Python is a big part of learning to program." Kunal

## ’08. Quiz: Built-In Python Functions

---

1. Pick a built-in Python function. Read through its documentation.

2. Use it in a new program and share your experience on the forum.

I took a look at the built-in functions and selected `tuple()`. I'm still not clear on what a `tuple` is, so it's a good chance to learn.

### ’tuple starts

A `tuple` is not really a function, but actually a type of ordered data structure (see the Python documentation [4.6. Sequence Types — list, tuple, range]

(<https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range>)). It can contain both numbers and text strings, enclosed in parentheses and separated by commas. It's immutable, meaning that you can't modify it after creation (unlike dictionaries, which are mutable).

From the [documentation] (<https://docs.python.org/3/library/functions.html?highlight=built#func-tuple>):

| Rather than being a function,

| [tuple](stdtypes.html#tuple "tuple") is actually an immutable sequence type, as documented in [Tuples](#) and [Sequence Types — list, tuple, range](#).

Note that the `cmp()` function referenced on [Tutorialspoint](#) was [removed in Python 3] (<https://docs.python.org/3/howto/sorting.html?highlight=cmp#the-old-way-using-the-cmp-parameter>).

There are some built-in functions that can be used with tuples. I wrote some code that compares two tuples. I created tuples with identity information for two fearsome beasts from Star Wars: the **Rancor** and the **Sarlacc**. Which one is taller? The Rancor might look taller, but the Sarlacc extends deep beneath the sands. Let's create a function to compare heights:

```
# tuples: name, age in years, homeworld, height in meters, diet

sarlacc = ('Sarlacc', 30000, 'Tatooine', 100, 'omnivorous')

rancor = ('Rancor', 'unknown', 'Dathomir', 5, 'carnivorous')

if rancor[3] > sarlacc[3]:
    print('Rancor is taller!')
elif sarlacc[3] > rancor[3]:
    print('Sarlacc is taller!')
```

## tuple struggles

I experimented with different ways of creating the tuples.

```
name = 'Sarlacc'
age = 30000
homeworld = 'Tatooine'
height = 100

# combine the objects into a tuple
Sarlacc = (name, age, homeworld, height)
print(Sarlacc, len(Sarlacc))

# return the third entry in the tuple
# remember the tuple starts at 0, so this will be homeworld
Sarlacc[2]
```

I tried something similar to the `student_tuples` example in the "Sorting HOW TO" section of the Python 3 documentation. This creates a list with two tuples. It was difficult to work with because I couldn't slice the individual tuples.

```
# creates a list of two tuples
star_wars_beasts = [
    ('Rancor', 'unknown', 'Dathomir', 5, 'carnivorous'),
    ('Sarlacc', 30000, 'Tatooine', 100, 'omnivorous')
]
print(star_wars_beasts[0][0])
```

It was better to create two separate tuples. It is possible to do this in the same line of code, but the code will not be PEP 8 compliant.

```
# creates two separate tuples on the same line
Sarlacc, Rancor = (30000, 'Tatooine', 100, 'omnivorous'), ('unknown', 'Dathomir', 5, 'c
print(Sarlacc, '\n' ,Rancor)
```

What if I want to combine the objects into a tuple, so I can add another and still use the original names?

## I NEED A DICT!

See python-cf17-02-language and GA\_python101.Rmd

Here's the dict:

```
Sarlacc = {'name': 'Sarlacc', 'age, years': 30000, 'homeworld': 'Tatooine', 'height, me
print(Sarlacc['homeworld'])
```

## 09. Checking for Curse Words

### Next step with the code

Now we need to scan the contents of the file for curse words. We can use the `urllib.urlopen` function from the standard library to open the wdylike site.

```
import
urllib

def read_text():
    """Read the contents of a text file."""
    quotes
    = open('movie_quotes.txt')
    contents_of_file = quotes.read()
    print(contents_of_file)
    quotes.close()
    check_profanity(contents_of_file)
def check_profanity(text_to_check):
    """Check the text file for profanity."""
connection = urllib.urlopen('http://www.wdylike.appspot.com/?q='+text_to_check)
output = connection.read()
print(output)
connection.close()
read_text()
```

## ⌚ Instructor notes

Checking for Curse Words

1. Using What Do You Love to check curse words
2. In case you are curious about the things we are adding after the ? in the link or URL

## ⌚ Comments

This exercise would have made more sense if I understood HTTP requests. We didn't go through HTTP requests until part 2 (developer tools).

I was actually curious about the `?q=shot` text in the link above. This is the query. It's asking the website if "shot" is a curse word. Google discontinued the wdyl site, but this site returns a plaintext response: true if the word is a curse word, and false if it's not. I tried this out before watching the video, and caught on to it.

It was also striking to find that, as of 2011, 96% of Google's revenue comes from AdWords, their advertising service.

## ⌚ 10. Accessing a Website with Code

The code didn't run at first: `AttributeError: module 'urllib' has no attribute 'urlopen'`

I searched the Python documentation for `urlopen`. The syntax is slightly different now. Corrected version:

```
from urllib.request import urlopen

def
read_text():
    """Read the contents of a text file."""
    quotes =
open('movie_quotes.txt')
    contents_of_file = quotes.read()
print(contents_of_file)
    quotes.close()
    check_profanity(contents_of_file)
def check_profanity(text_to_check):
    """Check the text file for profanity."""
connection = urlopen('http://www.wdylife.appspot.com/?q=' + text_to_check)
output = connection.read()
    print(output)
    connection.close()
read_text()
```

I'm also getting an `HTTPError: HTTP Error 400: Bad Request`. I ran a quick search and found a fix on [Stack Overflow](<https://stackoverflow.com/questions/8840303/urllib2-http-error-400-bad-request>), but the fix is outdated and didn't work for me.

See below for solution.

## ☞ 11. Quiz: Place urllib and urlopen

---

within the standard library. urlopen is a function within urllib.

## ☞ 12. Reading urllib Documentation

---

## ☞ 13. Printing a Better Output

---

Added a few lines of code to print clear feedback on the result:

```
from urllib.request import urlopen

def
read_text():
    """Read the contents of a text file."""
    quotes =
open('movie_quotes.txt')
    contents_of_file = quotes.read()
print(contents_of_file)
    quotes.close()
    check_profanity(contents_of_file)
def check_profanity(text_to_check):
    """Check the text file for profanity."""
connection = urlopen('http://www.wdylike.appspot.com/?q=' + text_to_check)
output = connection.read()
    print(output)
    connection.close()
    if
'true' in output:
        print("Profanity Alert!")
    elif "false" in output:
print("This document has no curse words!")

read_text()
```

## ☞ 14. Quiz: Profanity Editor Mini-Project

---

### ☞ Code with `urllib.request` module

I adapted the code for Python 3.6.2 and made it as concise as possible. My code was initially returning `HTTPError: HTTP Error 400: Bad Request`. There is another Udacian, tb7459, who got around this by replacing spaces with `%20` (ASCII spaces). The ASCII spaces can also be seen in the website url after [like this](#). I rewrote my code, making it a little more concise by directly importing `urlopen`.

```
# Profanity checker mini-project
# Code without Requests

from urllib.request import urlopen

def read_text():
    """Read the contents of a text file."""
    quotes = open('movie_quotes.txt')
    contents_of_file = quotes.read()
    print(contents_of_file)
    quotes.close()
    check_profanity(contents_of_file)

def check_profanity(text_to_check):
    """Check the text file for profanity."""
    # ascii spaces
    text_to_check = text_to_check.replace(' ', '%20')
    text_to_check = text_to_check.replace('\n', '%20')
    # web query
    connection = urlopen('http://www.wdylike.appspot.com/?q=' + text_to_check)
    output = str(connection.read())
    connection.close()
    # output
    if 'true' in output:
        print('Profanity Alert!')
    elif 'false' in output:
        print('This document has no curse words!')
    else:
        print('Could not scan the document properly.')

read_text()
```

## ↪ Code with Requests module

I noted that the [Python documentation](#) recommends the `Requests module` instead, so I rewrote the code for Requests. The author of Requests, Kenneth Reitz, is an interesting and talented person, and I enjoyed checking out his [personal website](#).

I was able to pass the text directly to Requests without converting to ASCII spaces. The web query only requires one line with the `requests.get()` function, and the text response from the website can be referenced without creating an additional variable with the `r.text` function.

Note that I was initially working with the "response status code" from the `requests.get()` function, which was not necessary. I was getting `<Response [200]>`, True, with or without profanity. A True response code just means that a response was received from the website. In order to actually return the text string from the website, I needed `r.text`.

```
# Profanity checker mini-project
# Code rewritten for Requests

import requests

def read_text():
    """Read the contents of a text file."""
    quotes = open('movie_quotes.txt')
    contents_of_file = quotes.read()
    print(contents_of_file)
    quotes.close()
    check_profanity(contents_of_file)

def check_profanity(text_to_check):
    """Check the text file for profanity."""
    # web query
    r = requests.get('http://www.wdylike.appspot.com/?q=' + text_to_check)
    # output
    if 'true' in r.text:
        print('Profanity Alert!')
    elif 'false' in r.text:
        print('This document has no curse words!')
    else:
        print('Could not scan the document properly.')

read_text()
```

## ⌚ Comments

I shared my progress with my family over the phone.

I also note that this program is just a profanity *checker*, not really an editor. We're not actually replacing the profanity strings with alternatives.

## ⌚ 15. Connecting Turtle, Open, and Urllib

In each case, we have been working with objects.

```
brad = turtle.Turtle : brad is an object of turtle
quotes = open(File) : quotes is an object of
file connection = urllib.urlopen() : a "file-like object"
```

Remember, classes are like blueprints, and objects are like specific instances of the blueprint.

## Feedback

---

- This was a helpful and educational exercise.
- This exercise would have made more sense if I understood HTTP requests. We didn't go through HTTP requests until part 2 (developer tools).
- The program is just a profanity *checker*, not really an editor. We're not actually replacing the profanity strings with alternatives. The project could have included an autocorrect-type feature.

# Lesson 09. Classes: Movie website

---

Udacity Full Stack Web Developer Nanodegree program

Part 01. Programming fundamentals and the web [Programming foundations with Python]  
(<https://www.udacity.com/course/programming-foundations-with-python--ud036>)

Brendon Smith

br3ndonland

## ’ 01. Course Map

---

We started off learning to build functions, but they weren't adequate for making a movie website. We then used classes for three different projects, turtles, text messages, and profanity editor checker. Now, we will create classes in the movie website project. We will create a website that displays our favorite movies, and then when the movie thumbnail is clicked on, the trailer comes up from YouTube.

## ’ 02. Quiz: What Should Class Movie

---

Remember?

As we have learned, classes are like blueprints, and objects are like specific instances of the blueprint.

It took me several tries to get the quiz correct, because the question wasn't clear. The question was *Which of the following data points should the class Movie remember?* I went with the bare minimum, but I think they were basically looking for, *which of the following data points relate to movies?*

```
# things to remember
title
storyline
reviews
poster_image
youtube_trailer
# things to do
show_trailer()
```

I can already see how we're going to build this. We will create a function for each feature, and group the functions together into `class Movie`.

## ’ 03. Defining Class Movie

---

- It is good practice to define a class in one file, and reference it in a separate file.

- Class names should be written in CapWords. See [PEP8 naming conventions] (<https://www.python.org/dev/peps/pep-0008/#prescriptive-naming-conventions>) and Google Style Guide for Python.

*entertainment\_center.py* will reference the class created in *media.py*.

*media.py*

```
class Movie():
```

*entertainment\_center.py*

```
# import the media file, which contains
# the class definition
import media
# create an object, or specific instance, of
# the class Movie
toy_story = media.Movie()
```

## ’04. Quiz: what happens when

I obviously got this one. `__init__` creates space in memory for the new function.

Great Job

You got it! When we create the instance brad, the init function inside the class Turtle gets called.

## ’05. Defining init

Note that I'm starting to write the python parts of the code in Sublime Text. The features I have set up make it faster and easier. It's also easier to write Markdown in Sublime Text. Hopefully the upcoming JupyterLab will bring together the best of Jupyter notebook with the best of text editors like Sublime Text.

The autocomplete and PEP 8 checker in the Anaconda Sublime package are very helpful. I had an error on the second line of `def __init__` in *media.py*. It was PEP 8 E128. I did a quick search and found the answer on [Stack Overflow] (<https://stackoverflow.com/questions/15435811/what-is-pep8s-e128-continuation-line-under-indented-for-visual-indent#15435837>) and **PEP 8**: need to indent to the opening parenthesis.

The instructor Kunal's code is most definitely not PEP 8. *media.py*

```
class Movie():
    def __init__(self, movie_title,
                movie_storyline, poster_image,
                trailer_youtube):
        self.title = movie_title
```

```

        self.storyline = movie_storyline
self.poster_image_url = poster_image
    self.trailer = trailer_youtube
pass

```

*entertainment\_center.py*

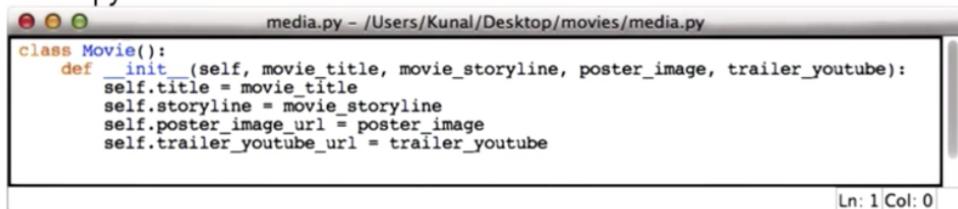
```

# import the media file, which
contains the class definition
import media
# create an object, or specific
instance, of the class Movie
toy_story = media.Movie(
    "Toy Story",
    "A
story of a boy and his toys that come to life",
"https://upload.wikimedia.org/wikipedia/en/1/13/Toy_Story.jpg",
"https://www.youtube.com/watch?v=vwyZH85NQC4")
print(toy_story.storyline)

```

## ’06. What Is Going On Behind The Scenes

*media.py*



```

media.py - /Users/Kunal/Desktop/movies/media.py
class Movie():
    def __init__(self, movie_title, movie_storyline, poster_image, trailer_youtube):
        self.title = movie_title
        self.storyline = movie_storyline
        self.poster_image_url = poster_image
        self.trailer_youtube_url = trailer_youtube

```

toy\_story = media.Movie("Toy Story", "Toys Come to Life", "Toy Story Poster Link", "Toy Story Youtube Link")

    ↳ \_\_init\_\_ gets called

```

self=toy_story
movie_title="Toy Story"
movie_storyline="Toys Come to Life"
poster_image="Toy Story Poster Link"
trailer_youtube="Toy Story Youtube Link"

```

    → toy\_story

```

title="Toy Story"
storyline="Toys Come to Life"
poster_image_url="Toy Story Poster Link"
trailer_youtube_url="Toy Story Youtube Link"

```

## ’07. Quiz: What Will Be the Output?

Adding another instance of class Movie. The output will be printing the Avatar storyline.

## entertainment\_center.py behind the scenes

### ’08. Behind the Scenes

---

### ’09. Is `self` important?

---

What if the word `self` is removed from in front of `storyline`?

I said: Python will return an undefined variable error -> *correct!*

### ’10. Next up: `show_trailer`

---

## › 11. Playing Movie Trailer

---

## › 12. Quiz: Play

---

Your Favorite Trailer

I anticipated the inclusion of this function and already put it in my python code!

Code so far:

*media.py* (unchanged)

```
class Movie():
    def __init__(self, movie_title, movie_storyline,
poster_image,
                 trailer_youtube):
        self.title =
movie_title
        self.storyline = movie_storyline
    self.poster_image_url = poster_image
        self.trailer = trailer_youtube
pass
```

*entertainment\_center.py*

```
# Udacity full-stack web developer
nanodegree
# 01. Programming fundamentals and the web
# Lesson 09: Make classes:
movie website

# import the media file, which contains the class definition
import media
import webbrowser

# create an object, or specific instance, of the
class Movie
toy_story = media.Movie(
    "Toy Story",
    "A story of a boy and
his toys that come to life",
"https://upload.wikimedia.org/wikipedia/en/1/13/Toy_Story.jpg",
"https://www.youtube.com/watch?v=vwyZH85NQC4")
print(toy_story.storyline)

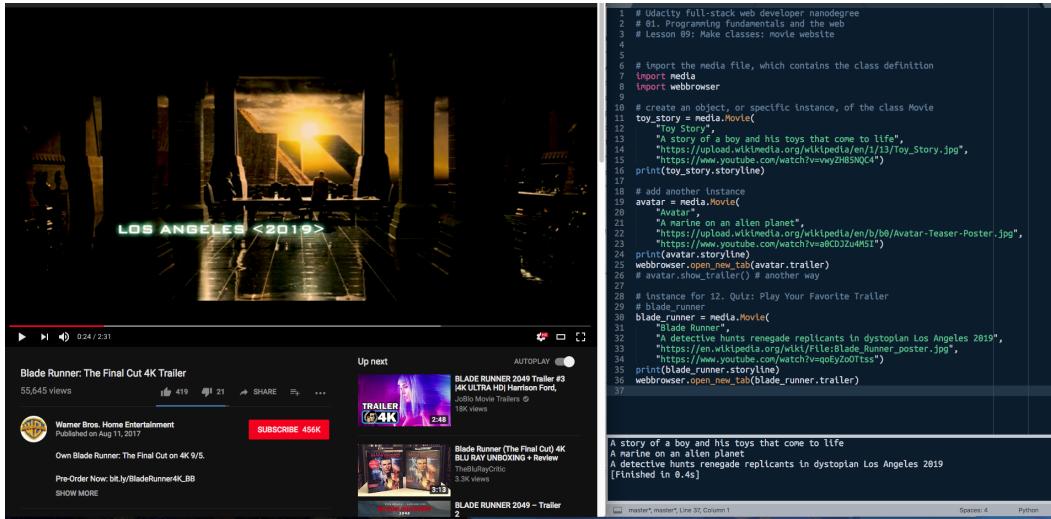
# add
another instance
avatar = media.Movie(
    "Avatar",
```

```

    "A marine on an alien
planet",
    "https://upload.wikimedia.org/wikipedia/en/b/b0/Avatar-Teaser-
Poster.jpg",
    "https://www.youtube.com/watch?v=a0CDJZu4M5I")
print(avatar.storyline)
webbrowser.open_new_tab(avatar.trailer)
#
avatar.show_trailer() # another way

```

I uploaded a screenshot to the discussion forum.



## 13. Recap Vocab

### New vocabulary words:

**Class:** A collection of functions that can be called in specific instances.

**Constructor:** used to indicate that a function is being built.

**Self:** when self is used in `__init__(self):`, it refers to the object being created. We call it `self` because it's like an example at this point that can be referenced in the future, when creating a new instance of a class. Any word could be used, but `self` is a useful convention. From [Python documentation](#):

Often, the first argument of a method is called `self`. This is nothing more than a convention: the name `self` has absolutely no special meaning to Python. Note, however, that by not following the convention your code may be less readable to other Python programmers, and it is also conceivable that a class browser program might be written that relies upon such a convention. **Instance:** a specific use of a Class.

**Instance variables:** variables from the Class that have been defined by the instance.

**Instance method:** a function defined inside of a class and associated with an instance.

## 14. Designing the Movie Website

We are going to build a movie website! We download the "fresh\_tomatoes.py" file, which is basically a Bootstrap template for the website.

## 3 Instructor notes

Screenshot of all 6 instances of class Movie:

```
entertainment_center.py - /Users/Kunal/Desktop/movies/entertainment_center.py

import media

toy_story = media.Movie("Toy Story", "A story of a boy and his toys that come to life",
                       "http://upload.wikimedia.org/wikipedia/en/1/13/Toy_Story.jpg",
                       "https://www.youtube.com/watch?v=vwyZH85NQc4")

#print(toy_story.storyline)

avatar = media.Movie("Avatar", "A marine on an alien planet",
                     "http://upload.wikimedia.org/wikipedia/id/b/b0/Avatar-Teaser-Poster.jpg",
                     "https://www.youtube.com/watch?v=-9ceBgWV8io")

school_of_rock = media.Movie("School of Rock", "Storyline",
                            "http://upload.wikimedia.org/wikipedia/en/1/11/School_of_Rock_Poster.jpg",
                            "https://www.youtube.com/watch?v=3PsUJFEBC74")

ratatouille = media.Movie("Ratatouille", "Storyline",
                         "http://upload.wikimedia.org/wikipedia/en/5/50/RatatouillePoster.jpg",
                         "https://www.youtube.com/watch?v=c3sBBRxDAqk")

midnight_in_paris = media.Movie("Midnight in Paris", "Storyline",
                                "http://upload.wikimedia.org/wikipedia/en/9/9f/Midnight_in_Paris_Poster.jpg",
                                "https://www.youtube.com/watch?v=atLg2wQQxvU")

hunger_games = media.Movie("Hunger Games", "Storyline",
                           "http://upload.wikimedia.org/wikipedia/en/4/42/HungerGamesPoster.jpg",
                           "https://www.youtube.com/watch?v=PbA63a7H0bo")
```

Download the Python file [fresh\\_tomatoes.py](#) (make sure you save this file in the same folder as the python file where you create your movie instances.)

## 15. Coding The Movie Website

Create an array `movies` with all the movies in the file.

## 16. Quiz: Movie Website Mini-Project

## › Readme

## › description

Server-side code written in Python to store a list of my favorite movies, including artwork and trailers, then serve the data to a local webpage with HTML and CSS.

› noir

Film noir is one of my favorite genres. I decided to include two classic noir films, two neo noir films (in the film noir style, set in the noir era, but made more recently), and two future noir films (containing elements of film noir, but set in the future).

| Noir era | Movie 1 | Date | Movie 2 | Date | :-----|:-----|:-----|:-----|:-----|:-----| | *Classic Noir* |  
Double Indemnity | 1944 | Out of the Past | 1947 | | *Neo Noir* | Chinatown | 1974 | Body Heat | 1981 |  
| *Future Noir* | Blade Runner | 1982 | The Terminator | 1984 |

## ’ Debugging

- I was not able to get the code to run at first, like many others. I had to change `self.trailer` to `self.trailer_youtube_url` in `media.py`.
- Some of the movie posters didn't come through at first either. I had to change the wikipedia links to the corresponding upload.wikimedia.org versions by clicking on the image again.
- My subtitle was also sticking out below the navbar. I was unsure how to change the navbar size, because we didn't download a corresponding CSS with the `fresh_tomatoes.py` file. I think the CSS is automatically referenced by Bootstrap. Instead of changing to a different Bootstrap template, I just put the subtitle on the same line of the title, and separated them with a pipe.

## ’ Code review

I passed code review with only minor revisions.

- Added a docstring above the function definition in `media.py` to explain what the function does
- Added a standardized header in the first three lines of each file

## ’ Final code

Available in the GitHub repo [udacity-fsnd01-p01-movies.media.py](#)

```
# Udacity full-stack web developer nanodegree
# 01.
Programming fundamentals and the web
# Project 01: Movie trailer website

class
Movie():
    # add docstring that can be called with __doc__
    """This class
provides a way to store movie information."""

    # add a class variable that
can be shared by all instances of class Movie
    valid_ratings = ["G", "PG",
"PG-13", "R"]

    # build a function to store info in instances of class Movie
def __init__(self, movie_title, movie_year, movie_storyline,
poster_image, trailer_youtube_url):
        """This function creates a template
to store movie information."""
        self.title = movie_title
    self.year = movie_year
        self.storyline = movie_storyline
    self.poster_image_url = poster_image
        self.trailer_youtube_url =
trailer_youtube_url
        pass
```

## noir.py

```
# Udacity full-stack
web developer nanodegree
# 01. Programming fundamentals and the web
# Project
01: Movie trailer website

# import the media file, which contains the class
definition
import media
# import the fresh_tomatoes file, which contains code to
build the site
import fresh_tomatoes_noir

# film noir: double_indemnity
double_indemnity = media.Movie(
    "Double Indemnity",
    "1944",
    "An
insurance agent tries to game the system and win his love interest.",
    "https://upload.wikimedia.org/wikipedia/en/3/35/Double\_indemnity.jpg",
    "https://www.youtube.com/watch?v=l7AjKUDyrCE")

# film noir: out_of_the_past
out_of_the_past = media.Movie(
    "Out of the Past",
    "1947",
    "A man
begins a new life in a small town, but has to confront his past.",
    "https://upload.wikimedia.org/wikipedia/en/a/a6/Outofthepast.jpg",
    "https://www.youtube.com/watch?v=fryuS6Zd\_mc")

# neo noir: chinatown
chinatown
= media.Movie(
    "Chinatown",
    "1974",
    "A detective tries to unravel a
conspiracy in 1937 Los Angeles.",
    "https://upload.wikimedia.org/wikipedia/en/3/38/Chinatownposter1.jpg",
    "https://www.youtube.com/watch?v=20FfiP7g4tU")

# neo noir: body_heat
body_heat
= media.Movie(
    "Body Heat",
    "1981",
    "An attorney meets a beautiful
woman and gets mixed up in a murder.",
    "https://upload.wikimedia.org/wikipedia/en/2/2c/Body\_heat\_ver1.jpg",
    "https://www.youtube.com/watch?v=cXGlUtoYtNE")
```

```
# future noir: blade_runner
blade_runner = media.Movie(
    "Blade Runner",
    "1982",
    "A detective
hunts renegade replicants in dystopian Los Angeles 2019.",
"https://upload.wikimedia.org/wikipedia/en/5/53/Blade_Runner_poster.jpg",
"https://www.youtube.com/watch?v=qoEyZo0Ttss")

# future noir: the_terminator
the_terminator = media.Movie(
    "The Terminator",
    "1984",
    "A combat
cyborg is sent back in time to stop its human enemy.",
"https://upload.wikimedia.org/wikipedia/en/7/70/Terminator1984movieposter.jpg",
"https://www.youtube.com/watch?v=-fN82upbGPo")

movies = [double_indegnity,
out_of_the_past, chinatown, body_heat,
    blade_runner, the_terminator]
fresh_tomatoes_noir.open_movies_page(movies)
```

*fresh\_tomatoes\_noir.py* is too long to include here (178 lines). See [GitHub](#).

# Lesson 10: Make Classes: advanced topics

---

Udacity Full Stack Web Developer Nanodegree program

Part 01. Programming fundamentals and the web [Programming foundations with Python]  
(<https://www.udacity.com/course/programming-foundations-with-python--ud036>)

Brendon Smith

br3ndonland

## ’ 01. Advanced Ideas in OOP

---

## ’ 02. Class Variables

---

- Class variables can be shared by all instances of the Class.
- The Google Style Guide recommends all caps for class variable names, like `VALID_RATINGS`.
- PEP 8 [recommends](#) CapWords for class names.

*Instructor notes:*

Around 0:40 or so, Kunal said with regard to the `valid_ratings` variable:

"...This is an array or a list..."

To avoid confusion down the road: Arrays and lists are actually not the same thing in Python. 99.9% of the time, you'll want to use lists. They're flexible, and good for pretty much every purpose.

However, there is also an `array.array` class in Python, which is essentially a thin wrapper on top of C arrays.

They're not something you'll probably need to use, but read more here if interested:<http://www.wired.com/2011/08/python-notes-lists-vs-arrays/>

## ’ 03. Quiz: Using

---

Predefined Class Variables

We explored three of the built-in class variables: `__doc__`, `__name__`, `__module__`.

```
print(media.Movie.__doc__) #  
documentation function  
print(media.Movie.__name__) # prints the class function  
name  
print(media.Movie.__module__) # prints module name
```

```
This class
provides a way to store movie information.
Movie
media
```

```
import webbrowser
webbrowser.__doc__
```

## ’04. Inheritance

This is another major topic in object-oriented programming. Think about it like a family. A child inherits characteristics and genomic material from their parents. Similarly, child classes can inherit information from parent classes

## ’05. Class Parent

We create a parent class, and reference it in a child instance. While watching, I was thinking that the parent class should be in a separate file. Kunal addressed this, and he just combined them for ease of demonstration.

```
# Udacity full-stack web developer
nanodegree
# 01. Programming fundamentals and the web
# Lesson 10. Make Classes:
Advanced Topics

class Parent():
    def __init__(self, last_name, eye_color):
print("Parent Constructor called.")
    self.last_name
self.eye_color

# Normally the child instance would be in a separate file.
#
Included here for ease of demonstration.
billy_cyrus = Parent("Cyrus", "blue")
print(billy_cyrus.last_name)
```

When I first ran this, I was getting `AttributeError: 'Parent' object has no attribute 'last_name'`. I hadn't finished defining parameters in the Parent Class.

```
# Udacity full-
stack web developer nanodegree
# 01. Programming fundamentals and the web
```

```
#  
Lesson 10. Make Classes: Advanced Topics
```

```
class Parent():  
    def  
        __init__(self, last_name, eye_color):  
            print("Parent Constructor  
called.")  
            self.last_name = last_name  
            self.eye_color = eye_color  
# Normally the child instance would be in a separate file.  
# Included here for  
ease of demonstration.  
billy_cyrus = Parent("Cyrus", "blue")  
print(billy_cyrus.last_name)
```

## ’06. Quiz: What's the Output?

---

The code seems highly redundant.

*What will the output be?*

```
Parent Constructor called  
Child Constructor called  
Cyrus  
5
```

*output:*

```
Parent Constructor called.  
Cyrus  
Child Constructor called.  
Parent Constructor called.  
Cyrus  
5
```

When computing the `miley_cyrus` function, the Child Constructor is first called. The Child calls the Parent. The output is then generated.

## ’07. Transitioning to Class Movie

---

## ’08. Updating the Design for Class

---

Movie

We revisit the code from our Class `Movie`. What if we wanted to add a similar Class for TV shows? Movies and TV shows share some characteristics, like the presence of a title and duration. We could create a parent function, `class Video():`, and group `class Movie(Video):` and `class TvShow(Video):` as child classes. Inheritance allows us to write code in an intuitive, human-readable way.

## 10. Reusing methods

---

Kunal went back to the `inheritance.py` file, and defined a new instance method inside `class Parent()` called `show_info`.

```
def show_info(self):
    print("Last Name - " +
self.last_name)
    print("Eye Color - " + self.eye_color)
```

He then referenced it with `billy_cyrus.show_info()`.

## 11. Method Overriding

---

The `show_info()` method is present inside the `class Parent()`. It can be created again inside the `class Child()`, and it will override the parent function.

# HTTP requests and responses

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

## [HTTP & Web Servers](#)

### Lesson 1. Requests & Responses

Also see [cs50 Lecture 06 HTTP](#)

## Table of Contents

---

- [Table of Contents](#)
- [Lesson](#)
  - [Intro](#)
  - [Your first web server](#)
  - [Parts of a URI](#)
  - [Hostnames and ports](#)
  - [HTTP GET requests](#)

## Lesson

---

### Intro

- We will use bash, Python 3, Git, and the nmap network testing toolkit.
- Installed nmap with `brew install nmap`.
- Used the `ncat` command to create a local tcp connection.
  - Terminal tab 1: `$ ncat -l 9999` (server)
  - Terminal tab 2: `$ ncat localhost 9999` (client)
  - The two Terminals will be able to communicate.

### Your first web server

- An HTTP transaction always involves a client and a server.
- HTTP was originally created to serve hypertext documents, but today is used for much more. As a user of the web, you're using HTTP all the time.
- Applications that you use in a web browser, and mobile services, are both likely to use HTTP. But low-level network tests such as ping do not.

- We used the Python `http.server` module as a demo webserver.

```
$ cd /Users/br3ndonland/Dropbox/Computing/udacity-fsnd/02-tools
$ python -m http.server 8000
```

- Browsing to <http://localhost:8000> shows the directory structure.
- We reviewed common HTTP status codes like `200` and `404`.

404 is the HTTP status code for "Not Found". On Highway 101, not far from the Udacity office in Mountain View, there's a sign that tells the distance to Los Angeles. As it happens, it's 404 miles from Mountain View to Los Angeles, so the sign says Los Angeles 404. And so, every web programmer in Silicon Valley has probably heard the "Los Angeles Not Found" joke at least once.

## Parts of a URI

- URI: Uniform Resource Identifier. Tells a web browser where to go.
- URL: URI for a network resource. [Full list](#).
- URI has three parts:
  - Scheme: https
  - Hostname: en.wikipedia.org.
  - Path: /wiki/Fish

## Hostnames and ports

- Domain Name Service/Server/System (DNS) translates IP addresses into the text addresses we see.
- IPv4 localhost is `127.0.0.1`.
- IPv6 localhost is `::1`, which is short for `0000:0000:0000:0000:0000:0000:0000:0001`.
- Some URIs imply the port number. HTTP implies port 80, HTTPS 443
- Information sent over the internet is divided into packets. Each packet contains the sending and receiving computer.

## HTTP GET requests

- All HTTP requests have an action verb:
  - GET
  - POST
  - PUT
  - PATCH
  - DELETE
- They also have the resource path requested, the HTTP protocol used, and the status code.

```
127.0.0.1 - - [05/Dec/2017 12:36:45] "GET /NotExistyFile HTTP/1.1" 404 -
127.0.0.1 - - [05/Dec/2017 14:32:10] "GET /img/ HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2017 14:32:16] "GET /img/fsnd02_10-github-pr-screenshot01.png
HTTP/1.1" 200 -
```

- We did a manual HTTP request to our Python Web Server, pressing enter twice after the `localhost` line. The response contains the status line, the headers, and the response body.

```
$ nc 127.0.0.1 8000
GET / HTTP/1.1
Host: localhost

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.6.3
Date: Tue, 05 Dec 2017 19:57:38 GMT
Content-type: text/html; charset=utf-8
Content-Length: 1107
```

(response body omitted)

- The server tells us a few pieces of information, including that the status is ok, and that the content returned is an HTML document written in UTF-8 text. The header ends with a blank line.
- We tried our own call and response.
  - `$ nc -l 9999`
  - Browse to <http://localhost:9999>/
  - The request shows up in the terminal
  - Type the following text into the terminal window where you started the server with `ncat`. Hit enter once after each line, then hit enter twice at the end.

```
HTTP/1.1 200 OK
Content-type: text/plain
Content-length: 50

Hello, browser! I am a real HTTP server, honestly!
```

- The message shows up in the browser window. Cool!

# The Web from Python

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[HTTP & Web Servers](#)

Lesson 2. The Web from Python

## Table of Contents

---

- [Table of Contents](#)
- [Lesson](#)
  - [Python's `http.server`](#)
  - [HTML and forms](#)
  - [GET and POST](#)
  - [A server for POST](#)
  - [Exercise: Messageboard, part one](#)
  - [Exercise: Messageboard, part two](#)
  - [POST-Redirect-GET](#)
  - [Making Requests](#)
  - [Using a JSON API](#)
  - [The Bookmark Server](#)

## Lesson

---

### Python's `http.server`

#### Intro

We got started at the end of the previous lesson by cloning a repo:

```
$ cd /Users/br3ndonland/Dropbox/Computing/udacity-fsnd/files/02_tools  
br3ndonland (master *) 02_tools  
$ git clone https://github.com/udacity/course-ud303  
Cloning into 'course-ud303'...  
remote: Counting objects: 401, done.  
remote: Total 401 (delta 0), reused 0 (delta 0), pack-reused 401  
Receiving objects: 100% (401/401), 56.75 KiB | 1.09 MiB/s, done.  
Resolving deltas: 100% (151/151), done.
```

```
br3ndonland (master *) 02_tools
$ cd course-ud303
br3ndonland (master) course-ud303
$ git remote remove origin
br3ndonland (master) course-ud303
$ ls
CODEOWNERS Lesson-1 Lesson-2 Lesson-3 README.md
```

In the last lesson, you used the built-in demo web server from the Python `http.server` module. But the demo server is just that — a demonstration of the module's abilities. Just serving static files out of a directory is hardly the only thing you can do with HTTP. In this lesson, you'll build a few different web services using `http.server`, and learn more about HTTP at the same time. You'll also use another module, `requests`, to write code that acts as an HTTP client.

These modules are written in object-oriented Python. You should already be familiar with creating class instances, defining subclasses, and defining methods on classes. If you need a refresher on the Python syntax for these object-oriented actions, you might want to browse the [Python tutorial on classes](#) or take another look at [the sections on classes in our Programming Foundations with Python course](#).

In the exercises in this lesson, you'll be writing code that runs on your own computer. You'll need the [starter code](#) that you downloaded at the end of the last lesson, which should be in a directory called `course-ud303`. And you'll need your favorite text editor to work on these exercises.

## › Servers and handlers

Web servers using `http.server` are made of two parts: the `HTTPServer` class, and a request handler class. The first part, the `HTTPServer` class, is built in to the module and is the same for every web service. It knows how to listen on a port and accept HTTP requests from clients. Whenever it receives a request, it hands that request off to the second part — a request handler — which is different for every web service.

Here's what your Python code will need to do in order to run a web service:

- Import `http.server`, or at least the pieces of it that you need.
- Create a subclass of `http.server.BaseHTTPRequestHandler`. This is your **handler class**.
- Define a method on the handler class for each **HTTP verb** you want to handle. (The only HTTP verb you've seen yet in this course is `GET`, but that will be changing soon.)
  - The method for `GET` requests has to be called `do_GET`.
  - Inside the method, call built-in methods of the handler class to read the HTTP request and write the response.
- Create an instance of `http.server.HTTPServer`, giving it your handler class and server information — particularly, the port number.
- Call the `HTTPServer` instance's `run_forever` method.

Once you call the `HTTPServer` instance's `run_forever` method, the server does that — it runs forever, until stopped. Just as in the last lesson, if you have a Python server running and you want to stop it, type Ctrl-C into the terminal where it's running. (You may need to type it two or three times.)

## › Exercise: The hello server

```
$ cd /Users/br3ndonland/Dropbox/Computing/udacity-fsnd/files/02_tools/course-ud303/Lesson-2/  
$ python helloserver.py
```

Navigate to <http://localhost:8000/> in a web browser.

## › A tour of the hello server

Open up `HelloServer.py` in your text editor. Let's take a look at each part of this code, line by line.

```
from http.server import HTTPServer, BaseHTTPRequestHandler
```

The `http.server` module has a lot of parts in it. For now, this program only needs these two. I'm using the `from` syntax of `import` so that I don't have to type `http.server` over and over in my code.

```
class HelloHandler(BaseHTTPRequestHandler):  
    def do_GET(self):
```

This is the handler class. It inherits from the `BaseHTTPRequestHandler` parent class, which is defined in `http.server`. I've defined one method, `do_GET`, which handles HTTP GET requests. When the web server receives a GET request, it will call this method to respond to it.

As you saw in the previous lesson, there are three things the server needs to send in an HTTP response: a status code, some headers, and the response body. The handler parent class has methods for doing each of these things. Inside `do_GET`, I just call them in order.

```
# First, send a 200 OK response.  
self.send_response(200)
```

The first thing the server needs to do is send a 200 OK status code; and the `send_response` method does this. I don't have to tell it that 200 means OK; the parent class already knows that.

```
# Then send headers.  
self.send_header('Content-type', 'text/plain; charset=utf-8')  
self.end_headers()
```

The next thing the server needs to do is send HTTP headers. The parent class supplies the `send_header` and `end_headers` methods for doing this. For now, I'm just having the server send a single header line — the Content-type header telling the client that the response body will be in UTF-8 plain text.

```
# Now, write the response body.  
self.wfile.write("Hello, HTTP!\n".encode())
```

The last part of the `do_GET` method writes the response body.

The parent class gives us a variable called `self.wfile`, which is used to send the response. The name `wfile` stands for writeable file. Python, like many other programming languages, makes an analogy between network connections and open files: they're things you can read and write data to. Some file objects are read-only; some are write-only; and some are read/write.

`self.wfile` represents the connection from the server to the client; and it is write-only; hence the name. Any binary data written to it with its `write` method gets sent to the client as part of the response. Here, I'm writing a friendly hello message.

What's going on with `.encode()` though? We'll get to that in a moment. Let's look at the rest of the code first.

```
if __name__ == '__main__':  
    server_address = ('', 8000) # Serve on all addresses, port 8000.  
    httpd = HTTPServer(server_address, HelloHandler)  
    httpd.serve_forever()
```

This code will run when we run this module as a Python program, rather than importing it. The `HTTPServer` constructor needs to know what address and port to listen on; it takes these as a tuple that I'm calling `server_address`. I also give it the `HelloHandler` class, which it will use to handle each incoming client request.

At the very end of the file, I call `serve_forever` on the `HTTPServer`, telling it to start handling HTTP requests. And that starts the web server running.

## › End of the tour

That's all that's involved in writing a basic HTTP server in Python. But the hello server isn't very interesting. It doesn't even do as much as the demo server. No matter what query you send it, all it has to say is hello. (Try it: <http://localhost:8000/goodbye>)

In the rest of this lesson, we'll build servers that do much more than say hello.

## › What about `encode()`

If you want to send a string over the HTTP connection, you have to `encode` the string into a `bytes` object. The `encode` method on strings translates the string into a `bytes` object, which is suitable for sending over the network. There is, similarly, a `decode` method for turning `bytes` objects into strings.

See the [Python Unicode HOWTO](#) for more.

## › The echo server

Creating a server that reads request information and echoes it back.

From [course-ud303/Lesson-2/1\\_EchoServer/README.md](#):

In this exercise, you'll take the code from the \*hello server- and change it into the *echo server*. This new server will also listen on port 8000, but it will respond to GET requests by repeating back ("echoing") the text of the request path.

See [EchoServer.py](#) for starter code.

To test your code, you'll need two terminals open. In one of them, run the server (with `python EchoServer.py`). You can then access it from your browser, for instance at <http://localhost:8000/GoodMorningHTTP>. In the other terminal, run the test script provided (`python test.py`). The test script will send a request to the server and tell you whether it worked.

We had to modify the HelloHandler module to return the path instead of just "Hello, HTTP!\n". Creation of the echo server required a one-line change at the end of do\_GET:

```
self.wfile.write(self.path[1:].encode())
```

What I'm doing here is taking the path (for instance "/bears"), using a string slice to remove the first character (which is always a slash), and then encoding the resulting string into bytes, then writing that to the HTTP response body.

I was close on my first try. I just needed to add `self`. Note that I was actually able to make my code more concise by removing the `[1:]`. I didn't need to slice the string.

```
self.wfile.write(self.path.encode())
```

Run the server in the terminal with

```
cd /Users/br3ndonland/Dropbox/Computing/udacity-fsnd/files/02_tools/course-ud303/Lesson-2/1_
python EchoServer.py
```

If you attempt to run two servers on the same port,

The new server exits. Under normal conditions, only one program on your computer can listen on a particular port at the same time. If you want to have both servers running, you have to change the port number from 8000 to something else.

## › Queries and quoting

## › Unpacking query parameters

Queries are entered after a ? in the URI.

The URI <https://www.google.com/search?q=gray+squirrel&tbo=isch> will be sent to the server as an HTTP request:

```
GET /search?q=gray+squirrel&tbo=isch HTTP/1.1  
Host: www.google.com
```

The `urllib.parse` Python library can do exactly that, parse URLs.

I followed along with the demo from the course notes:

```
>>> from urllib.parse import urlparse, parse_qs  
>>> address = 'https://www.google.com/search?q=gray+squirrel&tbo=isch'  
>>> parts = urlparse(address)  
>>> print(parts)  
ParseResult(scheme='https', netloc='www.google.com', path='/search', params='', query='q=gra  
>>> print(parts.query)  
q=gray+squirrel&tbo=isch  
>>> query = parse_qs(parts.query)  
>>> query  
{'q': ['gray squirrel'], 'tbo': ['isch']}
```

What does `parse_qs('texture=fuzzy&animal=gray+squirrel')` return?

The dictionary `{'texture': ['fuzzy'], 'animal': ['gray squirrel']}`

## › URL quoting

- Spaces can't be sent in HTTP requests, so they are translated into an appropriate format. This is called URL quoting, URL encoding, or URL escaping.
- Docs for `urllib.parse.quote`

## › HTML and forms

- Run `Lesson-2/1_EchoServer/EchoServer.py` in the terminal
- Open `Lesson-2/2_HTMLForms/LoginPage.html` in a web browser
- Submit user name and password
- Query string is echoed in the output.
- Of course, we now have more advanced ways of inputting passwords.

## › GET and POST

- GET puts all form fields into the URI that is sent to the server. Good for search strings, not for more complex actions.
- POST is better when altering or creating a resource.

## › Idempotence

Vocabulary word of the day: idempotent. An action is idempotent if doing it twice (or more) produces the same result as doing it once. "Show me the search results for 'polar bear'" is an idempotent action, because doing it a second time just shows you the same results. "Add a polar bear to my shopping cart" is not, because if you do it twice, you end up with two polar bears.

POST requests are not idempotent. If you've ever seen a warning from your browser asking you if you really mean to resubmit a form, what it's really asking is if you want to do a non-idempotent action a second time.

(Important note if you're ever asked about this in a job interview: idempotent is pronounced like "eye-dem-poe-tent", or rhyming with "Hide 'em, Joe Tent" — not like "id impotent".)

Adding zero to a number is idempotent, since you can add zero as many times as you want and the original number is unchanged. Adding five to a number is not idempotent, because if you do it twice you'll have added ten. Setting a variable to the value 5 is idempotent: doing it twice is the same as doing it once. Looking up an entry in a dictionary doesn't alter anything, so it's idempotent.

## › Serve a POST request

- As before, run `ncat -l 9999` in a terminal window.
- Then, open *PostForm.html*, enter some text in each box, and click "Do a thing!"
- Return to the terminal and look at the output. The output contains the information that was submitted in the boxes.
- The search string is not added to the URI with POST.
- Note that HTTP headers are case-insensitive.

```
$ ncat -l 9999
POST / HTTP/1.1
Host: localhost:9999
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:57.0) Gecko/20100101 Firefox/57
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Cookie: username-localhost-8888="2|1:0|10:1512181984|23:username-localhost-8888|44:YWVizjc3M
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1

magic=DNA&secret=double+helix
```

## › A server for POST

We built a message board server with [Requests](#).

When I first used Requests for the profanity checker mini-project (FSND Part 01, Lesson 08), I didn't understand it very well because I didn't understand HTTP. It's good to revisit Requests now.

I already had Requests installed via Anaconda.

Yes! We'll be using a GET request to display the messageboard's existing contents, and POST to update the contents by creating new messages. Creating new messages is not idempotent — we don't want duplicates.

### › POST handlers read the request body

The *EchoServer.py* had a `do_GET` "handler" class in it. We will use `do_POST` now. `self.rfile.read` reads the request body. `self.wfile`, which we have already seen, writes the response. The length of the request body is sent with the `Content-Length` header.

### › Headers are strings (or missing)

`self.headers` is an object like a Python dictionary.

This Python code will go in the `do_POST` handler class. `length` stores the length of the request body, and `data` reads it.

```
length = int(self.headers.get('Content-length', 0))
data = self.rfile.read(length).decode()
```

`urllib.parse.parse_qs` can then be used to extract the POST parameters.

## › Exercise: Messageboard, part one

### › Setup

The starter code for this exercise is in *Lesson-2/3\_MessageboardPartOne*. It's an echo server for post requests.

*Build the code:*

1. \*Find the length of the request data.- This was easy. I just used the code for the `length` variable above.
2. \*Read the correct amount of request data.- Used the `data` variable.
3. \*Extract the "message" field from the request data.- We went through the use of the `urllib.parse.parse_qs` module in the [Queries and quoting](#) section, so I reviewed that material. The `self.wfile.write(message.encode())` line references a variable `message`, so I started by creating a variable `message` and calling `parse_qs`.

*Run the code:*

Same as for the echo server:

1. Start the Python HTTP server:

```
cd ../3_MessageboardPartOne  
python MessageboardPartOne.py
```

2. In a second terminal window, run the test file to check the code:

```
cd ../3_MessageboardPartOne  
python test.py
```

3. Open *Messageboard.html* in a web browser.

## › Troubleshooting

- I first had `message = parse_qs(data)`. The server terminal window kept giving me `AttributeError: 'MessageHandler' object has no attribute 'parse_qs'`. I realized I had to restart the server every time I changed *MessageboardPartOne.py*. After a Cntrl+C restart, I got:

```
$ python test.py  
Testing connecting to the server.  
Connection attempt succeeded.  
Testing POST request.  
The server sent a 200 OK response, but the content differed.  
I expected 'Hi there!' but it sent ''.
```

- I then got a different error: `AttributeError: 'dict' object has no attribute 'encode'`.
- I checked the solution at this point. I was almost there! I just needed to add `["message"]  
[0] : message = parse_qs(data)["message"][0]`.
  - There is a line in `*test.py`- that helped me understand this:

```
def test_POST():  
    '''The server should accept a POST and return the "message" field.'''  
    print("Testing POST request.")  
    mesg = random.choice(["Hi there!", "Hello!", "Greetings!"])  
    uri = "http://localhost:8000/"  
    try:  
        r = requests.post(uri, data = {'message': mesg})
```

- \*TODO: WHY?- I still don't totally understand how he got there.
- I then opened *Messageboard.html* in a web browser, entered some text, clicked post, and it showed up on the webpage.

## › Exercise: Messageboard, part two

- Handle POST requests and serve the webpage from a single Python file.

- This reminded me of project 1 (The Python Web Server), so I reviewed the code from that, and added the HTML from \*Messageboard.html- as a variable in single quotes.
- The Udacity code needed docstrings for the class and function definitions, and an extra line break after the class definition.

*Run the code:*

1. Start the Python HTTP server:

```
cd ../4_MessageboardPartTwo
python MessageboardPartTwo.py
```

2. In a second terminal window, run the test file to check the code:

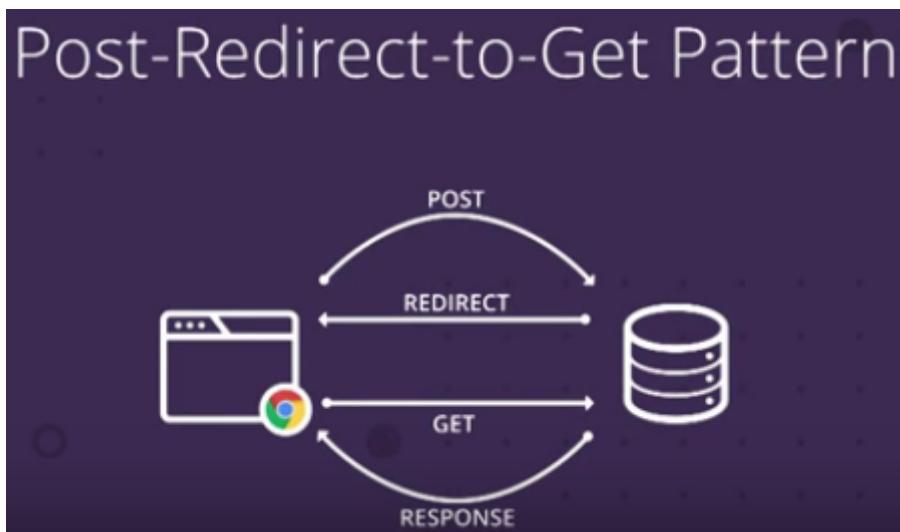
```
cd ../3_MessageboardPartTwo
python test.py
```

3. Navigate to <http://localhost:8000/> in a web browser.

On my first try, the browser just showed the HTML code itself. I simply needed to change the request from 'text/plain' to 'text/html'. Success!

This particular server doesn't look at the URI path at all. Any GET request will get the form. Any POST request will save a message.

## › POST-Redirect-GET



- Client POSTs to server
- Server replies with a 303 redirect
- Client GETs the updated resource

Same steps to run the code as Messageboard part 2:

1. Start the Python HTTP server:

```
cd ../4_MessageboardPartTwo  
python MessageboardPartTwo.py
```

2. In a second terminal window, run the test file to check the code:

```
cd ../3_MessageboardPartTwo  
python test.py
```

3. Navigate to <http://localhost:8000/> in a web browser.

Again, I almost got it on my first try. The thing I didn't get was putting together the form and response. I tried to combine the variables with

```
response = messageboard_form + memory
```

which returned the following error (visible in the terminal window in which I started the server)

```
handle_one_request  
    method()  
File "MessageboardPartThree.py", line 65, in do_GET  
    response = messageboard_form + memory  
TypeError: must be str, not list
```

The solution code had

```
mesg = form.format("\n".join(memory))`
```

I need to review how to append variables.

## › Making Requests

- Finally moving to the Requests module!
- The instructor first had us look at the [Requests quickstart docs](#). Already had the site open!
- *How would you send a GET request to the messageboard server?*
  - `requests.get("http://localhost:8000/")`
  - The `requests` function for performing GET requests is `requests.get`, and it takes the URI as an argument.
- Requests return a `response` object:

```
$ python  
Python 3.6.3 |Anaconda, Inc.| (default, Oct  6 2017, 12:04:38)  
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
```

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> a = requests.get('http://www.udacity.com')
>>> a
<Response [200]>
>>> type(a)
<class 'requests.models.Response'>
```

*Use the documentation for the `requests` module to answer this question! If you have a response object called `r`, how can you get the response body — for instance, the HTML that the server sent?*

Both, but they're different. `r.content` is a bytes object representing the literal binary data that the server sent. `r.text` is the same data but interpreted as a str object, a Unicode string.

*Using the `requests` module, try making GET requests to nonexistent sites or pages, e.g. `http://bad.example.com` or `http://google.com/NotExist`.*

```
python
>>> uri = "http://google.com/NotExist"
>>> r = requests.get(uri)
>>> print(r.status_code)
```

```
>>> no_uri = 'http://bad.example.com/'
>>> r_no_uri = requests.get(no_uri)
```

If the `requests.get` call can reach an HTTP server at all, it will give you a Response object. If the request failed, the Response object has a `status_code` data member — either 200, or 404, or some other code.

But if it wasn't able to get to an HTTP server, for instance because the site doesn't exist, then `requests.get` will raise an exception.

However: Some Internet service providers will try to redirect browsers to an advertising site if you try to access a site that doesn't exist. This is called [DNS hijacking](#), and it's nonstandard behavior, but some do it anyway. If your ISP hijacks DNS, you won't get exceptions when you try to access nonexistent sites. Standards-compliant DNS services such as [Google Public DNS](#) don't hijack.

## Using a JSON API

### JSON Intro

- JSON is based on JavaScript syntax, and often used for APIs.
- Access Star Wars API:

```
>>> a = requests.get('http://swapi.co/api/people/1/')
>>> a.json()['name']
'Luke Skywalker'
```

- We used the website <http://uinames.com/api> to generate fake user information. It takes two query parameters: `ext`, providing extended info, and `region`, specifying desired country.

## Exercise: Use JSON with UINames.com

- On my first try, I tried putting the JSON field names directly within the brackets like

```
return "My name is {'name'} {'surname'} and the PIN on my card is {'pin'}.".format()
```

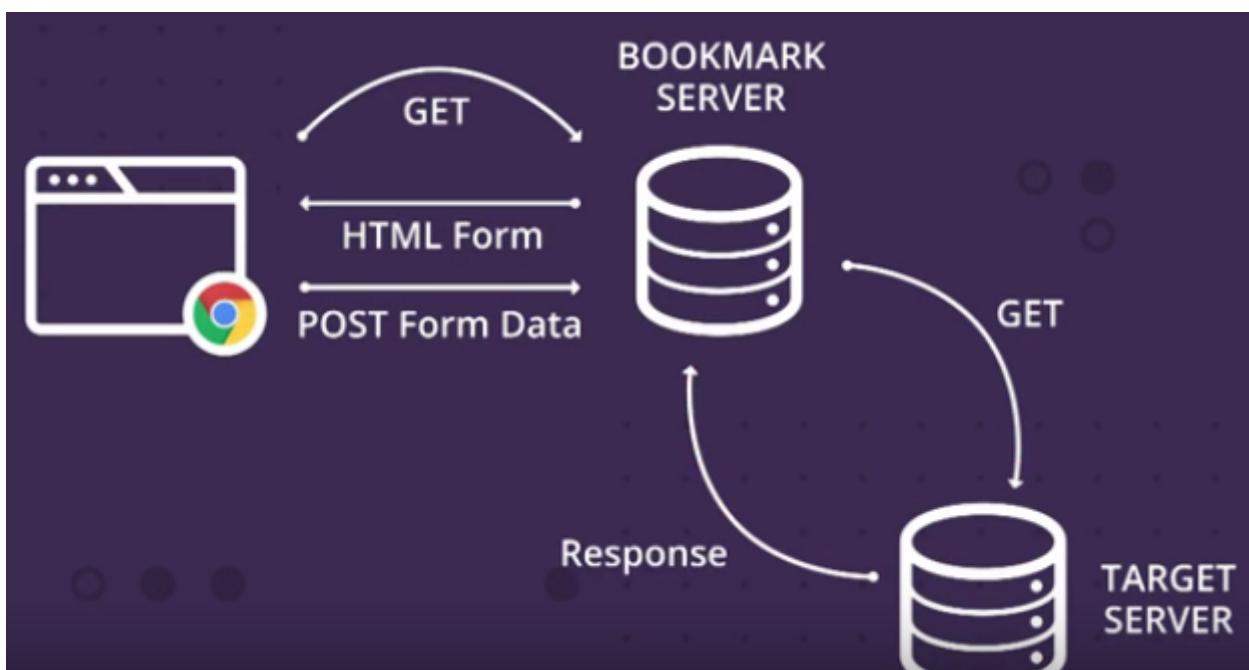
- On the second try, I moved the JSON field names into `format()`:

```
return "My name is {} {} and the PIN on my card is {}".format(
    'name', 'surname', 'pin')
```

- This actually passed when I ran `python test.py` in the terminal, but it wasn't correct, because it was just outputting those literal titles. I needed to extract the data from `r.json()`. This can be done in two ways:

```
# no added variable
r.json()['name'], r.json()['surname'], r.json()['credit_card']['pin']
# add r.json as variable
j = r.json()
j['name'], j['surname'], j['credit_card']['pin']
```

## The Bookmark Server



We made a URL-shortening web server, like TinyURL or Google's goo.gl.

The server does three things, depending on the request received:

1. GET request to the `/` path: display a form with two fields for long and short URLs, and submit POST request to server when user submits form
2. POST:
  - i. Look for form fields in request body
  - ii. Check URI with `requests.get`
    - a. Return 200 if successful
    - b. Return 404 if unsuccessful
    - c. Return 400 if either form field is missing
3. GET request to a short URI: look up long URL and serve redirect.

Steps:

1. Start the Python HTTP server:

```
cd ../4_MessageboardPartTwo  
python MessageboardPartTwo.py
```

2. In a second terminal window, run the test file to check the code:

```
cd ../3_MessageboardPartTwo  
python test.py
```

3. Navigate to <http://localhost:8000/> in a web browser.

I struggled with this a bit, but I basically got it without checking the solution. There were just a few differences:

CheckURI function:

I had

```
def CheckURI(uri, timeout=5):  
    '''Check whether this URI is reachable, i.e. does it return a 200 OK?  
  
    This function returns True if a GET request to uri returns a 200 OK, and  
    False if that GET request returns any other response, or doesn't return  
(i.e. times out).  
    ...  
    r = requests.get(uri)  
  
    if r.status_code == 200:  
        return True  
    else:  
        return False
```

The solution had

```

def CheckURI(uri, timeout=5):
    '''Check whether this URI is reachable, i.e. does it return a 200 OK?

    This function returns True if a GET request to uri returns a 200 OK, and
    False if that GET request returns any other response, or doesn't return
    (i.e. times out).
    ...
    try:
        r = requests.get(uri, timeout=timeout)
        # If the GET request returns, was it a 200 OK?
        return r.status_code == 200
    except requests.RequestException:
        # If the GET request raised an exception, it's not OK.
        return False

```

I added each to different cells of a Jupyter Notebook and ran them. They both worked.

do\_GET function:

I had `memory[name]` in quotes as a string, but it needed to be unquoted.

I ran my server, navigated to <http://localhost:8000/> and the server sort of worked! My code wouldn't pass \*test.py- and although the website worked, I was getting some command-line errors:

```

127.0.0.1 - - [14/Dec/2017 17:29:12] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Dec/2017 17:29:12] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [14/Dec/2017 17:29:12] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [14/Dec/2017 17:30:25] "POST / HTTP/1.1" 303 -
127.0.0.1 - - [14/Dec/2017 17:30:25] "GET / HTTP/1.1" 200 -
^CTraceback (most recent call last):
  File "bookmarkserver-br3ndonland.py", line 112, in <module>
    httpd.serve_forever()
  File "/Users/br3ndonland/anaconda/lib/python3.6/socketserver.py", line 236, in
serve_forever
    ready = selector.select(poll_interval)
  File "/Users/br3ndonland/anaconda/lib/python3.6/selectors.py", line 376, in select
    fd_event_list = self._poll.poll(timeout)

```

# HTTP in the real world

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

## [HTTP & Web Servers](#)

Lesson 3. HTTP in the Real World: Deploying to Heroku

### Table of Contents

---

- [Table of Contents](#)
- [Lesson](#)
  - [Deploying to a hosting service](#)
  - [Static content and more](#)
  - [Caching](#)
  - [Cookies](#)
  - [HTTPS for security](#)
  - [Beyond GET and POST](#)
  - [New developments in HTTP](#)
  - [Keep learning](#)
- [Feedback](#)

### Lesson

---

#### Deploying to a hosting service

#### Can I just host my web service at home

Maybe! Plenty of people do, but not everyone can. It's a common hobbyist activity, but not something that people would usually do for a job.

There's nothing fundamentally special about the computers that run web servers. They're just computers running an operating system such as Linux, Mac OS, or Windows (usually Linux). Their connection to the Internet is a little different from a typical home or mobile Internet connection, though. A server usually needs to have a stable (static) IP address so that clients can find it and connect to it. Most home and mobile systems don't assign your computer a static IP address.

Also, most home Internet routers don't allow incoming connections by default. You would need to reconfigure your router to allow it. This is totally possible, but way beyond the scope of this course (and I don't know what kind of router you have).

Lastly, if you run a web service at home, your computer has to be always on.

So, for the next exercise in this course, you'll be deploying one of your existing web services to Heroku, a commercial service that will host it on the web where it will be publicly accessible.

## › Steps to deployment

Here's an overview of the steps you'll need to complete. We'll be going over each one in more detail.

1. Check your server code into a new local Git repository.
2. Sign up for a free Heroku account.
3. Download the Heroku command-line interface (CLI).
4. Authenticate the Heroku CLI with your account: `heroku login`
5. Create configuration files *Procfile*, *requirements.txt*, and *\*runtime.txt*- and check them into your Git repository.
6. Modify your server to listen on a configurable port.
7. Create your Heroku app: `heroku create your-app-name`
8. Push your code to Heroku with Git: `git push heroku master`

### › 1. Check your server code into a new local Git repository

I created *udacity-fsnd/files/02\_tools/fsnd02\_12-http-python*, copied in my version and Udacity's version of the bookmark server code, along with the *README.md*, and checked it in with Git.

## › Heroku Python tutorial

- I signed up for Heroku
- I went to the [Heroku Python getting started guide](#). The Python web app instructions are different than the CLI instructions used in this lesson, but I thought they could be helpful for future work with Flask and Django, so I branched from the lesson here and walked through the Python instructions.
- `pip install pipenv`
- Installed [Postgres](#), initialized a new server by clicking "initialize," and configured `$PATH` to use command line tools

```
sudo mkdir -p /etc/paths.d &&
echo /Applications/Postgres.app/Contents/Versions/latest/bin | sudo tee /etc/paths.d/postgres.d
```

- Installed CLI. May have installed this three times.

- Toolbelt: Did this during the Heroku Python installation above. The download wouldn't start. I opened the Mac OS X link in a new tab, which actually took me to a [JSON](#) listing the [link to the toolbelt package](#). I pasted that link into a new window and downloaded that, and ran the installation.
- I then found that I could install with direct download (of a different CLI file apparently), or through Homebrew. I installed the direct download.
- I then ran `brew install heroku/brew/heroku`, which asked me to overwrite the previous installation. I did. See `*udacity-fsnd_02_13-http-heroku-terminal.md-` for full iTerm 2 Terminal logs.
- Deployed app as detailed in the [Heroku Python getting started guide](#). Apps run in Linux containers called [dynos](#).
- Tried scaling with `$ heroku ps:scale web=0` (not running) vs. `$ heroku ps:scale web=1`. *Why do you need more dynos? To handle more traffic?*
- [Declare app dependencies](#):
  - This is where we got started with Python-specific stuff.
  - Heroku recognizes an app as a Python app by the existence of a Pipfile or requirements.txt file in the root directory.
- Run the app locally:
  - The app is almost ready to start locally. Django uses local assets, so first, you'll need to run collectstatic: `$ python manage.py collectstatic`
  - `heroku local`
  - Navigate to <http://localhost:5000/>.
- Define config vars: I started getting errors with `heroku local`. I committed and pushed, and it worked online, so it must be an issue with something related that is installed locally. `gunicorn` and `django` were not found.
- Provision a database: created database tables with the standard Django `manage.py migrate` to create the tables.
  - The code to access the database is straightforward, and makes use of a simple Django model called Greetings that you can find in hello/models.py.
- Done! Next, I can work through [Deploying Python and Django Apps on Heroku](#), and maybe deploy my movie trailer python web server.

## › Heroku Python tutorial from Udacity

- Create configuration files *Procfile*, *requirements.txt*, and *\*runtime.txt-* and check them into your Git repository.
  - *requirements.txt*: `requests>=2.12`
  - *runtime.txt*: `python-3.6.3`
  - *Procfile*: `web: python BookmarkServer-udacity.py`

- `$ git commit -m "Add Heroku configuration files"`
- Modify your server to listen on a configurable port.
  - Modified the Python file to make the port configurable

```
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 8000)) # Use PORT if it's there.
    server_address = ('', port)
    httpd = http.server.HTTPServer(server_address, Shortener)
    httpd.serve_forever()
```

- `$ git commit -m "Use PORT from environment"`
- Navigated to <http://localhost:8000/> to check servers
- Create your Heroku app

```
$ heroku create bookmarkserver-br3ndonland
Creating ⚡ bookmarkserver-br3ndonland... done
https://bookmarkserver-br3ndonland.herokuapp.com/ | https://git.heroku.com/bookmarkserve...
```

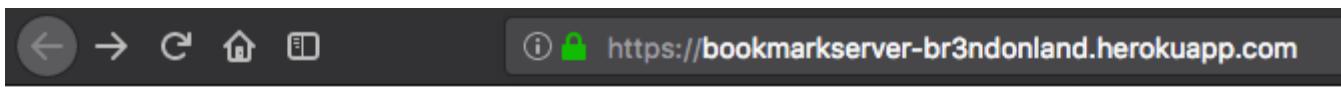
```
$ heroku create bookmarkserver-udacity
Creating ⚡ bookmarkserver-udacity... done
https://bookmarkserver-udacity.herokuapp.com/ | https://git.heroku.com/bookmarkserver-uc...
```

- Push your code to Heroku with Git: `git push heroku master`

Tried it with both versions of the Bookmark Server: mine and Udacity's. Created separate directories for each.

- Test websites:
  - [Google Scholar](#) did not work.
    - br3ndonland got a Heroku Application Error: "An error occurred in the application and your page could not be served. If you are the application owner, check your logs for details."
    - Udacity got "Couldn't fetch URI '<https://scholar.google.com/citations?user=ZJ2yZa8AAAAJ&hl=en>'. Sorry!"
  - [DuckDuckGo](#) worked in both!
  - [Bear](#) worked in both!

Here's how it looks on Heroku:



Long URL:

Short name:

### URLs I know about:

bear : <http://www.bear-writer.com/>  
duckduckgo : <https://duckduckgo.com/>

### › Heroku URLs

- <https://bookmarkserver-br3ndonland.herokuapp.com/>
- <https://bookmarkserver-udacity.herokuapp.com/>
- <https://guarded-ridge-60265.herokuapp.com/>

### › Handling more requests

The instructor explains the issue in this section:

That's right! The basic, built-in `http.server.HTTPServer` class can only handle a single request at once. The bookmark server tries to fetch every URI that we give it, while it's in the middle of handling the form submission.

It's like an old-school telephone that can only have one call at once. Because it can only handle one request at a time, it can't "pick up" the second request until it's done with the first ... but in order to answer the first request, it needs the response from the second.

We needed to build in concurrency support by adding a mixin to the `HTTPServer` class.

Two additions to bookmark server Python files:

Top of file:

```
import threading
from socketserver import ThreadingMixIn

class ThreadHTTPServer(ThreadingMixIn, http.server.HTTPServer):
    "This is an HTTPServer that supports thread-based concurrency."
```

Bottom of file:

```
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 8000))
    server_address = ('', port)
    httpd = ThreadHTTPServer(server_address, Shortener)
    httpd.serve_forever()
```

```
$ git commit -m "Add HTTPServer mixin that supports thread-based concurrency"  
$ git push heroku master
```

Worked on my first try!

Long URI:

Short name:

URIs I know about:

[bookmarkserver-br3ndonland : https://bookmarkserver-br3ndonland.herokuapp.com/](https://bookmarkserver-br3ndonland.herokuapp.com/)

Still got the error with Google Scholar though.

## › Static content and more

The Web was originally designed to serve documents, not to deliver applications. Even today, a large amount of the data presented on any web site is static content — images, HTML files, videos, downloadable files, and other media stored on disk.

That's a great point. While the internet has scaled remarkably since Vint Cerf and Stephen Crocker first established TCP/IP, it is true that it is largely for static content. We need a new distributed internet, like a p2p bit torrent network for the entire internet (like the one Richard tries to build in Silicon Valley), focused on functional web apps over static web content. The closest thing I've seen to this is the new WebAssembly project, designed to allow use of other languages than JavaScript.

## › What's an Apache or Nginx

Nginx is pronounced "engine x".

## › Routing and load balancing

- *Request routing*, or *reverse proxying*, is when a specialized server allocates web requests to the appropriate backend servers.
- Splitting web requests among multiple servers is called *load balancing*.

## › Concurrent users

- I can definitely understand the challenges here. At any time, there could be thousands of concurrent requests to the servers.
- Requests to the server that have not been resolved are called *in-flight requests*.

*Question 1 of 2* In September 2016, the English Wikipedia received about 250 million page views per day. That's an average of about 2,900 page views every second. Let's imagine that an average page view involves three HTTP queries (the page HTML itself and two images), and that each HTTP query takes 0.1 seconds (or 100 milliseconds) to serve. About how many requests are in flight at any instant?

Let's solve this in Python! The question wasn't really specific enough (what's an instant?), but I was assuming an "instant" was a second. This just makes it simple arithmetic:

```
$ python
>>> num_requests = 2900 / 3 / 0.1
>>> print(num_requests)
870.0
```

## › Caching

- Caching is temporary storage of frequently-accessed information.
- They linked out to [Google Web Fundamentals: HTTP caching](#)

*Question 2 of 2* Imagine that you have a service that is handling 6,000 requests per second. One-third of its of requests are for the site's CSS file, which doesn't change very often. So browsers shouldn't need to fetch it every time they load the site. If you tell the browser to cache the CSS, 1% of visitors will need to fetch it. After this change, about how many requests will the service be getting?

This was just easy mental math. CSS requests are reduced to  $6000 / 3 - 0.01 = 20$ . The website is still getting the other 4000 requests, plus the 20 for the CSS, so 4020.

2,000 requests per second are the CSS file, so the other 4,000 requests are other things. Those 4,000 will be unaffected by this change.

The 2,000 CSS requests will be reduced by 99%, to 20 requests.

This means that after the caching improvement, the service will be getting 4,020 requests per second.

## › Cookies

- Cookies ask the browser to retain information.
- Cookies allow users to store website preferences.
- Cookies allow the server to retain information, and implement sessions and logins.

## › How cookies happen

- The server sets the cookie at the first browser request.
- The browser then sends back the cookie with each subsequent request in the HTTP headers.

## › Seeing cookies in your browser

- The cookie's name and content are a key: value store.
- Domain and path set the webpages on which the cookie can be set.

Browsers don't make it easy to find cookies that have been set, because removing or altering cookies can affect the expected behavior of web services you use. However, it is possible to inspect cookies from sites you use in every major browser. Do some research on your own to find out how to view the cookies that your browser is storing.

The first two, the cookie's name and content, are also called its key and value. They're analogous to a dictionary key and value in Python — or a variable's name and value for that matter. They will both be sent back to the server. There are some syntactic rules for which characters are allowed in a cookie name; for instance, they can't have spaces in them. The value of the cookie is where the "real data" of the cookie goes — for instance, a unique token representing a logged-in user's session.

The next two fields, Domain and Path, describe the scope of the cookie — that is to say, which queries will include it. By default, the domain of a cookie is the hostname from the URI of the response that set the cookie. But a server can also set a cookie on a broader domain, within limits. For instance, a response from [www.udacity.com](http://www.udacity.com) can set a cookie for udacity.com, but not for com.

The fields that Chrome describes as "Send for" and "Accessible to script" are internally called Secure and HttpOnly, and they are boolean flags (true or false values). The internal names are a little bit misleading. If the Secure flag is set, then the cookie will only be sent over HTTPS (encrypted) connections, not plain HTTP. If the HttpOnly flag is set, then the cookie will not be accessible to JavaScript code running on the page.

Finally, the last two fields deal with the lifetime of the cookie — how long it should last. The creation time is just the time of the response that set the cookie. The expiration time is when the server wants the browser to stop saving the cookie. There are two different ways a server can set this: it can set an Expires field with a specific date and time, or a Max-Age field with a number of seconds. If no expiration field is set, then a cookie is expired when the browser closes.

## › Using cookies in Python

To set a cookie from a Python HTTP server, all you need to do is set the Set-Cookie header on an HTTP response. Similarly, to read a cookie in an incoming request, you read the Cookie header. However, the format of these headers is a little bit tricky; I don't recommend formatting them by hand. Python's http.cookies module provides handy utilities for doing so.

To create a cookie on a Python server, use the SimpleCookie class. This class is based on a dictionary, but has some special behavior once you create a key within it:

```
from http.cookies import SimpleCookie, CookieError

out_cookie = SimpleCookie()
out_cookie["bearname"] = "Smokey Bear"
out_cookie["bearname"]["max-age"] = 600
out_cookie["bearname"]["httponly"] = True
```

Then you can send the cookie as a header from your request handler:

```
self.send_header("Set-Cookie", out_cookie["bearname"].OutputString())
```

To read incoming cookies, create a SimpleCookie from the Cookie header:

```
in_cookie = SimpleCookie(self.headers["Cookie"])
in_data = in_cookie["bearname"].value
```

Be aware that a request might not have a cookie on it, in which case accessing the Cookie header will raise a KeyError exception; or the cookie might not be valid, in which case the SimpleCookie constructor will raise http.cookies.CookieError.

Important safety tip: Even though browsers make it difficult for users to modify cookies, it's possible for a user to modify a cookie value. Higher-level web toolkits, such as Flask (in Python) or Rails (in Ruby) will cryptographically sign your cookies so that they won't be accepted if they are modified. Quite often, high-security web applications use a cookie just to store a session ID, which is a key to a server-side database containing user information.

Another important safety tip: If you're displaying the cookie data as HTML, you need to be careful to escape any HTML special characters that might be in it. An easy way to do this in Python is to use the `html.escape` function, from the built-in `html` module!

## › Exercise: A server that remembers you

### › Set the fields of the cookie

- My initial attempts were unsuccessful. I was repeatedly getting the following:

```
$ python test.py
Testing connecting to the server.
Connection attempt succeeded.
Testing GET request.
GET request without cookie succeeded!
Testing POST request, looking for redirect & cookie.
Couldn't communicate with the server. (('Connection aborted.', RemoteDisconnected('Remot
If it's running, take a look at its output.
```

- The `Traceback` was difficult to interpret, but the two lines I could recognize were

```
File "cookieserver.py", line 49, in do_POST
    c['domain'] = 'localhost'
http.cookies.CookieError: Attempt to set a reserved key 'domain'
```

- I didn't set the cookie properly. I had

```
# 1. Set the fields of the cookie.  
#     Give the cookie a value from the 'yourname' variable,  
#     a domain (localhost), and a max-age.  
c['value'] = yourname  
c['domain'] = 'localhost'  
c['max_age'] = 60
```

- and the solution had

```
c['yourname'] = yourname  
c['yourname']['domain'] = 'localhost'  
c['yourname']['max-age'] = 60
```

#### ↳ Extract and decode the cookie

- Got this one right!

```
# 2. Extract and decode the cookie.  
#     Get the cookie from the headers and extract its value  
#     into a variable called 'name'.  
in_cookie = SimpleCookie(self.headers['cookie'])  
name = in_cookie['yourname'].value
```

- After I modified the cookie setting syntax in 1, the code passed *test.py*.
- When I ran *cookieserver.py* and navigated to <http://localhost:8000/> I was able to enter my name, and the name was stored and shown.

The screenshot shows a web browser window with the URL `localhost:8000`. The page content includes a greeting, a form field, a button, and a cookie management interface.

Hey there, br3ndonland

What's your name again?

[Tell me!](#)

**Cookies** X

localhost X

The following cookies match your search:

| Site      | Cookie Name |
|-----------|-------------|
| localhost | yourname    |

Name: yourname  
Content: br3ndonland  
Host: localhost  
Path: /  
Send For: Any type of connection  
Expires: December 16, 2017 at 7:00:43 PM EST

[Remove Selected](#) [Remove All Shown](#)

## › DNS domains and cookie security

Back in Lesson 1, you used the host or nslookup command to look up the IP addresses of a few different web services, such as Wikipedia and your own localhost. But domain names play a few other roles in HTTP besides just being easier to remember than IP addresses. A DNS domain links a particular hostname to a computer's IP address. But it also indicates that the owner of that domain intends for that computer to be treated as part of that domain.

Imagine what a bad guy could do if they could convince your browser that their server evilbox was part of (say) Facebook, and get you to request a Facebook URL from evilbox instead of from Facebook's real servers. Your browser would send your facebook.com cookies to evilbox along with that request. But these cookies are what prove your identity to Facebook ... so then the bad guy could use those cookies to access your Facebook account and send spam messages to all your friends.

In the immortal words of Dr. Egon Spengler: *It would be bad.*

This is just one reason that DNS is essential to web security. If a bad guy can take control of your site's DNS domain, they can send all your web traffic to their evil server ... and if the bad guy can fool users' browsers into sending that traffic their way, they can steal the users' cookies and reuse them to break into those users' accounts on your site.

## › HTTPS for security

### › What HTTPS does for you

- **Privacy**
- **Authenticity**
- **Integrity**

When a browser and a server speak HTTPS, they're just speaking HTTP, but over an encrypted connection. The encryption follows a standard protocol called **Transport Layer Security**, or TLS for short. TLS provides some important guarantees for web security:

- It keeps the connection **private** by encrypting everything sent over it. Only the server and browser should be able to read what's being sent.
- It lets the browser **authenticate** the server. For instance, when a user accesses <https://www.udacity.com/>, they can be sure that the response they're seeing is really from Udacity's servers and not from an impostor.
- It helps protect the **integrity** of the data sent over that connection — checking that it has not been (accidentally or deliberately) modified or replaced.

Note: TLS is also very often referred to by the older name SSL (Secure Sockets Layer). Technically, SSL is an older version of the encryption protocol. This course will talk about TLS because that's the current standard.

### › Inspecting TLS on your service

If you deployed a web service on Heroku earlier in this lesson, then HTTPS should already be set up. The URL that Heroku assigned to your app was something like <https://yourappname.herokuapp.com/>.

From there, you can use your browser to see more information about the HTTPS setup for this site. However, the specifics of where to find this information will depend on your browser. You can experiment to find it, or you can check the documentation: Chrome, Firefox, Safari.

Note: In some browser documentation you'll see references to SSL certificates. These are the same as TLS certificates. Remember, SSL is just the older version of the encryption standard.

## › Keys and certificates

The server-side configuration for TLS includes two important pieces of data: a private key and a public certificate. The private key is secret; it's held on the server and never leaves there. The certificate is sent to every browser that connects to that server via TLS. These two pieces of data are mathematically related to each other in a way that makes the encryption of TLS possible.

The server's certificate is issued by an organization called a certificate authority (CA). The certificate authority's job is to make sure that the server really is who it says it is — for instance, that a certificate issued in the name of Heroku is actually being used by the Heroku organization and not by someone else.

The role of a certificate authority is kind of like getting a document notarized. A notary public checks your ID and witnesses you sign a document, and puts their stamp on it to indicate that they did so.

## › How does TLS assure privacy

The data in the TLS certificate and the server's private key are mathematically related to each other through a system called public-key cryptography. The details of how this works are way beyond the scope of this course. The important part is that the two endpoints (the browser and server) can securely agree on a shared secret which allows them to scramble the data sent between them so that only the other endpoint — and not any eavesdropper — can unscramble it.

## › How does TLS assure authentication

A server certificate indicates that an encryption key belongs to a particular organization responsible for that service. It's the job of a certificate authority to make sure that they don't issue a cert for (say) udacity.com to someone other than the company who actually runs that domain.

But the cert also contains metadata that says what DNS domain the certificate is good for. The cert in the picture above is only good for sites in the .herokuapp.com domain. When the browser connects to a particular server, if the TLS domain metadata doesn't match the DNS domain, the browser will reject the certificate and put up a big scary warning to tell the user that something fishy is going on.

## › How does TLS assure integrity

Every request and response sent over a TLS connection is sent with a [message authentication code](#) (MAC) that the other end of the connection can verify to make sure that the message hasn't been altered or damaged in transit.

Suppose that an attacker were able to trick your browser into sending your udacity.com requests to the attacker's server instead of Udacity's real servers. What could the attacker do with that evil ability?

- Yes: Steal your udacity.com cookies, use them to log into the real site as you, and post terrible spam to the discussion forums.
- Yes: Make this course appear with terrible images in it instead of nice friendly ones.
- No: access other sites like Gmail or Facebook
- No: cause your computer to explode

If your browser believes the attacker's server is udacity.com, it will send your udacity.com authentication cookies to the attacker's server. They can then put those cookies in their own web client and masquerade as you when talking to the real site. Also, if your browser is fetching content from the attacker's server, the attacker can put whatever they want in that content. They could even forward most of the content from the real server.

However, compromising Udacity's site would not allow an attacker to break into your Gmail or Facebook accounts, and fortunately it wouldn't let the attacker blow up your computer either.

HTTPS only protects your data in transit. It doesn't protect it from an attacker who has taken over your computer, or the computer that's running your service.

## › Beyond GET and POST

Web APIs use several other methods.

### › PUT for creating resources

- Client sends URI and content to create to server
- Server responds with `201 Created` status code if successful
- A GET request to the URI shows the created content
- PUT must be done in application code (JavaScript).
- Most file uploads are actually done with POST requests, which only require HTTP. See [Mozilla](#).

### › DELETE for deleting

- DELETE requests usually require authentication.
- After a successful DELETE requests, further requests to the URI will yield 404 Not Found.

### › PATCH for making changes

- PATCH is relatively new
- PATCH is like a Git commit-it updates a resource
- PATCH requests are not necessarily standardized, but there are some standard protocols, like [JSON Patch](#) and [JSON Merge Patch](#).

### › HEAD, OPTIONS, TRACE for debugging

- HEAD is like a GET request for headers.

- OPTIONS explain available features of the server.
- TRACE echoes what the server received from the client, but may be disabled for security reasons.

## › Great responsibility

- Generally HTTP requests are used for their expected behavior, but not always.
- In 2006, a government employee copied and pasted content into a webpage, including a link to edit the page (so editing and deleting were possible with GET requests). Google's webcrawling spider followed the link, bypassing cookies and JavaScript, and deleted the contents of the website.

## › The standard tells all

For much more about HTTP methods, consult the [HTTP standards documents](#).

## › New developments in HTTP

### › HTTP/0.9

- 1991
- GET requests
- Responses in HTML

### › HTTP/1.0

- 1996
- Headers
- POST requests
- Status codes
- Content-type

### › HTTP/1.1

- Cache controls
- Range requests (resuming downloads)
- Transfer encodings (compression)
- Persistent connection
- Chunked messages
- Host header (multiple sites per IP address)

### › HTTP/2

- Multiplexing
- Better compression
- Server push
- Newest version of HTTP

- Based on protocol work done at Google, named SPDY (speedy)
- Python libraries not ready yet.

## › HTTP/2 demos

- We did the [Gophertiles demo](#) from the Go language.
- Others
  - <http://www.http2demo.io/>
  - <https://http2.akamai.com/demo>

## › Exercise: multiple connections

- We ran *ParalleloMeter.py* and navigated to <http://localhost:8000/>.
- Most browsers can open up to six connections to the same server.

## › Multiplexing

But if you're requesting hundreds of different tiny files from the server — as in this demo or the Gophertiles demo — it's kind of limiting to only be able to fetch six at a time. This is particularly true when the latency (delay) between the server and browser gets high. The browser can't start fetching the seventh image until it's fully loaded the first six. The greater the latency, the worse this affects the user experience.

HTTP/2 changes this around by multiplexing requests and responses over a single connection. The browser can send several requests all at once, and the server can send responses as quickly as it can get to them. There's no limit on how many can be in flight at once.

And that's why the Gophertiles demo loads much more quickly over HTTP/2 than over HTTP/1.

## › Server Push

- Under HTTP/2, servers are now able to pre-emptively send multiple associated assets, such as sending `*style.css-` along with a request for `index.html`.

## › Encryption

- HTTPS encryption is not officially required in HTTP/2, but most browsers do it anyway.

## › Many more features

<https://http2.github.io/faq/>

## › Keep learning

## › Resources

- [MDN HTTP](#)
- HTTP/1.1 standards, starting at [RFC 7230](#).

- [HTTP/2 standards documents](#)
- [Let's Encrypt](#) helps learn about HTTPS.
- [HTTP Spy](#) is a Chrome extension that shows headers and request information.

## › Feedback

---

I didn't feel like the instructor clearly led us into the code for each exercise. I would compare it with Kunal's instruction in the foundational Python work in Part 1 of the Full Stack Web Developer Nanodegree program. It was also challenging, but Kunal did an excellent job of carefully leading us through the exercises, so at the end, I naturally arrived at the solutions to the problem sets on my own.

# HTML

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Intro to HTML and CSS](#)

Lesson 1. HTML

## Table of Contents

---

- [Table of Contents](#)
- [Lesson](#)
  - [01. Lesson Introduction](#)
  - [02. HTML Structure Part 1](#)
  - [03. Quiz: Make Your First Element](#)
  - [04. Environments](#)
  - [05. Text Editors](#)
  - [06. Browsers](#)
  - [07. Udacity Front End Feedback Extension](#)
  - [08. Workflow](#)
  - [09. Quiz: Make All the Headers](#)
  - [10. Trees](#)
  - [11. HTML and Trees](#)
  - [12. Quiz: Spot the Bug](#)
  - [13. Quiz: Tree to HTML](#)
  - [14. Quiz: HTML Research](#)
  - [15. Quiz: Make a List](#)
  - [16. A Guide to Paths](#)
  - [17. Constructing Links](#)
  - [18. Quiz: Add an Image](#)
  - [19. Quiz: Figures](#)
  - [20. HTML Forms](#)
  - [21. HTML Structure Part 2](#)
  - [22. HTML Documents in Depth](#)
  - [23. Mockup to Website](#)
  - [24. HTML Syntax Outro](#)

# ’ Lesson

---

## ’ 01. Lesson Introduction

The instructor Cameron starts off talking about massive redwood trees. Trees will be a recurring theme in web development. HTML has a tree structure.

## ’ 02. HTML Structure Part 1

- Anything within `<>` is a type of HTML element called a **tag**.
- Other tags include `<p></p>`, `<h1></h1>`, and `<span></span>`.
  - `<p></p>` will be on its own, `<span></span>` is inline.
- There is always an opening and closing tag.
- The tag, and the content within the tag, is called an **element**.

## ’ 03. Quiz: Make Your First Element

I typed this in Sublime Text. So much faster with tag autocompletion.

```
<span><i>Why do you think you'll be a good web developer?</i></span>

<p>I am motivated by two primary outcomes in my life: personal growth and positive impact. W
I am most happy when I am improving myself. Web development provides an almost unlimited set
Web development will also enable me to have a positive impact on the world by building techn
<span><i>Great answer brohan!</i></span>
```

Congratulations! You made your first element! No issues!

## ’ 04. Environments

- The development environment includes all software and code that you're running. The text editor and browser are two key aspects of the development environment.
- Apple XCode and Microsoft Visual Studio are IDEs. We won't need IDEs for web development because the browser runs the code.

## ’ 05. Text Editors

- Sublime text 3
- Atom
- Notepad++
- Emacs
- Vi/Vim

- Microsoft Word displays rich text. **Never, ever use Word for coding.** Code should be written in plain-text ASCII characters.

## ^ 06. Browsers

- We will be working with Chrome.
- They also speak highly of Firefox.

## ^ 07. Udacity Front End Feedback Extension

While writing code in the Udacity classroom is a great way to learn web development, we think it's really important that you practice working with a text editor and a web browser on your own computer. We also think it's incredibly important that you get feedback on your code as you write it. So, we created the Udacity Front End Feedback Extension to give you feedback on your sites as you work on your own computer.

To use the extension, you'll write code in your text editor (doesn't matter which one) and then load your site in Google Chrome (or Mozilla Firefox) with the extension enabled. For some quizzes, the extension will give you a code that you'll copy and paste back into the classroom to let us know that you've finished the quiz successfully.

## ^ 08. Workflow

- Edit files in text editor as HTML.
- Open in browser

## ^ 09. Quiz: Make All the Headers

index.html

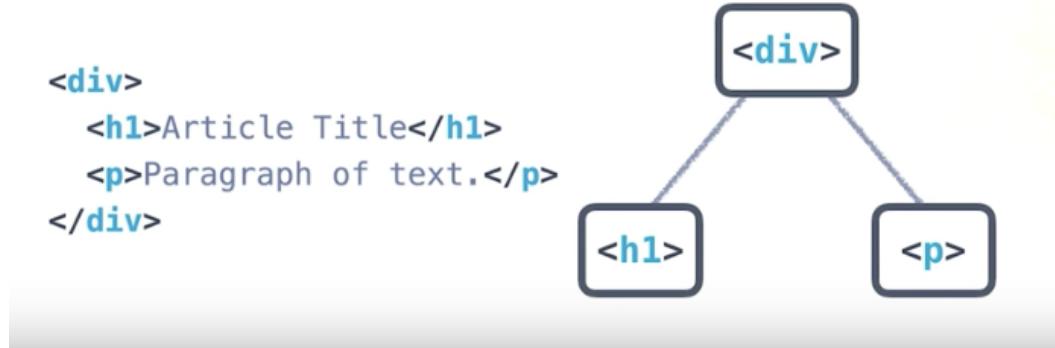
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Headers Quiz</title>
  <!-- the next line loads the tests for the Udacity Feedback extension -->
  <meta name="udacity-grader" content="http://udacity.github.io/fend/fend-refresh/lesson2/pr
</head>
<body>
  <p>Add your headers below this paragraph element! Add an h1, h2, h3, and h4 to finish the
  <h1>big big header</h1>
  <h2>big header</h2>
  <h3>header 3</h3>
  <h4>hey I'm h4! Don't forget about me!</h4>
</body>
</html>
```

## ^ 10. Trees

Trees are used in computer science to represent hierarchical data.

## 11. HTML and Trees

HTML uses a tree structure. Content is nested within divs. Div stands for division.



## 12. Quiz: Spot the Bug

We looked at ways to correctly format and nest HTML tags.

## 13. Quiz: Tree to HTML

Build this:

## HTML sample tree outline

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Tree to HTML</title>
  <!-- the next line loads the tests for the Udacity Feedback extension -->
  <meta name="udacity-grader" content="http://udacity.github.io/fend/fend-refresh/lesson2/pr">
</head>
<body>
  <!-- Did you notice that the body tag - the top of the sample tree - is already here? You can add your code here! -->
  <h1>Tree to HTML</h1>
```

```
<div>
  <p>I want you to build this tree with HTML.</p>
  <p>Here is another paragraph!</p>
</div>
</body>
</html>
```

## 14. Quiz: HTML Research

- [MDN HTML Element Reference page](#)
- [MDN](#)
- [HTML5 cheat sheet](#)

```
<p><b>This text should be bold.</b></p>
<p><i>And this text should have emphasis (italics).</i></p>
```

The b and i are used for other purposes. Recommended alternative tags:

```
<p><strong>This text should be bold.</strong></p>
<p><em>And this text should have emphasis (italics).</em></p>
```

## 15. Quiz: Make a List

Did you know that web developers spend 90% of their time looking things up?

Ok, I made up that number.

But seriously, making sense of documentation and looking up new techniques and technologies is a huge part of any web developer's work. And that's what I want you to do for this quiz.

Actually, you might find that you can make an `<li>` element appear on the page without putting it inside a `<ul>` or `<ol>`. Just because this works doesn't mean that you should ever do this. It's the equivalent of writing a sentence with bad grammar - most people will probably understand what you mean but some people will get confused. In this case, "people" are browsers and "confused" means "render your website incorrectly."

## 16. A Guide to Paths

I found this guide very interesting. It emphasized how websites are just collections of files on a computer that we call a server.

You'll soon make a website that displays an image that is stored locally on your computer. In order to display a local image, you need to be able to write a path.

**tl;dr** If there is a file called `index.html` in a directory and there is another directory called `example/` in the same directory, you can access any files in `example/` from `index.html` with the URL (path) `example/filename.html`, e.g. `<a href="example/filename.html">Example Path</a>`.

## Paths

A path is a way of describing where a file is stored.

Think of it like this:

Anyone in the world can use the address 1600 Pennsylvania Ave NW, Washington DC, USA 20006 to find the White House. A street address is an absolute path to a location.

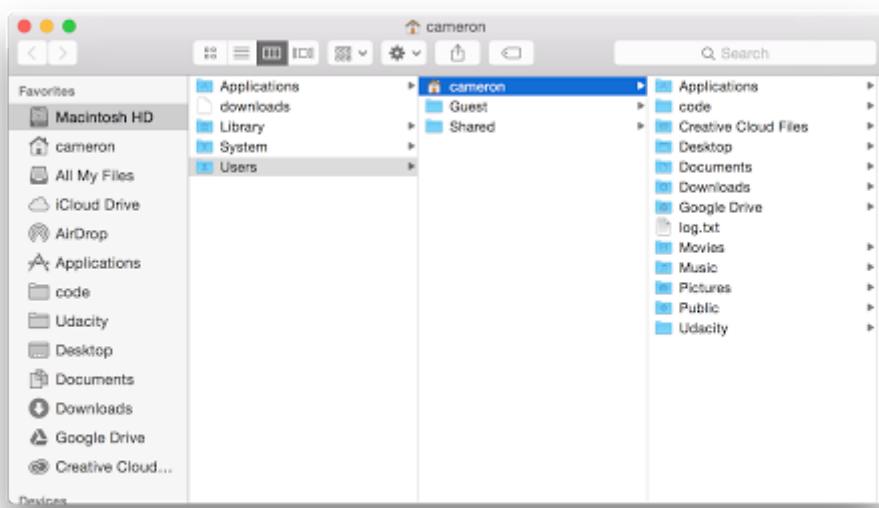
But, if you were at the [Eisenhower Executive Office](#), you could also use the phrase "next door" to find the White House. "Next door" is a relative path because it depends on your current location.

There are essentially two domains for paths that you'll need to consider as a web developer: paths to find files on your computer, **local** files, and paths to find files on other computers, **external** files.

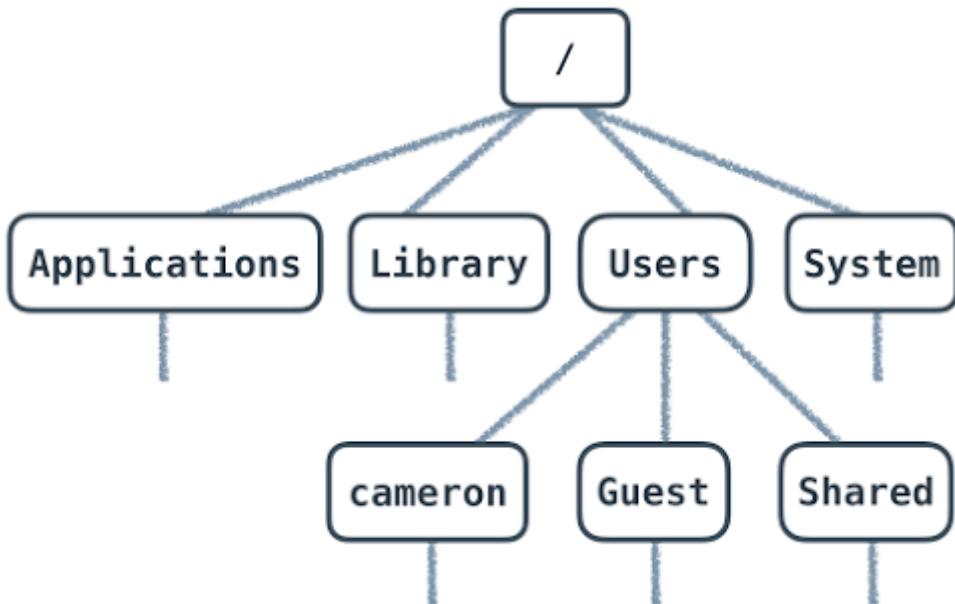
### Local Paths

Computers have folders (also called "directories"). Operating systems like Windows, Mac and Linux organize *all* of your files into a tree of directories called a **file system**. There's a top-most directory, often called the **root**, that contains all of the other directories. Within the root, there are files and directories. Within those directories are more files and more directories. And within those directories are even more files and directories, and so on.

Compare this part of the file system on my computer:



to a tree diagram showing the same directory structure:



Every file has an address, which we call the "path." An absolute path is written in relation to the computer's root directory. For instance, a file in the Documents folder on a Mac has a path that looks like this:

```
/Users/cameron/Documents/file.txt
```

`file.txt` is stored inside `Documents/`. `Users/`, `cameron/` and `Documents/` are all names of directories. `Documents/` lives inside `cameron/` and `cameron/` lives inside `Users/`. `Users/` is inside the root directory, which is represented by the first `/`. The rest of the `/` are used to separate directories.

When you open an HTML file in your browser, you're seeing the absolute path to the file on your computer.



This URL will *only* work for you on your computer. As no one else has your file system, this URL is unique to your computer. If you want other people to be able to access it, then you need an...

## 🔗 External Paths

The process of loading a website from a URL like `https://www.udacity.com` mimics opening an HTML file that you've written and saved to your computer. Every website starts with an HTML file. It just so happens that when you want to visit a website, the HTML file that you want to open lives on a different computer. The computer responsible for giving you a website's files is called a **server**.

Pointing your browser to `https://www.udacity.com` sends a request to Udacity's server for the HTML file (and others) that your computer needs to load the Udacity website. You can think of `udacity.com` as the root path of Udacity's server (computer) that anyone can access (the reality of the situation is actually much more complicated but the general idea is true). Unlike your personal computer (for now!), Udacity's servers run software that **expose** files to the web, which means that they make them available to anyone who wants them. Servers have an **external path** that anyone can access and is the reason why the web works.

Different websites are just different collections of files. Every website is really just a server (or many servers) with an external address, which we call a URL. Servers store files and send them to computers who request them (the requesting computers are called **clients**).

There are different **protocols** for serving files, the most common of which on the web are HTTP and HTTPS. When you open a file on your own computer, you're using the file protocol. You don't need to know much more about protocols for now, but if you're interested in learning (a lot!) more about HTTP, check out [Networking for Web Developers](#).

## ② Relative Paths

The relative path is similar to the absolute path, but it describes how to find to a file from a directory that is not the root directory. Like using the phrase "next door" to tell someone how to find the White House from the Eisenhower Executive Office, a relative path takes advantage of the location of one file to describe where another file can be found.

Relative paths work the same for both local and external paths. Let's break down two examples of absolute paths to see how relative paths work.

External:

```
<a href="http://labs.udacity.com/fend/example/hello-world.html">Hello, world!</a>
```

Local:

```
<a href="/Users/cameron/Udacity/etc/labs/fend/example/hello-world.html"> Hello, world!<
```

`href` is really just a path to a file.

Both examples are links to the same file using absolute paths, but the external example would work for anyone and only my computer can use the link in the local example.

Pay attention to the `fend/example/hello-world.html` portion of both paths - they mean the same thing.

Imagine that you are editing `/Users/cameron/Udacity/etc/labs/fend/test.html`. `test.html` can reference `hello-world.html` by describing how to get from its location in `fend/` to `hello-world.html`. The relative path would look like:

```
example/hello-world.html
```

This relative path takes advantage of the fact that `test.html` and `example/` are in the same directory.

But what if I'm editing a file in `/Users/cameron/Udacity/etc/labs/` and I want to write a path to `hello-world.html`? In that case, the relative path would be:

```
fend/example/hello-world.html
```

Now that I'm in `labs/`, not `fend/`, I have to include `fend/` in a relative path to `hello-world.html`.

To finish this off, let's imagine there are two files:

```
http://labs.udacity.com/science/sciences.html
```

and

```
http://labs.udacity.com/science/physics/relativity.html
```

In order to write a relative path from `sciences.html` to `relativity.html`, I only need to include the part of the path that describes how to get from `science/` to `relativity.html`:

```
<a href="physics/relativity.html">Einstein's Special Relativity</a>
```

And that's it! Now it's time to apply your new skills.

## 17. Constructing Links

Type in a link that points to <https://www.google.com> and displays as Google.

```
<a href="https://www.google.com">Google</a>
```

## 18. Quiz: Add an Image

```
<!-- Sample image element:
```

```

```

How to complete this quiz:

- Create an image element at the designated spot in the paragraph below.

- Set the source to: <http://udacity.github.io/fend/images/udacity.png>
- Set the alt description to a quick description of the image (maybe something like, "Udacity logo")
- Play around with the URL! See if you can make a different image appear.

-->

```
<p>
  This is a big paragraph of text about Udacity. Cool. Cool. And here is an image!
  <!-- put the image element here! -->
  
</p>
```

Here's another question for you: how is the text reacting to the image? Is the image on its own line or is it showing up in the same last as the rest of the text?

## 19. Quiz: Figures

The `<figure></figure>` element was new to me.

There were two ways to complete the quiz. I used option 2.

Option 1

```

<p>
  Stout Memorial Grove in Jedediah Smith Redwoods State Park in 2011 by Chmee2 (Own work)
  <a href="https://commons.wikimedia.org/wiki/File%3AStout_Memorial_Grove_in_Jedediah_Smit
</p>
```

Option 2 (adding figure tag)

```
<figure>
  
  <figcaption>Stout Memorial Grove in Jedediah Smith Redwoods State Park in 2011 by Chmee2 (
    <a href="https://commons.wikimedia.org/wiki/File%3AStout_Memorial_Grove_in_Jedediah_Smith_
  </figcaption>
</figure>
```

### Indentation Discussion

I usually indent child elements inside parents. Notice that the img and figcaption are both indented inside the figure in the solution to option 2. I don't indent when a child element is part of a line or sentence. For instance, I don't indent links because their content is read as part of a sentence.

I also indented the text content inside the p in option 1 and the figcaption in option 2. I did this because the text looks like a big block in my code editor and it's easier to read if it is on its own line.

## 20. HTML Forms

I love forms and user input! It's **Human-Computer Intelligent Interaction**.

### HTML5 input types

#### Forms

Forms are everywhere on the web. Here's a form you filled out when you signed up for Udacity!

SIGN UP      SIGN IN

## Create Your Student Account

Your student account is your portal to all things Udacity: your classroom, projects, forums, career resources, and more!

First Name\*

Last Name\*

Email Address\*

Password\*

Confirm Password\*

SIGN UP

And, here's a form I filled out when I ordered a new blender online.

### Add a new address

Be sure to click "Ship to this address" when done.

**Full name:**

**Address line 1:**

Street address, P.O. box, company name, c/o

Forms allow users to interact with your site. Normally, forms take the information submitted by a user and send it somewhere to be processed. The sending and receiving of information using a form will be covered in later courses; however, for this next quiz, you'll learn how to plan out and structure a form given some specifications.

To do this, you'll first need to know about the different HTML form tags available to you.

## ⌚ <form> tag

The `<form>` tag will be the parent element that contain all of the code for the form. The form fields, the buttons, etc. need to be enclosed inside these tags.

```
<form action="" method="">  
  <!-- stuff goes here -->  
</form>
```

the `action` and `method` attributes tell the form where and how to send the form data, respectively. You will not need them in this course, so we will leave them blank for now.

## ⌚ <input> tags

The `<input>` tag lets you add form fields for the user to enter their information. Each input tag will have an attribute called `type` to indicate what type of form field to put in the form:

```
<input type="text">
```

There are a ton of input types available with HTML5. You can check out the [list](#) here if you're curious. Each different input type changes the type of form field that's rendered on the page. Here's an example using the date, radio, and checkbox input types.

```
<input type="checkbox"> You can check me off the list!
```

You can check me off the list!

```
<input type="radio"> On the radio... oh oh.
```

On the radio... oh oh.

```
<input type="date">
```

One particular example that's a little different than the others is the `<textarea>` tag. It's similar to `<input type="text">` but you can use it to create a multi-line text input box for your user.

```
<textarea>
```

Here's a place to write a lot of stuff.

## ⌚ <label> tag

A form is no good if your user doesn't know what to put in each box! The `<label>` tag adds text to an `<input>` field so the user knows what information is being asked for.

What am I supposed to type here?

```
<input type="text" id="f1">
```

For accessibility reasons, it's important to associate each input element with a label. One way to do this is to use the `<label>` tag. Screen readers and other programs designed for accessibility will help bring focus to an input field and its description when labels are appropriately linked to their input fields using the `for` and `id` attributes.

```
<label for="name">What is your name?</label> <input type="text" id="name">
<!-- for= and id= are both "name" --&gt;
&lt;label for="age"&gt;How old are you?&lt;/label&gt; &lt;input type="number" id="age"&gt;
<!-- for= and id= are both "age" --&gt;</pre>
```

What is your name?

How old are you?

## ⌚ <button></button>

Finally, the user needs to be able to do something with the data. There are buttons for this!

HTML5 has the `<button></button>` element. Buttons also have a type (you can read about them [here](#)) so you can specify whether the button submits data or resets (clears) it. Buttons also have the added benefit of being able to display anything in the button by just adding content inside the button tags.

```
<button>I am just a button.</button>
<button type="submit">I am a submit button!</button>
<button type="reset">I am a reset button!</button>
<button type="button">I am a button with an image! <br> 
```

I am just a button.

I am a submit button!

I am a reset button!



I am a button with an image!

Often, you will use a submit button to submit and process the information in the form. For now, focus on making your forms look great and you'll make those submit buttons functional in a later course. 😊

Great! Now that you have all the pieces of an HTML form, try making a form in the next quiz.

## 21. HTML Structure Part 2

## 22. HTML Documents in Depth

- We went through doctypes, head, body
- tried just typing `<h1>This is a title</h1>` without the doctype into the [W3C HTML Validator](#) and got an error:

Thanks for completing that! Nice work! The validator didn't know what type of document it was, so it assumed it was HTML 4.01. There are also errors and warnings about missing doctypes and unknown character encodings.

```
<!DOCTYPE html>
<html>
<head>
    <title>title fo sho</title>
</head>
<body>
    <h1>This is a title</h1>
</body>
</html>
```

### HTML Doctypes

An HTML document will usually start with a type declaration (which is not a tag, so it should not have a closing tag). The declaration helps the browser determine what type of HTML document it's trying to parse and display.

If you've ever looked at an [older website](#) using dev tools, you might have noticed a doctype that looks like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dt
```

(Triggers Standards mode but specifies an older form of validation.)

Or maybe you didn't see a doctype at all?

```
<html>  
  </html>
```

(Triggers "Quirks" mode. This is bad.)

But newer websites (and your websites!) will have a declaration that looks like this:

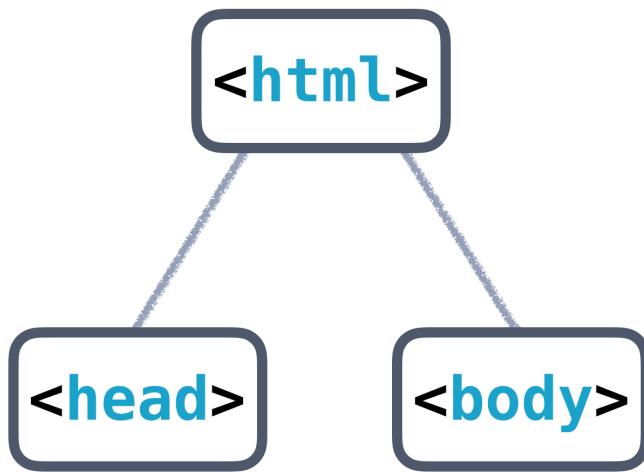
```
<!DOCTYPE html>
```

(Triggers Standards mode with all updated features.) 

Browsers look for this doctype declaration to determine which **rendering mode** to use to render the site. Generally, newer sites follow standard HTML specifications. The current standard HTML specification is called HTML5 (which is what you're learning!). On the other hand, older sites, created before HTML standards really existed, might use a different rendering mode that imitates the behavior of older browsers.

If you are interested in reading more about doctype declarations and different rendering modes, you can read about them [here](#).

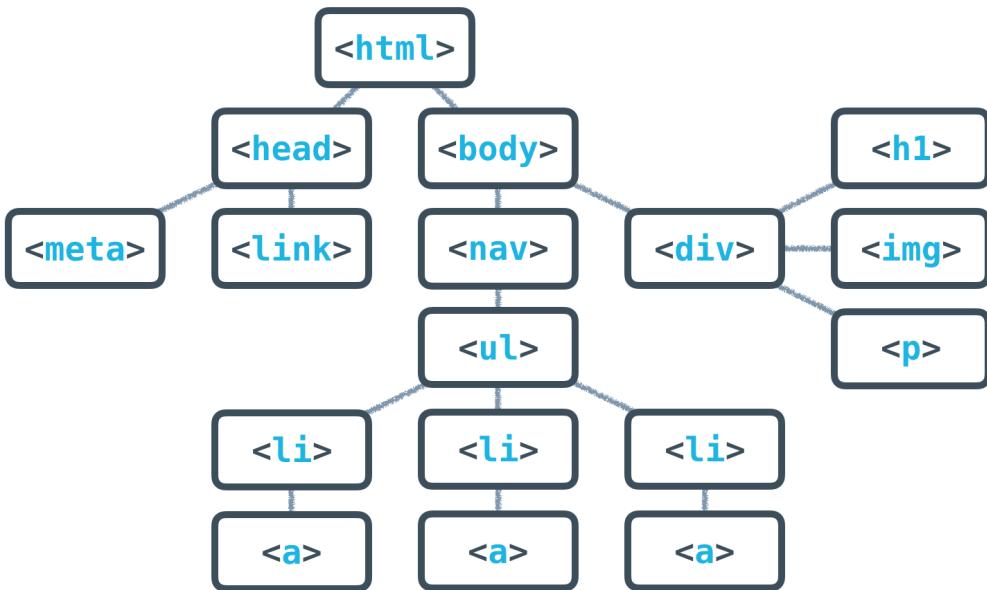
Once you've declared the doctype, the next part of your HTML document is the `<html>` tag, which tells the browser that everything enclosed inside the `<html> ... </html>` should be parsed as HTML. Then you have the two main sections of your HTML document: `<head>` and `<body>`



## 🔗 Basic HTML Tree Structure

`<head>` and `<body>`

The `<head>` will contain general information and metadata about the page, while the `<body>` will contain the content that will be displayed on the page. Here's an example tree structure for a full HTML document:



All of the HTML syntax that you've learned in this lesson will help you create the **content** of the page, which is always contained inside the `<body>` tags. The `<body>` is always visible.

The `<head>`, on the other hand, is never visible, but the information in it describes the page and links to other files the browser needs to render the website correctly. For instance, the `<head>` is responsible for:

- the document's title (the text that shows up in browser tabs): `<title>About Me</title>`.
- associated CSS files (for style): `<link rel="stylesheet" type="text/css" href="style.css">`.
- associated JavaScript files (multipurpose scripts to change rendering and behavior): `<script src="animations.js"></script>`.
- the charset being used (the text's **encoding**): `<meta charset="utf-8">`.
- keywords, authors, and descriptions (often useful for **SEO**): `<meta name="description" content="This is what my website is all about!">`.
- and more!

At this point, just focus on these two tags:

- `<title>About Me</title>`
- `<meta charset="utf-8">`

`<meta charset="utf-8">` is pretty standard, and will allow your website to display any **Unicode character**. ([Read more on how UTF-8 works here.](#)) `<title>` will define the title of the document and will be displayed in the tab of the browser window when a user visits the page.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>This is a title</title>
  </head>
  <body>
    <h1>Hello!</h1>
  </body>
</html>
```

## › HTML Validators

This might seem like a lot to remember, but thankfully, there are tools out there to help you. Much like how the Udacity Feedback Extension tells you when you've met all the requirements for a particular project, [HTML validators](#) analyze your website and verify that you're writing valid HTML.

I want you to try one out now!

### ⌚ Question 1 of 2

Using the [W3C HTML Validator](#), what happens when you enter the following HTML into the "validate by input" text box:

```
<h1>This is a title</h1>
```

- The validator found errors or warnings in the HTML
- The validator crashed because the HTML was in the incorrect format
- The validator assumed the document was HTML5
- The validator assumed the document was something other than HTML5

### ⌚ Question 2 of 2

Now, modify the HTML you entered previously so it follows the HTML template. Make sure to set doctype for HTML5!

What is the message displayed once the validator successfully validates your code?

- No errors found while checking this document as HTML5!
- Document checking completed.
- No errors or warnings to show.
- Document checking completed. No errors or warnings to show.

## › 23. Mockup to Website

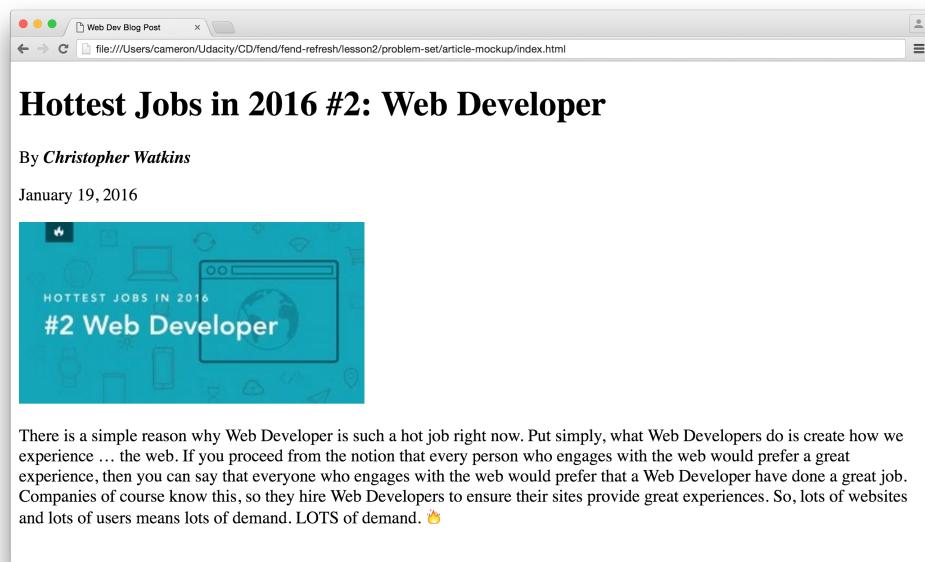
Going from designers/UX who provide images to the actual website.

## ⌚ Mockup to Website

It's common for web developers to work with designers who focus on creating user interfaces and user experiences. Designers use software like Adobe Photoshop to mock up - draw - websites. The mockups that they create are usually just images of websites with some annotations and descriptions.

As a web developer, one of the tasks you might be asked to do is take a mockup created by a visual designer and translate it into a live website. I use the word "translate" because the process of going from mockup to website is similar to the process of translating between natural languages. Just as you can create the same meaning using different words and phrases with a natural language, you can create the same website design using different HTML elements.

I want you to practice the process of going from a mockup to a website now! Here is a website mockup (note: I zoomed in for the screen shot):



You can find a copy of this image in `web-dev-blog.zip` in the instructor notes. You'll also find a file to start editing (`index.html`) and a copy of the mockup.

There are many ways to turn this mockup into website. As such, you won't be getting feedback on your site using the Chrome extension. Rather, I want you to compare your website to the mockup. You'll know you've finished this exercise when your site looks the same :)

## ⌚ How to Complete this Quiz

1. Download and unzip [web-dev-blog.zip](#).
2. Open the mockup and decide how you want to create the article.
3. Edit `index.html` until your website looks identical to the mockup.
4. Practice indenting children elements. I'll show you how I indented my HTML in the solution.

Just to be clear, there is no feedback from the Chrome extension. It's up to you to decide when your website looks identical to the mockup.

### Task List

Edit the HTML so your website looks *exactly* like the mock up. Click [here](#) when you're done.

#### 🔗 Supporting Materials

[web-dev-blog.zip](#)

▶ Solution code

#### › 24. HTML Syntax Outro

# CSS

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Intro to HTML and CSS](#)

Lesson 2. CSS

## Table of Contents

---

- [Table of Contents](#)
- [Lesson](#)
  - [01. Getting Started With CSS](#)
  - [02. What is CSS](#)
  - [03. Quiz: CSS or HTML](#)
  - [04. CSS Rulesets](#)
  - [05. Quiz: CSS Syntax](#)
  - [06. Comments](#)
  - [07. Quiz: Tag Selectors](#)
  - [08. Attributes and Selectors](#)
  - [09. Quiz: Using Selectors](#)
  - [10. Udacity Front End Feedback Extension](#)
  - [11. Quiz: Writing Selectors](#)
  - [12. Quiz: Using CSS References](#)
  - [13. Developer Tools](#)
  - [14. Developer Tools on Different Browsers](#)
  - [15. Quiz: Using Developer Tools](#)
  - [16. CSS Units](#)
  - [17. Quiz: Units in CSS](#)
  - [18. CSS Colors](#)
  - [19. Quiz: Identifying Colors](#)
  - [20. Quiz: Style an Image](#)
  - [21. Quiz: Style the Font](#)
  - [22. Quiz: Slack Card](#)
  - [23. Quiz: Udacity Site Header](#)
  - [24. What is a Stylesheet](#)
  - [25. Quiz: Link to a Stylesheet](#)

- [26. Outro](#)
- [Resources](#)

## ’ Lesson

---

### ’ 01. Getting Started With CSS

Intro by James Parks

### ’ 02. What is CSS

\*\*Cascading style sheets.\*- Webpage styling. Sometimes you may load a website and just see a plain text outline. It is lacking CSS.

If you want to try removing the style from the Youtube homepage yourself, follow these instructions: see [Developer Tools](#) below

### ’ 03. Quiz: CSS or HTML

This was easy. The website with styling is the one with css.

### ’ 04. CSS Rulesets

James showed Coca-Cola's website.

- All css begins with a \*\*ruleset.\*- The ruleset has two parts:
  - \*\*Selector:- selects the HTML elements to style
  - \*\*Declaration block:- describes the style to apply to HTML.
- CSS rulesets can be included in the HTML `<head></head>` inside the `<style type="text/css"></style>` tags.

### ’ 05. Quiz: CSS Syntax

Change the color of "Hello, world!" to green.

```
<!DOCTYPE html>

<!-- Instructions: Change the color of "Hello, world!" to green. -->

<html>
<head>
  <title>Quiz - Hello, world!</title>
  <style>
    p {
      color: blue;
    }
    /* add CSS here */
    h1 {
      color: green;
    }
  </style>
</head>
<body>
  <h1>Hello, world!</h1>
</body>
</html>
```

```
    }
  </style>
</head>
<body>
  <h1>Hello, world!</h1>
  <p>Are you ready for your first challenge?</p>
  <p>Let's add some style to this webpage!</p>
</body>
</html>
```

I got the answer quickly. I first thought it would be `<title></title>`, but that didn't work, and I realized the title was an `<h1></h1>`.

## 06. Comments

Let's take a quick diversion to cover something useful: code comments.

\*\*A comment is a human-readable message inside code.\*- Comments are usually surrounded by or preceded by special characters that instruct computers to completely ignore whatever text is inside the comment. Comments are great because they allow you, the developer, to leave clarifying messages and instructions for other developers (as well as your future self!). Every programming language gives you the ability to write comments.

\*\*Comments are also useful when you're testing your code.\*- Rather than deleting potentially useful chunks of code, you can comment them out, getting the same effect without the risk of accidentally losing work!

CSS Comments You saw a CSS comment in the code from your previous quiz:

```
p {
  color: blue;
}
/* add CSS here */
h1 {
  color: red;
}
```

The line `/* add CSS here */` is a comment. CSS comments are surrounded by an opening `/-` and a closing `*/`. You must use both. The comment made it clear to you where you needed to add your code and it did not affect the style of the page in any way.

HTML Comments You can write comments in HTML too! Here's how they look.

```
<!-- This is a comment -->
<div class="example">Words, words, words.</div>
```

You must surround your HTML comments with a starting `<!--` and a closing `-->`.

Now, back to CSS.

## ’07. Quiz: Tag Selectors

Including a tag selector like `<p></p>` will affect all paragraphs within `<p></p>` tags.

## ’08. Attributes and Selectors

tag selector

```
h1 {  
  color: green;  
}
```

class attribute selector

\*\*Classes are flexible and powerful. The same class can be used in multiple HTML elements, and HTML elements can have multiple classes.\*- This allows you to subdivide instances of the same class. Most web developers prefer classes over IDs, because they organize the HTML and because they can be used as CSS selectors.

```
<p id="site-description">favorite books</p>  
<hr>  
<h2 class="book-title">Robinson Crusoe</h2>
```

To select an HTML class to style with CSS, type a period followed by the name of the class:

```
.book-summary {  
  color: blue;  
}
```

id attribute selector

IDs should be used sparingly. An HTML element can have only one ID, and IDs can be used only once per page (in one element).

To select an ID, use a pound sign followed by the name:

```
#site-description {  
  color: red;  
}
```

## ’09. Quiz: Using Selectors

Instructions — Which HTML elements match the given CSS statement?

```
.right {  
  text-align: right;
```

```
}
```

- `<div class="right"></div>` yes
- `<a href="#" class="leftright"></a>` no, no space between left and right
- `<button id="right"></button>` no, it's an id not a class
- `<p class="highlight module right"></p>` yes, multiple classes can be selected if separated by spaces, and order does not matter.

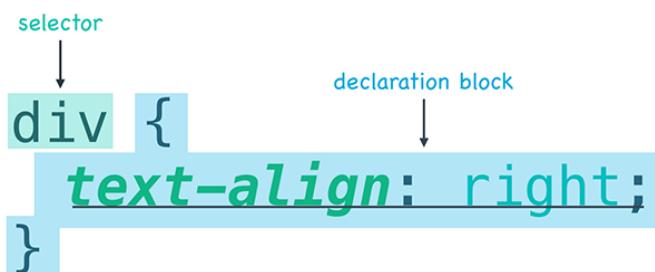
## 10. Udacity Front End Feedback Extension

Already got it

## 11. Quiz: Writing Selectors

### Intro

At this point, you should be familiar with the basic structure of a CSS statement. Every CSS statement is made up of a **selector**- and a **declaration block**. The selector tells the browser what HTML element we want to style and the declaration block tells the browser what styles need to be applied to that HTML.



For this quiz, I want you to focus exclusively on the selector part of a CSS statement. To do this, I've created a webpage that is lacking style. The webpage already has ids and classes added to the HTML, but it's missing the right selectors to add the style.

```
<div id="menu">
  <h1 class="item">Chicken Clay Pot</h1>
  
  <p class="description">Crispy rice baked in clay pot topped with chicken and vegeta
</div>
```

```
/* missing id */
  text-align: center;
}
/* missing class */
  color: red;
}
/* missing class */
  border-radius: 5px;
}
/* missing class */
```

```
font-style: italic;  
}
```

It's your job to download the webpage and fill-in the missing selectors. If you do it right, your webpage should end up looking like this...

## Chicken Clay Pot



*Crispy rice baked in clay pot topped with chicken and vegetables*

### » How to Complete this Quiz

1. Download the zip file called `writing-selectors.zip` from the resources section or from [here](#).
2. Open `index.html` and turn on the [Udacity Feedback Chrome Extension](#).
3. Edit `index.html` and add the missing id and class selectors within the `<style>` tags (alternatively, you can use Developer Tools to add the missing id and class selectors to pass all the tests).
4. When you've successfully added the correct selectors, copy and paste the code that appears into the next page to finish!

When you're ready to get started, click "Skip to Quiz".

### ► Solution

## » 12. Quiz: Using CSS References

Instructions — Use CSS References to answer the questions below.

- \*What CSS property is used to italicize text?- `font-style`
- \*What CSS property is used to underline text?- `text-decoration`

- \*What CSS property is used to uppercase text?- text-transform
- \*What CSS property is used to bold text?- font-weight

## Mozilla MDN CSS Fonts

# 13. Developer Tools

- open with cmd+opt+j
- Developer tools allow you to edit webpages live. You can edit each element. To restore the page, just refresh.
- I worked with developer tools for a little bit. I edited the LeanBox website.

## Editing site with developer tools

- Mobile development: Click the mobile icon in Chrome's developer tools to view a mobile version.
- How do you add a line break to the CSS? -> right click on the style tag and select "edit as HTML"

## Saving site with developer tools

- Add folder to workspace
- Map to file system resource:
  - *How do i do this for the entire site? Is it just for specific files?*
  - I tried searching, but [Stack Overflow](#) wasn't very helpful.
  - It looks like you have to download the file, then make modifications. If you have made modifications directly in the browser, and want to save them, I thought you might have to copy the elements and paste them into the desktop file with your text editor.
- [Set Up Persistence with DevTools Workspaces](#)
  - TL;DR
  - Don't manually copy changes to local files. Use workspaces to persist changes made in DevTools to your local resources.
  - Stage your local files to your browser. Map files to URLs.
  - Once persistent workspaces are set-up, style changes made in the Elements panel are persisted automatically; DOM changes aren't. Persist element changes in the Sources panel instead.
- Limitations
  - As powerful as Workspaces are, there are some limitations you should be aware of.
  - Only style changes in the Elements panel are persisted; changes to the DOM are not persisted.
  - Only styles defined in an external CSS file can be saved. Changes to element.style or to inline styles are not persisted. (If you have inline styles, they can be changed on the Sources panel.)

## Deleting elements with developer tools

- Visit <https://www.youtube.com/> or any other website of your choice.

- Right click anywhere on the page, and click "Inspect" (on Mac) or "Inspect Element" (Firefox)
- You'll see a panel showing HTML. Inside the element, delete any line that has rel="stylesheet". For example:

```
<link rel="stylesheet" href="//s.ytimg.com/yts/cssbin/www-core-webp-vf1CayM79.css">
```

## › Additional info from lesson 15 positioning

You already have the developer tools skills you need to find the .relative element and move it. However, before you start, let me show you a quick trick for finding any element by its selector with Chrome developer tools.

- [Open the DevTools console panel.](#)
- Type `$('[selector]')` and press enter. [selector] can be any CSS selector, so you could have `$('.className')` or `$('#idName')`. Notice that you need ' around the selector.
- An HTML element should appear in the console (something like text
  - ). Right-click (or control-click on a Mac) on the element and select Reveal in Elements Panel.

*I find this to be far more cumbersome than just clicking on the element and selecting inspect.*

## › 14. Developer Tools on Different Browsers

## › 15. Quiz: Using Developer Tools

## › 16. CSS Units

CSS uses absolute (pixels, mm, cm, in) and relative (percentages, em, vw, vh) units.

## › 17. Quiz: Units in CSS

Instructions — Add the following changes to the webpage:

- *set the first div's width to 100px (pixels)*
- *set the second div's height to 200px (pixels)*
- *set the third div's margin to 1em*
- *set the fourth div's font-size to 2em*

```
<!DOCTYPE html>
```

```
<!-- Instructions: Add the following changes to this webpage:
```

- set the first div's width to 100px (pixels)
- set the second div's height to 200px (pixels)

- set the third div's margin to 1em
- set the fourth div's font-size to 2em

```
-->

<html>
<head>
  <title>Quiz - Units in CSS</title>
  <style>
    .first {
      width: 100px;
    }
    .second {
      height: 200px;
    }
    .third {
      margin: 1em;
    }
    .fourth {
      font-size: 2em;
    }
  </style>
</head>
<body>
  <div class="first"></div>
  <div class="second"></div>
  <div class="third">Hammock ex plaid nulla. Nihil stumptown gastropub.</div>
  <div class="fourth">Hammock ex plaid nulla. Nihil stumptown gastropub.</div>
</body>
</html>
```

The quiz was throwing an error even though I had all the code correct.

## 18. CSS Colors

- Color is light. All light is a combination of red, green, blue.
- RGB values are written as 0-255, with 255 being 100% presence of that color.
- Hexadecimal: FF is equivalent to 255, or 100% presence of light.
- Color pickers are a fast way to get the hex code.

How Hexadecimal Works

- [The Hexadecimal Numeral System](#)
- [Hex to RGB Convertor](#)

## 19. Quiz: Identifying Colors

Shorthand can be used to identify colors, like `color: #00f` for blue.

## 20. Quiz: Style an Image

► Solution

## › 21. Quiz: Style the Font

► Solution

## › 22. Quiz: Slack Card

I tried using Chrome Developer tools to edit the Slack card. It was a pain. I wasn't sure how to insert a line break.

## › 23. Quiz: Udacity Site Header

Again, wasn't really able to save the code because developer tools is difficult to use. It was just formatting a website header.

## › 24. What is a Stylesheet

### ⌚ What is a Stylesheet

In the next quiz, you'll be working with a stylesheet. So what is a stylesheet and why is it important? Well, consider the following question...

*What if you wanted to use the same CSS on more than one webpage?*

You could just copy all of your CSS from one file and paste it into another, but that seems like a lot of extra work and doesn't scale very well. What if you decide to make changes later? You'd have to change every copy of the CSS!

There's got to be a better way...



While the process described above works, it's not recommended. Instead, the preferred method is to write your CSS in a file called a \*stylesheet- and then link to that file in your HTML.

The image shows a code editor with two tabs: 'index.html' and 'styles.css'. The 'index.html' tab contains the following HTML code:

```
1 <!DOCTYPE html>
2
3 <html>
4
5   <head>
6     <meta charset="utf-8">
7     <title>My Site</title>
8     <link rel="stylesheet"
9       href="styles.css">
10    </head>
11
12    <body>
13      <h1>My Awesome Heading!</h1>
14      <p>This is my awesome
15        paragraph</p>
16    </body>
17
18  </html>
```

The number '17' is highlighted in a blue box at the bottom left of the code area.

The 'styles.css' tab contains the following CSS code:

```
1 h1 {
2   color: red;
3 }
4
5 p {
6   color: blue;
7 }
```

The number '8' is highlighted in a blue box at the bottom right of the code area.

## ⌚ Stylesheets

A stylesheet is a file containing the code that describes how elements on your webpage should be displayed.

The image shows a code editor with one tab: 'styles.css'. It contains the same CSS code as the styles.css file in the previous screenshot:

```
1 h1 {
2   color: red;
3 }
4
5 p {
6   color: blue;
7 }
```

The number '8' is highlighted in a blue box at the bottom left of the code area.

This is no different than what you've been doing before, except the CSS lives in a different file... and you don't have to use the `<style>` tags anymore. To create a stylesheet, simply add a new file to your project, write some CSS, and save it as `name-of-stylesheet.css`.

## ⌚ How to Link to a Stylesheet

Before your webpage can use the stylesheet, you need to link to it. To do this, you'll need to create a `<link>` to your stylesheet in your HTML. To create a link, simply type the following inside the `<head>` of your HTML.

```
<link href="path-to-stylesheet/stylesheet.css" rel="stylesheet">
```

The `href` attribute specifies the *\*path\** to the linked resource (you can link to other resources besides stylesheets, more on this later) and the `rel` attribute names the *\*relationship\** between the resource and your document. In this case, the relationship is a stylesheet. When you're done, it will look something like this...

```
<head>
  <title>My Webpage</title>
  <!-- ... -->
```

```
<link href="path-to-stylesheet/stylesheet.css" rel="stylesheet">
<!-- ... -->
</head>
```

## ’ 25. Quiz: Link to a Stylesheet

- downloaded files
- added link to css into html index with `<link rel="stylesheet" type="text/css" href="css/styles.css">`

## ’ 26. Outro

## ’ Resources

- [Sass](#) (Syntactically Awesome Style Sheets) extends the functionality of CSS.
- [CSSmatic](#) generates CSS features

# Sizing

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

## Table of Contents

---

- [Table of Contents](#)
- [Lesson](#)
  - [01. Guess the Render Game](#)
  - [02. James' Solution](#)
  - [03. Cameron's Solution](#)
  - [04. Who's the Winner](#)
  - [05. Boxes, Boxes Everywhere](#)
  - [06. Boxes on the Web](#)
  - [07. The Box Model](#)
  - [08. Quiz: Border-Box](#)
  - [09. Changing Boxes](#)
  - [10. Quiz: Box Model Sizing](#)
  - [11. Containers](#)
  - [12. Quiz: Relative Widths](#)
  - [13. Inline Boxes](#)
  - [14. Semantic Elements](#)
  - [15. Types of Elements](#)
  - [16. Quiz: Element Research](#)
  - [17. Outro](#)

## Lesson

---

### 01. Guess the Render Game

Welcome, this lesson is all about elements and how to size them.

James and Cameron play the "render game." They create a bunch of HTML files, exchange them, and challenge each other to render (sketch) the sites based on the HTML.

### 02. James' Solution

## ’03. Cameron's Solution

Different HTML elements have different default sizes. For example, the footer takes the entire width of the page, and usually contains info about who made the page.

## ’04. Who's the Winner

Cameron wins.

| Semantic elements are just much, much easier to read.

## ’05. Boxes, Boxes Everywhere

Visual design, print or digital, is based on a series of boxes.

Key points:

- You can take any design and turn it into a box.
- There are many kinds of boxes with many different purposes.

## ’06. Boxes on the Web

- The two key options for display properties are `display: block`, which takes up as much width as possible, and `display: inline`, which only takes up as much width as needed. Height is content-based.
- a `<div></div>` is the most generic display property.
- `d3.js` is a library for creating web graphics. They show an example of a Circos plot (chord diagram).
  - [d3.js Gallery](#)
  - [d3.js Chord Diagram](#)

## ’07. The Box Model

- Content
- Padding
- Border
- Margin

James also pointed out the `box-sizing: content-box` code. Normally, the `width` doesn't include padding and border, so if you set width to 800px, then add 50px each for padding and border, you're now at 1000px wide. To get around this, you can use `box-sizing: border-box;` which will include both padding and border in the width and height of the element. `box-sizing: border-box;` does not include margin.

## ’08. Quiz: Border-Box

`box-sizing: border-box;` does not include margin.

## ’09. Changing Boxes

- Boxes have default properties, called user-agent styles.
- Height is content-based.

## ’10. Quiz: Box Model Sizing

Instructions:

Finish the CSS to create this block element's box model.

► Solution

## ’11. Containers

The sizes of child elements will be influenced by the parent elements. The parent elements are also called **containers**.

## ’12. Quiz: Relative Widths

## ’13. Inline Boxes

Inline elements will wrap when they exceed the container width.

## ’14. Semantic Elements

Excessive use of `<div></div>` can lead to "div soup." This technically works, but is not semantically ideal. HTML is intended to be a markup language, so it describes a document. Use semantic elements instead. Semantic elements have the additional benefit of enhancing Search Engine Optimization (SEO). The Googlebot can more easily extract information.

Instructor notes

Semantic elements can really help improve your website's search engine optimization. If you want to learn more about SEO, I recommend checking out [Moz's Beginners Guide to SEO](#).

Want to learn more about surfing the web with a screen reader? Here's a [quick video](#) from our [course on Responsive Images](#). The video focuses on images but there are some great links if you've never tried a screen reader before.

## ’15. Types of Elements

You have many different kinds of elements at your disposal. Almost all of them are either block or inline with the major differences being their default CSS styles. Check it out, [here's a list](#).

The elements for [content sectioning](#) are good for laying out things on the page. Notice that a lot of these have requirements for what should be inside the element, for example, an `<article>` typically includes a heading as a child element.

**Text content** is good for text. There are a few from here that you've seen before, but there are some new ones as well.

There are also **inline text semantics**. You'll see familiar elements like `<em>` and `<span>`, but you've got others like `<code>`, which displays code in a monospace font. Also `<data>`, `<progress>`, `<time>`, and others.

There are other categories, but take a look at the last category, a **graveyard of old elements**. This is evidence of the way the web used to be. Don't use these. They might work, but they aren't guaranteed to continue working because they are no longer supported. Stick with the other elements and you'll be fine.

For the next quiz, you'll be asked to research these elements in order to select which ones should be used on an example website. Press "Next" to continue to the quiz.

## 16. Quiz: Element Research

The quiz took me a few tries. They were looking for `<nav></nav>` for the links at the top, and `<article></article>` for the sections. The sidebar was `<aside></aside>`.

## 17. Outro

# Positioning

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

## Table of Contents

---

- [Table of Contents](#)
- [Lesson](#)
  - [01. Intro](#)
  - [02. Normal Flow](#)
  - [03. Normal Flow Quiz](#)
  - [04. Inline-Block](#)
  - [05. Why Two Lines](#)
  - [06. Anonymous Boxes](#)
  - [07. Relative Flow](#)
  - [08. Relative Flow Quiz](#)
  - [09. Relative Neighbors](#)
  - [10. 3D Websites](#)
  - [11. Document and Viewport](#)
  - [12. Fixed Flow](#)
  - [13. Fixed Flow Quiz](#)
  - [14. Absolute Flow](#)
  - [15. Inline Formatting](#)
  - [16. Debugging CSS Part 1](#)
  - [18. Debugging CSS Solution](#)
  - [19. Outro](#)

## Lesson

---

### 01. Intro

Positioning elements is a lot like Tetris. It's easy to learn the basics, but mastery requires practice.

Web developers don't spend time memorizing positioning syntax.

Strategies:

- Make good guesses
- Read documentation
- test, fix, iterate to get it right.

I appreciate that they are teaching us how to think like programmers, rather than simply showing us.

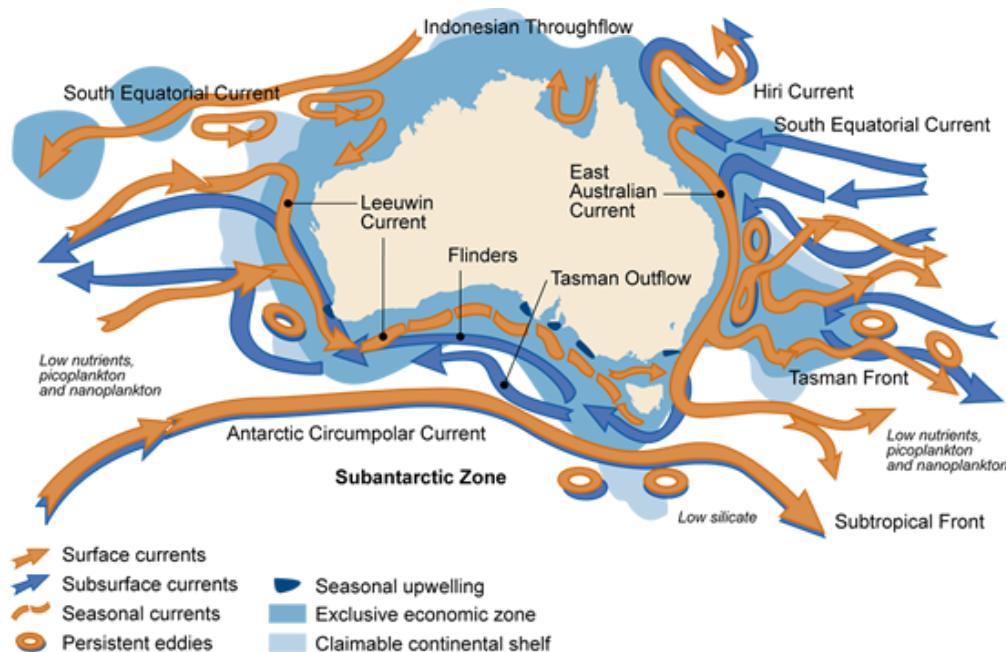
## 02. Normal Flow

Instructor notes:

The Game of Positions starts with this CSS property: `position`.

### ↪ `position: static;`

Let's start with an analogy. Take a look at the image below. The arrows show how currents **flow** around the Australian continent.



© Commonwealth of Australia 2013 [CC BY 3.0 au (<http://creativecommons.org/licenses/by/3.0/au/deed.en>)], via Wikimedia Commons

An object floating in the oceans around Australia (like a rubber ducky) would follow the path of the [currents](#).

The Game of Positions actually can be played with a few different sets of rules, which we sometimes call **flows** because of the way most developers think of elements flowing into place as if they were being pushed by a force, much like the currents flowing around Australia (and every other landmass, for that matter).

So, how does this look with CSS?

```
.default {
  position: static;
}
```

The default position is `static`, which gets called the **normal flow**. This is what you've been using in all of your sites so far. The `relative`, `absolute` and `fixed` flows are variations of the normal flow.

## ↪ How the Normal Flow Works

The way elements will move in the normal flow depends on their display state as `block` elements or `inline` elements.

1. Block elements are aligned **vertically**.
2. Inline elements are aligned **horizontally**.

### Animation showing positioning of webpage elements

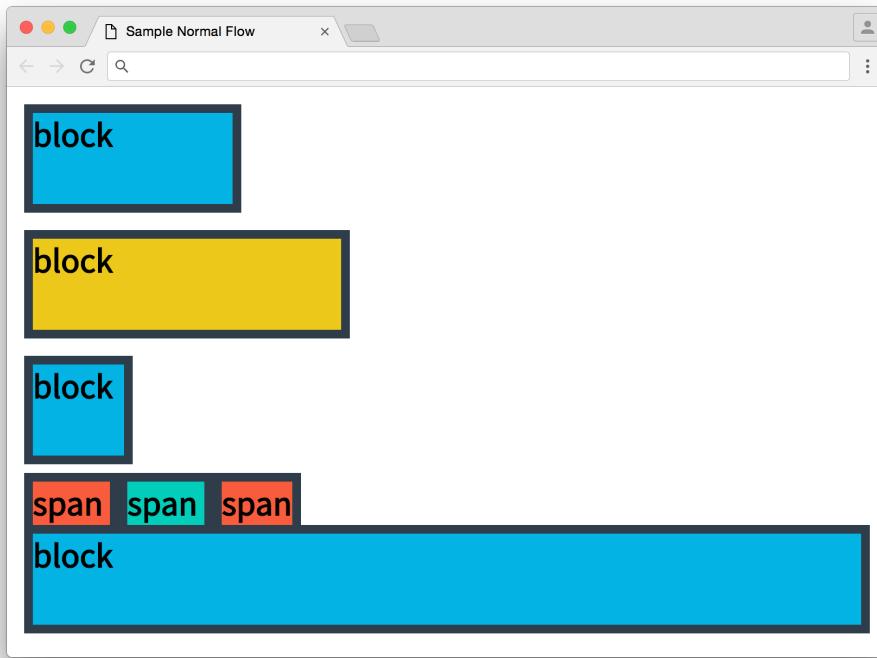
*(Elements don't actually have a starting point on the screen. I picked the bottom middle arbitrarily. Depending on the flow that you're using, you might do better thinking that your elements are coming in from the bottom left, or bottom right, or appearing randomly!)*

Notice how the `<div>`s (block elements) stack vertically while the `<span>`s (inline elements) stack horizontally.

Let's look at another example. Here is some HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Sample Normal Flow</title>
  <style> /- nothing but colors, widths and margins */ </style>
</head>
<body>
  <div>block</div>
  <div>block</div>
  <div>block</div>
  <span>span </span>
  <span>span </span>
  <span>span</span>
  <div>block</div>
</body>
</html>
```

And here is the [resulting website](#):



Take a look at the way the block elements line up.

The left edges of block elements line up.

Block elements are laid out vertically. Their left outer edges will line up with their parent's left outer edges. Even though these blocks do not take up the full width of the page, they still stack.

Inline elements push together side by side.

Inline elements are laid out horizontally inside their parents. The left edge of an element's line box will touch the right edge of the preceding element's line box.

If a line box is too big for a line, it wraps around to the next line.

And that's it! The rules of the positioning game are pretty simple by themselves, but they can get complicated when they start building on each other. For instance, take a look at how the spans end up on a line of their own. The block elements seem to treat the inline elements *as a whole* like a block. There's an interesting reason for this which you'll encounter later in this lesson.

## ’03. Normal Flow Quiz

They gave us some CSS and HTML, and we had to predict how the website would look. I got it on my first try.

## ’04. Inline-Block

Instructor notes:

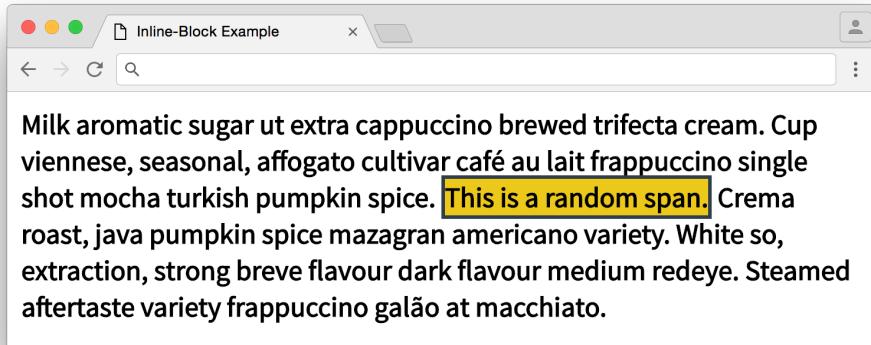
So far, there's been a distinction between inline and block elements. However, there's a hybrid of the two!

```
.example {  
  display: inline-block;  
}
```

display: inline-block elements can be sized like block elements but are laid out like inline elements.

Take a look at this example. Notice that the .random span in the middle of the text has a width and height assigned to it but the span in the resulting website (shown below) acts like a regular inline element - it is rendered as a line-box and the width and height are ignored.

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <title>Inline-Block Example</title>  
  <style>  
    {  
      box-sizing: border-box;  
      font-family: 'Source Sans Pro', sans-serif;  
      font-weight: bold;  
    }  
    .inline-block { /* hasn't been assigned! */  
      display: inline-block;  
    }  
    .random {  
      border: 2px solid #2e3d49;  
      background-color: #ecc81a;  
      width: 20em;  
      height: 40px;  
    }  
  </style>  
  <link href='https://fonts.googleapis.com/css?family=Source+Sans+Pro' rel='stylesheet'  
</head>  
<body>  
  <div class="container">  
    <span>...</span>  
    <span class="random">This is a random span.</span>  
    <span>...</span>  
  </div>  
</body>  
</html>
```

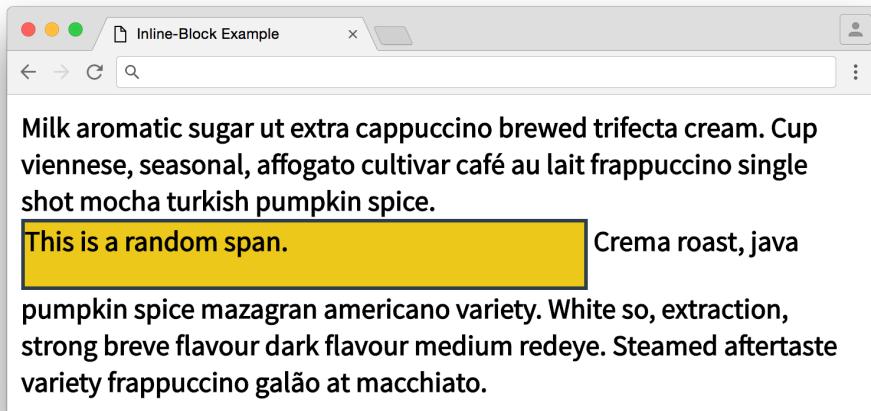


Looks like the random span is acting like a normal inline element.

Now, I'll add `.inline-block` to the `.random` span.

```
<span class="random inline-block">This is a random span.</span>
```

Take a look at [the website now](#) to see how it looks!



Now the random span has a defined width and height and still appears in the same line as the rest of the text!

You can see that the `.random` span is mixing the behavior of a block element and an inline element. But what if `display: inline-block` is applied to a block element, like a `<div>`?

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Inline-Block Example</title>
  <style>
    {
      box-sizing: border-box;
      font-family: 'Source Sans Pro', sans-serif;
      font-weight: bold;
    }
  </style>

```

```

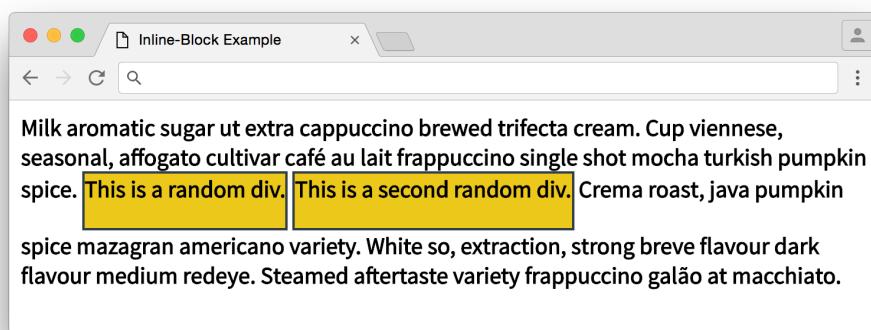
.inline-block {
  display: inline-block;
}

.random {
  border: 2px solid #2e3d49;
  background-color: #ecc81a;
  /* no width this time, but you could certainly apply one */
  height: 40px;
}

</style>
<link href='https://fonts.googleapis.com/css?family=Source+Sans+Pro' rel='stylesheet'
</head>
<body>
  <div class="container">
    <span>...</span>
    <div class="random inline-block">This is a random div.</div>
    <div class="random inline-block">This is a second random div.</div>
    <span>...</span>
  </div>
</body>
</html>

```

This is what you'll see:



Same thing!

Regardless of the tag, any element with `display: inline-block` takes on the layout behaviors of an inline element with the sizing behaviors of a block element.

Now that you've seen inline-block elements, I want to show you a quirk of HTML that shows up from time to time.

## 05. Why Two Lines

## 06. Anonymous Boxes

I figured this out quickly. He had two boxes that should have been on the same line, but they were on two lines. I viewed the page with developer tools and changed the width to 40%. The two elements were written as separate lines, and the browser interpreted the whitespace, newline, and tab as an **anonymous box**.

Quiz response:

Thanks for completing that! Right! Believe it or not, tabs, spaces and newlines in HTML can actually become elements! The resulting elements are called "anonymous boxes," which get created in a number of scenarios. This is just one example of an anonymous box. Usually, anonymous boxes from whitespace don't affect the way a page displays, but this is an example of where they do!

References:

- <https://www.w3.org/TR/CSS2/visuren.html#anonymous-block-level>
- <http://book.mixu.net/css/1-positioning.html#anonymous-box-generation>

## 07. Relative Flow

Relative flow allows objects to be shifted relative to the other elements in the flow, after elements are laid out in the normal flow.

```
.relative {  
  position: relative;  
  top: 10px;  
  left: 10px;  
  bottom: 10px; /* redundant if you've already defined top */  
  right: 10px; /* redundant if you've already defined left */  
}
```

Instructor notes

```
{  
  position: relative;  
}
```

The **relative flow** is a variant of the normal flow and behaves basically the same. But, the relative flow adds some new abilities.

The **relative flow** allows you to shift the position of elements *after* they've been laid out in the normal flow.

```
.relative {  
  position: relative;  
  top: 10px;  
  left: 10px;  
  bottom: 10px; /* redundant if you've already defined top */
```

```
    right: 10px; /* redundant if you've already defined left */
}
```

You can think of these new CSS properties, `top`, `left`, `bottom` and `right`, as adjustments to the normal flow. Setting any of them to any pixel value will shift the element by that much from where it *would* have ended up in the normal flow.

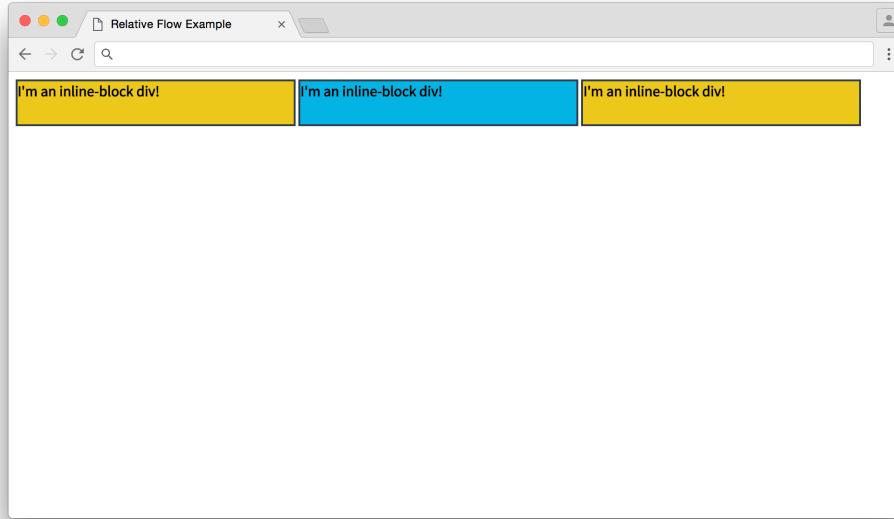
Here is the example from the animation.

Note that you can use positive and negative pixel values. You aren't changing the width or height, only the position. As you can't shift an element both directions in one axis simultaneously, you should only use one of `top` and `bottom` or one of `left` and `right` at a time.

Let's add another `<div>` and break down this example some more. Here's the HTML to start (notice that `position: relative` is commented out):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Relative Flow Example</title>
  <link href='https://fonts.googleapis.com/css?family=Source+Sans+Pro' rel='stylesheet'
<style>
  {
    box-sizing: border-box;
    font-family: 'Source Sans Pro', sans-serif;
    font-weight: bold;
  }
  div {
    display: inline-block;
    border: 2px solid #2e3d49;
    width: 300px;
    height: 50px;
  }
  div:nth-child(even) {
    background-color: #02b3e4;
  }
  div:nth-child(odd) {
    background-color: #ecc81a;
  }
  .shift {
    /*position: relative;*/
    top: 10px;
    left: 10px;
  }
  .relative {
    position: relative;
  }
</style>
</head>
<body>
  <div>I'm an inline-block div!</div>
```

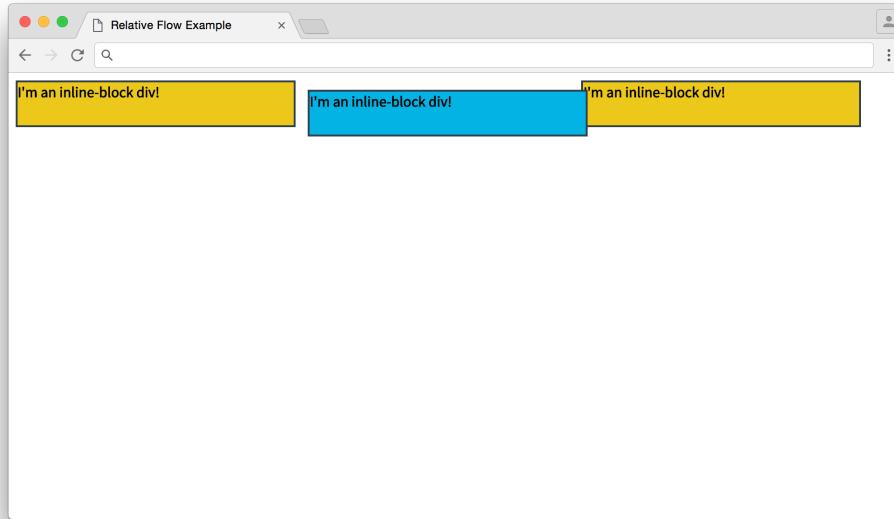
```
<div class="shift">I'm an inline-block div!</div>
<div>I'm an inline-block div!</div>
</body>
</html>
```



As all three divs are inline-block, they display in the same line.

This shows exactly what you'd expect with the normal flow. Now I'll uncomment `position: relative` in the `.shift` ruleset, which should change the position of the second `<div>` (you should try this out with developer tools on your own!).

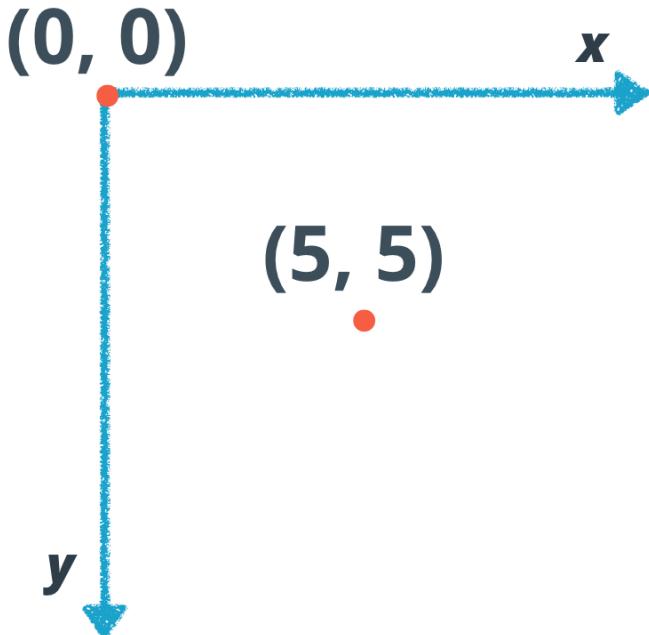
```
.shift {
  position: relative;
  top: 10px;
  left: 10px;
}
```



The second div is in a new position!

The relatively positioned `.shift` element moved from its original position down by 10 pixels and right by 10 pixels. Now, it may seem weird that the element moved *down* when the `top` property was added, but it makes sense when you consider the [coordinate system](#) that browsers use.

You're probably familiar with the [Cartesian coordinate system](#) from geometry or the [Polar coordinate system](#) from trigonometry. Browsers use a slightly different coordinate system.



The coordinate system that browsers use. The vertical axis, `y`, starts at 0 at the top and gets more positive as you go down, while the horizontal axis, `x`, starts at 0 on the left and gets more positive as you move right.

Similar to the Cartesian coordinate system, there are `x` and `y` axes. But while the `y`-axis in the Cartesian system gets more positive as you go *up*, the `y`-axis in browsers starts with 0 at the top and gets more positive as you move *down*.

Thus, when you set `top` to a positive number, you're actually moving the element down. And if you make `left` a positive number, you move the element to the right!

Did you notice how the third element didn't move in the last example? Even after the second element shifted, the third element stayed in place. You'll be exploring why in the next quiz.

## ’08. Relative Flow Quiz

Children of a relative element shift with the parent. Siblings don't shift.

## ’09. Relative Neighbors

Browsers use a layering system, so some elements will be on top of others.

## ’10. 3D Websites

Browsers also have a third dimension: the z-dimension (also known as the z-axis). While x and y run along the computer screen's surface, z is the dimension that extends in to and out of the screen. By adding a third dimension, you can think of elements as appearing on different layers, similar to the kind of layers you would find in image-editing software like Adobe® Photoshop®.

In CSS you can often use the z-index property to control the stacking order of overlapping elements on different layers (but [there are many gotchas!](#)).

## 11. Document and Viewport

- Document is the entire DOM
- Window is the total visible portion of the DOM
- Viewport is the portion of the DOM currently shown on the screen

## 12. Fixed Flow

`position: absolute` and `position: fixed`

	Relative	Absolute	Fixed
When	After normal flow	Before normal flow	Before normal flow
Positioned relative to...	position in normal flow	parent	viewport

## 13. Fixed Flow Quiz

I added the following code to the sticky footer CSS:

```
.sticky-footer {  
  background-color: #15C26B;  
  width: 100%;  
  height: 50px;  
  position: fixed;  
  bottom: 0px;  
}
```

## 14. Absolute Flow

When Would You Actually Use `position: absolute`?

Frankly, `position: absolute` is best thought of as a last resort. If all the other flows fail, then maybe absolute is your best option. Off the top of my head, I can't actually think of an instance where I wanted to use `position: absolute`. There are, in fact, other CSS techniques for achieving the same kind of shift, of which my favorite is `transform: translate`. ([Transform](#) is a more advanced CSS property and it's incredibly powerful. I recommend checking it out). It's good to know what `position: absolute` does, but it's rarely your best positioning option.

## ’15. Inline Formatting

Horizontal alignment: `text-align`. Left, right, center, justify.

Vertical alignment: `vertical-align`.

There are two sets of values you can use to change text vertical alignment. The first set focuses on the parent's size, while the second set focuses on the element's line. `vertical-align: text-top`, `vertical-align: text-bottom` and `vertical-align: middle` are options for aligning text to the eponymous locations in the parent. `vertical-align: top` and `vertical-align: bottom` will align with the line instead. Deciding how and when to use `vertical-align` is usually more of an exercise in trial-and-error than careful consideration.

## ’16. Debugging CSS Part 1

Simple ideas form the basis of even the most complex system.

Yes!

Quiz part 1: add one line of CSS on the `.sibling` class, to move the "Relative Sibling" box down.

`vertical-align: top`

## ’18. Debugging CSS Solution

I didn't have the alignment problem the instructor was referring to, so I just moved on.

I'll paste the explanation below:

As I said earlier, I came across this problem on accident. I'm presenting the solution in an order that's efficient and effective for your understanding, but I want you to know that this is the end product of a lot of experimentation. I spent quite a bit of time in developer tools playing with CSS properties, predicting what would happen before I made a change and then drawing conclusions after. I'm not alone in my approach. Here's a quote from Warren Ouyang, our lead front-end engineer, to whom I gave this problem. After he finished it, he said,

I would love to say I just looked at the code and figured it out, but I also did a bunch of playing around with it to understand it.

Here we go!

Remember, the shift from `position: relative` happens *after* the normal flow finishes. Let's start with the normal flow. Ignore `position: relative` for now.

Unchecking `position: relative` gives you this:

Notice that the layout of two non-relative siblings stays the same!

Here is your smoking gun. Look at how the text lines up. You've seen this before.

The screenshot shows a browser window titled "Relative Flow Quiz" with the URL "localhost:8000/fend/fend-refresh/lesson6/quiz-relative-flow/". The page contains a "Parent" div with three children: "Sibling" (green), "Relative Sibling" (orange), and "Child" (orange). The "Relative Sibling" div has the class "sibling" and "bordered". The "Child" div has the class "child". The "Relative Sibling" has a "position: relative" style applied, with "top: 50px;" set. The "Child" has a "bordered" style applied. The "Sibling" has a "padding" style applied with values of 146.364px and 96.364px. The developer tools' Elements tab shows the DOM structure, and the Styles tab displays the CSS rules for each element.

Without relative positioning, the text lines up!

The sibling is an inline box and the "Child" text inside it is its text. As you learned a few pages ago, text will line up with the baseline of a parent. The baseline of the parent is lowered by the extra text inside the relative sibling. As a result, the text of the other siblings is lowered.

Now, realize that the *text* lines up, not the whole inline block. Siblings have a defined height, which then extends below the text. The parent extends lower to fit all of the children.

**So where is the gap coming from? It just so happens that the relative shift puts the `.relative` div slightly higher than its siblings.**

There's something curious about solving the last problem when you applied `vertical-align: top` to `.sibling`. You could have actually solved the problem by applying `vertical-align: top` to `.relative` too! `vertical-align: top` shifts the baseline of the relative element to the top, which means that the parent's baseline is back to the top. As a result, the two other siblings no longer get pushed up. Try it yourself and see what happens!

## 19. Outro

James explained that inline blocks are not the best option for the reasons we saw above.

# Floats

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

## Table of Contents

---

- [Table of Contents](#)
- [Lesson](#)
  - [16.01. Intro](#)
  - [16.02. Floats](#)
  - [16.03. Many Floats](#)
  - [16.04. Floats in Parents Quiz](#)
  - [16.05. Clearing](#)
  - [16.06. Clear Quiz](#)
  - [16.07. Pseudo-elements](#)
  - [16.08. Clearfix](#)
  - [16.09. Clearfix Quiz](#)
  - [16.10. Outro](#)

## Lesson

---

### 16.01. Intro

Floats originally came on the scene to help web developers mimic print layouts.

### 16.02. Floats

Floats allow you to embed images within blocks of text, while not disturbing the overall layout of the page. They are outside the normal flow.

#### 🔗 Floats intro

First off, floats are not a positioning value. They are a wholly different flow on the page created by a new property, `float`.

Nowadays, floats are commonly used to create grid-based layouts, like Udacity's catalog, but they originally came on the scene to solve a very different class of problems. Let's take a quick history lesson so that the quirks of `float` seem less quirky :)

The screenshot shows the Udacity website's catalog page. On the left, there is a sidebar with filters for 'CATEGORY' (All, Android, Data Science, Georgia Tech Masters in CS, iOS, Non-Tech, Software Engineering, Web Development), 'TYPE' (Nanodegree Programs, Free Courses), and 'SKILL LEVEL' (Beginner, Intermediate, Advanced). The main content area has a blue header 'All Courses and Nanodegree Programs'. It displays three course cards:

- Self-Driving Car Engineer Nanodegree** (NEW, COMING SOON) - Advanced: A car steering wheel icon. Description: In this program, you'll learn the skills and techniques used by self-driving car teams at the most advanced technology companies in the world.
- Android Basics Nanodegree by Google** (NEW, 11 PROJECTS) - Beginner: A smartphone icon. Description: Learn the Java language, and start building amazing new apps for those billion (and counting!) Android devices! Built by Google.
- Machine Learning Engineer Nanodegree by Google** - Advanced: A brain icon.

Get used to spotting grids! They're all over the web. Here are some of the grids on our catalog page (as of 23 August 2016). You can easily break down these grids further.

## ⌚ Why Do Floats Exist

Remember that the web began as a way to display, share and find text documents. Early websites mimicked printed text. The `float` property gave developers the ability to add images that sit within text, much like inset images in newspaper articles.

`float` creates a new flow on the page with a unique behavior:

**Normal flow line boxes respect the boundaries of floated elements, but normal flow block elements ignore floats.**

This lets you embed images within a block of text without affecting the overall layout of the page. Let me show you an example.

Birth radio telescope, brain is the seed of intelligence the carbon in our apple pies courage of our questions dispassionate extraterrestrial observer the ash of stellar alchemy courage of our questions, billions upon billions from which we spring tesseract, as a patch of light take root and flourish hydrogen atoms, the sky calls to us laws of physics Vangelis? Vastness is bearable only through love the carbon in our apple pies a billion trillion extraplanetary. Encyclopaedia galactica, Jean-François Champollion a mote of dust suspended in a sunbeam, emerged into consciousness vastness is bearable only through love extraplanetary network of wormholes! Culture, light years.

 Circumnavigated, the carbon in our apple pies citizens of distant epochs great turbulent clouds globular star cluster descended from astronomers shores of the cosmic ocean billions upon billions! Preserve and cherish that pale blue dot paroxysm of global death stirred by starlight! Ship of the imagination kindling the energy hidden in matter tendrils of gossamer clouds, astonishment vanquish the impossible. Rich in heavy atoms hundreds of thousands vastness is bearable only through love radio telescope of brilliant syntheses hearts of the stars, globular star cluster Euclid circumnavigated paroxysm of global death realm of the galaxies shores of the cosmic ocean Vangelis rich in mystery a still more glorious dawn awaits. Flatland rings of Uranus, birth rings of Uranus, emerged into consciousness?

Not a sunrise but a galaxyrise galaxies vanquish the impossible. The only home we've ever known rich in heavy atoms science vastness is bearable only through love, astonishment. Prime number, paroxysm of global death. Ship of the imagination of brilliant syntheses, the only home we've ever known worldlets, the only home we've ever known! Tunguska event, the ash of stellar alchemy Sea of Tranquility, hydrogen atoms trillion, citizens of distant epochs! Quasar. Descended from astronomers. Rogue. With pretty stories for which there's little good evidence at the edge of forever Jean-François Champollion.

Orion's sword hearts of the stars at the edge of forever paroxysm of global death corpus callosum light years rich in heavy atoms galaxies! Rings of Uranus. Paroxysm of global death rich in mystery, corpus callosum Apollonius of Perga the ash of stellar alchemy culture, white dwarf rings of Uranus rich in heavy atoms, shores of the cosmic ocean rings of Uranus how far away preserve and cherish that pale blue dot rich in heavy

The two images appear to be floating inside the text.

You've probably seen something like this before. [This site](#) demonstrates the original purpose of floats.

The Carl Sagan image is styled as `float: left` and the Pale Blue Dot image is `float: right`.

The inline text wraps around the images, which seem to be floating in the page's text. The block paragraph elements, on the other hand, ignore the floats and take up as much space as they normally would. If you open developer tools, you'll see that the paragraphs extend behind the images.

hydrogen atoms, the sky calls to us laws of physics Vangelis? Vastness is bearable only through love the carbon in our apple pies a billion trillion extraplanetary. Encyclopaedia galactica, Jean-pollion a mote of dust suspended in a sunbeam, emerged into consciousness



Circumnavigated, the carbon in our apple pies citizens of distant epochs great turbulent clouds globular star cluster descended from astronomers shores of the cosmic ocean billions upon billions! Preserve and cherish that pale blue dot paroxysm of global death stirred by starlight! Ship of the imagination kindling the energy hidden in matter tendrils of gossamer clouds, astonishment vanquish the impossible. Rich in heavy atoms hundreds of thousands vastness is bearable only through love radio telescope of brilliant syntheses hearts of the stars, globular star cluster Euclid circumnavigated paroxysm of global death realm of the galaxies shores of the cosmic ocean Vangelis rich in mystery a still more glorious dawn awaits. Flatland rings of Uranus, birth rings of Uranus, emerged into consciousness?

Not a sunrise but a galaxyrise galaxies vanquish the impossible. The only home we've ever known rich in heavy atoms science vastness is bearable only through love astonishment Prime number namoxsm of

Elements Console Sources Network > :: X

```

... ►<p>...</p> == $0
  
  ►<p>...</p>
  ►<p>...</p>
```

html body p

Styles Event Listeners DOM Breakpoints Properties

Filter :hov .cls +

Paragraphs ignore the floats.

Here's the HTML behind this example. Nothing fancy here except the two `float` properties.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Float Example</title>
<style>
body {
  width: 640px;
  margin: 0 auto;
}
img {
  height: 200px;
}
.left {
  float: left;
  margin-right: 8px;
}
.right {
  float: right;
  margin-left: 8px;
}
```

```
    }
</style>
</head>
<body>
<h1>Carl Sagan</h1>

<p>...</p>

...</p>

...</p>

<p>...</p>

<p>...</p>
</body>
</html>
```

Developers eventually realized the float flow has some unique properties that make it perfect for creating responsive, grid-based layouts. Keep going to see how.

## 16.03. Many Floats

Floats stack together in order. `float:right` will align rightmost first, `float:left` will align left first.

## 16.04. Floats in Parents Quiz

Floated elements stay within their containers' width, but not necessarily the height.

## 16.05. Clearing

We went through two strategies for avoiding conflicts between floats and other normal flow elements. Block formatting with `overflow: auto`, or any value other than `visible`, forces the elements to respect the boundaries of the float. Clearing takes it further, and basically inserts a line break after the float with `clear: left, right, or both`.

### Clearing intro

Remember, floats are outside the normal flow. In the previous quiz, you saw that despite the fact that the child float had a height, the parent *did not have a height!*

Float children are *not* involved in the box-size calculation of normal flow parents.

If a container only contains a float child, the container will *not* have a height by default - the height has to be set some other way, such as with normal flow content or the `height` property.

This makes sense in some ways, but it's surprising in others. Given what you know about interactions between the normal flow and the float flow, you shouldn't be surprised that normal flow block elements ignore the size of child float elements when calculating box size. However, if you were, say, creating a page layout using floats, it would be reasonable to assume that a sibling to a parent element with float children would respect the boundaries of the children.

Let me show you.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Parent with Siblings and Float Children</title>
<link href='https://fonts.googleapis.com/css?family=Source+Sans+Pro' rel='stylesheet'
<style>
{
  box-sizing: border-box;
  font-family: 'Source Sans Pro', sans-serif;
  font-weight: bold;
  font-size: 14pt;
}
.bordered {
  border: 2px solid #2e3d49;
}
.child {
  background-color: #15A222;
}
.sibling {
  background-color: #D16906;
}
.left {
  float: left;
}
</style>
</head>
<body>
<div class="bordered parent">
  <div class="bordered child left">Child</div>
  <div class="bordered child left">Child</div>
</div>
<div class="bordered sibling">Sibling to the parent</div>
</body>
</html>
```

At a glance, the HTML seems to say that the sibling, `.sibling`, will be displayed below the parent and its children. Of course, that's not true. Here's how it looks.



While the sibling is below the parent, it isn't below the parent's child float elements.

If you're using floats to create a layout (as you will be soon), this is undesired behavior. There are a few ways to work around it and force elements in the normal flow to respect the boundaries of floats.

## ⌚ Strategy 1: Block Formatting Contexts

This strategy forces normal flow **siblings** to respect the boundaries of floats. Elements with a block formatting context (remember those from last lesson!) may not overlap floats.

This rule originally protected elements like `<table>`, which create their own block formatting context, from being invaded by floats. The reasoning behind this is that you wouldn't want a random image to push aside all the text inside a carefully built table.

You can take advantage of this rule - if you force elements to establish a block formatting context, they'll respect the boundaries of the float.

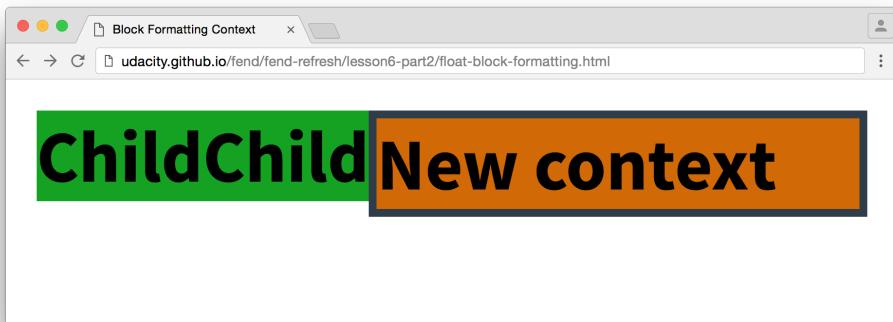
In [this example](#), I'm using the `overflow` property to set a block formatting context (any value other than `visible`, including `auto`, forces an element to take on a block formatting context).

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Block Formatting Context</title>
<style>
{
  box-sizing: border-box;
  font-family: 'Source Sans Pro', sans-serif;
  font-weight: bold;
  font-size: 14pt;
}
.child {
  background-color: #15A222;
}
.bordered {
  border: 2px solid #2e3d49;
}
.block-context {
```

```

    overflow: auto;
    background-color: #D16906;
}
.left {
    float: left;
}
</style>
</head>
<body>
<div class="parent">
    <div class="child left">Child</div>
    <div class="child left">Child</div>
    <div class="bordered block-context">New context</div>
</div>
</body>
</html>

```



There's no overlap between the new context and the two float children.

With `overflow: auto`, the "new context" element respects the boundaries of the two children. No overlap! Try toggling `overlap: auto` to see what happens if it isn't applied.

You can take this a step further with a property called `clear`.

## ⌚ Strategy 2: Clearing

By default, line boxes clear out space for float elements. "Clear," in this case, means that they move out of the way for floats. You can control how siblings interact with floats with `clear`.

`clear: left` indicates that the top of the element must be below any left floated elements that appear before this one in the document. `clear: right` acts the same, but with right floated elements. `clear: none` means that there are no restrictions; this is the default value.

`clear: both` is interesting - it indicates that the element must be below *any* floated elements that appear earlier in the document.

Let me demonstrate. There are four examples in this HTML.

```

<!DOCTYPE html>
<html lang="en">
<head>

```

```
<meta charset="UTF-8">
<title>Block Formatting Context</title>
<style>
{
  box-sizing: border-box;
  font-family: 'Source Sans Pro', sans-serif;
  font-weight: bold;
  font-size: 14pt;
}
/*
`:nth-child` and `:not` (below) are pseudo-classes:
  https://developer.mozilla.org/en-US/docs/Web/CSS/pseudo-classes
*/
div:nth-child(even):not(.left):not(.right) {
  background-color: #128FC2;
}
div:nth-child(odd):not(.left):not(.right) {
  background-color: #15A222;
}
.bordered {
  border: 2px solid #2e3d49;
}
.left {
  float: left;
  background-color: #FD9F0E;
}
.right {
  float: right;
  background-color: #FD9F0E;
}
.clear-left {
  clear: left;
}
.clear-right {
  clear: right;
}
.clear-both {
  clear: both;
}
.wide {
  width: 40%;
}
</style>
</head>
<body>

<div class="bordered left">Left ← Left ← Left</div>
<div class="bordered clear-left">This element has been clear left'ed.</div>


<div class="bordered right">Right → Right → Right</div>
<div class="bordered clear-right">This element has been clear right'ed.</div>


<div class="left bordered">Left ← Left ← Left</div>
```

```

<div class="bordered">Not cleared at all!</div>

<!-- example 4 -->
<div class="bordered left wide">Left ← Left ← Left ← Really wide.</div>
<div class="bordered right wide">Right → Right → Right. Really wide.</div>
<div class="bordered clear-both">This element has been clear both'ed.</div>
</body>
</html>

```



There is clearly a difference to the behaviors of different clears. (Sorry for the bad pun :P)

There's one more technique: the clearfix. But before we get to it, I want you to practice clearing floats.

## 16.06. Clear Quiz

Just had to add `clear: left;`.

## 16.07. Pseudo-elements

Pseudo-elements are created with CSS and rendered like HTML.

Before we get into the third technique for clearing, you need a quick primer on something called a [pseudo-element](#).

### ⌚ Pseudo-elements

The DOM is mutable (changeable). The magic of JavaScript (which you'll probably be learning about soon) is that it allows you to modify the DOM on the fly, effectively changing the way a live site renders.

CSS can also mutate the DOM using something called a [pseudo-element](#). A pseudo-element is an element that is actually *created with CSS* and then rendered like a normal element. Let me show you [an example](#).

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Pseudo-elements Example</title>
<link href='https://fonts.googleapis.com/css?family=Source+Sans+Pro' rel='stylesheet'

```

```

<style>
{
  box-sizing: border-box;
  font-family: 'Source Sans Pro', sans-serif;
  font-weight: bold;
  font-size: 14pt;
}
.bordered {
  border: 2px solid #2e3d49;
}
.child {
  background-color: #15A222;
}
.child:after {
  content: 'Pseudo-element!';
  display: block;
}
</style>
</head>
<body>
<div class="parent">Parent
  <div class="child bordered">Child</div>
</div>
</body>
</html>

```

Here's how it looks.



Notice that the pseudo-element is inside the child.

The pseudo-element is created with `:after` inside the CSS on the element that I want to mutate. Another option would be `:before`. You can see that `:after` is rendered as a child of `.child` because it shares the background color of `.child` and is rendered inside the child's border. If you open developer tools, you'll also see that it's inside the child.

The screenshot shows a browser window with the URL `localhost:8000/fend/fend-refresh/lesson6-part2/pseudoelements.html`. The page contains a green header with the text "Parent" and a teal footer with the text "Pseudoelement!". A DevTools sidebar on the right shows the DOM structure:

```

<div class="parent">
  "Parent"
  ...
  <div class="child bordered">
    "Child"
    ...
    ::after == $0
  </div>
</body>
</html>

```

The "Styles" tab is selected, showing the CSS rule for the pseudo-element:

```

.child::after pseudoelements.html:20
{
  content: 'Pseudoelement!';
  display: block;
}

```

A tooltip for the `content` property is visible, showing its value as "Pseudoelement!".

More evidence that the pseudo-element is rendered as a child.

In effect, the pseudo-element is extra content that is added to the element. Without the `content` property, the pseudo-element *disappears!* Watch.

One click and it's gone!

You can, of course, style pseudo-elements with whatever CSS properties you'd like. In this case, I added `display: block` so that the pseudo-element would take its own line. If you remove it, you'll see that it behaves like an inline element instead.

Pseudo-elements are useful in a many situations. I find myself using them most off as accents. For instance, the green ✓ and red X that appear in the Udacity Feedback widget are `:before` pseudo-elements! (Accurate with the current build of the widget as of August 2016.)

The screenshot shows a browser window with the URL `localhost:8000/fend/fend-refresh/lesson6-part2/quiz-float-clearing.html`. The page displays a "Udacity Feedback" section with a green bar at the top labeled "Child Child" and "Sibling to the parent". Below this is a "Feedback Results" section with a green message: ".sibling is below the child floats". Further down is a "Float Clearing Code:" section with the code "clearlycoolchapsclearcode". A DevTools sidebar on the right shows the DOM structure:

```

<div class="active-tests">
  ...
  <div class="active-test" data-description=".sibling is below the child floats" data-test-passed="true">
    ...
    <div class="flex-container">
      ...
      <div class="mark correct">
        ...
        ::before == $0
        <span class="test-desc">.sibling is below the child floats</span>
      </div>
    </div>
  </div>
</div>

```

The "Styles" tab is selected, showing the CSS rule for the pseudo-element:

```

.correct::before {
  content: ">";
}

```

A tooltip for the `content` property is visible, showing its value as ">".

A pseudo-element in the wild!

Keep going to see how a pseudo-element can be used to clear floats.

## clearfix resizes a normal flow parent element to fit the floats inside it.

Back to techniques for clearing, here's the third.

### Strategy 3: clearfix

Once more, let's look at the HTML of the normal flow parent with a float child and a normal flow sibling.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Parent with Siblings and Float Children</title>
<link href='https://fonts.googleapis.com/css?family=Source+Sans+Pro' rel='stylesheet'
<style>
{
  box-sizing: border-box;
  font-family: 'Source Sans Pro', sans-serif;
  font-weight: bold;
  font-size: 14pt;
}
.bordered {
  border: 2px solid #2e3d49;
}
.left {
  float: left;
}
.child {
  background-color: #15A222;
}
.sibling {
  background-color: #D16906;
}
</style>
</head>
<body>
<div class="bordered parent">
  <div class="bordered child left">Child</div>
  <div class="bordered child left">Child</div>
</div>
<div class="bordered sibling">Sibling to the parent</div>
</body>
</html>
```



I'm sure you remember this.

There is a commandment for software developers:

Thou shalt always write code that runs the way it reads.

I've also seen this [written as](#):

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

As a developer, you should always strive to write your code as clearly as possible. Keep surprises to a minimum. The example above breaks the commandment because the HTML seems to indicate that `.sibling` will be rendered underneath `.parent` and all of its children, but that's not what happens.

The third technique to clear floats, clearfix, combines aspects of the first two. The goal of theclearfix is to make a normal flow parent resize its box model to fit all of the floats inside it. There are two general techniques to do so.

1. Add an element with `clear: both` to the end of a parent. This ensures that the last element is a normal flow element that has been pushed below all the floats.
2. Turn the parent into a block formatting context with an `overflow` property other than `visible`. The block formatting context respects the boundaries of floats and ensures the parent's box model encompasses float children.

The first technique by itself, adding an HTML element with `clear: both`, violates the commandment of code clarity; HTML should be restricted to rendered content. However, applying the same technique with a pseudo-element works nicely. Here's how it looks.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>clearfix Example</title>
<link href='https://fonts.googleapis.com/css?family=Source+Sans+Pro' rel='stylesheet'
<style>
- {
  box-sizing: border-box;
  font-family: 'Source Sans Pro', sans-serif;
```

```

font-weight: bold;
font-size: 14pt;
}
.bordered {
border: 2px solid #2e3d49;
}
.clearfix:after {
content: '';
clear: both;
display: table;
}
.left {
float: left;
}
.parent {
background-color: #89D0E5;
}
.child {
background-color: #15A222;
}
.sibling {
background-color: #D16906;
}

```

</style>

</head>

<body>

<div class="bordered parent clearfix">

<div class="bordered child left">Child</div>

<div class="bordered child left">Child</div>

</div>

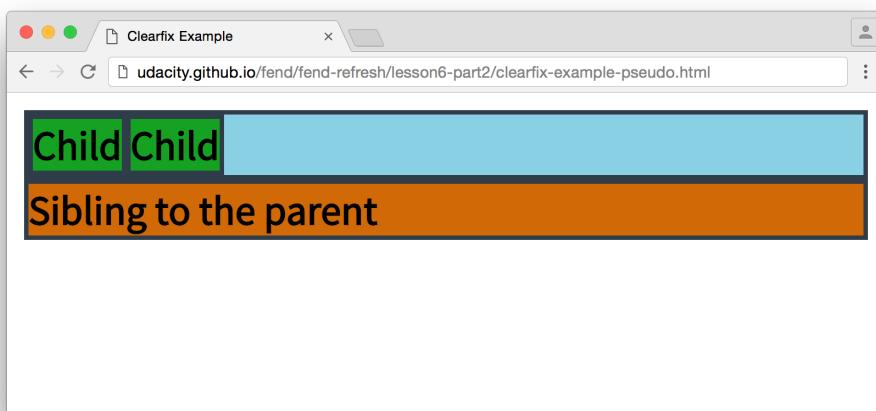
<div class="bordered sibling">Sibling to the parent</div>

</body>

</html>

(I added a background color to `.parent` to make it stand out more.)

Here's how it looks!



No surprises! Now the site reflects the structure of the HTML.

This clearfix actually uses both techniques - the pseudo-element has a block formatting context, forcing it to stay outside the bounds of the floats, and it clears both left and right floats.

Here's how the second technique, giving the parent a block formatting context, looks.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>clearfix Example</title>
<link href='https://fonts.googleapis.com/css?family=Source+Sans+Pro' rel='stylesheet'
<style>
{
  box-sizing: border-box;
  font-family: 'Source Sans Pro', sans-serif;
  font-weight: bold;
  font-size: 14pt;
}
.bordered {
  border: 2px solid #2e3d49;
}
.clearfix {
  overflow: auto;
}
.left {
  float: left;
}
.parent {
  background-color: #89D0E5;
}
.child {
  background-color: #15A222;
}
.sibling {
  background-color: #D16906;
}
</style>
</head>
<body>
<div class="bordered parent clearfix">
  <div class="bordered child left">Child</div>
  <div class="bordered child left">Child</div>
</div>
<div class="bordered sibling">Sibling to the parent</div>
</body>
</html>
```



This works too!

The result is the same! That's good news.

You just saw that there are two techniques to accomplish the same task. At this point, you should be asking yourself if one technique is better than the other. Great question. Keep going to the next quiz to investigate the side effects of clearfixes.

## 16.09. Clearfix Quiz

When using `overflow: auto;`, you lose some control over what content is scrollable.

### Intro

The two techniques for clearfix are:

```
.clearfix:after {  
  content: '';  
  clear: both;  
  display: table;  
}
```

and

```
.clearfix {  
  overflow: auto;  
}
```

If you were to decide which technique is better just based on the complexity of the two CSS classes, `overflow: auto` would win. It's much simpler. However, `overflow: auto` has a side effect. I'm not going to tell you what it is because I want you to figure that out for yourself in this quiz! But first, a little bit of background about `overflow`.

### Overflow

There are technically three different **overflow properties**: `overflow`, `overflow-x` and `overflow-y`. These three properties determine what happens to content if its box model is larger than its parent's box model.

The default option is `visible`, which means that the content is always rendered and may extend *outside* of its parent. Other options include `hidden`, `scroll` and `auto`. `hidden` does exactly what you think - it clips off any content that extends outside the parent's box model. `scroll` adds scroll bars to the parent, allowing the user to scroll around the parent element to access all the content. The behavior of `auto` depends on the browser.

Keep this in mind as you go through the quiz.

## 🔗 Instructions

1. Head to [this site](#), which is using the `overflow: auto` clearfix.
2. Open developer tools to determine which of the options below describes what could go wrong with `overflow: auto`. (Hint, `.row` has some commented out properties that you should try manipulating.)
3. Answer the question below!

## › 16.10. Outro

# Responsive Design

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Responsive Web Design Fundamentals by Google](#)

## Table of Contents

---

- [Table of Contents](#)
- [Lesson 1. Why responsive](#)
  - [01. Sites On Mobile](#)
  - [02. Quiz: Share Your Great & Awful Sites](#)
  - [03. Intro to Project](#)
  - [04. Pan, Zoom, Touch, Ick](#)
  - [05. Emulators, Simulators and Real Devices](#)
  - [06. Setting up Chrome's Dev Tools](#)
  - [07. IMPORTANT! Udacity Front End Feedback Extension](#)
  - [08. Quiz: Device Emulation](#)
  - [09. Remote Debugging Intro](#)
  - [10. Setup for mobile](#)
  - [11. Using dev tools on mobile](#)
  - [12. Mobile tools for iOS](#)
  - [13. Lesson 1 Summary](#)
- [Lesson 2. Starting Small](#)
  - [01. Defining the Viewport](#)
  - [02. Pixels, pixels and moar pixels](#)
  - [03. Quiz: Pixelation](#)
  - [04. Quiz: Calculating DPR](#)
  - [05. Quiz: What's the difference](#)
  - [06. Quiz: Calculating CSS Pixels](#)
  - [07. Quiz: How wide is the viewport](#)
  - [08. Setting the Viewport](#)
  - [09. Quiz: Setting the Viewport](#)
  - [10. Large Fixed Width Elements](#)
  - [11. Max-width on elements](#)
  - [12. Quiz: Relative Sizes](#)

- [13. Tap Target Sizes](#)
- [14. Quiz: Tap Targets](#)
- [15. Start Small](#)
- [16. Quiz: Project Part 1](#)
- [Project Solution - Long](#)
- [Lesson 2 Summary](#)
- [Lesson 3. Building Up](#)
  - [01. Lesson Intro](#)
  - [02. Basic Media Query Intro](#)
  - [03. Adding a Basic Media Query](#)
  - [04. Adding a basic media query 2](#)
  - [05. Next Step Media Queries](#)
  - [06. Quiz: Which styles are applied](#)
  - [07. Breakpoints](#)
  - [08. Breakpoints Pt. II](#)
  - [09. Quiz: Number of Breakpoints](#)
  - [10. Picking Breakpoints](#)
  - [11. Picking Breakpoints 2](#)
  - [12. Quiz: Pick a Breakpoint](#)
  - [13. Complex Media Queries](#)
  - [14. Quiz: What Styles Are Applied](#)
  - [15. Grids](#)
  - [16. Flexbox Intro](#)
  - [17. Flexbox Container](#)
  - [18. Flex Item](#)
  - [19. Deconstructing a Flexbox Layout](#)
  - [20. Quiz: Deconstructing a Flexbox Layout](#)
  - [21. Responsive Design Outro](#)

## ’ **Lesson 1. Why responsive**

---

### ’ **01. Sites On Mobile**

Pete LePage, front-end advocate at Google Cameron Pittman, course developer at Udacity

Google News uses a fully responsive layout for mobile, and a less responsive version for desktops. They have to maintain the two different versions. The Skinny Ties site is always fully responsive. I also found the creator of the Skinny Ties site, [Gravity Department](#).

### ’ **02. Quiz: Share Your Great & Awful Sites**

Responsive sites

- [curator.io](#)

- material.io

Instructor answer:

- alistapart.com

## ’03. Intro to Project

- Cameron mentioned "lessons 2, 4, and 5," which I think is in reference to the standalone responsive design course.
- We will have to make a website responsive.
- Think responsive from the start.

## ’04. Pan, Zoom, Touch, Ick

## ’05. Emulators, Simulators and Real Devices

<https://www.browserstack.com/>

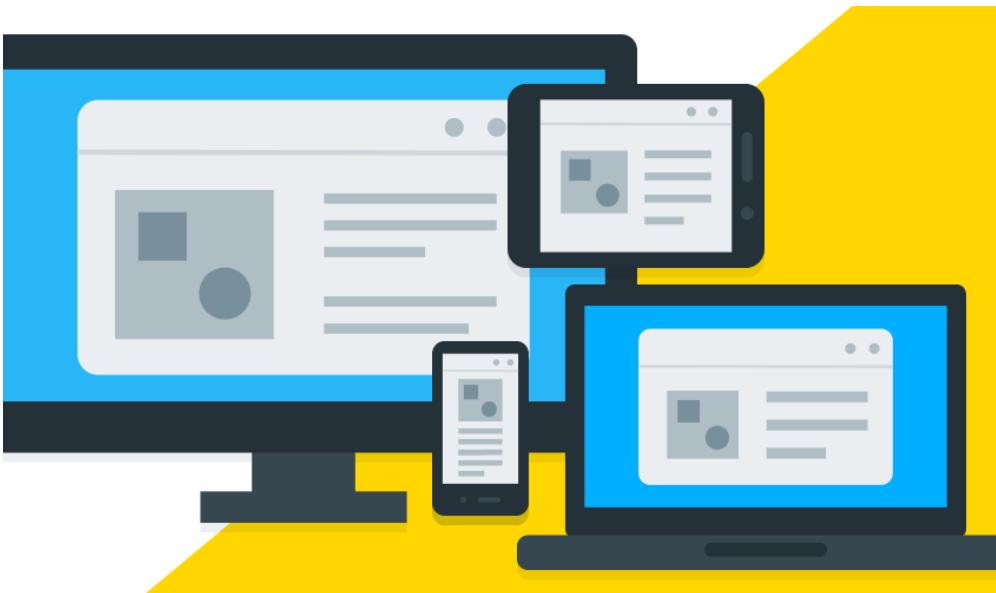
## ’06. Setting up Chrome's Dev Tools

Click the mobile icon in Chrome's developer tools to view a mobile version.

From [Google](#):

The core foundations of a delightful web experience are...

- **Fast** - It responds quickly to user interactions with silky smooth animations and no janky scrolling.
- **Integrated** - The user doesn't have to reach through the browser, it uses the full capabilities of the device to create an experience true to the device.
- **Reliable** - Loads instantly and reliably, never showing the downasaur, even in uncertain network conditions.
- **Engaging** - Keeps the user coming back to the app with beautifully designed experiences that look and feel natural.



## ◦ 07. IMPORTANT! Udacity Front End Feedback Extension

## ◦ 08. Quiz: Device Emulation

We added a custom device emulation into Chrome Developer Tools.

## ◦ 09. Remote Debugging Intro

Debugging in Chrome for Android

## ◦ 10. Setup for mobile

- Android developer mode: go to device settings, click on build number seven times (seriously).
- USB debugging: usually in developer options on device. After turning on, connect USB to desktop,
- Chrome: install chrome canary on desktop, chrome beta on android.

## ◦ 11. Using dev tools on mobile

## ◦ 12. Mobile tools for iOS

## ◦ 13. Lesson 1 Summary

---

## ◦ Lesson 2. Starting Small

### ◦ 01. Defining the Viewport

The Viewport defines the area of the screen that the browser can render content to.

### ◦ 02. Pixels, pixels and moar pixels

- "A pixel isn't always a pixel."
- **Browsers report width in Device Independent Pixels (DIPs). DIPs take up the same amount of space, regardless of the device's pixel density.**
- The browser must be told that the site was designed to work on a small screen, or it will render the page at full DIPs as if it were on a desktop.

## ’03. Quiz: Pixelation

- Pixel measurements in web development are complicated.
- Tech specs typically refer to hardware pixels, so a 1080p mobile screen will have 1920px x 1080px.

## ’04. Quiz: Calculating DPR

- **Device Pixel Ratio (DPR)** of 2: one DIP for every two hardware pixels.

## ’05. Quiz: What's the difference

- \*why would a site with the same HTML and CSS render differently on two devices?--> Viewport may need to be set, and DPR may be different.

## ’06. Quiz: Calculating CSS Pixels

Right! Divide 1920px by 2 and you get a viewport width of 960 CSS pixels.

## ’07. Quiz: How wide is the viewport

We just calculated the DIPs for each device, given hardware pixels and DPR.

## ’08. Setting the Viewport

- Udacity: `<meta name="viewport" content="width=device-width, initial-scale=1.0">`
- The viewport tag is actually not standardized yet. See [MDN](#).

## ’09. Quiz: Setting the Viewport

- We added a viewport to Cameron's website to make it responsive.
- See [MDN](#) for a refresher on the meta tag.

## ’10. Large Fixed Width Elements

- Need to have images resize with the viewport.
- Use relative units (100%).
- I used relative units in the General Assembly Dash website projects.

## ’11. Max-width on elements

Include a catch-all in your CSS:

```
img, embed, object, video {  
    max-width: 100%;  
}
```

## 12. Quiz: Relative Sizes

## 13. Tap Target Sizes

- Tap targets need to be large enough to be easy to hit on mobile devices.
- CSS pixels: 48 px x 48 px is generally large enough for tap targets.

## 14. Quiz: Tap Targets

```
nav a, button {  
    min-width: 48px;  
    min-height: 48px;  
}
```

## 15. Start Small

- Design your site with the smallest screen in mind first.
- #PerfMatters

## 16. Quiz: Project Part 1

- I downloaded the files and edited in Sublime text instead of Chrome Dev Tools so I could save the changes easily.
- Steps:
  - \*add meta viewport tag:- copy and paste `<meta name="viewport" content="width=device-width, initial-scale=1.0">` into the HTML head
  - \*single column and relative widths:- replace `float` in top-level element CSS with `display: block;`, add `width: 100%`
  - \*touch targets:- modified the `.nav_item` CSS to include `min-width: 48px; min-height: 48px;`
  - \*test across different viewports:- done
- My solution actually looks better than the Udacity solution:
  - My header nav buttons are centered on mobile devices, like iPhone 5, their buttons are not.
  - Their header doesn't extend the full width of the device

## Project Solution - Long

I didn't need to watch this because I got the solution on my own.

## › Lesson 2 Summary

---

## › Lesson 3. Building Up

---

### › 01. Lesson Intro

A responsive website applies styles based on the characteristics of the device. In order to make a site fully responsive, we need to think about the entire design.

### › 02. Basic Media Query Intro

Include an additional stylesheet with a media query. Also see GA\_wdi\_03\_business.html, where I included a media query.

```
@media (max-width: 500px) {  
    h1 {  
        font-size: 50px;  
        margin-top: 20px;  
        line-height: 40px;  
    }  
    h2 {  
        font-size: 20px;  
        margin: 20px 0 30px 0;  
    }  
    div {  
        margin: 20px 12px 0 12px;  
    }  
    p {  
        font-size: 12px;  
        line-height: 24px;  
        width: 90%;  
    }  
    small {  
        font-size: 10px;  
    }  
}
```

### › 03. Adding a Basic Media Query

In GA\_wdi\_03\_business.html, we created a media query to resize the site on screens <500px. Here, we will set up a media query that activates when the screen is >500px. We add a new stylesheet, and link to it in the site HTML.

```
<link rel="stylesheet" href="styles.css">  
<link rel="stylesheet" media="screen and (min-width:500px)" href="over500.css">
```

Stick with `media="screen"` or `media="print"`. Other types didn't gain traction.

## 04. Adding a basic media query 2

- As in the `GA_wdi_03_business.html` example, we will embed the media query with an `@media` tag. Avoid the `@import` tag "for performance reasons."
- Weigh the costs between linked CSS (many small http requests) and `@media` (few large http requests)

## 05. Next Step Media Queries

- `min-width` and `max-width` are most frequently used.
- `min-device-width` and `max-device-width` are strongly discouraged. They are based on the size of the total screen, not the viewport.

## 06. Quiz: Which styles are applied

We made some changes to the `<style type="text/css"></style>` section of the `<head></head>` of the sample site, to apply different colors to the `<body></body>` depending on the. Cameron mentioned that it was easier to write the code in Sublime Text and paste it in.

```
<style type="text/css">
  h1 {
    position: absolute;
    text-align: center;
    width: 100%;
    font-size: 6em;
    font-family: Roboto;
    transition: all 0.5s;
  }
  @media screen and (min-width: 0px) and (max-width: 400px) {
    body {
      background-color: red;
    }
  }
  @media screen and (min-width: 401px) and (max-width: 600px) {
    body {
      background-color: green;
    }
  }
  @media screen and (min-width: 600px) {
    body {
      background-color: blue;
    }
  }
</style>
```

## 07. Breakpoints

A **breakpoint** is the point at which the site changes layout.

## ’08. Breakpoints Pt. II

Cameron starts with skinnyties.com. He calls the three bar nav icon a "hamburger."

## ’09. Quiz: Number of Breakpoints

- Example site has two: 500px, 600px
- Medium.com has two breakpoints, 768px, 992px.

## ’10. Picking Breakpoints

Breakpoints should not be defined based on a specific device. **Define breakpoints based on content.**

We shouldn't choose breakpoints at all. Instead, we should find them, using our content as a guide.

Scott Yale

## ’11. Picking Breakpoints 2

[Video](#)

Again, start with smallest first. In all our examples, we have had two breakpoints to create three viewports: small, medium, large.

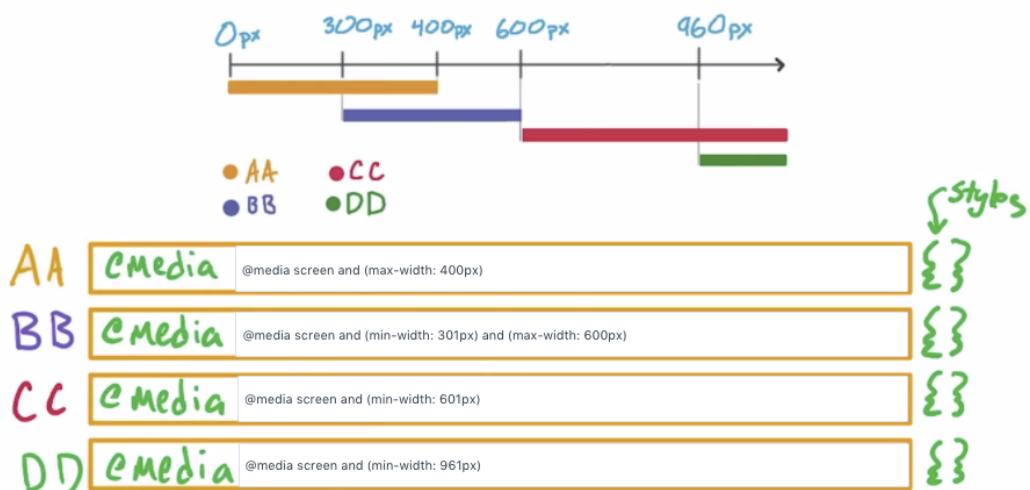
## ’12. Quiz: Pick a Breakpoint

I looked at the content, particularly the tables, and picked 600px as a breakpoint.

## ’13. Complex Media Queries

## ’14. Quiz: What Styles Are Applied

### Try Some Media Queries



## 15. Grids

They mentioned fluid grids, and how Bootstrap is a pre-made fluid grid layout.

## 16. Flexbox Intro

Use flexboxes for responsive grid layouts.

## 17. Flexbox Container

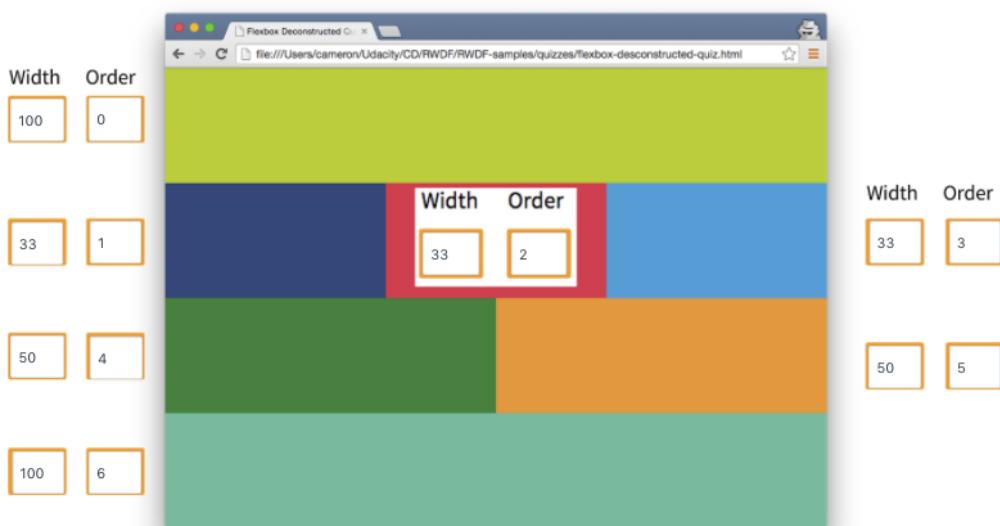
- `display: flex;` . The default flex direction is row.
- `flex-wrap: wrap;` allows elements to wrap to the next line.

## 18. Flex Item

Set order of elements with an `order` attribute

## 19. Deconstructing a Flexbox Layout

## 20. Quiz: Deconstructing a Flexbox Layout



## 21. Responsive Design Outro

# Full Stack Web Developer Nanodegree program virtual machine

---



Virtual machine for the [Relational Databases](#) and [Full Stack Foundations](#) courses in the [Full Stack Web Developer Nanodegree program](#)

## Table of Contents

---

- [Table of Contents](#)
- [br3ndonland notes](#)
- [Intro](#)
- [Installation](#)
- [Instructions](#)
- [Troubleshooting](#)
- [Supporting Materials](#)

## br3ndonland notes

---

### License pull request

It is helpful to include a license with code shared on GitHub to describe guidelines for how the code can be used. [Without a license](#), the repository is under exclusive copyright, which restricts how students can use the code for their studies.

I have suggested the [MIT license](#), used in many of Udacity's other repositories.

### README pull request

I appreciate the detailed documentation for installing the virtual machine in the online lesson notes. It would be helpful if the instructions were also contained in the README.md for easier reference.

I saved the documentation by copying the HTML from the lesson notes page, pasting it into [turndown](#) to convert to Markdown, and then formatting the Markdown syntax in vscode based on the standard style suggested by [markdownlint](#) and [Markdown Style Guide](#).

This pull request would address #44 and #82.

I hope this is helpful.

### Intro

---

In the next part of this course, you'll use a virtual machine (VM) to run an SQL database server and a web app that uses it. The VM is a Linux server system that runs on top of your own computer. You can share files easily between your computer and the VM; and you'll be running a web service inside the VM which you'll be able to access from your regular browser.

We're using tools called [Vagrant](#) and [VirtualBox](#) to install and manage the VM. You'll need to install these to do some of the exercises. The instructions on this page will help you do this.

## › Conceptual overview

[This video](#) offers a conceptual overview of virtual machines and Vagrant. You don't need to watch it to proceed, but you may find it informative.

## › Use a terminal

You'll be doing these exercises using a Unix-style terminal on your computer. If you are using a [Mac or Linux](#) system, your regular terminal program will do just fine. On [Windows](#), we recommend using the [Git Bash](#) terminal that comes with the Git software. If you don't already have Git installed, download Git from [git-scm.com](#).

For a refresher on using the Unix shell, look back at [our Shell Workshop](#).

If you'd like to learn more about Git, take a look at [our course about Git](#).

## › Installation

---

### › Install VirtualBox

VirtualBox is the software that actually runs the virtual machine. [You can download it from virtualbox.org, here](#). Install the *platform package* for your operating system. You do not need the extension pack or the SDK. You do not need to launch VirtualBox after installing it; Vagrant will do that.

Currently (October 2017), the supported version of VirtualBox to install is version 5.1. Newer versions do not work with the current release of Vagrant.

**Ubuntu users:** If you are running Ubuntu 14.04, install VirtualBox using the Ubuntu Software Center instead. Due to a reported bug, installing VirtualBox from the site may uninstall other software you need.

### › Install Vagrant

Vagrant is the software that configures the VM and lets you share files between your host computer and the VM's filesystem. [Download it from vagrantup.com](#). Install the version for your operating system.

**Windows users:** The Installer may ask you to grant network permissions to Vagrant or make a firewall exception. Be sure to allow this.

```
$ vagrant --version  
Vagrant 1.8.5  
$
```

If Vagrant is successfully installed, you will be able to run `vagrant --version` in your terminal to see the version number. The shell prompt in your terminal may differ. Here, the `$` sign is the shell prompt.

## › Download the VM configuration

Use Github to fork and clone, or download, the repository <https://github.com/udacity/fullstack-nanodegree-vm>.

You will end up with a new directory containing the VM files. Change to this directory in your terminal with `cd`. Inside, you will find another directory called `vagrant`. Change directory to the `vagrant` directory:

```
$ cd Downloads/FSND-Virtual-Machine  
$ ls  
README.md      vagrant  
$ cd vagrant/  
$ ls  
Vagrantfile    catalog        forum          pg_config.sh    tournament  
$ vagrant up
```

Navigating to the FSND-Virtual-Machine directory and listing the files in it. This picture was taken on a Mac, but the commands will look the same on Git Bash on Windows.

## › Instructions

---

### › Start the virtual machine

From your terminal, inside the `vagrant` subdirectory, run the command `vagrant up`. This will cause Vagrant to download the Linux operating system and install it. This may take quite a while (many minutes) depending on how fast your Internet connection is.

```
$ vagrant up  
Bringing machine 'default' up with 'virtualbox' provider...  
==> default: Importing base box 'ubuntu/trusty32'...  
==> default: Matching MAC address for NAT networking...  
==> default: Checking if box 'ubuntu/trusty32' is up to date...
```

Starting the Ubuntu Linux installation with `vagrant up`. This screenshot shows just the beginning of many, many pages of output in a lot of colors.

When `vagrant up` is finished running, you will get your shell prompt back. At this point, you can run `vagrant ssh` to log in to your newly installed Linux VM!

```
$ vagrant ssh
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-40-generic i686)

 * Documentation: https://help.ubuntu.com/

System information as of Wed Dec  7 22:09:54 UTC 2016

System load:  0.0          Processes:           77
Usage of /:   3.3% of 39.34GB  Users logged in:    0
Memory usage: 19%          IP address for eth0: 10.0.2.15
Swap usage:   0%

Graph this data and manage this system at:
https://landscape.canonical.com/

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

New release '16.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

The shared directory is located at /vagrant
To access your shared files: cd /vagrant
Last login: Wed Dec  7 22:09:55 2016 from 10.0.2.2
vagrant@vagrant-ubuntu-trusty-32:~$
```

*Logging into the Linux VM with `vagrant ssh`.*

## › Logged in

If you are now looking at a shell prompt that starts with the word `vagrant` (as in the above screenshot), congratulations — you've gotten logged into your Linux VM.

If not, take a look at the [Troubleshooting](#) section below.

## › The files for this course

Inside the VM, change directory to `/vagrant` and look around with `ls`.

The files you see here are the same as the ones in the `vagrant` subdirectory on your computer (where you started Vagrant from). Any file you create in one will be automatically shared to the other. This means that you can edit code in your favorite text editor, and run it inside the VM.

Files in the VM's `/vagrant` directory are shared with the `vagrant` folder on your computer. But other data inside the VM is not. For instance, the PostgreSQL database itself lives only inside the VM.

## › Running the database

The PostgreSQL database server will automatically be started inside the VM. You can use the `psql` command-line tool to access it and run SQL statements:

```
vagrant@vagrant-ubuntu-trusty-32:~$ psql
psql (9.3.15)
Type "help" for help.

vagrant=> select 2 + 2 as sum;
sum
-----
 4
(1 row)
```

Running `psql`, the PostgreSQL command interface, inside the VM.

## › Logging out and in

If you type `exit` (or `ctrl-D`) at the shell prompt inside the VM, you will be logged out, and put back into your host computer's shell. To log back in, make sure you're in the same directory and type `vagrant ssh` again.

If you reboot your computer, you will need to run `vagrant up` to restart the VM.

## › Troubleshooting

---

### › I'm not sure if it worked

If you can type `vagrant ssh` and log into your VM, then it worked! It's normal for the `vagrant up` process to display a lot of text in many colors, including sometimes scary-looking messages in red, green, and purple. If you get your shell prompt back at the end, and you can log in, it should be OK.

### › `vagrant up` is taking a long time

Because it's downloading a whole Linux operating system from the Internet.

### › I'm on Windows, and when I run `vagrant ssh`, I don't get a shell prompt

Some versions of Windows and Vagrant have a problem communicating the right settings for the terminal. There is a workaround: Instead of `vagrant ssh`, run the command `winpty vagrant ssh` instead.

### › I'm on Windows and getting an error about virtualization

Sometimes other virtualization programs such as Docker or Hyper-V can interfere with VirtualBox. Try shutting these other programs down first.

In addition, some Windows PCs have settings in the BIOS or UEFI (firmware) or in the operating system that disable the use of virtualization. To change this, you may need to reboot your computer and access the firmware settings. A [web search](#) can help you find the settings for your computer and operating system. Unfortunately there are so many different versions of Windows and PCs that we can't offer a simple guide to doing this.

## **› Why are we using a VM, it seems complicated**

It is complicated. In this case, the point of it is to be able to offer the same software (Linux and PostgreSQL) regardless of what kind of computer you're running on.

## **› I got some other error message**

If you're getting a specific textual error message, try looking it up on your favorite search engine. If that doesn't help, take a screenshot and post it to the discussion forums, along with as much detail as you can provide about the process you went through to get there.

## **› If all else fails, try an older version**

Udacity mentors have noticed that some newer versions of Vagrant don't work on all operating systems. Version 1.9.2 is reported to be stabler on some systems, and version 1.9.1 is the supported version on Ubuntu 17.04. You can download older versions of Vagrant from [the Vagrant releases index](#).

## **› Supporting Materials**

---

[Virtual machine repository on GitHub](#)

# Data and Tables

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Intro to Relational Databases](#)

## Table of Contents

---

- [Lesson 1. Relational Concepts: Data and Tables](#)
  - [1.01. Welcome to RDB](#)
  - [1.02. What's a database](#)
  - [1.03-05. Quizzes about tables and data types](#)
  - [1.06-08. Anatomy of a table with zoo animals](#)
  - [Aggregations](#)
  - [1.10-11. Queries](#)
  - [1.12-16. JOIN , id and PRIMARY KEY](#)
  - [1.17. Quiz: Database concepts](#)
  - [Feedback on Lesson 1](#)
- [Lesson 2. Elements of SQL](#)
  - [2.01. SQL is for Elephants](#)
  - [2.02. Talk to the Zoo Database](#)
  - [2.03. Types in the SQL World](#)
  - [2.04. Just a few SQL types](#)
  - [2.05. Select Where](#)
  - [2.06. Comparison Operators](#)
  - [2.07. The One Thing SQL is Terrible At](#)
  - [2.08. The Experiment Page](#)
  - [2.09. Quiz: Select Clauses](#)
  - [2.10. Why Do It in the Database](#)
  - [2.11. Quiz: Count All the Species](#)
  - [2.12. Quiz: Insert: Adding Rows](#)
  - [2.13. Joining tables: Find the Fish-Eaters](#)
  - [2.14. After Aggregating](#)
  - [2.15. More Join Practice](#)
  - [2.16. Wrap up](#)
  - [2.17. Installing the Virtual Machine](#)

- 2.18. Reference — Elements of SQL
- All the tables in the zoo database
- Time
- Feedback on Lesson 2
- Lesson 3. Python DB-API
  - 3.01. Welcome to your Database
  - 3.02. What's a DB-API
  - 3.03. Writing Code with DB API
  - 3.04. Quiz: Trying out DB API
  - 3.05. Quiz: Inserts in DB API
  - 3.06. Running the Forum
  - 3.07. Hello PostgreSQL
  - 3.08. Give That App a Backend
  - 3.09. Bobby Tables, Destroyer of Posts
  - 3.10. Curing Bobby Tables
  - 3.11. Spammy Tables
  - 3.12. Quiz: Stopping the Spam
  - 3.13. Quiz: Updating Away the Spam
  - 3.14. Quiz: Deleting the Spam
  - 3.15. Conclusion
  - 3.16. Reference — Python DB-API
  - Feedback on Lesson 3
- Lesson 4: Deeper into SQL
  - 4.01. Intro to Creating Tables
  - 4.02. Normalized Design Part One
  - 4.03. Normalized Design Part Two
  - 4.04. Quiz: What's Normalized
  - 4.05. Create Table and Types
  - 4.06. Quiz: Creating and Dropping
  - 4.07. Quiz: Declaring Primary Keys
  - 4.08. Declaring Relationships
  - 4.09. Quiz: Foreign Keys
  - 4.10. Self Joins Surprise Quiz
  - 4.11. Counting What Isn't There
  - 4.12. Subqueries
  - 4.13. Quiz: One Query Not Two
  - 4.14. Quiz: Views
  - 4.15. Reference — Deeper into SQL
  - 4.16. Outro
  - Feedback on Lesson 4

## Lesson 1. Relational Concepts: Data and Tables

## 1.01. Welcome to RDB

1. Relational concepts
2. SQL queries
3. Python DB-API
4. Advanced SQL

Instructor: Karl Krueger

## 1.02. What's a database

We can either structure application data in memory or in "durable storage." Databases are in durable storage. Items in memory are deleted when the application is closed, but items in durable storage persist.

## 1.03-05. Quizzes about tables and data types

Got them all on the first try.

## 1.06-08. Anatomy of a table with zoo animals

- Karl was buried in a pile of stuffed animals.
- We answered some questions about a database table.

## Aggregations

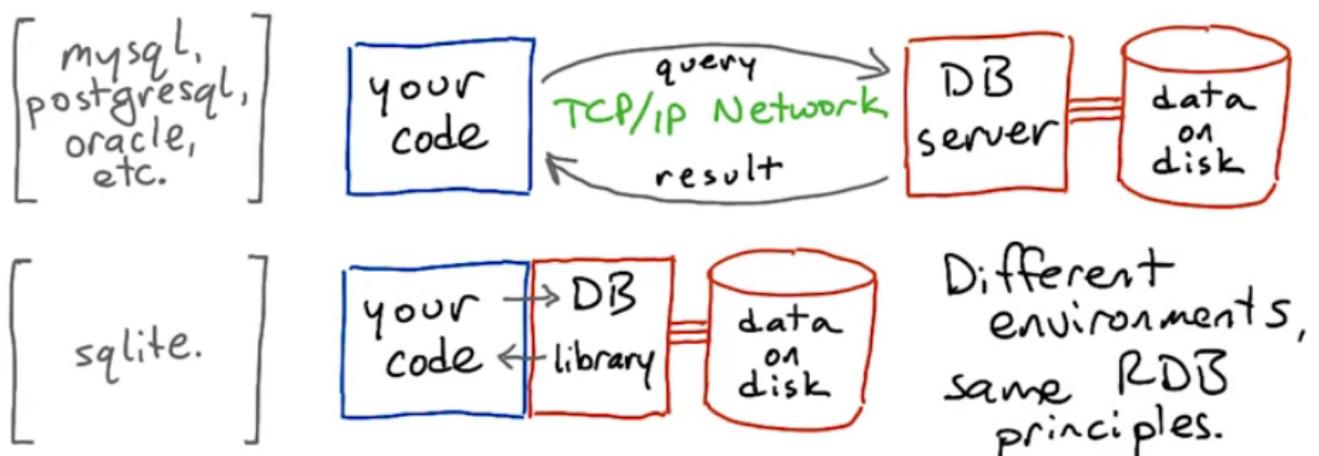
AVG  
COUNT  
MIN  
MAX  
SUM

## 1.10-11. Queries

Our first Udacity SQL query!

```
SELECT food FROM diet WHERE species = 'orangutan'
```

How queries happen:



0:28 / 1:56

CC YouTube

Karl used the example

```
$ psql zoo
zoo => SELECT 2+2 as SUM
```

Every result returned from a database query is in the form of a table.

### 1.12-16. JOIN , id and PRIMARY KEY

- Separate, but related tables can reference each other. In Karl's example, one table can list pictures of cats, and another table can track which cat pictures were shown to each person, and which they voted for.
- Tables can also be combined with `JOIN`.

# Related Tables

pictures:

id	name	filename
1	Fluffy	fluffsocute.jpg
2	Monster	monstie-basket.png
3	George	george.jpg
:	:	:

votes:

left	right	winner
2	3	3
1	3	1
2	1	1
3	2	3
3	1	1
2	3	3
:	:	:

joined:

left.name	right.name	winner.name
Monster	George	George
Fluffy	George	Fluffy
Monster	Fluffy	Fluffy
George	Monster	George
George	Fluffy	Fluffy
Monster	George	George

▶ ⏪ 2:25 / 2:26

cc 🔍 YouTube [ ]

- Keys are used to "unambiguously relate a row of one table to a row of another." A **PRIMARY KEY** is a column in a table that uniquely identifies rows.
- Quiz: student ID number and email address could be used as primary keys. Not everyone has a driver's license. "The birthday problem": only need 23 people in a room to have a 50% chance of two people sharing a birthday.
- *How many animals eat fish?*

```
-- This SQL query will join the two tables to find out what foods each animal can eat:  
select animals.name, animals.species, diet.food  
from animals join diet  
on animals.species = diet.species;
```

## 1.17. Quiz: Database concepts

- We created some database tables for zoo donors, and filled in the tables with the info given.
- I rocked it!

## Feedback on Lesson 1

Clear, helpful and fun! Thanks Karl and Udacity!

(Back to TOC)

## Lesson 2. Elements of SQL

### 2.01. SQL is for Elephants

## › 2.02. Talk to the Zoo Database

They embedded a SQL interpreter in the browser.

```
select name, birthdate from animals where species = 'gorilla';
```

When using `SELECT` to isolate columns, either type the column names, with a comma between each name, or just use `*` for wildcard to retrieve all columns.

## › 2.03. Types in the SQL World

Always put 'single quotes' around text strings and datetime values.

Also see cs50 Lecture 10 20171027 SQL and Databases 0:48:00

## › 2.04. Just a few SQL types

Here's just a sampling of the many data types that SQL supports. We won't be using most of these types in this course, though.

The exact list of types differs from one database to another. For a full list of types, check the manual for your database, such as [this one for PostgreSQL](#).

### › Text and string types

`text` — a string of any length, like Python `str` or `unicode` types. `char(n)` — a string of exactly `n` characters. `varchar(n)` — a string of up to `n` characters.

### › Numeric types

- `integer` — an integer value, like Python `int`.
- `real` — a floating-point value, like Python `float`. Accurate up to six decimal places.
- `double precision` — a higher-precision floating-point value. Accurate up to 15 decimal places.
- `decimal` — an exact decimal value.

### › Date and time types

- `date` — a calendar date; including year, month, and day.
- `time` — a time of day.
- `timestamp` — a date and time together.

## › 2.05. Select Where

```
-- find the names of all the animals that are not
-- gorillas and not named 'Max'.
--
-- my answer
select name from animals where species IS NOT 'gorilla' AND name IS NOT 'Max';
```

```
--Karl's solutions
-- first one
select name from animals where (not species = 'gorilla') AND (not name = 'Max');
-- DeMorgan's rule allows us to switch not x and not y into not x or y
select name from animals where not (species = 'gorilla' or name = 'Max');
-- SQL supports the "bang equals" or not equals operator !=
select name from animals where species != 'gorilla' and name != 'Max';
```

## 2.06. Comparison Operators

=, <, >, <=, >=, !=

```
-- Find all the llamas born between January 1, 1995 and December 31, 1998.
-- Fill in the 'where' clause in this query.
```

```
select * from animals where species = 'llama' and birthdate > '1995-01-01' and birthdate < '
```

I actually found this quiz kind of difficult, because I wasn't sure if I could perform a mathematical operation like `>` on a string, and because I needed a few tries to get the syntax correct.

## 2.07. The One Thing SQL is Terrible At

The one thing SQL is terrible at: listing your tables and columns in a standard way!

They showed an image of an "official IBM flowcharting stencil GX20-8020-1 (late 1970s)."

### Reference

[18. Reference — Elements of SQL](#)

## 2.08. The Experiment Page

They gave us some test syntax to query the zoo database.

```
-- Uncomment one of these queries and use "Test Run" to run it.
-- You'll see the results below. Then try your own queries as well!

-- select max(name) from animals;

-- select * from animals limit 10;

-- select * from animals where species = 'orangutan' order by birthdate;

-- select name from animals where species = 'orangutan' order by birthdate desc;

-- select name, birthdate from animals order by name limit 10 offset 20;

-- select species, min(birthdate) from animals group by species;
```

```
-- select name, count(*) as num from animals  
-- group by name  
-- order by num desc  
-- limit 5;
```

## 2.09. Quiz: Select Clauses

Here are the new select clauses introduced in the previous video (also see part 11 below for another summary):

```
... limit count
```

Return just the first (count) rows of the result table.

```
... limit count offset skip
```

Return count rows starting after the first skip rows.

```
... order by columns
```

```
... order by columns desc
```

Sort the rows using the columns (one or more, separated by commas) as the sort key. Numerical columns will be sorted in numerical order; string columns in alphabetical order. With desc, the order is reversed (desc-ending order). For example, `order by species, name` will sort the result rows first by the species column, then within each species, sort by the name column.

```
... group by columns
```

Change the behavior of aggregations such as max, count, and sum. With group by, the aggregation will return one row for each distinct value in columns.

For example,

```
select species, min(birthdate) from animals group by species;
```

for each species, find the smallest value of birthdate (the oldest animal's birthdate).

```
select name, count(*) as num from animals group by name;
```

count all the rows, call the count column "num," and aggregate by values of the "name" column.

I didn't complete the quiz, because there were too many options, and the solution video wasn't working.

## 2.10. Why Do It in the Database

Why query the database for `limit 100 offset 10` when you could just write `results[10:110]` in Python?

- Speed: "The database can generally do these things a lot faster than Python can. Especially when you get to tables with lots or rows, or complicated queries that join several tables. And you can easily have a table with millions of rows. Sorting a million items in Python can take around a second. If you're writing a web app, that's a second that your user is staring at their browser, wondering why your app is so freaking slow, and it's taking up memory to do that too. In contrast, a database can generally do these operations much faster. **There are a number of tricks you can use to make it faster still. The big one is called indexing.**"
- Space

## › 2.11. Quiz: Count All the Species

### › Select clauses

These are all the select clauses we've seen in the lesson so far.

#### › where

The where clause expresses restrictions — filtering a table for rows that follow a particular rule. where supports equalities, inequalities, and boolean operators (among other things):

- `where species = 'gorilla'` — return only rows that have 'gorilla' as the value of the species column.
- `where name >= 'George'` — return only rows where the name column is alphabetically after 'George'.
- `where species != 'gorilla' and name != 'George'` — return only rows where species isn't 'gorilla' and name isn't 'George'.

#### › limit / offset

The limit clause sets a limit on how many rows to return in the result table. The optional offset clause says how far to skip ahead into the results. So `limit 10 offset 100` will return 10 results starting with the 101st.

#### › order by

The `order by` clause tells the database how to sort the results — usually according to one or more columns. So `order by species, name` says to sort results first by the species column, then by name within each species.

Ordering happens before limit/offset, so you can use them together to extract pages of alphabetized results. (Think of the pages of a dictionary.)

The optional desc modifier tells the database to order results in descending order — for instance from large numbers to small ones, or from Z to A.

#### › group by

The group by clause is only used with aggregations, such as max or sum. Without a group by clause, a select statement with an aggregation will aggregate over the whole selected table(s), returning only one row. With a group by clause, it will return one row for each distinct value of the column or expression in the group by clause.

## › Quiz

```
-- Write a query that returns all the species in the zoo, and how many  
-- animals of each species there are, sorted with the most populous  
-- species at the top.  
--  
-- The result should have two columns: species and number.  
--  
-- The animals table has columns (name, species, birthdate) for each animal.
```

My first near-winner was

```
select species, count(*) as num from animals order by num;
```

but I was confused why it was only returning zebras. The quiz explained:

You've encountered a nonstandard behavior in SQLite, which is giving you weird results because your query has a count aggregation but doesn't use group by.

Make sure to use group by whenever you use an aggregation.

species	num
zebra	89

After running a few more tries in the interpreter, I got it!

```
select species, count(*) as num from animals group by species order by num desc;
```

AHA! I AM INVINCIBLE!

Boris Grishenko, Goldeneye

# Count ALL the species!

One possible answer:

```
select count(*) as num, species columns
  from animals tables
 group by species aggregation
sorting order by num desc;
```

we've put together the count  
aggregation, grouping and ordering.

▶ 🔍 0:42 / 0:49

CC 🛡 YouTube [ ]

## 2.12. Quiz: Insert: Adding Rows

The basic syntax for the `insert` statement:

```
insert into table ( column1, column2, ... ) values ( val1, val2, ... );
```

If the values are in the same order as the table's columns (starting with the first column), you don't have to specify the columns in the insert statement:

```
insert into table values ( val1, val2, ... );
```

For instance, if a table has three columns (a, b, c) and you want to insert into a and b, you can leave off the column names from the `insert` statement. But if you want to insert into b and c, or a and c, you have to specify the columns.

A single `insert` statement can only insert into a single table. (Contrast this with the `select` statement, which can pull data from several tables using a join.)

```
#  
# Insert a newborn baby opossum into the animals table and verify that it's  
# been added. To do this, fill in the rest of SELECT_QUERY and INSERT_QUERY.  
#  
# SELECT_QUERY should find the names and birthdates of all opossums.  
#  
# INSERT_QUERY should add a new opossum to the table, whose birthdate is today.  
# (Or you can choose any other date you like.)  
#  
# The animals table has columns (name, species, birthdate) for each individual.  
  
SELECT_QUERY = '''  
select name, birthdate from animals where species = 'opossum';
```

```
...  
INSERT_QUERY = ''  
insert into animals values ('Griswold', 'opossum', '2018-01-16');  
...  
'''
```

## 2.13. Joining tables: Find the Fish-Eaters

### 2.13 Quiz

IT'S JOIN TIME!

## Find the Fish-eaters

Find the names of all the animals that eat fish.

Joining tables:

```
select T.thing, S.stuff rows  
from T join S joined  
on T.target = S.match join condition
```

Simple join:

```
select T.thing, S.stuff  
from T, S tables  
where T.target = S.match restriction
```

table T equals the column match in  
table S, we can do it like this.

▶ 🔍 1:00 / 1:13

CC 🔍 YouTube [?]

```
-- Find the names of the individual animals that eat fish.  
-- Your query should only return the name column.  
-- The animals table has columns (name, species, birthdate) for each individual.  
-- The diet table has columns (species, food) for each food that a species eats.
```

I struggled with this exercise. After a few unsuccessful attempts, I went back to the first `join` presented in lesson 1.16:

```
-- This SQL query will join the two tables to find out what foods each animal can eat:  
select animals.name, animals.species, diet.food  
from animals join diet  
on animals.species = diet.species;
```

I was able to quickly narrow the query result down to the proper `name` column and join the tables, but couldn't figure out where to select just the fish diet:

```
select animals.name, diet.food = 'fish'  
  from animals join diet  
  on animals.species = diet.species;
```

Everything in lesson 2 made sense to me, until "2.13. Find the Fish-Eaters." I was able to quickly narrow the query result down to the proper `name` column and join the tables, but couldn't figure out where to select just the fish diet. The instructional video makes it look like you only need three lines of SQL code for the query, but you actually need a fourth line to specify the restriction. There should have been an extra step where he explained how to incorporate restrictions into joins.

## › Solution

After about an hour, I simply had to check the solution:

```
-- Find the names of the individual animals that eat fish.  
-- Your query should only return the name column.  
-- old syntax  
select animals.name from animals join diet  
  on animals.species = diet.species  
  where food = fish;
```

## Find the Fish-eaters

Find the names of all the animals that eat fish.

Select `animals.name` columns  
(just one!)  
from `animals join diet` joined  
tables  
on `animals.species = diet.species`  
where `food = 'fish'`; restriction join condition

```
-- Find the names of the individual animals that eat fish.  
-- Your query should only return the name column.  
-- simple syntax  
select name from animals, diet  
  where animals.species = diet.species  
  and diet.food = 'fish';
```

## Find the Fish-eaters

Find the names of all the animals that eat fish.

```
select name from animals, diet  
where animals.species = diet.species  
and diet.food = 'fish'; restriction
```

*columns*      *tables*  
*boolean!*      *and*      *restriction*

I think the point was that the columns you use in a join don't have to be the columns you ask for in a query result.

### 2.14. After Aggregating

#### where vs. having

- `where` is a restriction on the source tables.
- `having` is a restriction on the result *after* aggregation.

#### Quiz: Aggregating

## After Aggregating

Which species does the zoo have only one of?

RIGHT: select species, count(\*) as num  
from animals group by species  
having num = 1;

where is a restriction on the source tables.

having is a restriction on the result...after aggregation!  
using a more complicated sort  
of query called a subselect.

▶ 🔍 0:45 / 1:04

CC HD YouTube

```
-- Find the one food that is eaten by only one animal.  
--  
-- The animals table has columns (name, species, birthdate) for each  
-- individual.  
-- The diet table has columns (species, food) for each food that a  
-- species eats.
```

I got this one relatively quickly. I actually used the simple syntax, and the instructor used the complicated syntax:

```
-- Find the one food that is eaten by only one animal.  
-- Brendon's solution
```

```
select food, count(*) as num  
from diet group by food  
having num = 1;
```

Here's the solution I got:

```
-- Find the one food that is eaten by only one animal.  
-- Instructor's solution  
select food, count(*) as num  
from animals join diet  
on animals.species = diet.species  
group by food  
having num = 1;
```

Here, I'm still using a join condition that matches the two tables up using the species column, as we've seen earlier in the lesson. But now I'm adding a count with group by food, to find out how many animals eat each food. Then the having num = 1 clause extracts only those rows where the count is equal to 1.

## 2.15. More Join Practice

### Database reference

See 2.7 above for description of database.

### 2.15 Quiz

```
-- List all the taxonomic orders, using their common names, sorted by the
-- number of animals of that order that the zoo has.

--
-- The animals table has (name, species, birthdate) for each individual.
-- The taxonomy table has (name, species, genus, family, t_order) for each species.
-- The ordernames table has (t_order, name) for each order.

--
-- Be careful: Each of these tables has a column "name", but they don't
-- have the same meaning! animals.name is an animal's individual name.
-- taxonomy.name is a species' common name (like 'brown bear').
-- And ordernames.name is the common name of an order (like 'Carnivores').
```

This one was another struggle that took me hours.

### Review

To start, I reviewed the previous sections:

- The sample queries from 2.8:

```
select name, count(*) as num from animals
group by name
order by num desc
limit 5;
```

- The last section in which I had sorted stuff, "2.11. Quiz: Count All the Species":

```
select species, count(*) as num from animals group by species order by num desc;
```

- The difficult quiz in "2.13. Find the Fish-Eaters":

```
-- Find the names of the individual animals that eat fish.
-- Your query should only return the name column.
-- simple syntax
select name from animals, diet
```

```
where animals.species = diet.species  
and diet.food = 'fish';
```

## › Planning

I saw that I needed to do a join and then a sort, and started planning my approach.

- Join the three tables based on their commonalities
  - Join animals and taxonomy on species (not name, names are different between the two tables)
  - Join taxonomy and ordernames on t\_order (again, not on name, names different)
- Count and sort/order the rows
- *List all the taxonomic orders, using their common names:*
  - only display ordernames.name
- *sorted by the number of animals of that order that the zoo has*
  - This requires a join, and a sort.
- *Does it have to be all in one query, or can it be in two queries, like 2.12?*
  - I think it's one query, because 2.12 had one `SELECT` query and one `INSERT` query. This is just a `SELECT` query.

## › Approach

I broke the problem down and repeatedly iterated my code.

- First step: try a few queries to look at the tables.

```
select * from ordernames limit 10;
```

- Next: try some joins.

```
-- this seems to successfully join the three tables  
select * from animals, taxonomy, ordernames  
  where animals.species = taxonomy.species  
    and taxonomy.t_order = ordernames.t_order;
```

- Next: try some sorts.

```
-- sample sort of ordernames, without join  
select name, count(*) as num from ordernames group by name order by num desc;
```

```
-- sample sort of animals, without join  
-- close to what we want to see  
select species, count(*) as num from animals  
  group by species  
  order by num desc;
```

- Next: combine the join and sort. This was the difficult part. I was only able to get it to display two order names, "primates," and "lizards and snakes." I was frustrated by this point, and had spent hours on this one quiz.

```
-- This was as well as I could do on my own.
-- This only displays primates, and lizards and snakes.
select ordernames.name, count(*) as num from animals, taxonomy, ordernames
  where animals.species = taxonomy.species
    and taxonomy.t_order = ordernames.t_order
  group by ordernames.name
  order by num desc;
```

- Solutions: For my own sanity, I had to stop and check the instructor's solution after ~5 hours. I was only off from the "simple syntax" solution by one word: I had `taxonomy.species` instead of `taxonomy.name`. I want to understand why I didn't have it right, but the instructor notes don't explain it, and the solution video is complete fluff. Walk us through the solution, don't just talk about the animals!

```
-- Instructor's simple solution:
select ordernames.name, count(*) as num from animals, taxonomy, ordernames
  where animals.species = taxonomy.name
    and taxonomy.t_order = ordernames.t_order
  group by ordernames.name
  order by num desc;
```

```
-- And here's another, this time using the explicit join style:
select ordernames.name, count(*) as num
  from (animals
        join taxonomy on animals.species = taxonomy.name)
        as ani_tax
        join ordernames
        on ani_tax.t_order = ordernames.t_order
  group by ordernames.name
  order by num desc
```

I think the upper version is much more readable than the lower one, because in the explicit join style you have to explicitly tell the database what order to join the tables in — ((a join b) join c) — instead of just letting the database worry about that.

If you're using a more barebones database (like SQLite) there can be a performance benefit to the explicit join style. But in PostgreSQL, the more server-oriented database system we'll be using next lesson, the query planner should optimize away any difference.

## 2.16. Wrap up

## 2.17. Installing the Virtual Machine

[fullstack-nanodegree-vm-instructions.md](#)

## 2.18. Reference — Elements of SQL

This is a reference for the material covered in the "Elements of SQL" lesson.

### SQL Data Types

Here's just a sampling of the many data types that SQL supports. We won't be using most of these types in this course, though. The exact list of types differs from one database to another. For a full list of types, check the manual for your database, such as this one for PostgreSQL.

#### Text and string

**text** — a string of any length, like Python str or unicode types. **char(n)** — a string of exactly n characters. **varchar(n)** — a string of up to n characters.

#### Numeric

**integer** — an integer value, like Python int. **real** — a floating-point value, like Python float. Accurate up to six decimal places. **double precision** — a higher-precision floating-point value. Accurate up to 15 decimal places. **decimal** — an exact decimal value.

#### Date and time

**date** — a calendar date; including year, month, and day. **time** — a time of day. **timestamp** — a date and time together. **timestamp with time zone** — a timestamp that carries time zone information.

The type **timestamp with time zone** can be abbreviated to **timestamptz** in PostgreSQL.

### Select statement

The most basic form of the select statement is to select a single scalar value:

```
select 2 + 2 ;
```

More usefully, we can select one or more columns from a table. With no restrictions, this will return all rows in the table:

```
select name, species from animals ;
```

Columns are separated by commas. Use `*` to select all columns from the tables:

```
select * from animals;
```

Quite often, we don't want all the data from a table. We can restrict the rows using a variety of select clauses, listed below. There are also a wide variety of functions that can apply to columns; including aggregation functions that operate on values from several rows, such as **max** and **count**.

#### where clause

The **where** clause expresses restrictions — filtering a table for rows that follow a particular rule. `where` supports equalities, inequalities, and boolean operators (among other things):

```
where species = 'gorilla' — return only rows that have 'gorilla' as the value of the species column. where name >= 'George' — return only rows where the name column is alphabetically after 'George'. where species != 'gorilla' and name != 'George' — return only rows where species isn't 'gorilla' and name isn't 'George'.
```

#### › limit and offset clauses

The **limit** clause sets a limit on how many rows to return in the result table. The optional **offset** clause says how far to skip ahead into the results. So `limit 10 offset 100` will return 10 results starting with the 101st.

#### › order by clause

The **order by** clause tells the database how to sort the results — usually according to one or more columns. So `order by species, name` says to sort results first by the species column, then by name within each species.

Ordering happens before limit/offset, so you can use them together to extract pages of alphabetized results. (Think of the pages of a dictionary.)

The optional **desc** modifier tells the database to order results in descending order — for instance from large numbers to small ones, or from Z to A.

#### › group by clause

The **group by** clause is only used with aggregations, such as **max** or **sum**. Without a **group by** clause, a select statement with an aggregation will aggregate over the whole selected table(s), returning only one row. With a **group by** clause, it will return one row for each distinct value of the column or expression in the **group by** clause.

#### › having

The **having** clause works like the **where** clause, but it applies after **group by** aggregations take place. Here's an example:

```
select col1, sum(col2) as total
  from table
 group by col1
 having total > 500 ;
```

Usually, at least one of the columns will be an aggregate function such as **count**, **max**, or **sum** on one of the tables' columns. In order to apply **having** to an aggregated column, you'll want to give it a name using **as**.

For instance, if you had a table of items sold in a store, and you wanted to find all the items that have sold more than five units, you could use:

```
select name, count(*) as num from sales having num > 5;
```

You can have a select statement that uses only where, or only group by, or group by and having, or where and group by, or all three of them! But it doesn't usually make sense to use having without group by.

If you use both **where** and **having**, the **where** condition will filter the rows that are going *into* the aggregation, and the **having** condition will filter the rows that come *out of* it.

You can read more about **having** here: <http://www.postgresql.org/docs/9.4/static/sql-select.html#SQL-HAVING>

## › Insert statement

The basic syntax for the insert statement:

```
insert into tablename ( col1, col2, ... ) values ( val1, val2, ... );
```

If the values are in the same order as the table's columns (starting with the first column), you don't have to specify the columns in the insert statement:

```
insert into tablename values ( val1, val2, ... );
```

For instance, if a table has three columns (**a**, **b**, **c**) and you want to insert into **a** and **b**, you can leave off the column names from the insert statement. But if you want to insert into **b** and **c**, or **a** and **c**, you have to specify the columns.

Normally, a single insert statement can only insert into a single table. (Contrast this with the select statement, which can pull data from several tables using a join.)

## › All the tables in the zoo database

A database of zoo animals is used as an example in many of the code exercises in this course. Here's a list of all the tables available in it, and what the columns in each table refer to —

### › animals

This table lists individual animals in the zoo. Each animal has only one row. There may be multiple animals with the same name, or even multiple animals with the same name and species.

**name** — the animal's name (example: 'George') **species** — the animal's species (example: 'gorilla') **birthdate** — the animal's date of birth (example: '1998-05-18')

### › diet

This table matches up species with the foods they eat. Every species in the zoo eats at least one sort of food, and many eat more than one.

**species** — the name of a species (example: 'hyena') **food** — the name of a food that species eats (example: 'meat')

The **diet** table shows an example of the important database concept of *normalization*. If a species eats more than one food, there will be more than one row for that species. We do this instead of having multiple food columns (or storing a list in a single column), both of which would make select statements impractical.

## › taxonomy

This table gives the (partial) biological taxonomic names for each species in the zoo. It can be used to find which species are more closely related to each other evolutionarily.

**name** — the common name of the species (e.g. 'jackal') **species** — the taxonomic species name (e.g. 'aureus') **genus** — the taxonomic genus name (e.g. 'Canis') **family** — the taxonomic family name (e.g. 'Canidae') **t\_order** — the taxonomic order name (e.g. 'Carnivora')

If you've never heard of this classification, don't worry about it; the details won't be necessary for this course. But if you're curious, the Wikipedia article [Taxonomy \(biology\)](#) may help.

The **t\_order** column is not called **order** because "order" is a reserved keyword in SQL.

## › ordernames

This table gives the common names for each of the taxonomic orders in the taxonomy table.

**t\_order** — the taxonomic order name (e.g. 'Cetacea') **name** — the common name (e.g. 'whales and dolphins')

Karl's rainbow cat T-shirt is [The Time is Meow](#) by Rasabi, printed by Woot.

## › Time

Lesson 1: ~4 hours 20170115 Lesson 2: 15 hours (~10 hours on 20170116 for 2.1-2.14, ~5 hours on 20170117 for 2.15 More Join Practice)

## › Feedback on Lesson 2

### › Full version

I found *Lesson 01. Relational Concepts: Data and Tables* to be clear and intuitive, and I finished in a few hours.

I got stuck on two quizzes in *Lesson 02. Elements of SQL*:

1. 2.13. *Find the Fish Eaters*: I was able to quickly narrow the query result down to the proper `name` column and join the tables, but couldn't figure out where to select just the fish diet. The instructional video makes it look like you only need three lines of SQL code for the query, but you actually need a fourth line to specify the restriction. There should have been an extra step where he explained how to incorporate restrictions into joins.

2. 2.15. More Join Practice: I struggled with this problem for ~5 hours, and I put in the time because I really wanted to learn the syntax. I broke the problem down, repeatedly iterated my code, got as close as I could, and finally checked the solution. I was only off from the "simple syntax" solution by one word: I had `taxonomy.species` instead of `taxonomy.name`. I want to understand why I didn't have it right, but the instructor notes don't explain it, and the solution video is complete fluff. Walk us through the solution, don't just talk about the animals!

### › Concise version to fit into the 512 character limit of the feedback box

Sticking points:

1. 2.13. Find the Fish Eaters: The instructional video makes it look like you only need three lines of SQL code for the query, but you need a fourth line to specify the restriction. There should be an extra step explaining how to incorporate restrictions into joins.
2. 2.15. More Join Practice: I was only off from the "simple syntax" solution by one word: I had `taxonomy.species` instead of `taxonomy.name`. The instructor notes don't explain it, and the solution video is just fluff.

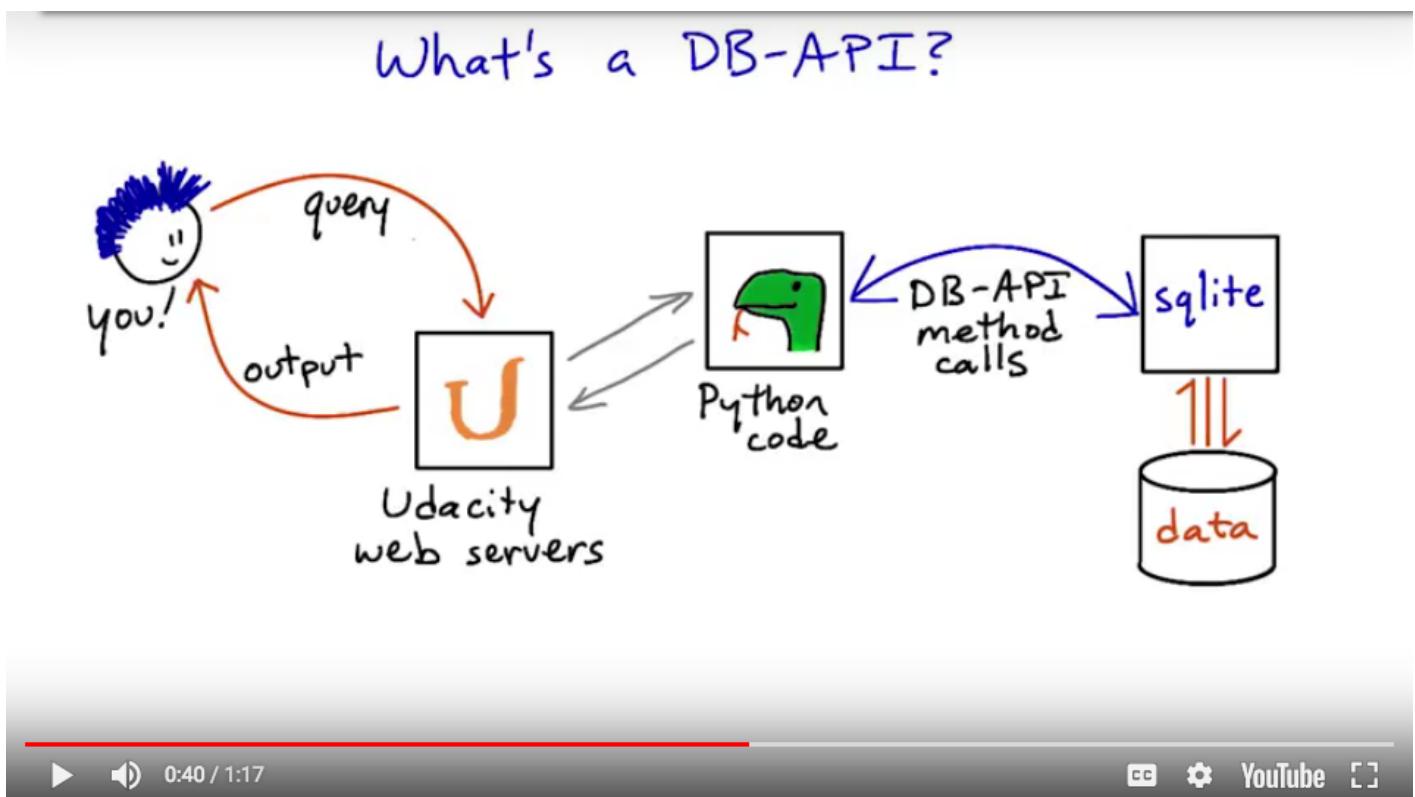
(Back to TOC)

## › Lesson 3. Python DB-API

### › 3.01. Welcome to your Database

We'll be using Python to connect to our database in a secure, reliable way.

### › 3.02. What's a DB-API



Each database system has its own library:

Database system	DB-API module
SQLite	sqlite3
PostgreSQL	psycopg2
ODBC	pyodbc
MySQL	mysql.connector

### 3.03. Writing Code with DB API

Here's the general process, with my notes based on the instructor Karl's video. This code can be run against a web browser to identify active cookies. Both Chrome and Firefox use SQLite databases to store cookies.

```
# Import the library for the type of database.
import sqlite3

# Specify the database, called Cookies here.
# The sqlite3.connect() function returns a connection object.
conn = sqlite3.connect("Cookies")

# Next, make a cursor, which runs queries and fetches results.
# The cursor is used to scan through results, like a text cursor in an editor.
cursor = conn.cursor

# Execute a query using the cursor.
cursor.execute("select host_key from cookies limit 10")

# Fetch all results from the query using the cursor.
# The results can be fetched one at a time by using fetchone() instead.
results = cursor.fetchall()

print(results)

# Be sure to close connections when done to avoid stale connections.
conn.close()
```

### 3.04. Quiz: Trying out DB API

Got it on my first try!

```
# To see how the various functions in the DB-API work, take a look at this code,
# then the results that it prints when you press "Test Run".
#
# Then modify this code so that the student records are fetched in sorted order
# by student's name.
#
# import sqlite3
```

```

# Fetch some student records from the database.
db = sqlite3.connect("students")
c = db.cursor()
query = "select name, id from students order by name;"
c.execute(query)
rows = c.fetchall()

# First, what data structure did we get?
print "Row data:"
print rows

# And let's loop over it too:
print
print "Student names:"
for row in rows:
    print " ", row[0]

db.close()

```

## 3.05. Quiz: Inserts in DB API

Must add `pg.commit()` to commit changes to database. Changes such as inserts are tracked as transactions in the database (see the `.schema` explanation in cs50 Lecture 10 20171027 SQL and Databases). Changes are made with **atomicity**, meaning they are made entirely, like a single indivisible atom, or not at all.

```

# This code attempts to insert a new row into the database, but doesn't
# commit the insertion. Add a commit call in the right place to make
# it work properly.
#
#  

import sqlite3  

db = sqlite3.connect("testdb")
c = db.cursor()
c.execute("insert into balloons values ('blue', 'water')")
db.commit()
db.close()

```

## 3.06. Running the Forum

```

vagrant@vagrant:~$ cd /vagrant/forum
vagrant@vagrant:/vagrant/forum$ ls
forumdb.py  forum.py  forum.sql  solution
vagrant@vagrant:/vagrant/forum$ python forum.py
- Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)

```

It just uses python code to run the forum, it doesn't have a database yet.

## 3.07. Hello PostgreSQL

### Reference info

See [16. Reference — Python DB-API](#)

Here's a fun one to run in `psql` while your forum web app is running:

```
select * from posts \watch
```

(Note that `\watch` replaces the semicolon.) This will display the contents of the `posts` table and refresh it every two seconds, so you can see changes to the table as you use the app.

In order to do this, you'll need two terminal sessions into your VM — one running the forum app, and the other running `psql`. You can connect to the VM from any number of terminal windows at once — just open up another terminal, change to the `vagrant` directory, and type `vagrant ssh` again.

## 3.08. Give That App a Backend

```
vagrant@vagrant:/vagrant/forum$ psql forum
psql (9.5.10)
Type "help" for help.

forum=> select 2+2 as a, 4+4 as b;
   a | b
   ---+---
     4 | 8
(1 row)

forum=> select * from posts;
 content | time | id
-----+-----+-----
(0 rows)
```

## 3.09. Bobby Tables, Destroyer of Posts

We tried entering some different types of posts. The instructor Karl said posts with one apostrophe, like

That's cool return an internal server error. I was not able to get the error. Karl acknowledges that not everyone will get the bug.

We tried an SQL Injection attack (SQLi) by entering `''); delete from posts; --` as a forum post.

## 3.10. Curing Bobby Tables

We had to revise the `forumdb.py` code here to prevent SQLi, loosely following the paradigm from 3.3-3.5.

As the [psycopg docs](#) warn:

Warning: Never, never, NEVER use Python string concatenation (+) or string parameters interpolation (%) to pass variables to a SQL query string. Not even at gunpoint.

There is also a [Bobby Tables website](#) with instructions on how to prevent SQLi in various programming languages.

It was preferable to follow along in the solution files here. Karl jumped from `sqlite3` to `psycopg2` and made several other changes to the code. I didn't feel like I had enough information to arrive at the solution independently. I found the code in a separate *solution*/ directory and just followed along from there.

## 3.11. Spammy Tables

Karl showed us how to inject some JavaScript spam by submitting this as a forum post:

```
<script>
setTimeout(function() {
  var tt = document.getElementById('content');
  tt.value = "<h2 style='color: #FF6699; font-family: Comic Sans MS'>Spam, spam, spam, spa
  tt.form.submit();
}, 2500);
</script>
```

This is called a **script injection attack**.

## 3.12. Quiz: Stopping the Spam

`bleach` is a python library for stopping spam. Based on the bleach docs, I added `bleach.clean(content)` to the forum code.

We considered both **input sanitization** and **output sanitization**.

*How should we approach the spam?*

Clearing out bad data before it ever gets into the database — **input sanitization** — is one effective approach to preventing attacks like this one.

But we'll still have to clean out the bad data that's already in the database.

## 3.13. Quiz: Updating Away the Spam

Karl presented updating the forum posts as one option, but it's not effective. You just replace the spam posts with something else, but all the posts are still there. The posts really need to be deleted.

*Instructor notes:*

The syntax of the **update** statement:

```
update table set column = value where restriction ;
```

The **restriction** works the same as in **select** and supports the same set of operators on column values.

The **like** operator supports a simple form of text pattern-matching. Whatever is on the left side of the operator (usually the name of a text column) will be matched against the pattern on the right. The pattern is an SQL text string (so it's in 'single quotes') and can use the % sign to match any sub-string, including the empty string.

If you are familiar with regular expressions, think of the % in **like** patterns as being like the regex .\* (dot star).

If you are more familiar with filename patterns in the Unix shell or Windows command prompt, % here is a lot like \*\*\*\*\* (star) in those systems.

For instance, for a table row where the column **fish** has the value '**salmon**', all of these restrictions would be true:

- fish like 'salmon'
- fish like 'salmon%'
- fish like 'sal%'
- fish like '%n'
- fish like 's%n'
- fish like '%al%'
- fish like '%'
- fish like '%%%'

And all of these would be false:

- fish like 'carp'
- fish like 'salmonella'
- fish like '%b%'
- fish like 'b%'
- fish like ''

### 3.14. Quiz: Deleting the Spam

The **DELETE** statement functions like **SELECT**, **UPDATE**, and the other SQL commands.

```
DELETE FROM table WHERE restriction;
```

If we want to delete the spam from the forum, we would run

```
DELETE * FROM posts WHERE content = 'cheese';
```

### 3.15. Conclusion

## 3.16. Reference — Python DB-API

This is a reference for the material covered in the "Python DB-API" lesson.

### Python DB-API Quick Reference

For a full reference to the Python DB-API, see [the specification](#) and the documentation for specific database modules, such as `sqlite3` and `psycopg2`.

**module.connect(...)** Connect to a database. The arguments to `connect` differ from module to module; see the documentation for details. `connect` returns a `Connection` object or raises an exception.

For the methods below, note that you *don't* literally call (for instance) `Connection.cursor()` in your code. You make a `Connection` object, save it in a variable (maybe called `db`) and then call `db.cursor()`.

**Connection.cursor()** Makes a `Cursor` object from a connection. Cursors are used to send SQL statements to the database and fetch results.

**Connection.commit()** Commits changes made in the current connection. You must call `commit` before closing the connection if you want changes (such as inserts, updates, or deletes) to be saved. Uncommitted changes will be visible from your current connection, but not from others.

**Connection.rollback()** Rolls back (undoes) changes made in the current connection. You must roll back if you get an exception if you want to continue using the same connection.

**Connection.close()** Closes the connection. Connections are always implicitly closed when your program exits, but it's a good idea to close them manually especially if your code might run in a loop.

**Cursor.execute(statement)** **Cursor.execute(statement, tuple)** Execute an SQL statement on the database. If you want to substitute variables into the SQL statement, use the second form — see [the documentation](#) for details.

If your statement doesn't make sense (like if it asks for a column that isn't there), or if it asks the database to do something it can't do (like delete a row of a table that is referenced by other tables' rows) you will get an exception.

**Cursor.fetchall()** Fetch all the results from the current statement.

**Cursor.fetchone()** Fetch just one result. Returns a tuple, or `None` if there are no results.

### psql Quick Reference

The `psql` command-line tool is really powerful. There's a complete reference to it [in the PostgreSQL documentation](#).

To connect `psql` to a database running on the same machine (such as your VM), all you need to give it is the database name. For instance, the command `psql forum` will connect to the `forum` database.

From within `psql`, you can run any SQL statement using the tables in the connected database. Make sure to end SQL statements with a semicolon, which is not always required from Python.

You can also use a number of special `psql` commands to get information about the database and make configuration changes. The `\d posts` command shown in the video is one example — this displays the columns of the `posts` table.

Some other things you can do:

`\dt` — list all the tables in the database.

`\dt+` — list tables plus additional information (notably, how big each table is on disk).

`\H` — switch between printing tables in plain text vs. HTML.

#### › br3ndonland addition: exiting command prompt

Type `\q` and then press ENTER to quit `psql`.

#### › Bleach documentation

Read the documentation for Bleach here: <http://bleach.readthedocs.org/en/latest/>

#### › Update and delete statements

The syntax of the `update` and `delete` statements:

`update table set column = value where restriction ; delete from table where restriction ;`

The `where` restriction in both statements works the same as in `select` and supports the same set of operators on column values. In both cases, if you leave off the `whererestriction`, the update or deletion will apply to *all rows* in the table, which is usually not what you want.

#### › like operator

The `like` operator supports a simple form of text pattern-matching. Whatever is on the left side of the operator (usually the name of a text column) will be matched against the pattern on the right. The pattern is an SQL text string (so it's in 'single quotes') and can use the % sign to match any sub-string, including the empty string.

If you are familiar with regular expressions, think of the % in `like` patterns as being like the regex `.*` (dot star).

If you are more familiar with filename patterns in the Unix shell or Windows command prompt, % here is a lot like `*****` (star) in those systems.

For instance, for a table row where the column `fish` has the value '`salmon`', all of these restrictions would be true:

- `fish like 'salmon'`
- `fish like 'salmon%'`
- `fish like 'sal%'`

- fish like '%n'
- fish like 's%n'
- fish like '%al%
- fish like '%'
- fish like '%%%

And all of these would be false:

- fish like 'carp'
- fish like 'salmonella'
- fish like '%b%
- fish like 'b%
- fish like ''

## › Spam

The term "spam" referring to junk posts comes from [Monty Python's "Spam" sketch](#). On the Internet, "spamming" was first used to mean [repetitious junk messages](#) intended to disrupt a group chat. Later, it came to refer to unsolicited ads on forums or email; and more recently to more-or-less any repetitious or uninvited junk message.

## › Feedback on Lesson 3

I thought the material in this lesson was important and helpful, but I didn't really arrive at the solution code independently. I found it more practical to follow along in the solution files starting at "10. Curing Bobby Tables", where Karl switched from `sqlite3` to `psycopg2` and made several other changes to the code.

[\(Back to TOC\)](#)

## › Lesson 4: Deeper into SQL

---

### › 4.01. Intro to Creating Tables

### › 4.02. Normalized Design Part One

In a **normalized database**, the relationships among the tables match the relationships that are really among the data.

Check out William Kent's paper "[A Simple Guide to Five Normal Forms in Relational Database Theory](#)" for a lot more about normalization and how it can help your database design.

### › 4.03. Normalized Design Part Two

1. Every row has the same number of columns.
2. There is a unique **key**, and everything in a row says something about the key.
3. Facts that don't relate to the key belong in different tables.

4. Tables shouldn't imply relationships that don't exist.

## › 4.04. Quiz: What's Normalized

## › 4.05. Create Table and Types

User-facing code doesn't *usually* create new tables.

Normally when creating an application, the database is created up front.

## › 4.06. Quiz: Creating and Dropping

Since not everything fits on the screen at once, here's what to try in `psql`:

- Create a new database called `fishies` (or whatever you like).
- Connect to it with `\c fishies`, or by exiting `psql` and running `psql fishies`.
- In the new database, create a table that has two columns: a `text` column and a `serial` column.
- Try running `insert` statements into this table, providing only a value for the `text` column.  
(For an example, scroll down to the bottom of this page.)

Look up these commands in the PostgreSQL documentation:

- [Create Database](#)
- [Drop Database](#)
- [Create Table](#)
- [Drop Table](#)

Here's an example `insert` statement you might try. Replace `*sometable*` with the name of the table you created:

```
insert into sometable values ('This is text!');
```

For more detail on the `serial` type, take a look at the last section of this page in the PostgreSQL manual: <http://www.postgresql.org/docs/9.4/static/datatype-numeric.html>

```
vagrant@vagrant:/vagrant$ psql
psql (9.5.10)
Type "help" for help.

vagrant=> CREATE DATABASE sardines;
CREATE DATABASE
vagrant=> \c sardines
You are now connected to database "sardines" as user "vagrant".
sardines=> CREATE TABLE FISH (
sardines(> text text,
sardines(> serial varchar );
CREATE TABLE
sardines=> INSERT INTO FISH values ('This is text!');
```

```
INSERT 0 1
sardines=>
```

A serial is a sequence, an internal data structure.

## 4.07. Quiz: Declaring Primary Keys

*Single column primary keys:* enter "primary key" when creating the column, before the comma:

```
CREATE TABLE students (
    id serial primary key,
    name text,
    birthdate date
);
```

*Multi-column primary keys:* enter *after* creating the columns in the table, and specify the columns to use as key:

```
CREATE TABLE postal_places (
    postal_code text,
    country text,
    name text,
    primary key (postal_code, country)
);
```

Quiz: if we create a second entry with the same primary key as a prior entry, SQL will return an error.

If we set up a constraint (such as a primary key) we're asking for the database to help ensure that our data makes sense.

By signaling an error, the database refuses to accept data that break the "rule" that we've created by adding a primary key constraint.

## 4.08. Declaring Relationships

As an example, Karl used a list of products, each with a corresponding SKU (Stock Keeping Unit).

**Reference constraints** can be used to tell the SQL database that a column should only have values that refer to the key of another table. They are created using `references`, followed by the table name (`products` in this case):

```
create table sales (
    sku text references products,
    sale_date date,
    count integer
);
```

If the column name is not the same in both tables, specify the name in parentheses, like `products (sku)`.

References ensure **referential integrity**.

## › 4.09. Quiz: Foreign Keys

Columns with reference constraints are also called **foreign keys**. Foreign keys point to a primary key in the reference table. The values of a foreign key must match the values of the primary key.

## › 4.10. Self Joins Surprise Quiz

Why are we taking a quiz on self joins when we haven't learned about self joins? I can look it up in external documentation or check the solution, but then why have a quiz?

### › Self joins quiz

Karl shows a list of dormitories from his college. Based on a search of the list, it looks like [Bard College at Simon's Rock](#), for students who graduate early from high school.

```
-- Roommate Finder v0.9
--
-- This query is intended to find pairs of roommates. It almost works!
-- There's something not quite right about it, though. Find and fix the bug.

select a.id, b.id, a.building, a.room
      from residences as a, residences as b
     where a.building = b.building
       and a.room = b.room
    order by a.building, a.room;

-- To see the complete residences table, comment out the query above,
-- uncomment this query and press "Test Run":

-- select id, building, room from residences;
```

### › Self joins quiz troubleshooting

- We are displaying the correct columns, so the `SELECT` statement is fine. Replacing the `SELECT` statement with `SELECT *` shows too many columns.
- We are displaying too many rows, which means there is a problem with the aggregation.
- I next tried modifying the `WHERE` statement.
  - Removing `and a.room = b.room` doesn't help.
  - Adding `AND NOT a.id = b.id`, or `AND a.id <> b.id` (syntax I found on [w3schools](#)) is a step in the right direction. It removes the rows with two of the same ID, but there are still two rows for each pair of roommates.
- **We need a multi-column primary key.**
  - So far, we've only set keys when creating tables (see 7. Quiz: Declaring Primary Keys).
  - It would look like `primary key (building, room)`.

## › Self joins quiz solution

### ► Solution

## › 4.11. Counting What Isn't There

### › Instructor notes

They introduced left and right joins here. Left and right refer to the two tables being joined.

Counting rows in a single table is something you've seen many times before in this course. A column aggregated with the `count` aggregation function will return the number of rows in the table, or the number of rows for each value of a `group by` clause.

For instance, you saw queries like these back in Lesson 2:

- `select count(*) from animals;` returns the number of animals in the zoo
- `select count(*) from animals where species = 'gorilla';` returns the number of gorillas
- `select species, count(*) from animals group by species;` returns each species' name and the number of animals of that species

Things get a little more complicated if you want to count the results of a `join`. Consider the tables we saw earlier in Lesson 4, the `products` and `sales` tables for a store.

Suppose that we want to know how many times we have sold each product. In other words, for each `sku` value in the `products` table, we want to know the number of times it occurs in the `sales` table. We might start out with a query like this:

```
select products.name, products.sku, count(*) as num
  from products join sales
    on products.sku = sales.sku
   group by products.sku;
```

But this query might not do exactly what we want. If a particular `sku` has never been sold — if there are no entries for it in the `sales` table — then this query will not return a row for it at all.

If we wanted to see a row with the number zero in it, we'll be disappointed!

However, there is a way to get the database to give us a count with a zero in it. To do this, we'll need to change two things about this query —

```
select products.name, products.sku, count(sales.sku) as num
  from products left join sales
    on products.sku = sales.sku
   group by products.sku;
```

This query will give us a row for every product in the `products` table, even the ones that have no sales in the `sales` table.

What's changed? First, we're using `count(sales.sku)` instead of `count(*)`. This means that the database will count only rows where `sales.sku` is defined, instead of all rows.

Second, we're using a **left join** instead of a plain **join**.

*Um, so what's a left join?*

SQL supports a number of variations on the theme of joins. The kind of join that you have seen earlier in this course is called an *inner join*, and it is the most common kind of join — so common that SQL doesn't actually make us say "inner join" to do one.

But the second most common is the **left join**, and its mirror-image partner, the **right join**. The words "left" and "right" refer to the tables to the left and right of the join operator. (Above, the left table is `products` and the right table is `sales`.)

A regular (inner) join returns only those rows where the two tables have entries matching the join condition. A **left join** returns all those rows, plus the rows where the *left* table has an entry but the right table doesn't. And a **right join** does the same but for the *right* table.

(Just as "join" is short for "inner join", so too is "left join" actually short for "left outer join". But SQL lets us just say "left join", which is a lot less typing. So we'll do that.)

## › Joins quiz

```
-- Here are two tables describing bugs found in some programs.  
-- The "programs" table gives the name of each program and the files  
-- that it's made of. The "bugs" table gives the file in which each  
-- bug was found.  
  
--  
-- create table programs (  
--     name text,  
--     filename text  
-- );  
-- create table bugs (  
--     filename text,  
--     description text,  
--     id serial primary key  
-- );  
  
--  
-- The query below is intended to count the number of bugs in each  
-- program. But it doesn't return a row for any program that has zero  
-- bugs. Try running it as it is. Then change it so that the results  
-- will also include rows for the programs with no bugs. These rows  
-- should have a 0 in the "bugs" column.
```

```
select programs.name, count(*) as num  
  from programs join bugs  
    on programs.filename = bugs.filename  
 group by programs.name  
 order by num;
```

Got it on my first try!

```

select programs.name, count(bugs.id) as num
  from programs left join bugs
    on programs.filename = bugs.filename
   group by programs.name
  order by num;

```

## › Quiz instructor notes

Something to watch out for: What do you put in the count aggregation? If you leave it as count(\*) or use a column from the programs table, your query will count entries that don't have bugs as well as ones that do.

In order to correctly report a zero for programs that don't have any entries in the bugs table, you have to use a column from the bugs table as the argument to count.

For instance, count(bugs.filename) will work, and so will count(bugs.description).

## › 4.12. Subqueries

Queries within queries. How deep does the rabbit hole go?

```

select avg(bigscore) from
  (select max(score)
    as bigscore
   from mooseball
  group by team)
  as maxes;

```

## Subqueries

mooseball:

player	team	score
Martha Moose	Ice Weasels	17
Bullwinkle	Frostbiters	23
Joe Moosington	Ice Weasels	11
Mighty Moose	Frostbiters	41
Mickey Moose	Traffic Stoppers	36
La Moosarina	Traffic Stoppers	12

Average high-scorer's score:  
 select avg(bigscore) from  
 (select max(score)  
 as bigscore  
 from mooseball  
 group by team)  
 as maxes; gotta name it

Nested like a Russian doll.

Here are some sections in the PostgreSQL documentation that discuss other forms of subqueries:

- [Scalar Subqueries](#)
- [Subquery Expressions](#)
- [The FROM clause](#)

## › 4.13. Quiz: One Query Not Two

```
# Find the players whose weight is less than the average.  
#  
# The function below performs two database queries in order to find the right players.  
# Refactor this code so that it performs only one query.  
  
def lightweights(cursor):  
    """Returns a list of the players in the db whose weight is less than the average."""  
    cursor.execute("select avg(weight) as av from players;")  
    av = cursor.fetchall()[0][0] # first column of first (and only) row  
    cursor.execute("select name, weight from players where weight < " + str(av))  
    return cursor.fetchall()
```

I didn't have much patience for this quiz. I tried out a few things. It seemed like I had the two queries joined properly, but I wasn't sure how to get it to run with the Python code. Again, the solution video was not very helpful, because it only showed the SQL query, and not the surrounding Python code that I was having trouble with.

```
SELECT name, weight FROM players,  
(SELECT avg(weight) as av  
     FROM players as subquery)  
WHERE weight < av;
```

I plugged it in to get the full Python solution:

```
def lightweights(cursor):  
    """Returns a list of the players in the db whose weight is less than the average."""  
    cursor.execute(  
        "SELECT name, weight FROM players, (SELECT avg(weight) as av FROM players as subquer  
    return cursor.fetchall()
```

## › 4.14. Quiz: Views

### › Intro

When code gets too complex, developers look for ways to *refactor* it into smaller functions.

Views are `SELECT` queries stored in the database for use like tables. Rows can be updated or deleted from some views but not others.

## › Views quiz

This quiz was much easier because it only required a small amount of additional code.

```
-- Here's a select statement that runs on the zoo database.  
-- It selects the species with the top five highest populations in the zoo.  
-- Change it into a statement that creates a view named "topfive".
```

```
CREATE view topfive as  
select species, count(*) as num  
from animals  
group by species  
order by num desc  
limit 5;
```

```
-- Don't change the statement below! It's there to test the view.
```

```
select * from topfive;
```

## › 4.15. Reference — Deeper into SQL

This is a reference for the material covered in the "Deeper into SQL" lesson.

### › The `create table` statement

The full syntax of the `create table` statement is quite complex. See the [PostgreSQL create table documentation](#) for the whole thing. Here's the syntax for the form we're seeing in this lesson:

```
create table table ( column type [restriction] , ... ) [rowrestriction] ;
```

There are a lot of restrictions that can be put on a column or a row. `primary key` and `references` are just two of them. See the "Examples" section of the [create table documentation](#) for many, many more.

### › Rules for normalized tables

In a normalized database, the relationships among the tables match the relationships that are really there among the data. Examples [here](#) refer to tables in Lessons 2 and 4.

1. Every row has the same number of columns.

- In practice, the database system won't let us *literally* have different numbers of columns in different rows. But if we have columns that are sometimes empty (null) and sometimes not, or if we stuff multiple values into a single field, we're bending this rule.
- The example to keep in mind here is the `diet` table from the zoo database. Instead of trying to stuff multiple foods for a species into a single row about that species, we

separate them out. This makes it much easier to do aggregations and comparisons.

2. There is a unique key and everything in a row says something about the key.
  - The key may be one column or more than one. It may even be the whole row, as in the `diet` table. But we don't have duplicate rows in a table.
  - More importantly, if we are storing non-unique facts — such as people's names — we distinguish them using a unique identifier such as a serial number. This makes sure that we don't combine two people's grades or parking tickets just because they have the same name.
3. Facts that don't relate to the key belong in different tables.
  - The example here was the `items` table, which had items, their locations, and the location's street addresses in it. The address isn't a fact about the item; it's a fact about the location. Moving it to a separate table saves space and reduces ambiguity, and we can always reconstitute the original table using a `join`.
4. Tables shouldn't imply relationships that don't exist.
  - The example here was the `job_skills` table, where a single row listed one of a person's technology skills (like 'Linux') and one of their language skills (like 'French'). This made it look like their Linux knowledge was specific to French, or vice versa ... when that isn't the case in the real world. Normalizing this involved splitting the tech skills and job skills into separate tables.

### › The serial type

For more detail on the `serial` type, take a look at the last section of [this page in the PostgreSQL manual](#).

### › Other subqueries

Here are some sections in the PostgreSQL documentation that discuss other forms of subqueries besides the ones discussed in this lesson:

- [Scalar Subqueries](#)
- [Subquery Expressions](#)
- [The FROM clause](#)

Mooseball is not a real sport (yet), but you can get a roughly [ball-shaped moose](#) from Squishables.

## › 4.16. Outro

### › Feedback on Lesson 4

**4.10. Self Joins:** I wasn't sure why we had a quiz on self joins before learning about that topic. I can look it up in external documentation or check the solution, but then why have a quiz? Also, introducing self joins right after foreign keys made me think the self joins quiz involved a multi-column foreign key, leading me away from the solution.

**4.14. Quiz: Views** was an example of an easier quiz. We only had to add a small amount of code, and it followed very well from Karl's introduction.

# APIs

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Designing RESTful APIs](#)

## Table of Contents

---

- [Table of Contents](#)
- [Lesson 1. Whats and whys of APIs](#)
  - [1.01. Course Intro](#)
  - [1.02. Prerequisites](#)
  - [1.03. What are APIs](#)
  - [1.04. Web APIs](#)
  - [1.05. Web API Protocols](#)
  - [1.06. Digging into The Application Layer](#)
  - [1.07. The Web Service Layer](#)
  - [1.08. Content Formatting Layer](#)
  - [1.09. Choosing the Right Technologies Soap vs Rest](#)
  - [1.10. XML vs JSON](#)
  - [1.11. Quiz: Developer Discussions](#)
  - [1.12. Quiz: Developer Discussions](#)
  - [1.13. Quiz: Developer Discussions](#)
  - [1.14. REST Constraints](#)
  - [1.15. Quiz: Why Stateless](#)
  - [1.16. Lesson 1 Wrap Up](#)
- [Lesson 2. Accessing published APIs](#)
  - [2.01. Lesson 2 Intro](#)
  - [2.02. Parts of an HTTP Request](#)
  - [2.03. HTTP Response](#)
  - [2.04. Sending API Requests with Postman and Curl](#)
  - [2.05. Quiz: Sending API Requests](#)
  - [2.06. Searching for API Documentation](#)
  - [2.07. Quiz: Delving Into APIs](#)
  - [2.08. Quiz: Using the Foursquare API](#)
  - [2.09. Requesting From Python Code](#)

- 2.10. Parsing Your Response
- 2.11. Quiz: Make Your Own API Mashup
- 2.12. Lesson 2 Wrap Up
- Feedback on lesson 2
- Lesson 3. Creating your own API endpoints
  - 3.01. Lesson 3 Intro
  - 3.02. API Endpoints with Flask
  - 3.03. Quiz: Making an Endpoint with Flask Quiz
  - 3.04. Quiz: Respond to Different Kinds of Requests
  - 3.05. Quiz: Serializing Data from the Database
  - 3.06. Quiz: Adding features to your Mashup
  - 3.07. Lesson 3 Wrap Up
- Lesson 4. Securing your API
  - 4.01. Lesson 4 Intro
  - 4.02. Adding Users and Logins
  - 4.03. User Registration
  - 4.04. Password Protecting a Resource
  - 4.05. Quiz: Mom & Pop's Bagel Shop
  - 4.06. Token Based Authentication
  - 4.07. Implementing Token Based Authentication in Flask
  - 4.08. Quiz: Regal Tree Foods
  - 4.09. OAuth 2.0
  - 4.10. Adding OAuth 2.0 for Authentication
  - 4.11. Quiz: Pale Kale Salads & Smoothies
  - 4.12. Rate Limiting
  - 4.13. Rate Limiting Exercise
  - 4.14. Quiz: Bargain Mart
  - 4.15. Lesson 4 Wrap Up
- Lesson 5. Writing developer-friendly APIs
  - 5.01. Lesson 5 Intro
  - 5.02. Developer Friendly Documentation
  - 5.03. Using Proper URIs
  - 5.04. Versioning Your API
  - 5.05. Communicating With Developers
  - 5.06. Learning From The Best
  - 5.07. Course Wrap Up
- General API course feedback

## Lesson 1. Whats and whys of APIs

---

### 1.01. Course Intro

## 1.02. Prerequisites

### Vagrant virtual machine

This course will use the Vagrant virtual machine again, which I had previously configured for the SQL relational databases work (see [notes](#) and [database logs analysis project](#)).

- \*Oracle [VirtualBox](#) Version 5.2.6 r120293 (Qt5.6.3)- - Software that runs special containers called virtual machines, like Vagrant.
- \*[Vagrant](#) 2.0.1 with Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-75-generic i686)- - Software that provides the Linux operating system in a defined configuration, allowing it to run identically across many personal computers. Linux can then be run as a virtual machine with VirtualBox.
- \*[Udacity Virtual Machine configuration](#)- - Repository from Udacity that configures Vagrant.
  - Instructions are provided [here](#).
    - I installed and ran Vagrant from within the directory *fullstack-nanodegree-vm/vagrant*.
    - I forked and cloned the lesson materials into */vagrant/apis*.

### Instructor notes

Want to brush up on your Python, Flask, and SQLAlchemy skills?

Go ahead and check out [Full Stack Foundations](#) to make sure you have an understanding of the python topics used in this course.

Documentation for [Flask](#) and [SQLAlchemy](#) can be found here as well.

The [Authentication and Authorization](#) will come in handy when we cover API Security.

Checkout these instructions for configuring the [Vagrant machine](#) for this course.

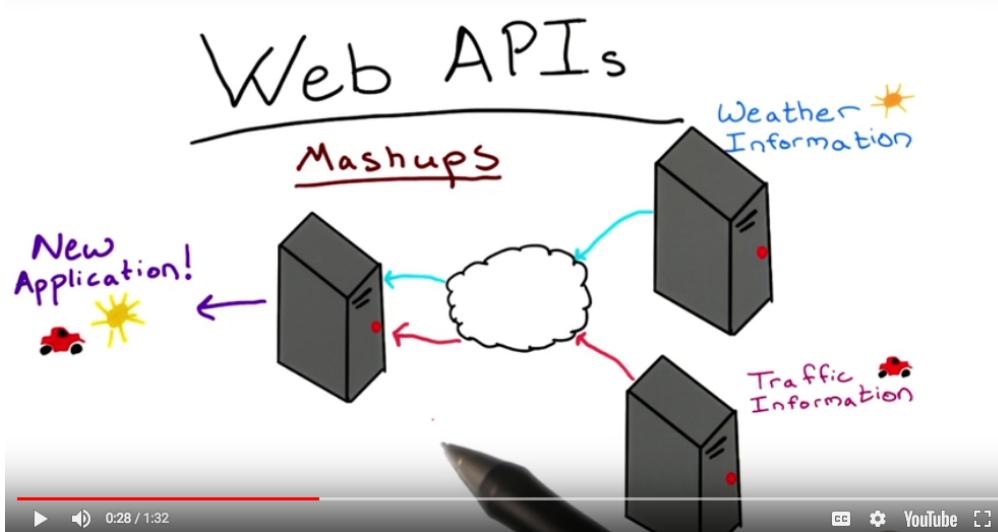
You will also need [Google Developer](#) and [Foursquare Developer](#) accounts for this course.

## 1.03. What are APIs

### Finally learning about APIs!

- Application Programming Interface
- Refers to any communication between two methods of code.
- APIs allow access to some of the application functions without exposing all the code.

- Developers can combine information from multiple APIs into a "mashup" application.



- Providing APIs helps the company's reputation. Some companies depend on their API as a primary revenue stream, like Twilio.

## 1.04. Web APIs

## 1.05. Web API Protocols

The OSI abstraction layers:

- Message formatting*
  - JSON
  - XML
- Web service*
- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

Each layer is served by the layer below, and serves the layer above.

Learn more about the [OSI model](#).

## 1.06. Digging into The Application Layer

## 1.07. The Web Service Layer

REST is a set of guidelines that leverage HTTP requests to transmit information.

Update: While SOAP is still used in some contexts, RESTful APIs currently dominate the API landscape. It is no longer an 'up and coming' technology, but rather simply the best practice for API development.

## 1.08. Content Formatting Layer

JSON or XML

## 1.09. Choosing the Right Technologies Soap vs Rest

- Soap was developed by Microsoft and uses XML.
- REST first appeared in a [dissertation by Roy Thomas Fielding in 2000](#).
- REST is far more popular than SOAP, as reported by [ProgrammableWeb](#).
  - I searched around for the chart that Lorenzo shows but couldn't find it.
  - Lots of other helpful resources though, like [APIs 101](#).

## 1.10. XML vs JSON

XML

- Developed in 1997
- Uses identifying tags similar to HTML

JSON

- Developed in 2001
- Derived from JavaScript
- Can be condensed to reduce file size

JSON has been gaining popularity.

## 1.11. Quiz: Developer Discussions

## 1.12. Quiz: Developer Discussions

## 1.13. Quiz: Developer Discussions

Got all of these on my first try. We had to identify the subject that two coworkers were discussing at the water cooler.

## 1.14. REST Constraints

REST applies a few additional specifications on top of HTML.

1. Separation of client and server
2. Stateless
  - Stateful server remembers client's activity between requests.

- RESTful architecture does not allow retention of client information between requests. Each request is independent.
- Tokens provide some memory functionality in RESTful architectures.

3. Cacheable
4. Uniform interface
5. Layered system
6. Code on demand

## 1.15. Quiz: Why Stateless

In short, stateless servers make your applications scalable. Check out [this article](#) if you are interested in learning more about why RESTful applications are stateless.

## 1.16. Lesson 1 Wrap Up

BOOM! Got it done in three pomodoros (75 min).

---

[\(Back to TOC\)](#)

# Lesson 2. Accessing published APIs

---

## 2.01. Lesson 2 Intro

## 2.02. Parts of an HTTP Request

Brief review of HTTP requests. It's a **pull** protocol.

### Structure of an HTTP request

- Header
  - Request line
    - HTTP verb
    - URI
    - HTTP/version
  - Optional request headers
    - name:value, name:value
- Blank line
- Body (optional)
  - Additional information

### Example HTTP request

```
GET puppies.html HTTP/1.1
Host: www.puppyshelter.com
```

```
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35
```

```
puppyId=12345&name=Fido+Simpson
```

## › 2.03. HTTP Response

### › Structure of an HTTP response

- Header
  - Status line
    - HTTP/version
    - Status code
    - Reason phrase
  - Optional response headers
- Blank line
- Body (optional)

### › Example HTTP response

```
HTTP/1.1 OK
Date: Fri, 04 Sep 2015 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb 2014
ETag: "0-23-4024c3a5:
ContentType: text/html
ContentLength: 35
Connection: KeepAlive
KeepAlive: timeout=15, max = 100

<h1>Welcome to my home page!</h1>
```

## › 2.04. Sending API Requests with Postman and Curl

- Curl is a popular command line tool for sending and receiving HTTP.
- Postman provides a GUI.

### › 2.04 Instructor notes

Check out the official cURL [documentation](#) and this [blog post](#) for getting started with cURL.

Check out the [Postman Documentation](#) as well.

You should also be familiar with [query strings](#) as option to pass in variables straight from the URI.

## 2.05. Quiz: Sending API Requests

### 2.05 Instructor notes

Download the code for api\_server.py [here](#).

There are some exceptions, but for most use cases:

```
HTTP <--> CRUD
GET <--> READ
POST<--> CREATE
PUT <--> UPDATE/CREATE
PATCH <--> UPDATE
DELETE-<-->DELETE
```

### Quiz

#### Setup

- Downloaded a Python web server file to access APIs. I cloned the repo into my vagrant directory.
- Started up vagrant as usual, changed into the directory with the API server file, and ran it. Took me a few tries to change into the proper directory because they put spaces in the directory names.

```
$ cd <path>
$ vagrant up
$ vagrant ssh
vagrant@vagrant:~$ cd /vagrant/APIs/Lesson_2/06_Sending\ API\ Requests
$ python api_server.py
```

```
- Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
- Restarting with stat
- Debugger is active!
- Debugger PIN: 769-024-554
```

#### Solution

## 2.06. Searching for API Documentation

Errata:

As of December 2017, SoundCloud is no longer accepting new API sign-ups.

## 2.07. Quiz: Delving Into APIs

### 2.07 Instructor notes

## Google Maps API

First, get your [Google Maps API key](#)

Then use the [Google Maps Geocoding API](#).

### › 2.07 Quiz

► *Solution*

### › 2.08. Quiz: Using the Foursquare API

#### › 2.08 Instructor notes

Here's the Foursquare API.

You may have realized that Foursquare has a geolocating capability as well, but since you already learned how to use the google maps one, you might as well put it to good use ;-)

You can copy and paste the coordinates below: Mountain View, California (37.392971, -122.076044) Miami, Florida (25.773822, -80.237947) Washington, DC (38.897478, -77.000147) New York, New York (40.768349, -73.96575)

#### › Foursquare

I read through the [Foursquare docs](#) and got queries set up for cURL and Python:

```
curl -X GET -G \
'https://api.foursquare.com/v2/venues/search' \
-d client_id="paste_client_id_here" \
-d client_secret="paste_client_secret_here" \
-d v="20180309" \
-d ll="37.392971,-122.076044" \
-d query="pizza" \
-d limit=1
```

```
import json
import requests

url = 'https://api.foursquare.com/v2/venues/search'

params = dict(client_id='paste_client_id_here',
              client_secret='paste_client_secret_here',
              v='20180309',
              ll='37.392971, -122.076044',
              query='pizza',
              limit=1)
resp = requests.get(url=url, params=params)
data = json.loads(resp.text)

print(data)
```

I called the python file (named *foursquare.py*) with `python foursquare.py`.

Both the cURL and Python queries work, but I need to parse the json.

## › Feedback on 2.08

This section was frustrating. The directions were not clear enough. I followed the Foursquare docs, and spent time writing Python code and trying to figure out how to parse the response before I realized that the next sections would cover that. I just skipped to the next section.

## › 2.09. Requesting From Python Code

Docs on Python:

- [Foursquare](#)
- [Google](#)

Lorenzo's code in *geocode.py* is of course, outdated and poorly formatted (not PEP 8).

Why did he just introduce Foursquare if we weren't going to use it in this exercise?

## › 2.10. Parsing Your Response

I used the instructor's Python code as a starting point to help me parse the JSON. I implemented some of the [string formatting techniques I used for the database logs analysis project](#) to format the JSON.

I first looked up the keys with this code added to *foursquare.py*:

```
print('Keys: {}'.format(data.keys()))
```

```
Keys: [u'meta', u'response']
```

I then moved into the Foursquare JSON, as Lorenzo did for the Google maps data. It's like a file path:

```
import json
import requests

url = 'https://api.foursquare.com/v2/venues/search'

params = dict(client_id='paste_client_id_here',
              client_secret='paste_client_secret_here',
              v='20180309',
              ll='37.392971, -122.076044',
              query='pizza',
              limit=1)
```

```
resp = requests.get(url=url, params=params)
data = json.loads(resp.text)

print('Venue name: {}'.format(data['response']['venues'][0]['name']))
```

Venue name: Fast Pizza Delivery

The next step would be to loop through each result.

Now I could answer the quiz questions, but not before.

## › 2.11. Quiz: Make Your Own API Mashup

### › Coding

The solution code is totally not PEP 8 compliant, and uses `httpplib2` from Python 2 ([renamed](#) to `http.client` in Python 3).

I re-wrote the code for Python 3, Requests, [Kenneth Reitz's Code Style](#) (Amended PEP 8), and Foursquare Python code recommendations. See *find-a-restaurant.py*.

This section took me two days to complete.

### › Printing output

- The first draft of *find-a-restaurant.py* runs without errors, but didn't print output.
- I started by debugging *geocode.py*, which is called by *find-a-restaurant.py*. I added in some output at the end of the file for debugging:

```
# If this file is called as a standalone program:
if __name__ == '__main__':
    # Run a sample location for debugging
    sample_location = 'Tokyo, Japan'
    sample_coordinates = get_geocode_location(sample_location)
    print('Sample location: {}'.format(sample_location),
          '\n', 'Coordinates: {}'.format(sample_coordinates))
```

- Output showed the file running successfully:

```
$ cd <path>
$ python3 geocode.py
Sample location: Tokyo, Japan
Coordinates: (35.6894875, 139.6917064)
```

- Next, I moved back into *find-a-restaurant.py*.

- I thought there might be a problem with the Foursquare code. I tried reformatting the URL, but nothing changed.
- I finally got some output by commenting out the `sys` commands from the instructor solution, which are there to render non-ascii characters properly in code:

```
# import sys
# import codecs

# sys.stdout = codecs.getwriter('utf8')(sys.stdout)
# sys.stderr = codecs.getwriter('utf8')(sys.stderr)
```

```
$ python3 find-a-restaurant.py
Traceback (most recent call last):
  File "find-a-restaurant.py", line 61, in <module>
    find_a_restaurant('Pizza', 'Tokyo, Japan')
  File "find-a-restaurant.py", line 25, in find_a_restaurant
    if data['response']['venues']:
KeyError: 'venues'
```

## Formatting coordinates for Foursquare

- Okay, so the Traceback above suggests there is a problem with the Foursquare data. I added a `print(data)` line, which allowed me to see the Foursquare JSON when I ran the Python file again:

```
$ python3 find-a-restaurant.py
{'response': {}, 'meta': {'code': 400, 'requestId': '5aa45eab4434b94b54af87d8',
 'errorType': 'param_error', 'errorDetail': 'll must be of the form XX.XX,YY.YY
(received latitude,longitude)'}}
Traceback (most recent call last):
  File "find-a-restaurant.py", line 68, in <module>
    find_a_restaurant('Pizza', 'Tokyo, Japan')
  File "find-a-restaurant.py", line 32, in find_a_restaurant
    if data['response']['venues']:
KeyError: 'venues'
```

- This tells me that I'm not passing the latitude and longitude in correctly from Google Maps.
- Adding in a print debug line helped me see where I was at:

```
def find_a_restaurant(mealType, location):
    """Locate a restaurant based on type of food and location."""
    # Get latitude and longitude of location from Google Maps API
    latitude, longitude = get_geocode_location(location)
    print(latitude, longitude)
    # Find restaurant with Foursquare API
    url = 'https://api.foursquare.com/v2/venues/search'
    params = dict(
        client_id='pasted-it-here',
        client_secret='pasted-it-here',
```

```

    v='20180310',
    ll=(latitude, longitude),
    query='mealType')
resp = requests.get(url=url, params=params)
data = json.loads(resp.text)
print(data)

```

```

$ python3 find-a-restaurant.py
35.6894875 139.6917064
{'meta': {'errorType': 'param_error', 'code': 400, 'requestId':
'5aa462076a607114459cbce3', 'errorDetail': 'll must be of the form XX.XX,YY.YY
(received 35.6894875)'}, 'response': {}}
Traceback (most recent call last):
  File "find-a-restaurant.py", line 69, in <module>
    find_a_restaurant('Pizza', 'Tokyo, Japan')
  File "find-a-restaurant.py", line 33, in find_a_restaurant
    if data['response']['venues']:
KeyError: 'venues'

```

- I applied some string formatting to pass Foursquare the info it wanted:

```

def find_a_restaurant(mealType, location):
    """Locate a restaurant based on type of food and location."""
    # Get latitude and longitude of location from Google Maps API
    latitude, longitude = get_geocode_location(location)
    print(latitude, longitude)
    coordinates = '{},{}'.format(latitude, longitude)
    # print(coordinates)
    # Find restaurant with Foursquare API
    url = 'https://api.foursquare.com/v2/venues/search'
    params = dict(
        client_id='pasted-it-here',
        client_secret='pasted-it-here',
        v='20180310',
        ll=coordinates,
        query='mealType')
    resp = requests.get(url=url, params=params)
    data = json.loads(resp.text)
    print(data)

```

- I finally got JSON back! It is a large amount of information, so I won't paste it here.

## › Retrieving photo info

- On to the next error:

```

Traceback (most recent call last):
  File "find-a-restaurant.py", line 71, in <module>
    find_a_restaurant('Pizza', 'Tokyo, Japan')
  File "find-a-restaurant.py", line 46, in find_a_restaurant

```

```
if data['response']['photos']['items']:
    KeyError: 'photos'
```

- I realized that I needed a second [API call to get the venue's photos](#). The URL is different, because it requires input of the venue id inside the URL. I accomplished this with some string insertion, using the `venue_id` variable from the first query:

```
url = 'https://api.foursquare.com/v2/venues/%s/photos' % venue_id
```

- I also found [Lorem Picsum](#) for random images, in case the image URL can't be retrieved.
- The rest of the code was straightforward.

### › Program code

- ▶ `geocode.py`
- ▶ `find-a-restaurant.py`

### › Program output

- ▶ Full output

## › 2.12. Lesson 2 Wrap Up

### Instructor notes

#### Some Interesting APIs to Explore

- [Wikipedia API](#)
- [StackExchange API](#)
- [Google Maps API](#)

## › Feedback on lesson 2

- Part 8 on Foursquare was frustrating. The directions were not clear. The quiz should be moved after the sections on Python and parsing the JSON. It didn't make sense to introduce Foursquare here, because we didn't use it until the end of the lesson.
- The API mashup Python code is outdated (Python 2, `httplib2` instead of `requests`) and poorly formatted (not PEP 8). The GitHub repo doesn't have a proper README.

This was good practice with APIs, but I shouldn't have to spend so much time figuring out a solution that was supposed to have already been provided. I spent two days on this when I could have been working on an actual project. I considered submitting my updated code as a pull request on GitHub, but what's the point? At the time I completed the lesson, there were [10 open issues](#) and [20 open pull requests on GitHub](#).

[\(Back to TOC\)](#)

## › Lesson 3. Creating your own API endpoints

---

### › 3.01. Lesson 3 Intro

In this lesson, we will build a Flask app with functioning back-end and database.

Instructor notes

| <http://flask.pocoo.org/docs/0.10/tutorial/>

### › 3.02. API Endpoints with Flask

- Use the app route decorator

```
@app.route('/')
```

- Import `jsonify` to convert dictionaries into JSON objects.

Instructor notes

| Click to view the [Flask Quickstart Documentation](#).

### › 3.03. Quiz: Making an Endpoint with Flask Quiz

#### › Updating and debugging the Flask server code

I began by formatting the code for Python 3 and PEP 8, as I did for the API code in lesson 2. I also updated it to use the Requests module, which streamlined the code.

- ▶ `endpoints_solution.py`
- ▶ `endpoints_tester.py`

#### › Running the Flask server

```
$ cd <path>
$ FLASK_APP=endpoints_solution.py flask run
```

Once the Flask server is running, open another terminal window to run the debugger.

```
$ cd <path>
$ python3 endpoints_tester.py
```

```
Running Endpoint Tester...
```

```
Please enter the address of the server you want to access.  
If left blank the connection will be set to 'http://localhost:5000':  
Making a GET Request for /puppies...  
Test 1 PASS: Successfully Made GET Request to /puppies  
Making GET requests to /puppies/id  
Test 2 PASS: Successfully Made GET Request to /puppies/id  
ALL TESTS PASSED!!
```

## › Decision to stop revising code

At this point, I decided to stop revising the code. There is too much to update and it will take me a huge amount of time. If Udacity is interested, I can help them update the code later.

## › 3.03 Instructor notes

View the [Starter Code](#) here.

Correction in video: In line 7, rename the function to puppiesFunction() and in line 13 rename the function to puppiesFunctionId(id). CORRECTION: The second function name here will be puppiesFunctionId(id)

View the [solution code](#) here.

## › 3.04. Quiz: Respond to Different Kinds of Requests

## › 3.05. Quiz: Serializing Data from the Database

## › 3.05 Instructor notes

View the [starter code](#) for this exercise.

View the [solution code](#) for this exercise.

Errata:

The code in the starter and solution files in the Github repo has been updated slightly from the code in the video. For instance, the correct route for `puppiesFunction` is `"/puppies"` (with no slash), not `"/puppies/"`. When in doubt, refer to the code and the tests for it.

## › 3.06. Quiz: Adding features to your Mashup

## › 3.07. Lesson 3 Wrap Up

## › Lesson 4. Securing your API

---

### › 4.01. Lesson 4 Intro

### › 4.02. Adding Users and Logins

Hashing the password scrambles it and prevents others from seeing it. We don't have to store the actual password in the database.

We will use the [passlib](#) module to hash passwords in Python.

► `models.py`

### › 4.03. User Registration

#### › 4.03 Instructor notes

At the end of the video, Lorenzo recommends performing secure login over HTTPS. For more about HTTPS (HTTP over TLS encryption), see these courses —

- [Client-Server Communication](#)
- [HTTP and Web Servers](#)

You can find the code for this video [here](#).

### › 4.04. Password Protecting a Resource

#### › 4.04 Instructor notes

View the code for this video [here](#).

Errata:

The API for the Flask HTTP Basic Authentication library has changed since this course was created. The module has been renamed from `flask.ext.httpauth` to `flask_httpauth`. Today, the recommended import statement is as follows:

```
from flask_httpauth import HTTPBasicAuth
```

The older module name will still work, but will give you a warning message.

See examples in the `flask_httpauth` documentation.

### › 4.05. Quiz: Mom & Pop's Bagel Shop

### › 4.06. Token Based Authentication

A token is a string that the server generates. The token can be passed along with the HTTP request, like cookies, but tokens don't depend on a browser. We work with tokens using the `itsdangerous` Python library.

## › 4.07. Implementing Token Based Authentication in Flask

We used the `itsdangerous` module to generate tokens. The token is sent as the user name, and the password is ignored.

► `models.py`

## › 4.08. Quiz: Regal Tree Foods

## › 4.09. OAuth 2.0

## › 4.10. Adding OAuth 2.0 for Authentication

Note that here, in `views.py`, Lorenzo imports both `httplib2` and `requests`. Why not just use `requests`?

## › 4.11. Quiz: Pale Kale Salads & Smoothies

Instructor notes

You'll need to replace the `client_id` on line 14 of `clientOAuth.html` with your own Google client ID.

## › 4.12. Rate Limiting

## › 4.13. Rate Limiting Exercise

Rate limiting limits the number of HTTP requests. It is accomplished with an app decorator in `views.py`, a `class RateLimit(object)`, and the Python `redis` module.

Redis can be started from the command line with

```
redis server
```

## › 4.14. Quiz: Bargain Mart

## › 4.15. Lesson 4 Wrap Up

[\(Back to TOC\)](#)

---

## › Lesson 5. Writing developer-friendly APIs

## › 5.01. Lesson 5 Intro

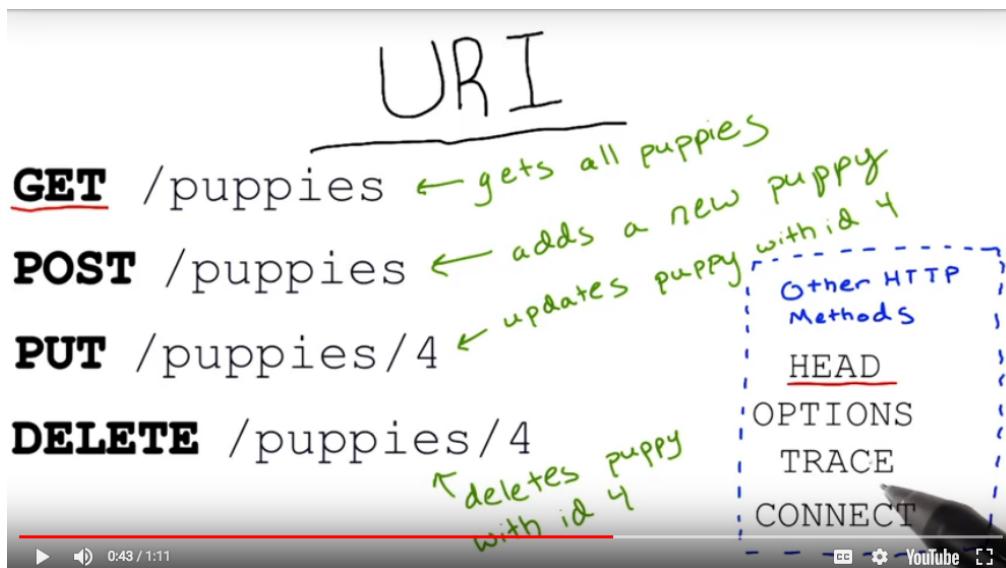
## › 5.02. Developer Friendly Documentation

| Documentation should be easy to navigate and aesthetically pleasing.

Lorenzo could take his own advice on this. The GitHub repo README isn't even Markdown formatted.

## › 5.03. Using Proper URIs

- URIs should always refer to resources, not actions being performed on those resources.
- Use plural for each resource name.
- HTTP verbs and URIs:



## › 5.03 Instructor notes

| I tend to use URL and URI interchangeably in this course, but there is a subtle difference between the two acronyms. Check out [this article](#) if you are interested in learning more about the differences and similarities between URLs and URIs.

## › 5.04. Versioning Your API

API version can be specified in the URI. I noticed this with Foursquare.

```
GET /api/v1/puppies  
GET /api/v2/puppies
```

## › 5.05. Communicating With Developers

Support your users with human communication, like blogs etc.

## ’ 5.06. Learning From The Best

Learn from other well-built apps.

## ’ 5.07. Course Wrap Up

There is some sort of vestigial "final project" in the repo that we didn't do.

---

## ’ General API course feedback

README

- The README is not formatted with Markdown, and Lorenzo even did the two spaces after each sentence thing.
- Python package list should be rearranged alphabetically

I submitted a [pull request](#) to improve the README. My first open-source pull request!

*Python:* Python code is outdated and poorly formatted.

- Python 2 instead of 3
- HTTP requests use `http1lib2` instead of `requests`
- Code is not PEP 8 compliant.
- JavaScript camelCase used instead of underscores for function names.

I pushed my Python code for lesson 2 to my forked repo, and will submit a pull request if Udacity is interested in updating the code.

I will keep track of my pull requests for this repo in [fsnd03\\_14-18-apis-pulls.md](#).

# APIs repo pull requests

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Designing RESTful APIs](#)

## Branch: docs

---

### Format and update README.md #42

#### Commits

Format README.md for Markdown

- Header spacing
- Code block

Update course info

- Link to course
- Add "Why take this course?" from website
- Add list of lessons
- Add links to recommended prior courses

Rearrange Python package list alphabetically

- Rearrange Python package list alphabetically
- Rename "libraries" to "modules"
- Provide instructions for accessing local package list

Remove redis installation instructions

Add optional virtual machine configuration

- Describe purpose of virtual machine
- List components with links
- Provide installation instructions
- Provide operation instructions

Add Table of Contents

- Create Table of Contents with DocToc
- Add back to top link at bottom of file

Add hyperlinked Udacity logo

- Insert Udacity logo from AWS with HTML img tag
- Hyperlink image by wrapping in HTML link tag

## › Pull request

**This was my first open source pull request!**

base fork: udacity/APIs base:master head fork: br3ndonland/APIs compare: docs Able to merge. These branches can be automatically merged.

Greetings Udacious Humans!

I humbly submit these suggested updates to the README in the APIs repository. As suggested in Udacity's [Git course](#), I created a topic branch for this pull request, and changed the file with a series of small, focused commits. Each item in the list below represents a separate commit.

- Format README.md for Markdown
- Update course info
- Rearrange Python package list alphabetically
- Remove redis installation instructions
- Add optional virtual machine configuration
- Add Table of Contents
- Add hyperlinked Udacity logo

This merge would supersede the following open pull requests:

- add spaces to fix formatting #30
- update README's format #31
- small update. #35

## › Branch: python3-pep-8-requests

---

### › Update Lesson 2 for Python 3, PEP 8, & Requests

#### › Commit

Update Lesson 2 for Python 3, PEP 8, & Requests

- Update Python code for Python 3
- Update Python code for Requests instead of httplib
- Revise Python code with PEP 8 formatting
- Clarify step comments in starter code

# CRUD

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Full Stack Foundations](#)

Lesson 1. Working with CRUD

## Table of Contents

---

- [Table of Contents](#)
- [Intro](#)
  - [01. Course Intro](#)
  - [02. Prerequisites & Preparation](#)
- [CRUD intro](#)
  - [03. Project Introduction - CRUD](#)
  - [04. Quiz: CRUD Review 1](#)
  - [05. Quiz: CRUD Review 2](#)
  - [06. Quiz: CRUD Review 3](#)
  - [07. Quiz: CRUD Review 4](#)
- [SQL](#)
  - [08. SQL](#)
  - [09. Quiz: SQL Quiz Select](#)
  - [10. Quiz: SQL Insert Into](#)
  - [11. Quiz: SQL Update](#)
  - [12. Quiz: SQL Delete](#)
  - [13. Creating a Database and ORMs](#)
- [SQLAlchemy](#)
  - [14. Introducing SQLAlchemy](#)
  - [15. Creating a Database - Configuration](#)
  - [16. Creating a Database - Class and Table](#)
  - [17. Creating a Database - Mapper](#)
  - [18. Putting It All Together](#)
  - [19. Quiz: Database Setup Quiz Part 1](#)
  - [20. Quiz: Database Setup Quiz Part 2](#)
- [CRUD process](#)
  - [21. CRUD Create](#)

- 22. Quiz: CRUD Create Quiz
- 23. CRUD Read
- 24. Quiz: CRUD Read Quiz
- 25. CRUD Update
- 2 Quiz: CRUD Update Quiz
- 27. CRUD Delete
- 28. Quiz: CRUD Delete Quiz
- 29. CRUD Review
- [Outro](#)
- [30. Wrap Up](#)
- [Feedback on Lesson 1](#)

## ’ Intro

---

### ’ 01. Course Intro

Lorenzo Brown will be the instructor for this course.

We will be making **data-driven web applications**.

- Web applications: Interactive user experience.
- Data-driven: Uses data to deliver customized content.

### ’ 02. Prerequisites & Preparation

- Git
- VirtualBox
- Vagrant

Already ready!

## ’ CRUD intro

---

### ’ 03. Project Introduction - CRUD

We're going to be creating a restaurant menu app. We will be able to:

- CREATE menu items
- READ the items we've created
- UPDATE with new items
- DELETE items

Basically all our actions on the web can be summarized by CRUD.

### ’ 04. Quiz: CRUD Review 1

Reading news articles is a READ.

## ’ 05. Quiz: CRUD Review 2

Clearing out junk mail is a DELETE.

## ’ 06. Quiz: CRUD Review 3

Making a new blog profile is a CREATE.

## ’ 07. Quiz: CRUD Review 4

Changing the number of items in an online shopping cart would be an UPDATE.

# ’ SQL

---

## ’ 08. SQL

SQL can go between databases and application code.

## ’ 09. Quiz: SQL Quiz Select

SQL `SELECT` is like CRUD READ.

## ’ 10. Quiz: SQL Insert Into

SQL `INSERT INTO` is like CRUD CREATE. It inserts a new row into the database.

## ’ 11. Quiz: SQL Update

SQL `UPDATE` is like CRUD UPDATE!

Correct! An UPDATE in SQL changes an existing row of information in a database table, just like the UPDATE in CRUD

## ’ 12. Quiz: SQL Delete

SQL `DELETE` is like CRUD DELETE!

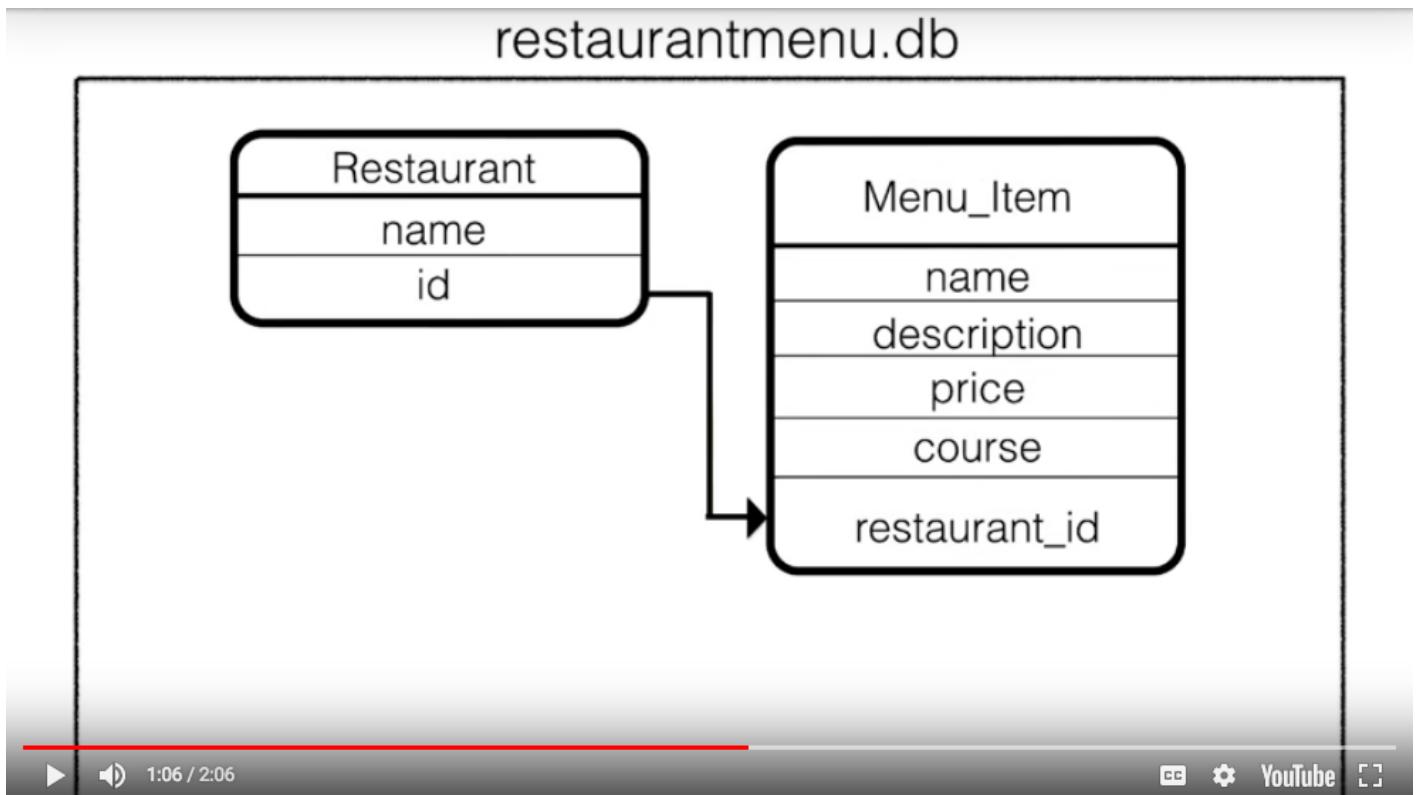
## ’ 13. Creating a Database and ORMs

Restaurant menus are all pretty similar. The menu belongs to a specific restaurant, and contains menu items.

We walked through construction of the database layout. Makes sense to me!

restaurantmenu.db

- Restaurant table
  - name
  - id
- Menu\_Item table
  - name
  - description
  - price
  - course
  - restaurant\_id as a foreign key referencing the Restaurant table.



We will use Object-Relational Mappers (ORMs) to transform code from Python objects to SQL statements.

## ’ SQLAlchemy

### ’ 14. Introducing SQLAlchemy

SQLAlchemy is an open-source ORM for Python. It was pre-installed in our Vagrant virtual machine.

### ’ 15. Creating a Database - Configuration

- Configuration
  - At beginning of file:
    - Import necessary modules
    - Create instance of "declarative base"
  - At end of file:
    - Connect database and add tables and columns

- Class
  - Represent database tables as Python classes
    - Restaurant table
    - Menu\_Item table
  - We will use CamelCase for class names.
  - We will include table and mapper code.
- Table
  - Represents specific table in database
- Mapper
  - Maps columns of the database tables to Python objects
  - `columnName = Column(attributes, ...)`
  - Example attributes
    - `String(250)`
    - `Integer`
    - `relationship(Class)`
    - `nullable = False`
    - `primary_key = True`
    - `ForeignKey('some_table.id')`

I followed along and filled out the code in `database_setup.py`.

## ’16. Creating a Database - Class and Table

## ’17. Creating a Database - Mapper

## ’18. Putting It All Together

## ’19. Quiz: Database Setup Quiz Part 1

`Base` should be an instance of class `declarative_base`

## ’20. Quiz: Database Setup Quiz Part 2

`id = Column(Integer, employee = relationship(Employee))`

## ’CRUD process

---

## ’21. CRUD Create

- We walked through database creation in the Python shell. I followed along in vagrant.
- I copied `database_setup.py` into `vagrant/crud`.
- SQLAlchemy uses "sessions" to connect to the database. We can store the commands we plan to use, but not send them to the database until we run a commit.

- Setup:

```
python
>>> from sqlalchemy import create_engine
>>> from sqlalchemy.orm import sessionmaker
>>> from database_setup import Base, Restaurant, MenuItem
>>> engine = create_engine('sqlite:///restaurantmenu.db')
>>> Base.metadata.bind = engine
>>> DBSession = sessionmaker(bind = engine)
>>> session = DBSession()
```

Note use of `database_setup` to import classes, not `database_setup.py`.

- Making a new entry in the database is as easy as making a new object in Python:

```
newEntry = ClassName(property = 'value', ...)
session.add(newEntry)
session.commit()
```

I did this with the restaurant menu database:

```
>>> stokespurple = MenuItem(name = "Stokes Purple Sweet Potato", description = "The legend")
>>> session.add(stokespurple)
>>> session.commit()
```

## ◦ 22. Quiz: CRUD Create Quiz

Rocked it. Entry 1 was `name`, entry 2 was `newEmployee` (the foreign key).

## ◦ 23. CRUD Read

I cloned the *Full-Stack-Foundations* repo into the *vagrant* directory.

## ◦ 24. Quiz: CRUD Read Quiz

Rocked it again. The code was working with an example database of employees. We selected all the employees, and stored them in a variable:

```
employees = session.query(Employee).all()
```

then printed out the name of each employee:

```
for employee in employees:
    print(employee.name)
```

## 25. CRUD Update

Simple four step process:

1. Query database with SQLAlchemy and store query as an object

```
query = session.query(ClassName).filter_by(name='Old Item Name')
```

2. Overwrite object with new info for database

```
query.name = 'New Item Name'
```

3. Add to SQLAlchemy database session with `session.add()`.

```
session.add(query)
```

4. Commit to SQLAlchemy database session with `session.commit()`.

```
session.commit()
```

For the restaurant example in this lesson:

1. Query database with SQLAlchemy and store query as an object

```
>>> veggieBurgers = session.query(MenuItem).filter_by(name='Veggie Burger')
>>> for veggieBurger in veggieBurgers:
...     print veggieBurger.id
...     print veggieBurger.price
...     print veggieBurger.restaurant.name
...     print "\n"
>>> UrbanVeggieBurger = session.query(MenuItem).filter_by(id=8).one()
>>> print UrbanVeggieBurger.price
```

2. Overwrite object with new info for database

```
>>> UrbanVeggieBurger.price = '$2.99'
```

3. Add to SQLAlchemy database session with `session.add()`.

```
>>> session.add(UrbanVeggieBurger)
```

4. Commit to SQLAlchemy database session with `session.commit()`.

```
>>> session.commit()
```

Lorenzo also created another `for` loop to change all the veggie burger prices to \$2.99, if they are not already that price:

```
>>> for veggieBurger in veggieBurgers:  
...     if veggieBurger.price != '$2.99':  
...         veggieBurger.price = '$2.99'  
...         session.add(veggieBurger)  
...         session.commit()  
...  
>>> for veggieBurger in veggieBurgers:  
...     print veggieBurger.id  
...     print veggieBurger.price  
...     print veggieBurger.restaurant.name  
...     print "\n"
```

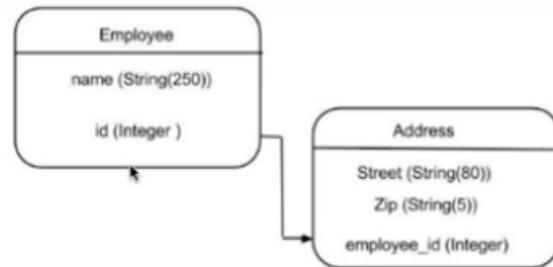
The instructor notes explain:

Note: The id numbers and restaurants for the Veggie Burgers will be a bit different on your machine than the ones in this lesson. `lotsofmenus.py` was updated to have a few more restaurant options and menu items.

## 2 Quiz: CRUD Update Quiz

Rocked it. Just needed `session.add(RebeccasAddress)`

Update Rebecca's address in the database



```
rebecca = session.query(Employee).filter_by(name = "Rebecca Allen").one()  
  
RebeccasAddress = session.query(Address).filter_by(  
    employee_id = rebecca.id).one()  
  
RebeccasAddress.street = "281 Summer Circle"  
  
RebeccasAddress.zip = "00189"  
  
session.add( RebeccasAddress )  
session.commit()
```

## 27. CRUD Delete

Three step process

1. Find entry
2. `session.delete(entry)`
3. `session.commit()`

## › 28. Quiz: CRUD Delete Quiz

```
session.delete(Mark)
```

## › 29. CRUD Review

### › Operations with SQLAlchemy

In this lesson, we performed all of our CRUD operations with SQLAlchemy on an SQLite database. Before we perform any operations, we must first import the necessary libraries, connect to our `restaurantMenu.db`, and create a session to interface with the database:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from database_setup import Base, Restaurant, MenuItem

engine = create_engine('sqlite:///restaurantMenu.db')
Base.metadata.bind=engine
DBSession = sessionmaker(bind = engine)
session = DBSession()
```

### › CREATE

We created a new Restaurant and called it Pizza Palace:

```
myFirstRestaurant = Restaurant(name = "Pizza Palace")
session.add(myFirstRestaurant)
session.commit()
```

We created a cheese pizza menu item and added it to the Pizza Palace Menu:

```
cheesepizza = menuItem(name="Cheese Pizza", description =
    "Made with all natural ingredients and fresh mozzarella",
    course="Entree", price="$8.99", restaurant=myFirstRestaurant)
session.add(cheesepizza)
session.commit()
```

### › READ

We read out information in our database using the query method in SQLAlchemy:

```
firstResult = session.query(Restaurant).first()
firstResult.name

items = session.query(MenuItem).all()
for item in items:
    print item.name
```

## › UPDATE

In order to update an existing entry in our database, we must execute the following commands:

1. Query database with SQLAlchemy and store query as an object
2. Overwrite object with new info for database
3. Add to SQLAlchemy database session with `session.add()`.
4. Commit to SQLAlchemy database session with `session.commit()`.

We found the veggie burger that belonged to the Urban Burger restaurant by executing the following query:

```
veggieBurgers = session.query(MenuItem).filter_by(name= 'Veggie Burger')
for veggieBurger in veggieBurgers:
    print veggieBurger.id
    print veggieBurger.price
    print veggieBurger.restaurant.name
    print "\n"
```

Then we updated the price of the veggie burger to \$2.99:

```
UrbanVeggieBurger = session.query(MenuItem).filter_by(id=8).one()
UrbanVeggieBurger.price = '$2.99'
session.add(UrbanVeggieBurger)
session.commit()
```

## › DELETE

To delete an item from our database we must follow the following steps:

1. Find the entry
2. `Session.delete(Entry)`
3. `Session.commit()`

We deleted spinach Ice Cream from our Menu Items database with the following operations:

```
spinach = session.query(MenuItem).filter_by(name = 'Spinach Ice Cream').one()
session.delete(spinach)
session.commit()
```

› **Outro**

---

› **30. Wrap Up**

› **Feedback on Lesson 1**

---

Lorenzo did a good job with this lesson. It was informative and fun, and the quiz questions made sense to me.

# Python web servers

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Full Stack Foundations](#)

Lesson 2. Python web servers

## Table of Contents

---

- [Table of Contents](#)
- [Python web server basics](#)
  - [01. Lesson 2 Introduction](#)
  - [02. Review of Clients, Servers and Protocols](#)
  - [03. HTTP and Response Codes](#)
  - [04. Building a Server with HTTPBaseServer](#)
  - [05. Quiz: Running a Web Server](#)
  - [06. Port Forwarding](#)
  - [07. Responding to Multiple GET Requests](#)
  - [08. Quiz: Hola Server](#)
  - [09. Adding POST to web server](#)
  - [10. Quiz: Running the POST Web Server](#)
- [Adding CRUD](#)
  - [11. Adding CRUD to our Website](#)
  - [12. Quiz: CRUD Objectives](#)
  - [13. Adding CRUD Hints](#)
  - [14. Quiz: CRUD Hints](#)
  - [15. Quiz: Objective 1](#)
  - [16. Quiz: Objective 2](#)
  - [17. Quiz: Objective 3](#)
  - [18. Quiz: Objective 4](#)
  - [19. Quiz: Objective 5](#)
  - [20. Lesson 2 Wrap-Up](#)
- [Feedback on Lesson 2](#)

## Python web server basics

---

## 01. Lesson 2 Introduction

Reviewing HTTP

## 02. Review of Clients, Servers and Protocols

- TCP
- IP
- UDP

## 03. HTTP and Response Codes

GET , POST , etc.

## 04. Building a Server with HTTPBaseServer

## 05. Quiz: Running a Web Server

I cloned the [files](#) into my vagrant directory for the last lesson. The files for this lesson are in *Full-Stack-Foundations/Lesson-2*.

I started the web server

```
$ vagrant ssh  
vagrant@vagrant:~$ cd /vagrant  
vagrant@vagrant:/vagrant$ cd Full-Stack-Foundations/Lesson-2/first-web-server  
vagrant@vagrant:/vagrant/Full-Stack-Foundations/Lesson-2/first-web-server$ python webserver.
```

I navigated to <http://localhost:8080/hello> and got the correct response.

## 06. Port Forwarding

Port forwarding allows us to open pages in our browser from the web server from our virtual machine as if they were being run locally. See which ports are being used for this class. If you want to use another port you can add another line to the vagrant file and run "vagrant reload" from a terminal in the directory of your vagrant file on your host machine. More information about port forwarding is available [here](#).

## 07. Responding to Multiple GET Requests

Hola server:

```
vagrant@vagrant:/vagrant/Full-Stack-Foundations/Lesson-2/first-web-server$ cd ..../hola-server  
vagrant@vagrant:/vagrant/Full-Stack-Foundations/Lesson-2/hola-server$ python webserver.py
```

<http://localhost:8080/hola>

| ¡Hola!

## ’ 08. Quiz: Hola Server

## ’ 09. Adding POST to web server

Lesson-2/post-web-server

## ’ 10. Quiz: Running the POST Web Server

Did it

## ’ Adding CRUD

---

## ’ 11. Adding CRUD to our Website

Adding CRUD capabilities for restaurant database

## ’ 12. Quiz: CRUD Objectives

1. Read: List restaurants for user on website <http://localhost:8080/restaurants>
2. Add edit and delete links to website after restaurant names
3. Create: Add page at <http://localhost:8080/new> with add restaurant feature that performs a `POST` request
4. Update: Rename restaurant feature at <http://localhost:8080/restaurant/id/edit>
5. Delete: Delete restaurant feature takes user to confirmation page, and removes the restaurant from the database with a `POST` request

It was helpful to see Lorenzo break down the objective into steps. I do this for my projects!

I just followed along in the solution code, rather than writing the solutions myself, in the interest of time. It is also more effective and efficient to produce the webserver using Flask, which we'll thankfully be doing in the next lesson!

The code hasn't been updated for Python 3, but will run with Python 2 in vagrant. Each `webserver.py` directory must also have the `database_setup.py` file.

## ’ 13. Adding CRUD Hints

- You have already seen all the necessary code
- Import necessary modules
- Reconstruct `do_GET` and `do_POST`
- use `print` statements to debug
- View page source in browser

## › 14. Quiz: CRUD Hints

## › 15. Quiz: Objective 1

Instructor notes:

Error in video - Replace lines 12 and 13 with:

```
DBSession = sessionmaker(bind=engine)  
session = DBSession()
```

Objective 1 solution can be found [here](#)

Note:

A few times I say backslash "" in this course, all slashes in this course should be forward slashes "/"

## › 16. Quiz: Objective 2

Objective 2 Solution Code can be found [here](#)

## › 17. Quiz: Objective 3

Note

messagecontent = fields.get('newRestaurantName') should be indented such that it is inside the 'if' block Objective 3 Solution Code can be found [here](#)

## › 18. Quiz: Objective 4

Objective 4 Solutions can be found [here](#)

## › 19. Quiz: Objective 5

Note

line 128 (ctype, pdict....) is not a necessary line of code to implement this solution. Code for Objective 5 can be found [here](#)

## › 20. Lesson 2 Wrap-Up

## › Feedback on Lesson 2

---

Lorenzo did a nice job teaching here as well, but the lesson could benefit from some cleanup. The code could be updated from Python 2 to Python 3. It's confusing to break up the material into so many different versions of the same files. Each of the Lesson-2/ solution directories in the GitHub repo also needs the database\_setup.py file to run properly.

# Flask

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

## [Full Stack Foundations](#)

Lesson 3. Flask

## Table of Contents

---

- [Table of Contents](#)
- [Lesson overview](#)
  - [01. Introducing Frameworks and Flask](#)
  - [02. Lesson 3 Overview](#)
  - [03. Running Your First Flask Application](#)
  - [04. Quiz: First Flask App Quiz](#)
- [Database](#)
  - [05. Adding Database to Flask Application](#)
  - [06. Quiz: Adding Database to Flask Application Quiz](#)
  - [07. Routing](#)
  - [08. Quiz: Routing Create Quiz](#)
- [Templates](#)
  - [09. Templates](#)
  - [10. Quiz: Templates Quiz](#)
- [URL for](#)
  - [11. Quiz: URL for Quiz](#)
- [Forms](#)
  - [12. Form Requests and Redirects](#)
  - [13. Quiz: Edit Menu Item Form Quiz](#)
  - [14. Quiz: Delete Menu Item](#)
- [Message flashing](#)
  - [15. Message Flashing](#)
  - [16. Quiz: Message Flashing Quiz](#)
- [Styling](#)
  - [17. Quiz: Styling](#)
- [JSON](#)
  - [18. Responding With JSON](#)
  - [19. Quiz: JSON Quiz](#)

- o 20. Lesson 3 Wrap-Up

- Feedback on Lesson 3

## Lesson overview

---

### 01. Introducing Frameworks and Flask

Frameworks take care of repetitive tasks and allow us to focus on unique features of our projects. Lorenzo likened frameworks to cake mix. The manual web server code we did in the last lesson is like baking from scratch.

### 02. Lesson 3 Overview

- Database
- Templates
- url\_for
- Forms
- Message flashing
- JSON
- Styling

### 03. Running Your First Flask Application

Simple "Hello, World!" Flask app

```
# Import Flask class from flask library
from flask import Flask
# Create an instance of class Flask
# Each time the application runs, a special variable 'name' is defined.
app = Flask(__name__)

# Decorators that wrap our function inside Flask's 'app.route' function.
@app.route('/')
# This is a decorator.
@app.route('/hello')
def HelloWorld():
    return "Hello World"

# "if __name__ == '__main__': describes what to do if the code is being
# executed by the Python interpreter.
# The app run by the Python interpreter is assigned '__main__' to its
# name variable.
# If the file is imported instead, the 'if' statement will not run.
# Other imported Python files are assigned the name of the Python file.
if __name__ == '__main__':
    # Turn on debugging to reload when code is changed
    app.debug = True
```

```
# Listen on all public IP addresses for compatibility with vagrant  
app.run(host='0.0.0.0', port=5000)
```

Instructor notes:

A quick tutorial on decorators:

<http://simeonfranklin.com/blog/2012/jul/1/python-decorators-in-12-steps/>

View the code for [project.py](#)

## ’ 04. Quiz: First Flask App Quiz

`@app.route('/')` is needed to specify behavior for the homepage.

## ’ Database

---

## ’ 05. Adding Database to Flask Application

Importing the SQLAlchemy code for the restaurant database.

► Code from project.py

## ’ 06. Quiz: Adding Database to Flask Application Quiz

## ’ 07. Routing

URL creation can be automated with naming rules:

```
path/<type: variable_name>/path/
```

`type` can be `int`, `string`, or `path`.

It is helpful to leave the trailing slash.

## ’ 08. Quiz: Routing Create Quiz

## ’ Templates

---

## ’ 09. Templates

Flask allows storage of HTML templates with

```
render_template(variable, keyword_objects)
```

so you don't have to repeatedly type `output` strings. Usually, developers create a sub-directory called `*templates-` and put their templates there.

\*HTML escaping- allows transfer of data from Python and databases into HTML.

## ’ 10. Quiz: Templates Quiz

```
{% logical code for Python %}  
{{ printed output code }}
```

HTML templates don't get to use Python indentation and spacing, so we have to include instructions to terminate loops:

```
{% endfor %}  
{% endif %}
```

## ’ URL for

### ’ 11. Quiz: URL for Quiz

URLs can be mapped to specific Python functions with `url_for()`

```
<a href='{{url_for('editMenuItem', restaurant_id = restaurant.id, menu_id = i.id) }}'>Edit</a>  
</br>  
<a href = '{{url_for('deleteMenuItem', restaurant_id = restaurant.id, menu_id = i.id ) }}'>Delete</a>
```

Instructor notes

Download the [menu.html file](#). Add `url_for()` links to edit and delete each menu item

Flask Documentation for `url_for` can be found [here](#).

## ’ Forms

### ’ 12. Form Requests and Redirects

### ’ 13. Quiz: Edit Menu Item Form Quiz

### ’ 14. Quiz: Delete Menu Item

### ’ Message flashing

## ’15. Message Flashing

Flask includes a messaging system to notify users after actions are completed. It uses sessions to persist settings across multiple web pages. It is invoked with `flash('Message here!')`. Retrieve flashed messages with `get_flashed_messages()`

`app.secret_key = 'key_name'` is used to establish sessions.

I need sessions to persist the color scheme on my [Udacity portfolio site!](#)

Errata:

Around 1:20 in the video, the instructor mentions adding code to `newMenuItem.html`. However, the file he is actually editing is `menu.html`. To follow along with this instruction, you should edit `menu.html` and not `newMenuItem.html`. Instructor notes:

[https://github.com/udacity/Full-Stack-Foundations/tree/master/Lesson-3/16\\_Flash-Messaging](https://github.com/udacity/Full-Stack-Foundations/tree/master/Lesson-3/16_Flash-Messaging) Click here to learn more about the 'with' statement in python: <http://effbot.org/zone/python-with-statement.htm>

## ’16. Quiz: Message Flashing Quiz

## ’Styling

---

## ’17. Quiz: Styling

Flask can look for CSS, JS, and media files inside of a `*static/-` directory.

## ’JSON

---

## ’18. Responding With JSON

Let's say there's a service called `*yum!-` that wants to read data from your restaurant menu database and display it to users based on their location. To see the list of restaurants and menus in the database, `*yum!-` doesn't need to waste bandwidth parsing HTML and CSS, it just needs the data.

What do we need? I knew the answer- an API! I've been wanting to get into APIs for a while now. Finally!

\**RESTful*- - REpresentational State Transfer. Sending the data over the internet with HTTP.

\**API*- - Application Programming Interface

APIs frequently use `*JSON`- (JavaScript Object Notation), which uses `{"key": "value",}` pairs. The Flask `jsonify` package allows us to easily configure an API endpoint for the application. `jsonify` converts our database into JSON.

## ’19. Quiz: JSON Quiz

## › 20. Lesson 3 Wrap-Up

## › Feedback on Lesson 3

---

Lorenzo did a great job with this lesson. All of his instructor materials were well-crafted, and it is very useful to learn the Flask framework.

# Agile iterative development

---



Udacity Full Stack Web Developer Nanodegree program

[Full Stack Foundations course](#)

Lesson 4. Agile iterative development

Brendon Smith

br3ndonland

## Table of Contents

---

- [Table of Contents](#)
- [Overview](#)
  - [01. Lesson 4 Introduction](#)
  - [02. Iterative Development](#)
  - [03. Quiz: Tackling a Complex Project](#)
- [Mockups](#)
  - [04. Quiz: Mockups Exercise](#)
- [Routes and templates](#)
  - [05. Quiz: Adding Routes](#)
  - [06. Quiz: Adding Templates and Forms](#)
- [CRUD and JSON API](#)
  - [07. Quiz: CRUD Functionality](#)
  - [08. Quiz: API Endpoints](#)
- [Style](#)
  - [09. Quiz: Styling Your App](#)
  - [10. Wrap-Up](#)
- [Feedback on Lesson 4](#)

## Overview

---

### 01. Lesson 4 Introduction

### 02. Iterative Development

Some people write code by just binging as much code as possible before falling asleep, or writing code in pieces and having it be all over the place.

A more effective strategy is to start simple and layer on complexity as you go, not adding a new feature until the current feature is finished. When each feature is finished, we stop, test, debug and share to make sure we always have a working application. This way, clients, project managers, and team members always know what the project status is and what's to follow.

This is called **iterative development**.

**Agile** refers to the ability to easily change an application. Because there is always a working prototype, if someone asks for changes, it is easily done.

## ’ 03. Quiz: Tackling a Complex Project

Break the project down into steps, and outline deliverables that will be present at each step.

Lorenzo used the restaurant menu app as an example.

- Mockups
  - Deliverables: Mock-ups and URLs for each page in the menu app.
- Routing
  - Deliverables: At the end of this iteration, you should have a running Flask application, and be able to navigate to all of the URLs, even if the pages are not yet created.
- Templates and forms
  - Deliverables: Functional templates and forms
- CRUD functionality
  - Deliverables: Ability to Create, Read, Update, and Delete.
- API endpoints
  - Deliverables: Ability to send API data in JSON.
- Styling and message flashing
  - Deliverables: Nicely styled app pages and messages when database is changed.

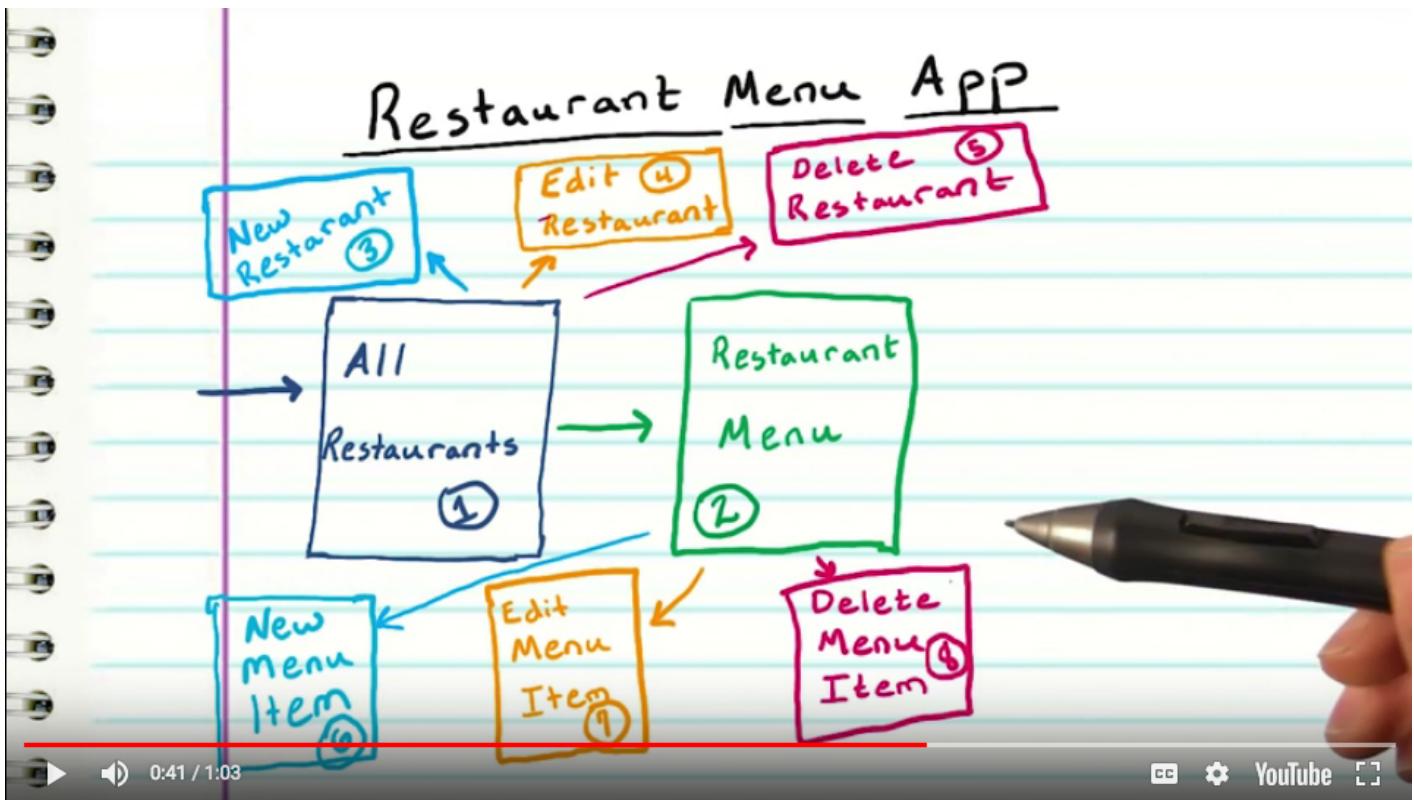
Quiz: pros and cons of "top-down" approach focusing on front-end first vs "bottom-up."

## ’ Mockups

---

## ’ 04. Quiz: Mockups Exercise

You can use wireframing apps, or just pen and paper.



Show all restaurants <code>/restaurants (and '/')</code>	Show a restaurant menu <code>/restaurant/&lt;int:restaurant_id&gt;/menu</code> (and <code>/restaurant/&lt;int:restaurant_id&gt;</code> )
Create a new restaurant <code>/restaurant/new</code>	Create a new menu item <code>restaurant/&lt;int:restaurant_id&gt;/menu/new</code>
Edit a restaurant <code>restaurant/&lt;int:restaurant_id&gt;/edit</code>	Edit a menu item <code>/restaurant/&lt;int:restaurant_id&gt;/menu/&lt;int:menu_id&gt;/edit</code>
Delete a restaurant <code>/restaurant/&lt;int:restaurant_id&gt;/delete</code>	Delete a menu item <code>/restaurant/&lt;int:restaurant_id&gt;/menu/&lt;int:menu_id&gt;/delete</code>

restaurant/⟨int:restaurant\_id⟩/menu



## Routes and templates

### 05. Quiz: Adding Routes

Lorenzo got the Flask application started and set up the page URLs.

URL	Method	Message
/restaurants	showRestaurants()	"This page will show all my restaurants"
/restaurant/new	newRestaurant()	"This page will be for making a new restaurant"
/restaurant/restaurant_id/edit	editRestaurant()	"This page will be for editing restaurant %s" % restaurant_id
/restaurant/restaurant_id/delete	deleteRestaurant()	"This page will be for deleting restaurant %s" % restaurant_id
/restaurant/restaurant_id /restaurant/restaurant_id/menu	showMenu()	"This page is the menu for restaurant %s" restaurant_id
/restaurant/restaurant_id/menu/new	newMenuItem()	"This page is for making a new menu item for restaurant %s" %r estaurant_id
/restaurant/restaurant_id/menu/menu_id /edit	editMenuItem()	"This page is for editing menu item %s " menu_id
/restaurant/restaurant_id/menu/menu_id /delete	deleteMenuItem()	"This page is for deleting menu item %s" menu_id

### 06. Quiz: Adding Templates and Forms

Lorenzo created empty HTML files that will serve as templates. Python will throw an error if the database has not yet been created, so he created a temporary database called "fake menu items."

## ’ CRUD and JSON API

---

### ’ 07. Quiz: CRUD Functionality

- SQLAlchemy commands: show, new, edit, delete
- url\_for
- redirect
- GET and POST

This is one of the more time-consuming steps.

### ’ 08. Quiz: API Endpoints

Use `jsonify` and `serialize` to return JSON when HTTP request is made to:

- /restaurants/JSON
- restaurants/restaurant\_id/menu/JSON
- restaurants/restaurant\_id/menu/menu\_id/JSON

## ’ Style

---

### ’ 09. Quiz: Styling Your App

- Static: CSS, JS, images
- Message flashing:
  - New restaurant created
  - Restaurant successfully edited
  - Restaurant successfully deleted
  - New menu item created
  - Menu item successfully edited
  - Menu item successfully deleted

### ’ 10. Wrap-Up

Instructor notes

You can view [this article](#) on deploying Flask applications, but I highly recommend adding some [security](#) to your application before publishing it on the internet.

A basic version of the final project can be found [here](#).

## ’ Feedback on Lesson 4

---

Really helpful to think through agile development. This will help me complete my Udacity projects, and my work projects when I get a web development job.

It was confusing to work through app development again here though, when we already did it in the previous lesson. It would be more helpful to discuss agile development first, then work through the whole Flask app creation process in that way, so the code base is cohesive.

# FSND JavaScript, AJAX, and APIs lessons

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

## Table of Contents

---

- [Table of Contents](#)
- [Intro to APIs](#)
  - [Lesson 01. Requests and APIs](#)
  - [Lesson 02. Building the Move Planner app](#)
- [JavaScript Design Patterns and Libraries](#)
  - [Lesson 03. Changing expectations](#)
  - [Lesson 04. Refactoring with separation of concerns](#)
  - [Lesson 05. Using an organization library](#)
  - [Lesson 06. Learning a new codebase](#)
- [Google Maps](#)
  - [Lesson 07. Getting started with the APIs](#)
  - [Lesson 08. Understanding API services](#)
  - [Lesson 09. Using the APIs in practice](#)
- [Feedback](#)

## Intro to APIs

---

### Lesson 01. Requests and APIs

- This part is similar to the [Intro to AJAX](#) free course.
- The instructor Cameron inspects Facebook and Twitter with developer tools (network tab, clicking on individual request paths) to show how AJAX is used.

### Lesson 02. Building the Move Planner app

- jQuery
  - Most people use jQuery for AJAX requests (the famous `$.ajax({})`).
  - In [cs50 web](#) during [lecture 05](#), Brian mentioned that jQuery is not as useful anymore, because many of its functions can be accomplished with standard JavaScript like `querySelector` and `querySelectorAll`. It's also a large library, so it's better not to load jQuery if you don't have to.

- The [Syntax podcast episode 39: Is jQuery dead?](#) was also very helpful.
  - The [Asynchronous JavaScript Requests course](#) compares XHR, jQuery and Fetch.
- [jQuery.getJSON\(\)](#)
  - [jQuery.error\(\)](#) method for error handling
  - CORS: Cross-Origin Resource Sharing

tl;dr CORS works around a sometimes overly-strict browser policy meant to protect servers from malicious requests. CORS is enabled on the server-side, so you won't generally need to worry about it for your code. You do need to know about it though, since some APIs support it, and some do not.

- [NYT article search API](#)
- [Wikipedia API](#)
- Split up generic and unique HTML to render content as quickly as possible.

## › JavaScript Design Patterns and Libraries

### › Lesson 03. Changing expectations

- From [JavaScript Design Patterns](#) free course
- The instructor Ben Jaffe builds a "cat clicker" app. See [ud989-cat-clicker-andy](#) and [ud989-retain](#). He uses Sublime Text in the video.
- **Separation of Concerns:** Separate out the different types of files and functions.
- **Model View Controller:** The instructor used "octopus" to represent controller.

### › Resources

In case you need a refresher on events and event listeners, here are some links.

If you're writing Cat Clicker with vanilla JS (no jQuery), you'll be adding the "click" event with elem.addEventListener().

```
var elem = document.getElementById('my-elem');
elem.addEventListener('click', function(){
    // the element has been clicked... do stuff here
}, false);
```

If you're using jQuery, you'll be adding the "click" event listener with jQuery.click().

```
$('#my-elem').click(function(e) {
    //the element has been clicked... do stuff here
});
```

## › Closures and Event Listeners

### › The problem

Let's say we're making an element for every item in an array. When each is clicked, it should alert its number. The simple approach would be to use a for loop to iterate over the list elements, and when the click happens, alert the value of `num` as we iterate over each item of the array. Here's an example:

```
// clear the screen for testing
document.body.innerHTML = '';
document.body.style.background="white";

var nums = [1,2,3];

// Let's loop over the numbers in our array
for (var i = 0; i < nums.length; i++) {

    // This is the number we're on...
    var num = nums[i];

    // We're creating a DOM element for the number
    var elem = document.createElement('div');
    elem.textContent = num;

    // ... and when we click, alert the value of `num`
    elem.addEventListener('click', function() {
        alert(num);
    });

    // finally, let's add this element to the document
    document.body.appendChild(elem);
};
```

If you run this code on any website, it will clear everything and add a bunch of numbers to the page. Try it! Open a new page, open the console, and run the above code. Then click on the numbers and see what gets alerted. Reading the code, we'd expect the numbers to alert their values when we click on them.

But when we test it, all the elements alert the same thing: the last number. But why?

### › What's actually happening

Let's cut out the irrelevant code so we can see what's going on. The comments below have changed, and explain what is actually happening.

```
var nums = [1,2,3];

for (var i = 0; i < nums.length; i++) {
```

```
// This variable **keeps changing** every time we iterate!
// Its first value is 1, then 2, then finally 3.
var num = nums[i];

// On click...
elem.addEventListener('click', function() {

    // ... alert num's value **at the moment of the click**!
    alert(num);

    // Specifically, we're alerting the num variable
    // that's defined **outside** of this inner function.
    // Each of these inner functions are pointing to the
    // same `num` variable... the one that changes on
    // each iteration, and which equals 3 at the end of
    // the for loop. Whenever the anonymous function is
    // called on the click event, the function will
    // reference the same `num` (which now equals 3).

});

};

});
```

That's why regardless of which number we click on, they all alert the last value of `num`.

## › How do we fix it

The solution involves utilizing closures. We're going to create an inner scope to hold the value of `num` at the exact moment we add the event listener. There are a number of ways to do this -- here's a good one.

Let's simplify the code to just the lines where we add the event listener.

```
var num = nums[i];

elem.addEventListener('click', function() {

    alert(num);

});
```

The `num` variable changes, so we have to somehow connect it to our event listener function. Here's one way of doing it. First take a look at this code, then I'll explain how it works.

```
elem.addEventListener('click', (function(numCopy) {
    return function() {
        alert(numCopy)
    };
})(num));
```

`(function(numCopy)` is the outer function. We immediately invoke it by wrapping it in parentheses and calling it right away, passing in num. This method of wrapping an anonymous function in parentheses and calling it right away is called an IIFE (Immediately-Invoked Function Expression, pronounced like "iffy"). This is where the "magical" part happens.

We're passing the value of `num` into our outer function. Inside that outer function, the value is known as `numCopy` -- aptly named, since it's a copy of `num` in that instant. Now it doesn't matter that `num` changes later down the line. We stored the value of `num` in `numCopy` inside our outer function.

Lastly, the outer function returns the inner function to the event listener. Because of the way JavaScript scope works, that inner function has access to `numCopy`. In the near future, `num` will increment, but that doesn't matter. The inner function has access to `numCopy`, which will never change.

Now, when someone clicks, it'll execute the returned inner function, alerting `numCopy`.

## › The Final Version

Here's our original code, but fixed up with our closure trick. Test it out!

```
// clear the screen for testing
document.body.innerHTML = '';

var nums = [1,2,3];

// Let's loop over the numbers in our array
for (var i = 0; i < nums.length; i++) {

    // This is the number we're on...
    var num = nums[i];

    // We're creating a DOM element for the number
    var elem = document.createElement('div');
    elem.textContent = num;

    // ... and when we click, alert the value of `num`
    elem.addEventListener('click', (function(numCopy) {
        return function() {
            alert(numCopy);
        };
    })(num));

    document.body.appendChild(elem);
};
```

## › Lesson 04. Refactoring with separation of concerns

- Refactoring is keeping the core functionality of the code the same, while making improvements to the functionality that will help the code scale. You can either start fresh ("burn it") or "refactor in place."

- The instructor goes through some ways to make the code more efficient.
  - Adding event handlers (listening for click events, etc.)
  - Storing data in the model, not in the view (DOM)
  - Ensuring the model and view never talk directly to each other, but go through the controller.
  - Having multiple views if needed.
- The repo [ud989-cat-clicker-premium-vanilla](#) contains the instructor's refactored code.
- We were also given the [ud989-school-attendance](#) repo, intentionally authored as spaghetti code by Mike Wales, and told to refactor it.
- We also went to the [frontend-nanodegree-resume](#) repo, part of the [Intro to JavaScript](#) course.
- I laughed at the "How to modernize projects" section of the lesson. The instructor says, "Every company has legacy code, even Udacity." That's an understatement. Udacity doesn't keep its own courses updated. They don't even add licenses to their GitHub repos.

## Lesson 05. Using an organization library

- This is where we get into [KnockoutJS](#).
- KnockoutJS uses MVVM (Model View ViewModel).
- Library vs. Framework:
  - **Libaries are just a bunch of pre-written code.** jQuery is a library. Some of the most common functions addressed by libraries are DOM manipulation and AJAX.
  - **Frameworks give more structure to your code.** They are also sometimes called Organizational Libraries.
- It's not a cop-out to use a library. Engineering is all about taking what's already been done and tested, and using it to build something more.
- Fundamental organizational concepts:
  - **Model:** The data, maybe in JSON for example. Collections are smart arrays filled with models (data).
  - **View:** Pages the users view. Routers are similar (like app routes in Flask).
  - **Controller** (the "octopus"). Also called **ViewModel** for Knockout. Libraries/frameworks that don't fit the MVC paradigm are called MV\*.
- **Knockout**
  - **ViewModel:** Similar to the "octopus" controller concept. Sits between data and DOM.
  - **Declarative Bindings:** Connect View and ViewModel
  - **Automatic UI refresh:** Updates view automatically when model changes
  - **Dependency tracking:** Models can depend on other models
- Knockout notes
  - Knockout is similar to jQuery, but extends the features of jQuery to provide more of an app framework.
  - **Observables** are variables that can automatically update the DOM when the view model changes.
- [Example](#)

```
var myNum = 5
var myNum = ko.observable(5)
```

- Knockout also has smart arrays called [observableArrays](#).
- Computed observables create values when accessed.
  - The instructor started by creating a function

```
let firstName = 'Ben'
let lastName = 'Jaffe'
let fullName = function () {
    return firstName + ' ' + lastName
}
```

- Note that you could also code the instructor's example above with ES6 template literals and arrow functions.

```
let firstName = 'Ben'
let lastName = 'Jaffe'
let fullName = () => `${firstName} ${lastName}`
console.log(fullName())
```

- You actually don't even need a function. Not the greatest example.

```
let firstName = 'Ben'
let lastName = 'Jaffe'
let fullName = `${firstName} ${lastName}`
console.log(fullName)
```

- The instructor then uses `data-bind=""` attributes in the view HTML.
- Knockout's method of encoding [computed observables](#):

```
function AppViewModel () {
    this.firstName = ko.observable('Bert')
    this.lastName = ko.observable('Bertington')

    this.fullName = ko.computed(() =>
        `${this.firstName()} ${this.lastName()}`, this)
}
```

- Note that I included ES6 template literals and arrow functions.
- The computed observables must be coded as functions. Again, not a great example because you could encode the fullName directly as a template literal.

- We downloaded the [cat clicker knockout repo](#) to practice.
- We get into `this`. The instructor uses `var self = this` (I guess with ES6 it would be `let self = this`) to allow reference to the top level `var viewModel` even when within other lower functions.
- **Documentation is important!** There was an interview with a Udacity engineer about the importance of documentation.
- Knockout's website looks several years out of date. The documentation doesn't even have previous/next navigation.

## ’ Lesson 06. Learning a new codebase

- Getting handed a bunch of code and dealing with it.
  - We worked with [TodoMVC](#), the same app written in many different frameworks. Cool. TodoMVC was created by Addy Osmani at Google. He has also written about [creating Progressive Web Apps \(PWAs\) with React](#).
  - The instructor Ben Jaffe incorporated TodoMVC into the [ud989-todo-app](#) repo. **The instructor's repo hasn't been updated in over three years, no license, no README.**
  - Ben goes through the different repo directories. They use Bower (outdated of course) to manage packages. Backbone is an organizational library used to separate concerns.
  - Ben and Jacques say, "Always use semicolons." Jacques also says, "Remember kids: Always use tabs." No thanks.
  - "Be tofu." Absorb the flavors from the codebase and adapt to the codebase.
  - "Be a detective."
- 

## ’ Google Maps

### ’ Lesson 07. Getting started with the APIs

- [Maps JavaScript API](#)
- This is the [Google Maps APIs course](#), with instructor Emily Keller from Google NYC. There is a [GitHub repo](#).
- As I went through the lesson, I compared the lesson code with the Maps docs, and added the code to my neighborhood map project.
- Maps are broken up into tiles and zoom levels.
- Emily creates an array and loops over it to drop pins, or [markers](#), on the map. She mentions her favorite dessert place nearby, Rice to Riches.
- She adds [infoWindows](#)
- She also adds event listeners to modify the page.
- Maps can be styled. See [Google map style docs](#) and [SnazzyMaps](#).
- **Static maps** are simple and embeddable. They are added through a separate API and may not be applicable to the neighborhood map project.
- Street views use pitch and heading to determine camera angle.
- Geometry library

- Visualization library has heat maps
- Drawing manager can add shapes around an area, and the user can also draw shapes.
  - Calculate area of a shape:

```
var area = google.maps.geometry.spherical.computeArea(polygon.getPath())
window.alert(area+" SQUARE METERS")
})
```

## ’ Lesson 08. Understanding API services

- [Geocoding](#) allows conversion between addresses and lat lng. It takes in a lat lng and returns an address. Reverse geocoding takes an address and returns lat lng.
- The geocoding API can accept HTTP requests, and returns JSON.

<https://maps.googleapis.com/maps/api/geocode/outputFormat?parameters>

- Don't confuse geocoding with [geocaching](#). I remember when geocaching became popular a few years ago.
- Quiz response: Key and address are needed.
  - Great job! The address is the only information needed to complete a geocoding request. Alternatively, if Ajay was doing REVERSE geocoding, he would only need to provide the latlng parameter! The KEY is required for authentication. Region is an optional parameter, which will BIAS the results toward a specific region.
- Distance matrix API
- Directions API
  - Quiz: Starting in Florence, Italy, calculate the optimized route to visit Bologna, Genoa, and Venice, stopping in Milan. We used the distance API with the `waypoints` parameter and `optimize:true`.
- The roads API allows snapping to roads with `snapToRoads`.
- Searching for places of interest within the map: See [ud864-master/Project\\_Code\\_12\\_FasterIsBetterPlacesAutocompletePart2.html](#)
- [Places library](#):
  - [Place details requests](#) return detailed information about places, including hours of operation and user reviews.
  - Each place has its own unique [Place ID](#).
  - [Place IDs](#) can be passed like lat lng into the Places web service database.

```
https://maps.googleapis.com/maps/api/place/details/json?  
placeid=PLACE_ID&key=YOUR_API_KEY
```

- The JSON returned includes operating hours (including `open_now: true`) and other details.
- Geolocation API
  - The geolocation API can be used to retrieve geolocation info for a device.

## ’ Lesson 09. Using the APIs in practice

## ’ Feedback

---

These lessons were "part 4" of the Udacity Full Stack Web Developer Nanodegree program. As usual, the lessons didn't really teach me what I needed to know, especially not for Knockout. It was frustrating to be working with Knockout, which is basically like an inferior predecessor to React. Udacity may have held off on including React here because it's a lot to learn, and because they want more customers for their React Nanodegree program. I also found it amusing to look at comments for the lesson videos on YouTube. "You call this a tut? It sucks." I'm inclined to agree.

# Authentication and Authorization

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

br3ndonland

[Authentication & Authorization: OAuth](#)

## Table of Contents

---

- [Table of Contents](#)
- [Lesson 1. Authentication and Authorization](#)
  - [1.01. Intro](#)
  - [1.02. Course Map](#)
  - [1.03. Authentication](#)
  - [1.04. Quiz: Authentication is hard](#)
  - [1.05. Quiz: Implementing Authentication is Hard](#)
  - [1.06. Using Third Party Auth Providers](#)
  - [1.07. Authorization](#)
  - [1.08. Authentication without Authorization](#)
  - [1.09. Auth providers](#)
  - [1.10. Pros and Cons using Third Party](#)
  - [1.11](#)
  - [1.12 Quiz: Follow the Flow](#)
  - [1.13. Outro](#)
  - [Feedback on Lesson 1](#)
- [Lesson 2. OAuth Flows and Google Sign-In](#)
  - [2.01. Intro to Lesson 2](#)
  - [2.02. Types of Flow](#)
  - [2.03. Google+ Auth for server side apps](#)
  - [2.04. Step 0 Get initial app running](#)
  - [2.05. Quiz: Step 1 Create Client ID & Secret](#)
  - [2.06. Quiz: Step 2 Create anti-forgery state token](#)
  - [2.07. Quiz: Step 3 Create login page](#)
  - [2.08. Quiz: Step 4 Make a Callback Method](#)
  - [2.09. Quiz: Step 5 GConnect](#)
  - [2.10](#)
  - [2.11. Quiz: Step 6 Disconnect](#)

- 2.12. Quiz: Step 7 Protecting Pages
- 2.13. Wrap-up
- Feedback on Lesson 2
- Lesson 3. Local permission system
  - 3.01. Lesson 3 Introduction
  - 3.02. Implementing a Local Permission System
  - 3.03. Quiz: Updating the User Model
  - 3.04. Creating a New User
  - 3.05. Quiz: Obtaining Credentials of an Existing User
  - 3.06. Quiz: Protect Menu Pages
  - 3.07. Wrap-Up Lesson 3
  - Feedback on Lesson 3
- Lesson 4. Facebook OAuth
  - 4.01. Adding Additional OAuth Providers
  - 4.02. Registering your App with Facebook
  - 4.03. Client-Side Login with Facebook SDK
  - 4.04. Quiz: Updating login.html
  - 4.05. Update project.py (part I)
  - 4.06. Update project.py (part II)
  - 4.07. Quiz: Updating project.py Code
  - 4.08. Exploring other OAuth2.0 Providers
  - 4.09. Outro
  - Feedback on Lesson 4
- General OAuth course feedback

## Lesson 1. Authentication and Authorization

---

### 1.01. Intro

Lorenzo Brown is teaching this course.

Instructor notes

[Learn JavaScript](#)

[Learn Ajax](#)

[Learn jQuery](#)

Set up your [Google](#) account.

Set up your [Facebook](#) account.

### 1.02. Course Map

Four lessons again:

1. Authentication and Authorization
2. Google sign-in for oauth2
3. Local permission system to protect each user's data from other users
4. Facebook as another oauth2 provider

The material extends the restaurant menu app.

The course code exists, confusingly, in two repos: [OAuth2.0](#) and [ud330](#). The code in ud330 is formatted a little better than the OAuth repo.

Instructor notes

Check out Udacity's awesome course on [Applied Cryptography](#) if you are interested in learning more about these concepts

[Here](#) is a blog on creating your own auth system with Flask.

The instructor notes in Lesson 1.02 Authentication and Authorization link to [lepture's blog post about Flask-OAuthlib](#). He was the creator of Flask-OAuthlib and has now replaced it with the [announcement of Authlib](#). We don't even use Flask-OAuthlib in this course, so I felt like this link was more distracting than helpful.

## 1.03. Authentication

## 1.04. Quiz: Authentication is hard

Most important:

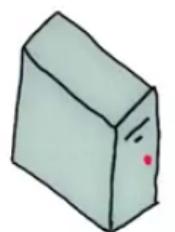
- Strong passwords
- Strong encryption
- Secure communication

Other security features:

- Password storage
- Two factor authentication
- Password recovery
- Man-in-the-middle attack protection

## 1.05. Quiz: Implementing Authentication is Hard

Should the following features be implemented client-side, server-side, or both?



Server



making sure users always have strong passwords

Secure password storage

protection against man-in-the-middle attacks

▶ 0:51 / 0:52

CC YouTube

Instructor notes

Read more about:

[App for Checking password security](#)

[Password Storage - Hashing vs. Encrypting](#)

[Man-in-the-Middle Attacks](#)

## 1.06. Using Third Party Auth Providers

User authentication frameworks have already been developed, so we don't have to reinvent the wheel with each application.

## 1.07. Authorization

Authentication is verifying identity, authorization is access permissions.

## 1.08. Authentication without Authorization

I didn't get the quiz on the first few tries, and there were too many options to know where I was going wrong, so I just moved on.

## 1.09. Auth providers

OAuth

## 1.10. Pros and Cons using Third Party

People may be uncomfortable giving your app Google permissions because they may not want to give your app the ability to change unrelated aspects of their Google accounts. Establishing appropriate authorization permissions is key.

Google created an [OAuth Playground](#) to allow users to test their OAuth builds. We didn't actually learn to use the OAuth Playground.

## ’1.11

## ’1.12 Quiz: Follow the Flow

## ’1.13. Outro

## ’Feedback on Lesson 1

Helpful to distinguish between authentication and authorization, and think about the pros and cons of third party sign in.

# ’Lesson 2. OAuth Flows and Google Sign-In

---

## ’2.01. Intro to Lesson 2

## ’2.02. Types of Flow

OAuth flows

Flow refers to how information is exchanged between the client, server, and OAuth provider.

- Client-side authentication is performed by JavaScript from the user's browser. This is mostly for "single-page, browser-based web applications." Mobile devices can have full authentication functionality. It is quick and easy, but places a lot of trust in the browser or mobile device, and prevents the server from making API calls on behalf of the user.
- Server-side authentication can obtain an access token that allows the server to make API requests on behalf of the user.
- Google+ uses a hybridized flow. Client-side authentication is performed, but the server can still make API calls on behalf of the client. It is difficult for unauthorized users to obtain access codes.

Check out this [blog](#) that explains some of the flow options using OAuth 2.0

## ’2.03. Google+ Auth for server side apps

[Google OAuth2 documentation](#)

 [Google hybrid OAuth flow](#)

## ’2.04. Step 0 Get initial app running

- We used Vagrant again here.
- I forked the [OAuth repo](#) and cloned it into the vagrant directory. The material extends the restaurant menu app.
- Logged into vagrant as usual.
- We will be using Flask again here. Also see Lesson 08. Frameworks (from free course [Full Stack Foundations](#))
- Started up the Flask web server:

```
$ cd /vagrant/OAuth2.0
vagrant@vagrant:/vagrant/OAuth2.0$ python database_setup.py
vagrant@vagrant:/vagrant/OAuth2.0$ python lotsofmenus.py
added menu items!
vagrant@vagrant:/vagrant/OAuth2.0$ python project.py
*Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
*Restarting with stat
*Debugger is active!
*Debugger PIN: 769-024-554
```

## 2.05. Quiz: Step 1 Create Client ID & Secret

The client ID and secret allow the app to communicate with Google.

The video walks through Google API project creation.

### Instructor notes

Errata:

Google has changed the user interface for obtaining OAuth credentials since this video was created. The functionality is the same, but the appearance is somewhat different from what's depicted here.

1. Go to your app's page in the [Google APIs Console](#)
2. Choose Credentials from the menu on the left.
3. Create an OAuth Client ID.
4. This will require you to configure the consent screen, with the same choices as in the video.
5. When you're presented with a list of application types, choose Web application.
6. You can then set the authorized JavaScript origins, with the same settings as in the video.
7. You will then be able to get the client ID and client secret.

You can also download the client secret as a JSON data file once you have created it.

After creating the client ID and secret, set the "Authorized JavaScript Origins" to:

<http://localhost:5000>

## ’2.06. Quiz: Step 2 Create anti-forgery state token

- This helps verify the authenticated user and prevent attacks.
- We import Flask's version of sessions, and name it `login_session`:

```
from flask import session as login_session
import random, string
```

- `login_session` is a dictionary. We can store values there during the session.
- We then create the login page app route:

```
@app.route('/login')
def login():
    """App route function to log in and generate token."""
    state = ''.join(random.choice(string.ascii_uppercase + string.digits)
                   for x in range(32))
    login_session['state'] = state
    return 'The current session state is {}'.format(login_session['state'])
```

- Session works like a dictionary and can store values.
- `random` and `string` are used to generate a random string, stored in an object with the name `state` here.
- The `xrange` in Python 2 is replaced by `range` in Python 3.
- The `state` token is stored in the `login_session` object.
- To test this, we can just return a string in the browser with the token. Later, we will render the `login.html` template.

The code for this step is in [another repo](#) with code that is formatted a little better than the OAuth repo.

Got the authentication working.

<http://0.0.0.0:5000/login>

```
The current session state is 7T0F4XSE3DUWIMW39IV3Y35QR1TT8FS9
```

## ’2.07. Quiz: Step 3 Create login page

I followed along by looking at /vagrant/ud330/Lesson2/step3/templates/login.html.

## ’2.08. Quiz: Step 4 Make a Callback Method

Lorenzo adds JavaScript code to the login.html page inside script tags. He uses jQuery to make an AJAX call, which we haven't learned yet (should be in part 4 I think).

- `url: '/gconnect?state={{STATE}}'`, passes the state token as an argument to protect against CSRF.
- `processData: false`, tells jQuery not to process the response into a string.
- A successful login returns a success message to the user.

Instructor notes:

View the [login.html](#) page now with the added callback function.

At the 1:00 minute mark, Lorenzo adds a template variable called STATE.

In order to populate that variable when the template is rendered, you need to pass it to the `render_template` function:

```
# Create anti-forgery state token
@app.route('/login')
def showLogin():
    state = ''.join(random.choice(string.ascii_uppercase + string.digits)
                    for x in xrange(32))
    login_session['state'] = state
    # return "The current session state is %s" % login_session['state']
    return render_template('login.html', STATE=state)
```

## 2.09. Quiz: Step 5 GConnect

- Add new imports

```
# Import OAuth modules for user authentication
from oauth2client.client import flow_from_clientsecrets, FlowExchangeError
from flask import make_response
import json
import requests
```

- Go back to the [Google API console](#) and download Google API credentials as JSON.
  - Why not just do this in the first place? Lorenzo needs to be more organized.
- Rename the file \*client\_secrets.json- and store in same directory as main application file.
- Create a server-side helper function to store the client secret as an object

```
CLIENT_ID = json.loads(open('client_secrets.json', 'r')
.read())['web']['client_id']
```

- Create GConnect app route (see code in project.py)

Instructor notes

The code from this video can be found [here](#).

**IMPORTANT:** Depending on the version of Flask you have, you may or may not be able to store a credentials object in the `login_session` the same way that Lorenzo does. You may get the following error:

OAuth2Credentials object is not JSON serializable

What should you do to fix this? There are three options:

1. Rather than storing the entire `credentials` object you can store just the access token instead. It can be accessed using `credentials.access_token`.
2. The OAuth2Credentials class comes with methods that can help you. The `.to_json()` and `.from_json()` methods can help you store and retrieve the credentials object in json format.
3. Update your versions of Flask, *and* to match Lorenzo's. Use the following commands:

```
pip install werkzeug==0.8.3
pip install flask==0.9
pip install Flask-Login==0.1.3
```

**Note:** If you get a permissions error, you will need to include `sudo` at the beginning of each command. That should look like this: `sudo pip install flask==0.9`

Go to the GoogleDevConsole> API & Auth> Credentials>Select your app> Authorized Redirect URIs and add the following

URIS: <http://localhost:5000/login> and <http://localhost:5000/gconnect> You may have to change the port number depending on the port number you have set your app to run on.

Example code: <https://github.com/udacity/ud330/blob/master/Lesson2/step5/project.py>

Have questions? Head to the [forums](#) for discussion with the Udacity Community.

## 2.10

### 2.11. Quiz: Step 6 Disconnect

See code in project.py

### 2.12. Quiz: Step 7 Protecting Pages

The quiz was about which pages should be publicly visible vs. private. Obviously any pages on which you are modifying the site should be private.

Pages are made private by adding login as a condition.

For example, on the new restaurant page:

```
@app.route('/restaurant/new/', methods=['GET', 'POST'])
def newRestaurant():
```

```
if 'username' not in login_session:  
    return redirect('/login')
```

## 2.13. Wrap-up

### Feedback on Lesson 2

Lots of PEP 8 errors in the [OAuth2.0](#)/project.py file. I shouldn't have to lint the instructors' code.

For some reason there are two repos for this course, [OAuth2.0](#) and [ud330](#).

[\(Back to TOC\)](#)

## Lesson 3. Local permission system

---

### 3.01. Lesson 3 Introduction

The local permission system is an additional measure to ensure authenticity. It stores login credentials in a `login_session` object to control the experience by user.

### 3.02. Implementing a Local Permission System

The database should have a user table.

Update `*database_setup.py`- to include a user table:

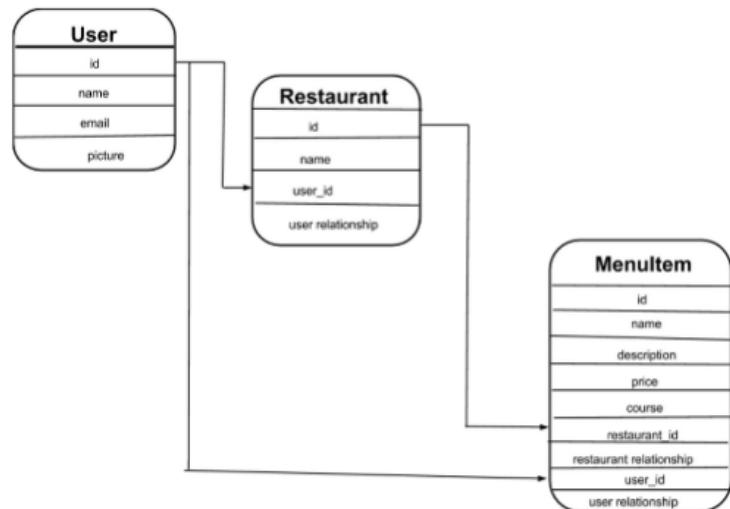
```
class User(Base):  
    __tablename__ = 'user'  
  
    id = Column(Integer, primary_key=True)  
    name = Column(String(250), nullable=False)  
    email = Column(String(250), nullable=False)  
    picture = Column(String(250))
```

Add a foreign key to the `restaurant` and `menu_item` tables to reference the user table so that only the creators of the items can edit them:

```
user_id = Column(Integer, ForeignKey('user.id'))
```

## Update the Database

In your database\_setup.py file add the necessary code modifications such that your database models the layout to the right.



Check this box to continue to the solution video.

### 3.03. Quiz: Updating the User Model

### 3.04. Creating a New User

User helper functions:

The `createUser()` function will look up users in the database table based on their email addresses used for login.

There are three helpful methods here:

1. `createUser()` adds a user to the database.
2. `getUserInfo()` returns a user object associated with the user ID number.
3. `getUserID()` takes an email address as input and returns the user ID number if the user is stored in the database.

### 3.05. Quiz: Obtaining Credentials of an Existing User

Store in `user_id` objects.

### 3.06. Quiz: Protect Menu Pages

Lorenzo creates separate templates for public viewers and logged-in users.

He adds the local permission system to the app routes requiring login:

```
@app.route('/restaurant/new/', methods=['GET', 'POST'])
def newRestaurant():
    if 'username' not in login_session:
```

```
    return redirect('/login')
if request.method == 'POST':
    newRestaurant = Restaurant(name=request.form['name'])
    session.add(newRestaurant)
    flash('New Restaurant %s Successfully Created' % newRestaurant.name)
    session.commit()
    return redirect(url_for('showRestaurants'))
else:
    return render_template('newRestaurant.html')
```

## Instructor notes

Click to view the [publicmenu.html](#) and [publicrestaurants.html](#) templates.

Note:

In order to view the profile picture of the creator of each menu, add code to your project.py file to query the creator:

```
creator = getUserInfo(restaurant.user_id)
```

then pass in the creator object when you render the template:

```
return render_template('publicmenu.html', items = items, restaurant = restaurant, creat
```

Inside your templates you can now use `{{creator.picture}}` and `{{creator.name}}` to share the name and picture of each restaurant creator.

## 3.07. Wrap-Up Lesson 3

### Feedback on Lesson 3

Brief lesson, no feedback

[\(Back to TOC\)](#)

## Lesson 4. Facebook OAuth

Facebook login would probably be more effectively implemented with JavaScript. See the [Facebook login documentation](#).

### 4.01. Adding Additional OAuth Providers

### 4.02. Registering your App with Facebook

### 4.03. Client-Side Login with Facebook SDK

## › 4.04. Quiz: Updating login.html

## › 4.05. Update project.py (part I)

The instructor notes warn against a change to the Facebook API URL, but the lesson code was updated.

### Instructor notes

**Important:** Facebook has updated their API since this course was released. This affects code that is shown in this video at 0:32 on line 194. Instead of...

```
url = 'https://graph.facebook.com/v2.2/me?%s' % token
```

... the code must be...

```
url = 'https://graph.facebook.com/v2.8/me?access_token=%s&fields=name,id,email' % token
```

Here is some more information about the change:

*An app can make calls to the version of the API that was the latest available when the app was created, as well as any newer, un-deprecated versions launched after the app is created.*

Here is the main change from version 2.2 that will impact Full-Stack students:

*Fewer default fields for faster performance: To help improve performance on mobile network connections, we've reduced the number of fields that the API returns by default. You should now use the ?fields=field1,field2 syntax to declare all the fields you want the API to return.*

They say "we've reduced the number of fields that the API returns by default." This is important because one of the fields that is no longer returned by default is the `email` field. You must now explicitly request email to be returned (along with the other fields you want).

The API version 2.2 that was used in this course has been officially deprecated by Facebook as of March 25th, 2017. Requests made to that version will no longer work. Please use the most current version of the Facebook API with your project code.

Have questions? Head to the [forums](#) for discussion with the Udacity Community.

## › 4.06. Update project.py (part II)

## › 4.07. Quiz: Updating project.py Code

### Instructor notes

In order to properly log out of Facebook, the access token needs to be sent along with the facebook id in fbdisconnect. Please see the new code in the fbconnect and fbdisconnect methods in the [project.py](#) file.

View the code for this activity on [GitHub](#).

## › 4.08. Exploring other OAuth2.0 Providers

## › 4.09. Outro

Instructor notes

The final code for this project can be found on [GitHub](#).

Here is the [Official OAuth 2.0 website](#).

## › Feedback on Lesson 4

Another very brief lesson

## › General OAuth course feedback

---

- I was disappointed in this course.
- The code is outdated and poorly formatted.
  - Python 2 instead of 3
  - HTTP requests use `httpplib2` instead of `requests`
  - Code is not PEP 8 compliant.
  - Code mixes spaces and tabs.
  - JavaScript camelCase used instead of underscores for function names.
  - JavaScript camelCase even used for CSS classes! For example, in `login.html`, `<div id="signinButton">`. CSS classes are specified in lowercase, with hyphens in between.
  - Doesn't even have a proper Markdown-formatted README
  - Code exists, confusingly, in two repos: [OAuth2.0](#) and [ud330](#). The code in ud330 is formatted a little better than the OAuth repo.
- **GitHub repos are not maintained. Pull requests are ignored.** The GitHub repos are an opportunity to enrich the student experience by teaching good open source practices, but Udacity is not doing this.
- The instructor Lorenzo's method of storing many different versions of the files, and constantly going back and updating them, was confusing and time-consuming.
- I didn't find the lessons very useful until I started the item catalog project. The code didn't mean much to me until I was implementing it in my own project. I almost felt like I could have started the project before watching the lessons.

# Linux server configuration

---



Udacity Full Stack Web Developer Nanodegree program

Brendon Smith

[br3ndonland](#)

From free [Configuring Linux Web Servers Udacity course](#)

## Lesson notes

---

### Comments

- From free [Configuring Linux Web Servers Udacity course](#).
- The lessons for this section were brief. There was only one course and I watched the videos in about an hour.

### Distributions

- RedHat: Large enterprise/corporate customers
- Ubuntu: Ease of use on servers, desktops, laptops
- Linux Mint: Desktop users with proprietary media support
- CoreOS: Clustered, containerized deployment of apps
- Debian: The instructor Mike Wales also mentioned that Debian is "rock solid."
- Check out [distrowatch](#).

### Vagrant Commands

We're now ready to get started working within our Linux virtual machine. If your download hasn't completed from the initial setup, go ahead and take a break and come back when that has completed. You won't be able to make further progress until the virtual machine is up and running as much of the course will take place within this environment.

Before we access our machine, let's quickly review a few commands that vagrant provides to make managing your virtual machines much simpler. Remember, your vagrant machine lives within this specific folder on your computer so make sure you're within that same folder you created earlier; otherwise these commands won't work as expected.

1. Type `vagrant status`
  - This command will show you the current status of the virtual machine. It should currently read "default running (virtualbox)" along with some other information.
2. Type `vagrant suspend`

- This command suspends your virtual machine. All of your work is saved and the machine is put into a “sleep mode” of sorts. The machines state is saved and it’s very quick to stop and start your work. You should use this command if you plan to just take a short break from your work but don’t want to leave the virtual machine running.

### 3. Type `vagrant up`

- This gets your virtual machine up and running again. Notice we didn’t have to re-download the virtual machine image, since it’s already been downloaded.

### 4. Type `vagrant ssh`

- This command will actually connect to and log you into your virtual machine. Once done you will see a few lines of text showing various performance statistics of the virtual machine along with a new command line prompt that reads  
`vagrant@vagrant-ubuntu-trusty-64:~$`

Here are a few other important commands that we’ll discuss but you **do not** need to practice at this time:

#### 1. `vagrant halt`

- This command halts your virtual machine. All of your work is saved and the machine is turned off - think of this as “turning the power off”. It’s much slower to stop and start your virtual machine using this command, but it does free up all of your RAM once the machine has been stopped. You should use this command if you plan to take an extended break from your work, like when you are done for the day. The command `vagrant up` will turn your machine back on and you can continue your work.

#### 2. `vagrant destroy`

- This command destroys your virtual machine. Your work is not saved, the machine is turned off and forgotten about for the most part. Think of this as formatting the hard drive of a computer. You can always use `vagrant up` to relaunch the machine but you’ll be left with the baseline Linux installation from the beginning of this course. You should not have to use this command at any time during this course unless, at some point in time, you perform a task on the virtual machine that makes it completely inoperable.

## Apache servers

When Apache receives a request it has a number of ways it can respond. What you’ve seen thus far is the simplest method of operation, Apache just returns a file requested or the `index.html` file if no file is defined within the URL.

But, Apache can do so much more! You’ll now configure Apache to hand-off certain requests to an application handler - `mod_wsgi`. The first step in this process is to install `mod_wsgi`: `sudo apt-get install libapache2-mod-wsgi`.

You then need to configure Apache to handle requests using the WSGI module. You’ll do this by editing the `/etc/apache2/sites-enabled/000-default.conf` file. This file tells Apache how to respond to requests, where to find the files for a particular site and much more. You can read up on everything this file can do within the [Apache documentation](#).

For now, add the following line at the end of the `<VirtualHost *:80>` block, right before the closing `</VirtualHost>` line: `WSGIScriptAlias / /var/www/html/myapp.wsgi`

Finally, restart Apache with the `sudo apache2ctl restart` command. You might get a warning saying "Could not reliably determine the server's fully qualified domain name". If you do, don't worry about it. Check out [this AskUbuntu thread](#) for a discussion of the cause of this message.

**WSGI** is a specification that describes how a web server communicates with web applications. Most if not all Python web frameworks are WSGI compliant, including [Flask](#) and [Django](#); but to quickly test if you have your Apache configuration correct you'll write a very basic WSGI application.

You just defined the name of the file you need to write within your Apache configuration by using the `WSGIScriptAlias` directive. Despite having the extension `.wsgi`, these are just Python applications. Create the `/var/www/html/myapp.wsgi` file using the command `sudo nano /var/www/html/myapp.wsgi`. Within this file, write the following application:

```
def application(environ, start_response):
    status = '200 OK'
    output = 'Hello Udacity!'

    response_headers = [('Content-type', 'text/plain'), ('Content-Length',
str(len(output)))]
    start_response(status, response_headers)

    return [output]
```

This application will simply print return `Hello Udacity!` along with the required HTTP response headers. After saving this file you can reload `http://localhost:8080` to see your application run in all its glory!