
LAPTOP SETUP

ICEBREAKER

CODE 101

Introduction to Software Development and Careers in Technology

ICEBREAKER

- Your name
- Why you’re here
- What you want to gain from this workshop

GOALS

- Get to know the life of a professional software developer.
- Understand the frameworks of modern websites.
- Code a complete website using HTML and CSS.
- Deploy your website on the Internet.
- Figure out whether coding is for you.

TODAY’S AGENDA

- 9:00 am Intros, Intro to the Modern Web and HTML
- 10:00 am Team Coding
- 12:00 pm Lunch and Alumnus talk
- 1:00 pm Troubleshooting
- 1:30 pm Intro to CSS and Team Coding
- 3:00 pm What is a “Software Developer”?
- 4:15 pm How to Share and Deploy Code
- 5:45 pm Dinner and Industry Speaker Talk
- 6:45 pm Presentations
- 7:30 pm Wrap-up and Survey

HOW YOU'LL LEARN

"Path" Learning:

- Leads you along
- Students are consumers of information
- Predictable outcomes
- Creates dependency
- The goal: an exchange of information

"Sandbox" Learning:

- Fosters exploration
- Students are co-creators of their learning experience
- Wide range of outcomes
- Creates autonomy
- The goal: learning and discovery

SKILLS FOR SANDBOX LEARNING

- Generating and selecting ideas: what do you want to learn now?
- Planning your learning: managing scope, finding resources
- Experimentation: keeping track of what you've tried, what's worked, and what hasn't
- Reflection: pausing every so often to tally what you've learned, and what new questions you have
- Finding help!

YOUR TOOLS

- Your new friends
- Your instructor and TAs
- The internet
- Slack

SLACK

YOUR TOOLS

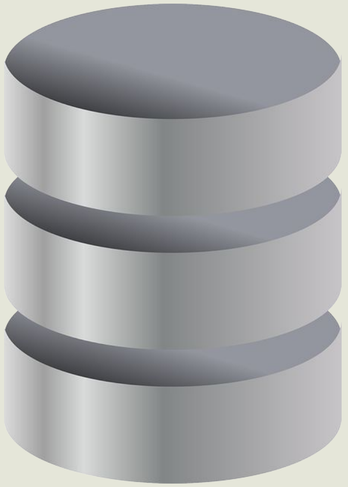
- Your new friends
- Your instructor and TAs
- The internet
- Slack
- Your feedback!

THE MODERN WEB

WEB APPLICATIONS

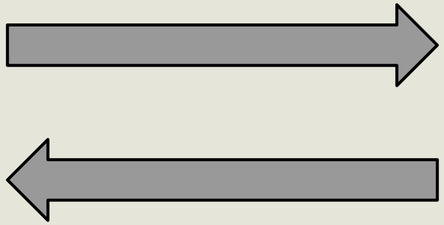
Client

Server

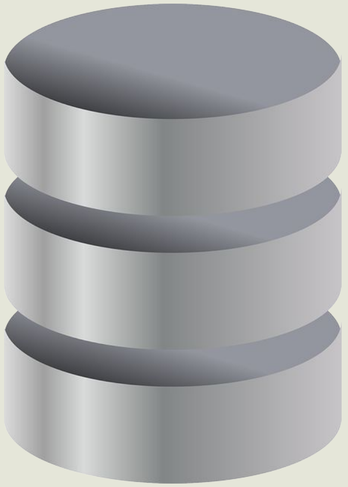


Database

Client



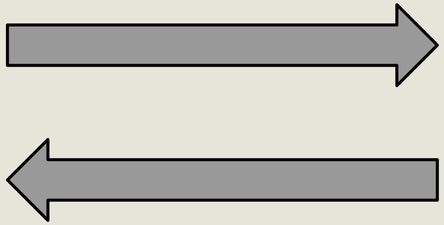
Server

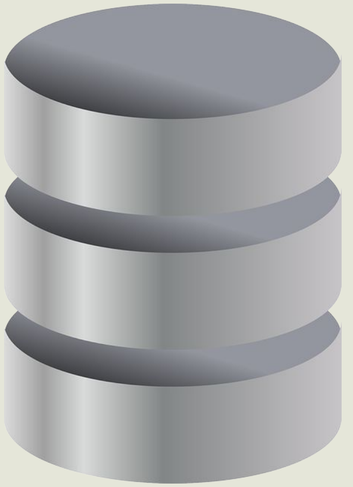


Database

Front End

Back End





Client

Server

Database

Front End

Back End

HTML/CSS/JS

iOS

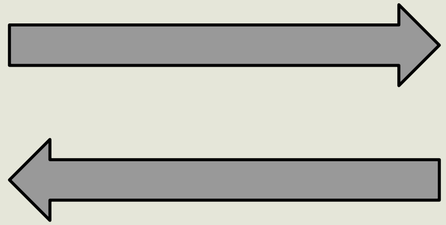
Python

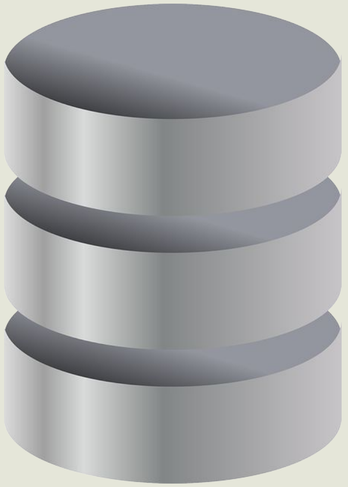
Ruby

JavaScript
("full stack")

MongoDB

MySQL





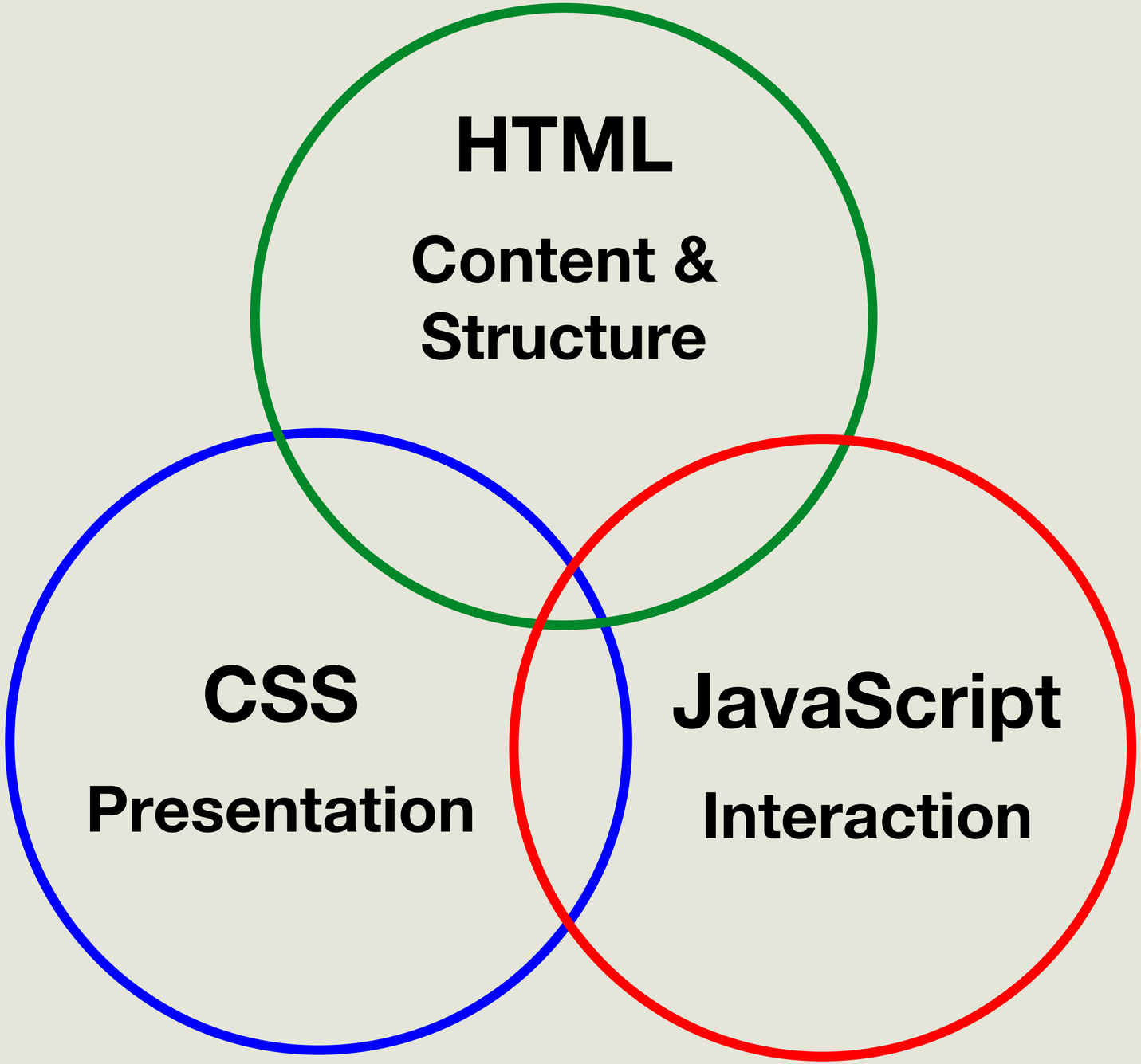
Client

Server

Database

APPLICATION EXAMPLES

THE FRONT END


HTML
Content &
Structure
CSS
Presentation
JavaScript
Interaction

HTML

- Tells the browser **what** the content is
    - Text
    - Headlines
    - Images
    - Links
- Communicates how that content is **organized**
    - Importance
    - Blocks

Earthquake Preparedness

North America

Look around places where you spend time. Identify safe place such as under a sturdy piece of furniture or against an interior wall in your home, office or school so that when the shaking

HTML - WHAT IT LOOKS LIKE

```
<body>

    <header>
        <h1>Earthquake Preparedness</h1>
    </header>

    <nav>
        <ul>
          <li><a href="">North America</a></li>
          <li><a href="">East Asia</a></li>
          <li><a href="">Southeast Asia</a></li>
        </ul>
    </nav>
```

CSS

- Tells the browser how the content should **look**
    - Fonts
    - Colors
    - Placement of content
    - Responsiveness

Earthquake Preparedness

North America

Look around places where you spend time. Identify safe place such as under a sturdy piece of furniture or against an interior wall in your home, office or school so that when the shaking starts, you Drop to the ground, Cover your head and neck with your arms, and if a safer place is nearby, crawl to it and Hold On.

Earthquake Preparedness

North America
East Asia
Southeast Asia

North America

Look around places where you spend time. Identify safe place such as under a sturdy piece of furniture or against an interior wall in your home, office or school so that when the shaking starts, you Drop to the ground, Cover your head and neck with your arms, and if a safer place is nearby, crawl to it and Hold On.

When the shaking stops, look around. If there is a clear path to safety, leave the building and go to an open space away from damaged areas.

CSS - WHAT IT LOOKS LIKE

```
nav li a {
    display: block;
    height: 25px;
    border-right: 1px solid #cccccc;
    padding: 12px 18px 8px 18px;
    vertical-align: middle;
}
a {
    color: black;
}
```

JavaScript

- Tells the browser what it should **do** when certain things happen
    - The page loads
    - The user clicks on a button
    - A form is submitted
    - The page needs to fetch extra data

Preparedness

Help earthquake victims in Nepal!

Amount:

Donate Now

JAVASCRIPT - WHAT IT LOOKS LIKE

```
$(function() {
  $("#donate").on("submit", function(e) {
    e.preventDefault();

    var amount = $("input[name=amount]").val();

    if (amount) {
      $.post("donate", {amount: amount},
        function(res) {
          var donateRes = res.msg;
          $("#donateRes").text(donateRes);
        });
```

YOUR APP: GETTING STARTED

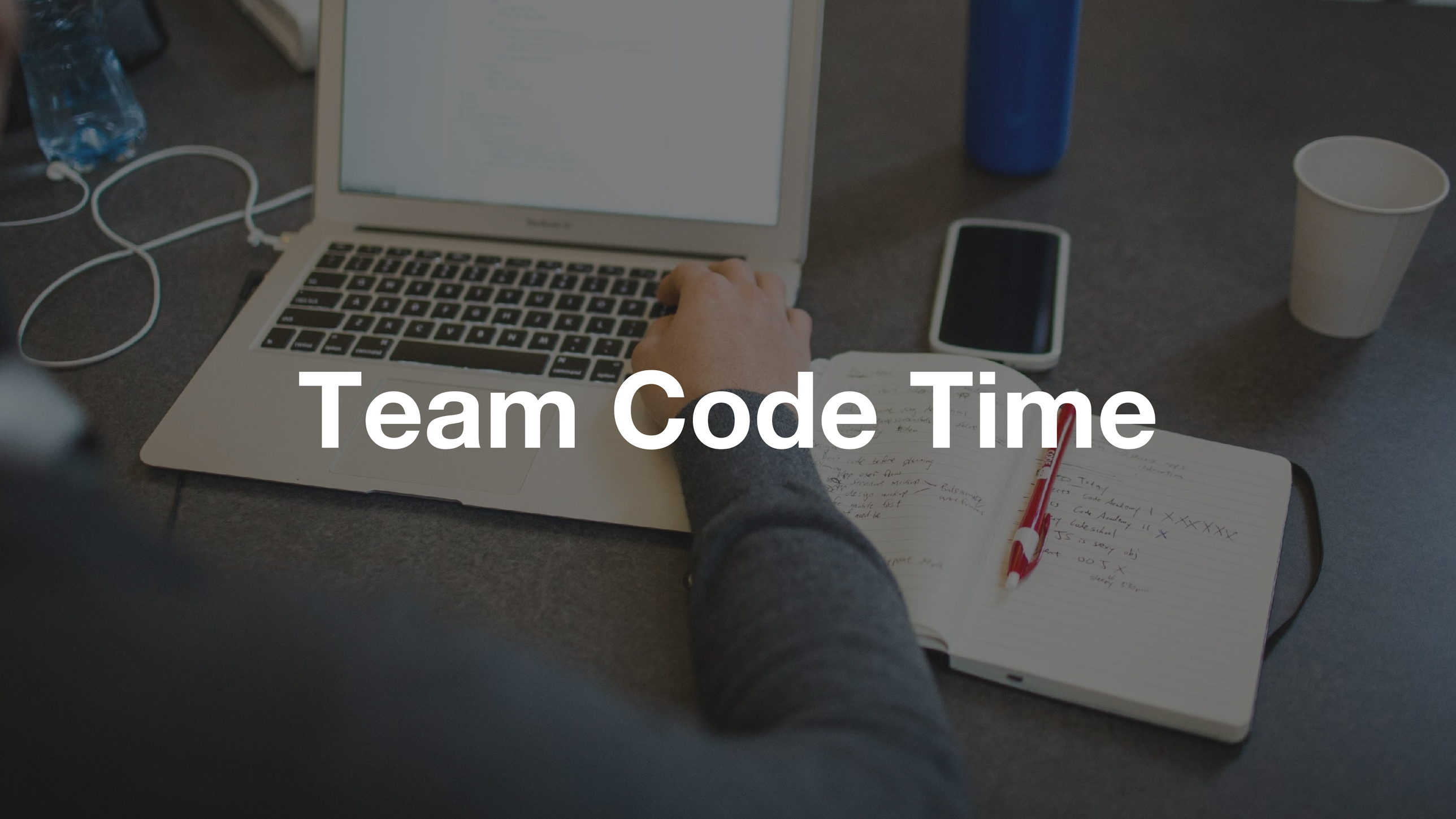
HOW TO START

- Identify your content. What do you need to communicate?

- Organize your content
  - Headlines
  - Paragraphs
  - Blocks - what content should be together?
  - Links - where do they go?

- Start sketching

- Eventually, write code


Team Code Time

OUTLINE YOUR APP

- Decide on a cause (fictitious, or fictitious name)
- Pitch: your topic, need it addresses, and pages
- Each pair takes one page and:
  - Outlines that page’s content (write it down)
  - Chooses a template
  - Organizes the content into the template of their choosing
  - Sketches it out!
- Make a slack channel for your team

HTML

HOW HTML WORKS

- “**Tags**” tell the browser what the content is
- Browsers display the content according to the rules of the tags
- This is called “marking up” the content
- Hyper Text Markup Language

TAGS

- Tags surround the content they describe. Like this:
    - My really cool paragraph!
- There are **beginning** tags:
- And **ending** tags:
- Browsers don’t care whether you use upper case or lower case. (Lower case is easier for humans to read.)

ANATOMY OF A WEB PAGE

```
<html>
    <head>
        <title>Welcome to Code Fellows</title>
    </head>
    <body>
        <!-- content of web page goes here -->
    </body>
</html>
```

ORGANIZE YOUR CONTENT

- HTML has tags that are applied to blocks of content.
    - <header></header>
    - <nav></nav>
    - <section></section>
    - <article></article>
    - <aside></aside>
    - <footer></footer>

WHITE SPACE

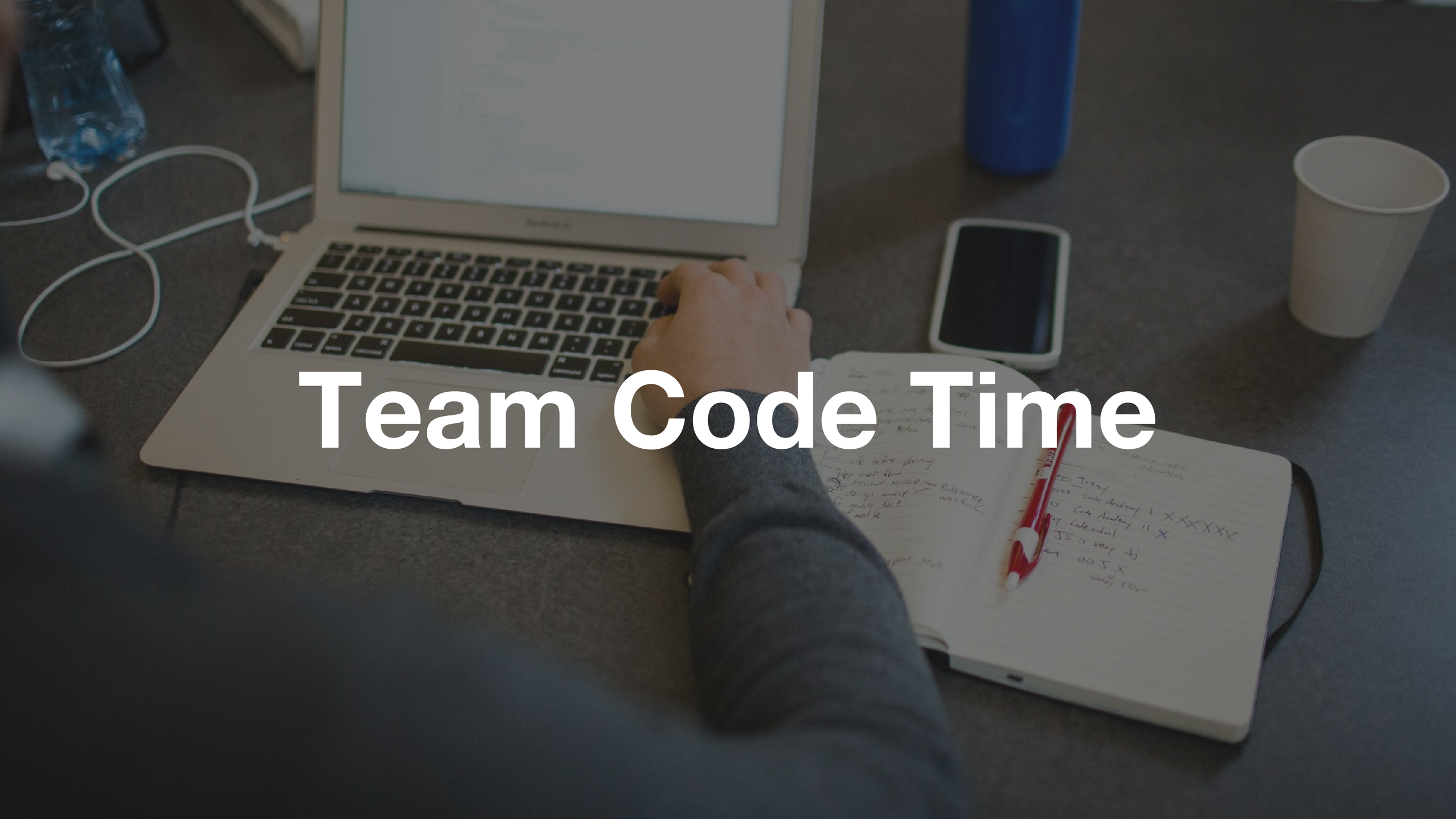
- White space = tabs, spaces, returns
- All white space characters show up as one space.
- Multiple white space characters show up as one space.
- We can use different tags to make:
    - Double-spaced lines My paragraph
    - Returns
    - Lists and

HEADINGS

- Tells the browser how important a line of text is
- <h1>I’m the biggest heading!</h1>
- <h6>And I’m the smallest heading.</h6>
- They are automatically shown in bold, in a bigger size, and with a line break after them.

ATTRIBUTES

- Tags can have **attributes** that give more information:
    - Where to find an image file
    - The URL of a link
- Attributes have different names, but the syntax is the same:
    - <img src=”logo.jpg”>
    - About Us


Team Code Time

WRITE YOUR HTML

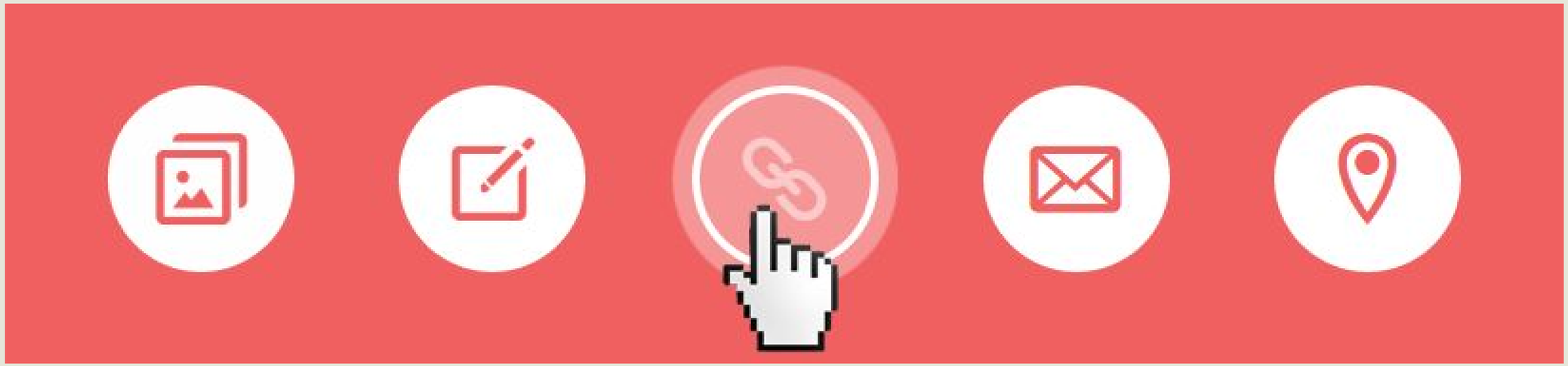
- In your pairs:
  - Create an html file with an appropriate name, like “about_us.html”. Save it in your projects folder.
  - One (and only one!) of your team’s files must be called “index.html”. Yes, the filename must be lowercase! Tip: don’t put spaces in your filenames.
  - Add <html>, <head> and <body> tags.
  - Inside the <body> tags, write your content.
  - Then, add sections, paragraphs, navs, headers, etc.
  - Use your team’s filenames in your nav bar links.
  - Use the wireframe you created (that sketch of your content) as a guide to create your html.

LUNCH

TROUBLESHOOTING

CSS

WHY



- Design and lay out your HTML elements. (The “look and feel”)
- Maintain consistent styles.

Image Source:
http://tympanus.net/Development/IconHoverEffects/

THE POWER OF CSS

- Zen Garden (Slack link)
- In 1 and 2, the HTML did not change. The CSS did.

1

2

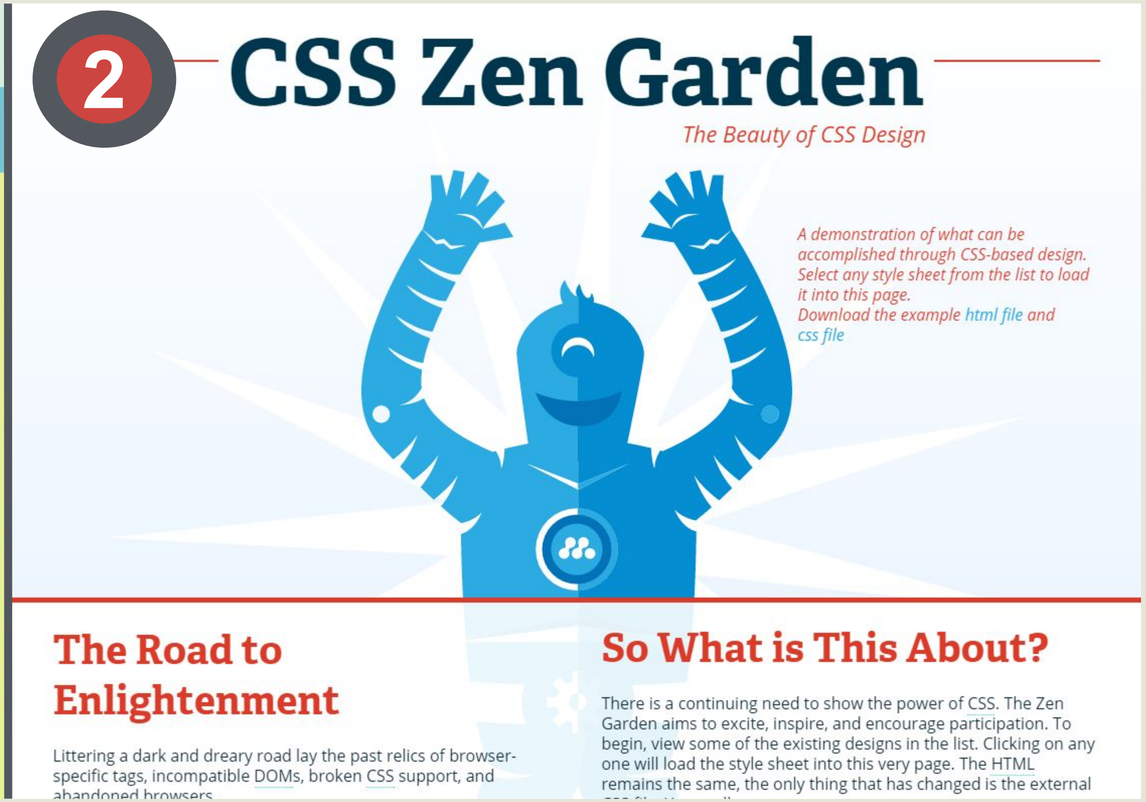


Image Source:
http://www.mezzoblue.com/zengarden/alldesigns/

WHAT TO EXPECT

- Those CSS Zen Garden examples? Those were created by advanced users of CSS. Your site will look something like this:

http://kbeame.github.io/

HOW CSS WORKS

- CSS = Cascading Style Sheets

- CSS works by:
  - **Selecting** elements to apply a **style** to
  - Defining what that **style** is: color, font, spacing, etc.

INCLUDING CSS IN YOUR HTML

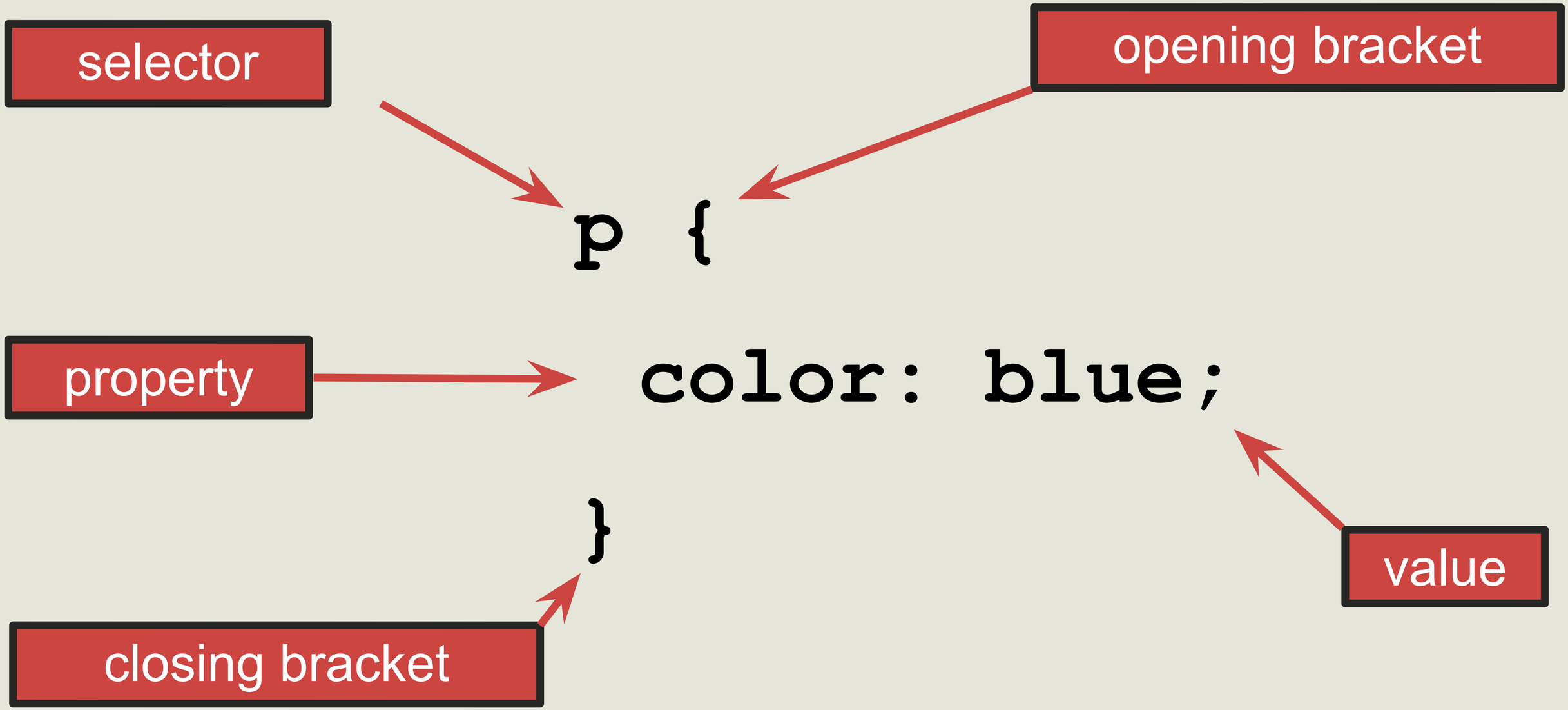
- CSS Styles can be written inline next to thing you want to style, in the header, or in an external file referenced in the header.
  - **inline - in the HTML’s body**

    link
  - **inline - in the HTML’s header**

    <style> a {color: #330066} </style>
  - **externally linked - in the HTML’s header (preferred)**

    <link rel="stylesheet" href="styles.css" type="text/css">

CSS: WHAT IT LOOKS LIKE



CSS: SELECTING BY TAG

- In your HTML
  - Content areas you want to style
    1. `<nav></nav>`
    2. `<li></li>`
    3. `<li><a></a></li>`

- In your CSS
  - Selecting content areas to style by **tag**
    1. `nav { }`
    2. `li { }`
    3. `li a { }`

CSS: SELECTING BY CLASS OR ID

- By **class**
  - Can be used multiple times within HTML.
  - The html: `<p class="buyNow">Buy now!</p>`
  - The css selector: `.buyNow`
- By **id**
  - Can only be used once within HTML.
  - The html: `<p id="buyNow">Buy now!</p>`
  - The css selector: `#buyNow`

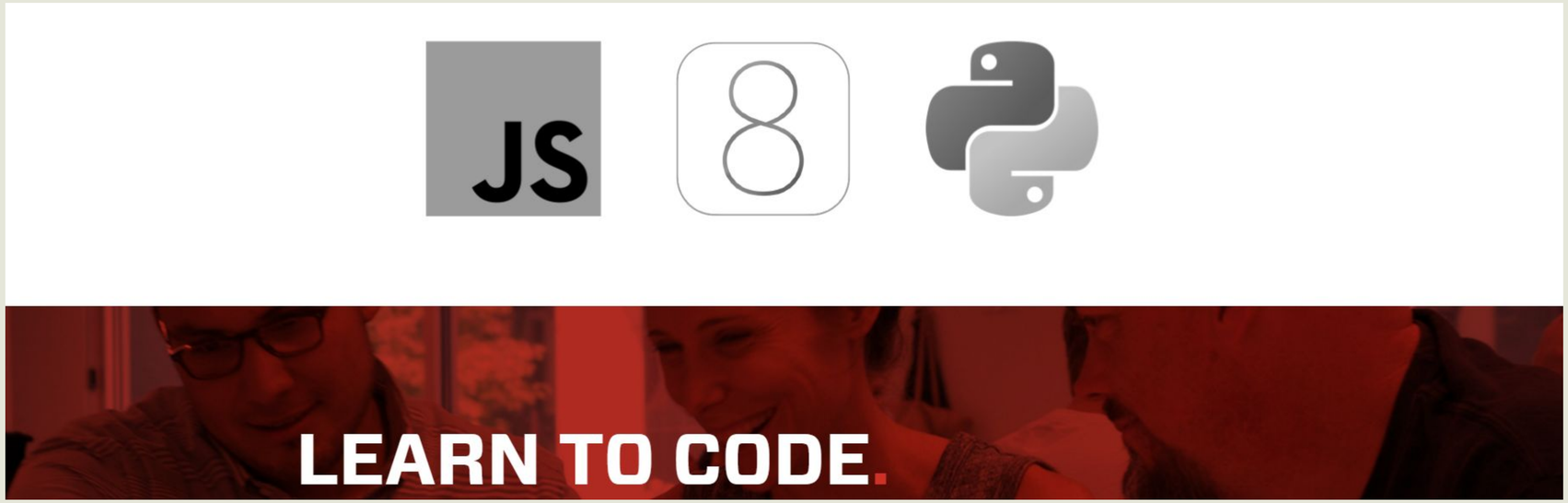
SOME CSS PROPERTIES

- padding: 5px;
- border: 2px solid black;
- color: crimson;
- background-color: beige;
- font-size: 14px;
- font-weight: bold;
- font-family: Arial, Helvetica, sans-serif;
- line-height: 24px;

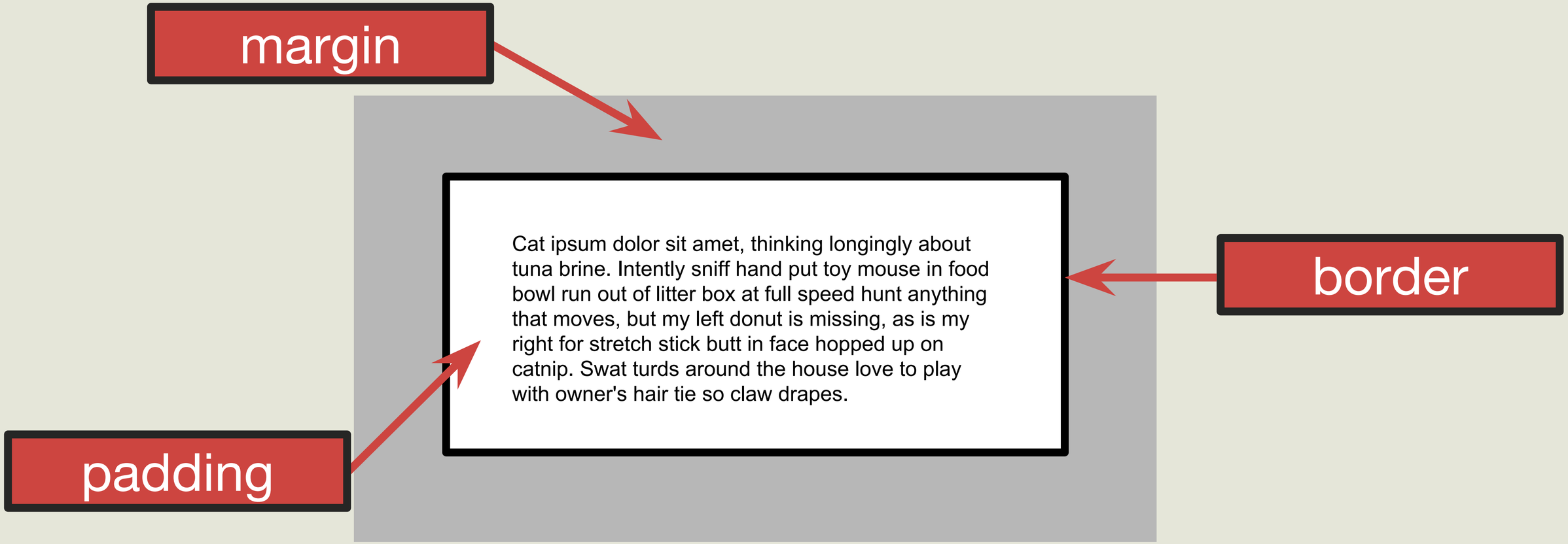
px = pixels

LAYOUT: THE BOX MODEL

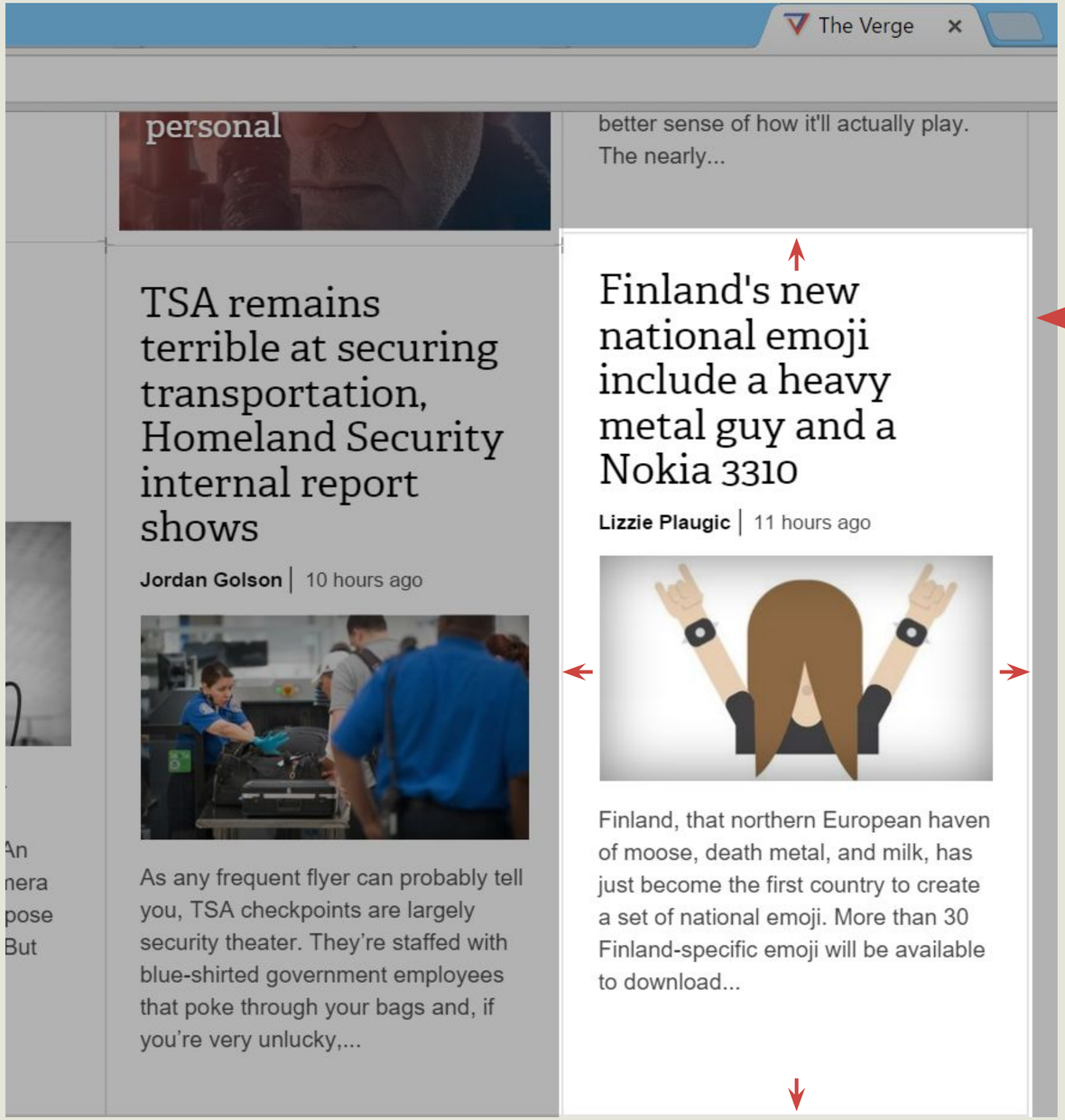
- It’s the **padding**, **border**, and **margin** separating your content from other areas of content on your page.
- Use these properties wisely to create negative space to improve your layout and user experience (UX).



LAYOUT: THE BOX MODEL (cont.)



LAYOUT: THE BOX MODEL (cont.)

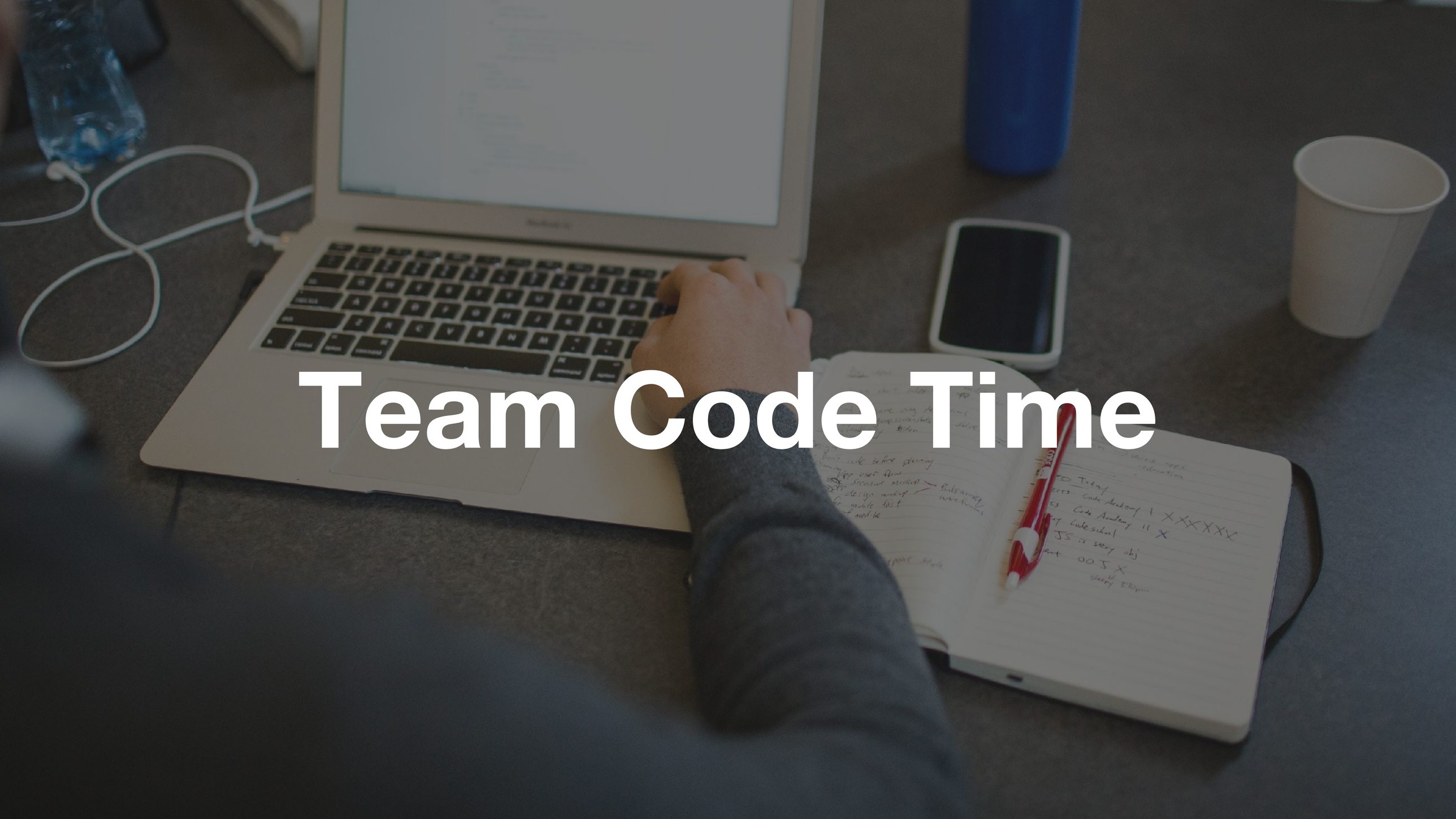


border: 1px;

padding: 19.5px;

margin: 0px;

Image Source:
http://theverge.com


Team Code Time

STYLE IT!

- Link one of the css files to your html file. Be sure to change that file's name to something unique, like "aboutus.css"
- Look at each of the styles and figure out which elements on the page they refer to
- Make some changes! Maybe you want to alter:
  - Text colors (color)
  - Background colors (background-color)
  - Link underlines (text-decoration)
  - Fonts (font-family)

the
DEVELOPER LIFE

WHAT'S IN A NAME?

- People who write code for a living go by many different names:
  - Web developer
  - Software developer
  - Front-end developer
  - Software engineer
  - UX Designer
- The names differ by stack, company, and where you spend most of your time (front end, back end, or both)

COMMONALITIES

- Regardless of where you work, developers usually:
  - Identify bugs or new features
  - Plan out how to fix the bug or make the new feature
  - Write code
  - Review code
  - Make sure code gets tested
  - Work with a team to design and implement solutions

A DAY IN THE LIFE OF A (FICTIONAL) TYPICAL DEVELOPER

A DAY IN THE LIFE

- 9:10 am: Get to work. See what happened overnight.
- 9:24 am: Grab some coffee/tea/morning beverage of choice.
- 9:30 am: Daily standup. You and your teammates share what you're working on, and what you need help with.
- 9:45 am: Help a teammate think through a problem.
- 10:11 am: Start on the problem you're working on. Search through the code base to see what the existing code does.
- 10:49 am: xkcd break! Feel smart because you understand it.

A DAY IN THE LIFE

- 10:52 am: Find some whiteboard space and start drawing out potential solutions.
- 11:21 am: Ask a teammate for their opinion. Ditch your drawings. Make new ones.
- 11:50 am: Talk through your latest drawing with your team. You all decide it’s the best approach.
- 12:15 pm: Lunch! Read some Hacker News.
- 12:45 pm: Someone brings back donuts. Do a dance of joy.

A DAY IN THE LIFE

- 12:49 pm: Start translating your drawings into code. Put on your headphones and get in the groove.
- 3:01 pm: Where’d the time go? Oops...the coffee maker broke.
  - Fix the coffee maker (startup).
  - Ask your office manager to fix the coffee maker (enterprise).
- 3:20 pm: Test your code. Find something that needs fixing. Rinse. Repeat.
- 3:52 pm: Submit your code for review. Ping-pong break.

A DAY IN THE LIFE

- 4:25 pm: Get feedback on your code. Talk through the code with the reviewer. Find things you can do better.
- 4:39 pm: Start changing your code.
- 5:32 pm: Submit your changes. Cross your fingers.
- 5:58 pm: Woo-hoo! Your code is accepted. Make sure your company’s next steps happen (testing, deployment to beta environments, etc.).
- 6:17 pm: Head home, content with another day.

RELATED JOBS

DEVELOPMENT-RELATED JOBS

- Testing: Make sure what's been developed actually works.
- Sales engineer: Use your strong interpersonal skills to help potential buyers understand the technical side of your product.
- Web producer: Manage the content of a site using a CMS (content management system) and some HTML/CSS/JavaScript.
- Technical writer: Create accurate, well-written documentation that helps customers use your product.
- Program manager: Herd cats. Keep the end goal in mind.

BECOMING A DEV

DEVELOPMENT MIGHT BE A GOOD FIT FOR YOU IF...

- You want a field where you are constantly learning
- Learning languages appeals to you
- You like to finish a problem once you tackle it
- You want to work with interesting people, who will challenge you to think differently and get better at what you do
- You think solving puzzles is fun
- You want to help make things that change the world (or at least one small corner of it)

GETTING THERE: SELF-STUDY

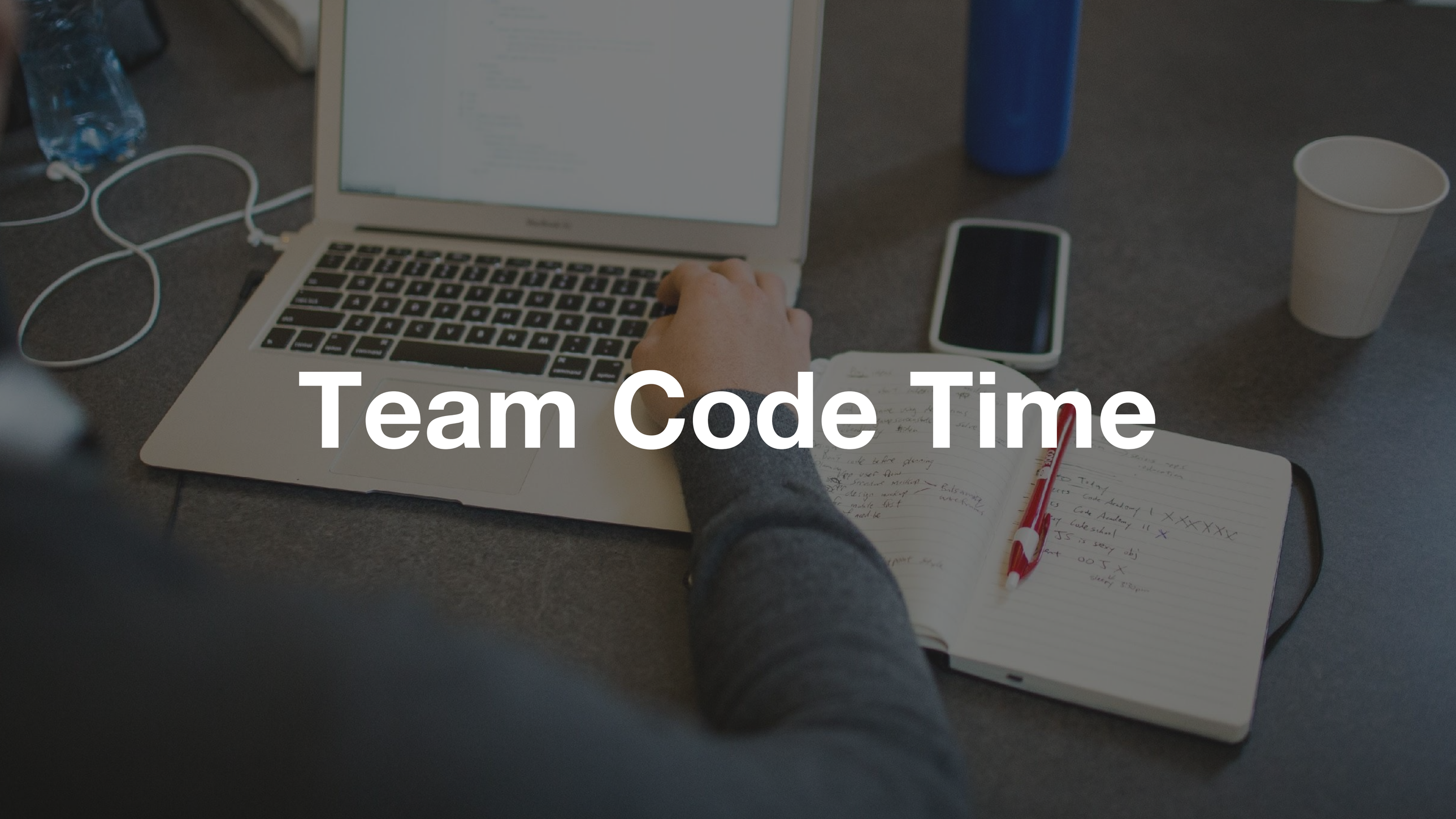
Online, books, meetups

- Pros:
  - Less expensive
  - Go at your own pace and schedule
- Cons:
  - Hard to get the depth of knowledge required to break into the industry
  - Fewer contacts to help you get a job

GETTING THERE: SCHOOLS

Specifically, code school programs like Code Fellows

- Pros:
  - Support from a community to meet your goals
  - You get industry-ready, up-to-the-minute skills
  - The school should actively support your job search
- Cons:
  - It requires commitment and motivation to get the most out of it
  - Expense (look for high placement rates and salaries for grads)


Team Code Time

FINISH IT!

- Each pair shares their HTML and CSS files on Slack.
- Individually, download all your team's files. Make sure you have everything.
- Test your site by opening one file in a browser. Your nav links should work now.
- If you have time left over, fine-tune the HTML and CSS.

SHARING CODE

ABOUT GIT

WHAT IS GIT?

It’s a **version control system.**

- It lets multiple developers work on the same code
- A history of changes to your files
- The ability to view, apply, and remove those changes
- Keep all your project files in one **repository**
- It makes collaboration possible!

WITHOUT VERSION CONTROL:

Look familiar?

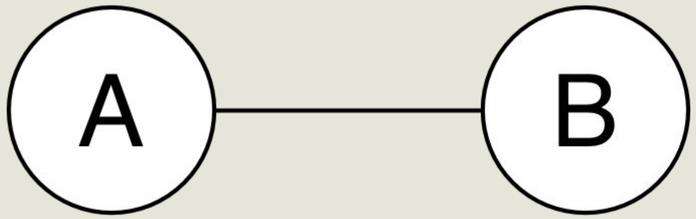
- `term_paper.docx`
- `term_paper2.docx`
- `term_paper2_with_footnotes.docx`
- `final_term_paper.docx`
- `term_paper_for_submission.docx`
- `term_paper_for_submission_for_real.docx`

SNAPSHOTS IN TIME

- **Commits** represent each successive version of a file or files.
- Commits are the Git equivalent of “Save As…”

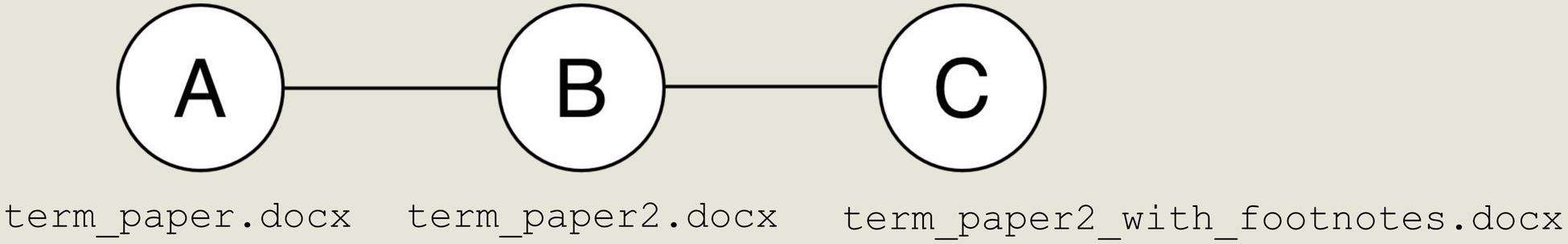
SNAPSHOTS IN TIME

Each successive version creates a new snapshot on the timeline of the project.



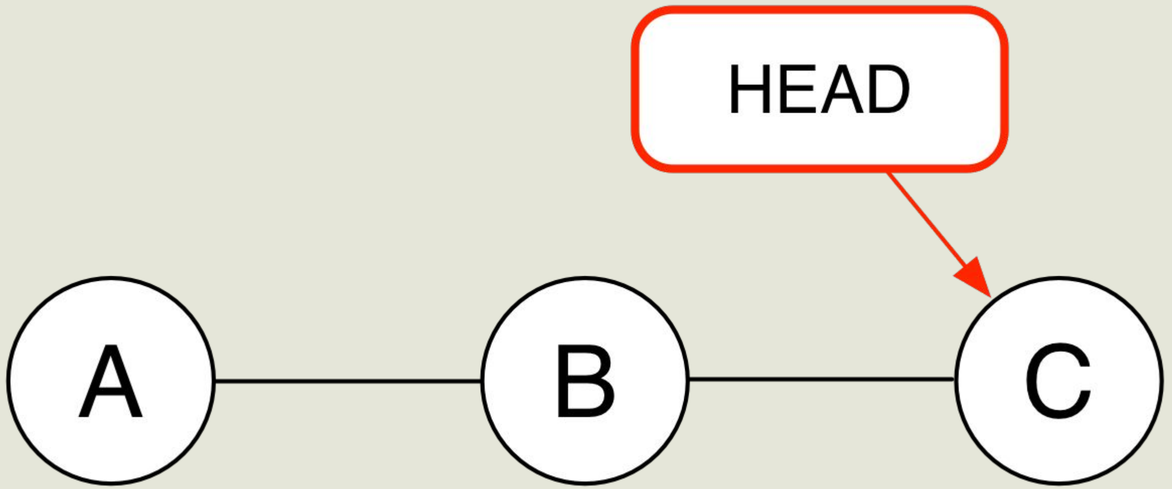
SNAPSHOTS IN TIME

Git keeps track of what the file looked like at different points in time.

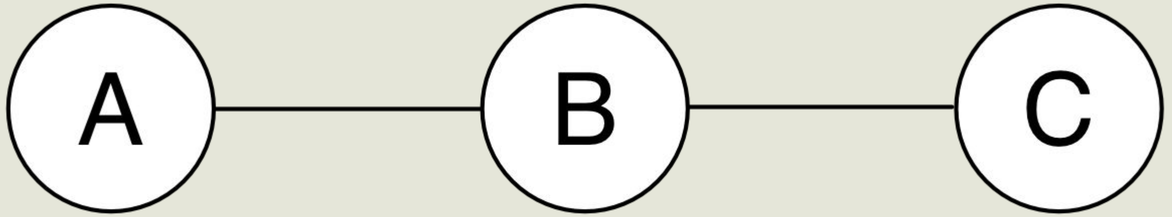


KEEPING TRACK

- Each **commit** (snapshot) has a label that points to it.
- HEAD = The label meaning “You Are Here”
- You can also assign **messages** to commits.
- **Messages** are like writing a caption for your snapshot.



A SUMMARY OF GIT



- You use Git to take snapshots of your code at points in time.
- Git keeps a history of what those snapshots look like.
- Git has a special label, called HEAD, that means “You Are Here.”
- Usually you give a snapshot a label, called a message.

GITHUB

WHAT'S GITHUB?

- A way to share code with others!
- An online place to store your code. (Backup is good!)
- It uses Git to help you manage your team's work:
  - Version tracking
  - Reviewing changes
  - Keep changes separate until you want to add them in

GIT + GITHUB = AWESOME

With Git (version control) and GitHub (online code storage), you can:

- Have lots of team members work on the same files, without messing each other up
- Keep a history of each file over time
- Work on code on your own computer, and sync it with what's online

REPOSITORIES (AKA “REPOS”)

WHAT'S A REPOSITORY?

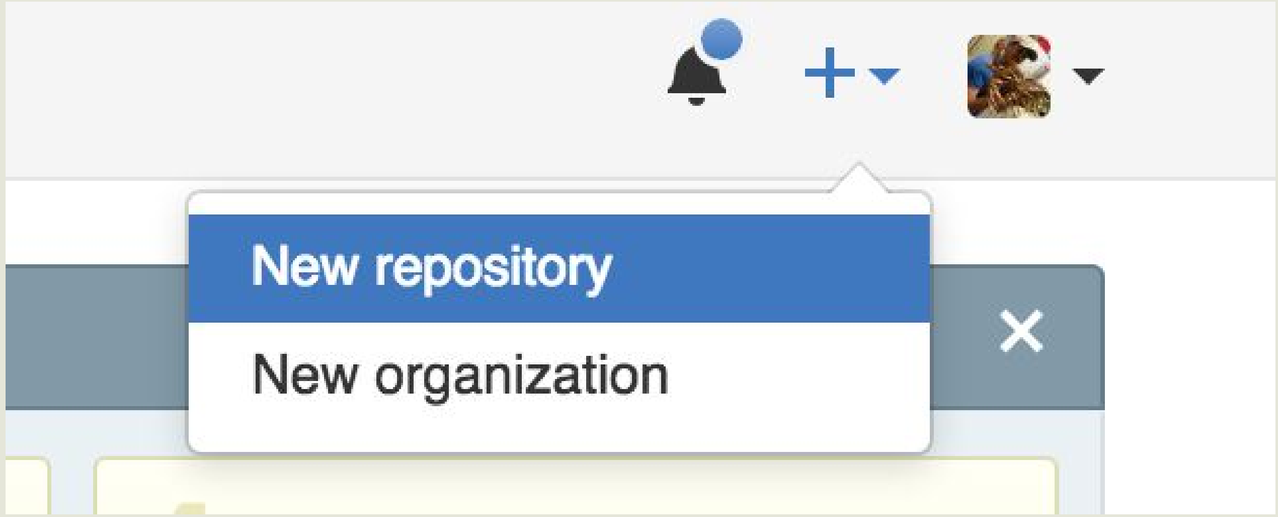
A repository is a collection of files that you've told Git to pay attention to.

- Usually, one project = one repository.
- Really large projects might have multiple repositories for different parts of their system (e.g. front end and back end).
- Repositories can live on GitHub or your computer.

Let's make one now!

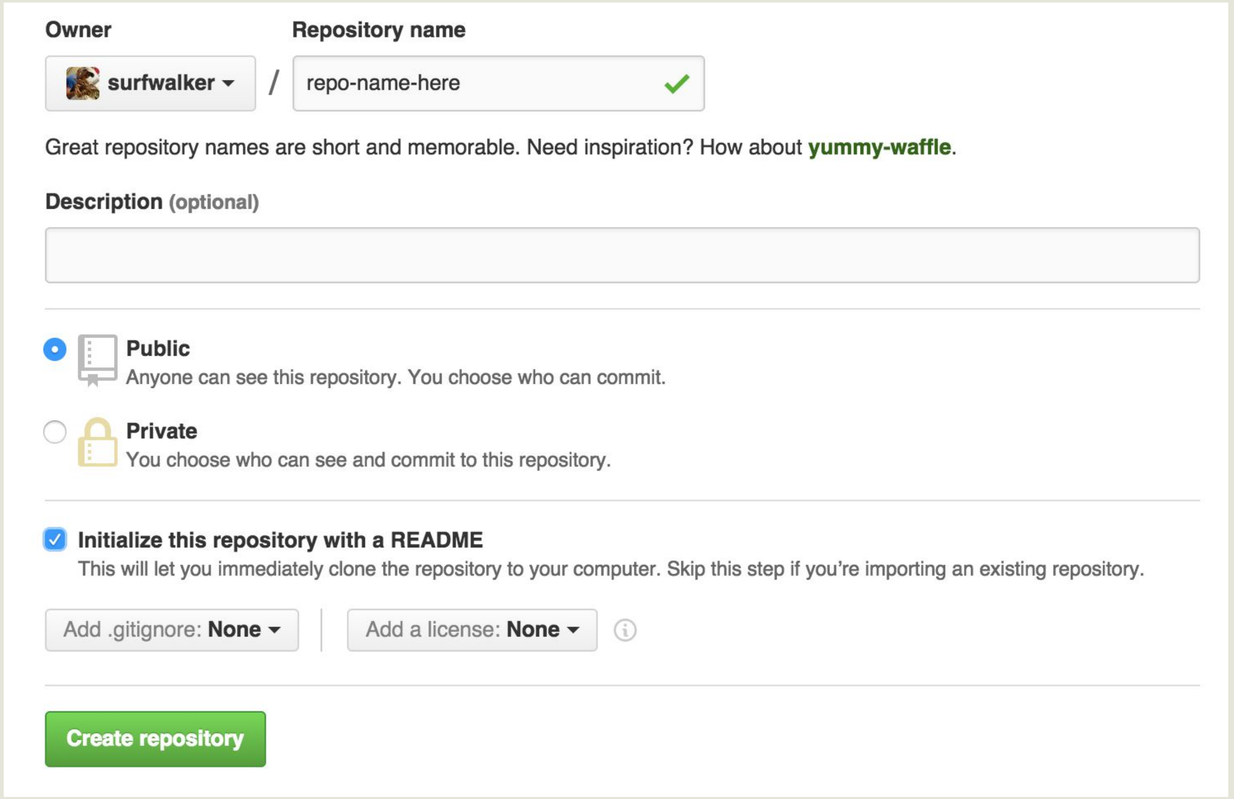
CREATING A REPO ON GITHUB

- Log in to GitHub.
- At the top right side of the window, look for your name and avatar.
- Next to it you'll find a small + sign. Click that.
- From the menu that opens, select *New repository.*



CREATING A REPO ON GITHUB

- Repos can be named anything.
- Today, name your repo this: **yourusername.github.io**
- Check *Initialize this repository with a README.*
- Click *Create repository*.



LINKING REPOS

- Congrats! You just made your first repo.
- Now we need to make a copy of this repo on our computers, and connect the two repos to each other.
- If they’re connected, they can give and receive code from the other repo.
- We’ll do this by **cloning**.

CLONE THAT REPO

- Look for a dropdown that says **HTTPS**, with a URL next to it. It's right above the blue bar, about ⅓ of the way down your screen.
- Then, copy the URL using either the *Copy to clipboard* button (the button on the right) or your keyboard shortcut.

GO TO YOUR PROJECTS FOLDER

- Open up your terminal
- Type **cd ~**
- That will take you to your home folder.
- Now type **ls** to list all your folders.
- You should see Documents (Mac), or a **projects** folder.
- Mac users, type: **cd Documents**. You should see a **projects** folder.
- Finally, type **cd projects** to navigate into that projects folder.

GIT COMMAND: `git clone`

In the projects directory in your terminal:

- Type **`git clone`**
- Paste in the link you just copied:
  - Mac: command + V
  - Windows: right-click on the terminal window, select Paste
  - Linux: ctrl + shift + V

```
$ git clone git@github.com:surfwalker/repo-name-here.git
```

- Hit Enter.

GIT COMMAND: `git clone`

What just happened?

- You made a local repo that is a copy of the one on GitHub
- You told that local repo that it can talk to the GitHub one.

  See the URL of the GitHub repo by typing **git remote -v**

```
$ git remote -v
```

- It made a directory that has all the files in it you had online
- Now move your site's files into this repo (this directory)!

COMMIT AND PUSH

GIT COMMAND: `git status`

Now that your files are in your repo, we need to make a commit (take a snapshot of them).

- Review the current status of your files by typing **git status**

```
$ git status
```

- It will tell you what files have changed since your last commit.
- In this case, you've only made one commit - when you told it to put a README in your repo (online).

GIT COMMAND: `git add`

Next: we need to tell git what files to commit.

- This is done by typing **git add** and then a filename.

```
$ git add index.html
```

- It will tell you what files have changed since your last commit.
- Repeat this command for each file that you added.
- This tells the file to get ready for the snapshot.
- Type **git status** again to see the difference!

GIT COMMAND: `git commit`

Finally, take that snapshot!

- Type **git commit -m “your message goes here”**

```
$ git commit -m “First commit”
```

- **git commit** is what takes the snapshot
- **-m** tells it that you want to create a message with the commit
- Think of the messages as being like photo captions.

GIT COMMAND: `git push`

Great! Now it's time to copy this code to your repo on GitHub.

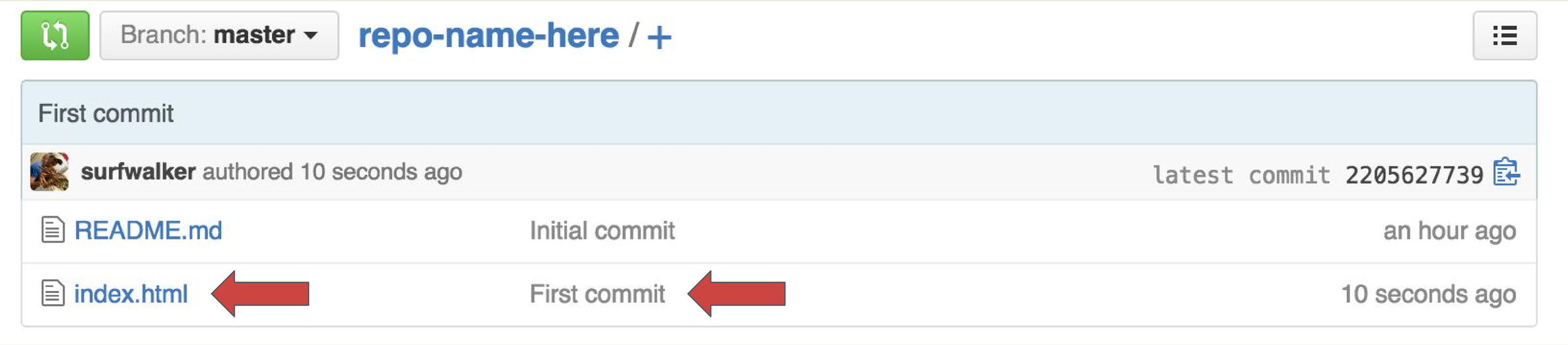
- Type **git push origin master**

```
$ git push origin master
Counting objects: 6, done.
…
To git@github.com:surfwalker/repo-name-here.git
* [new branch]      master -> master
```

- This sends this commit (this snapshot of your code) to GitHub.
- Go to your repo on GitHub, and look for your files!

VERIFY ON GITHUB

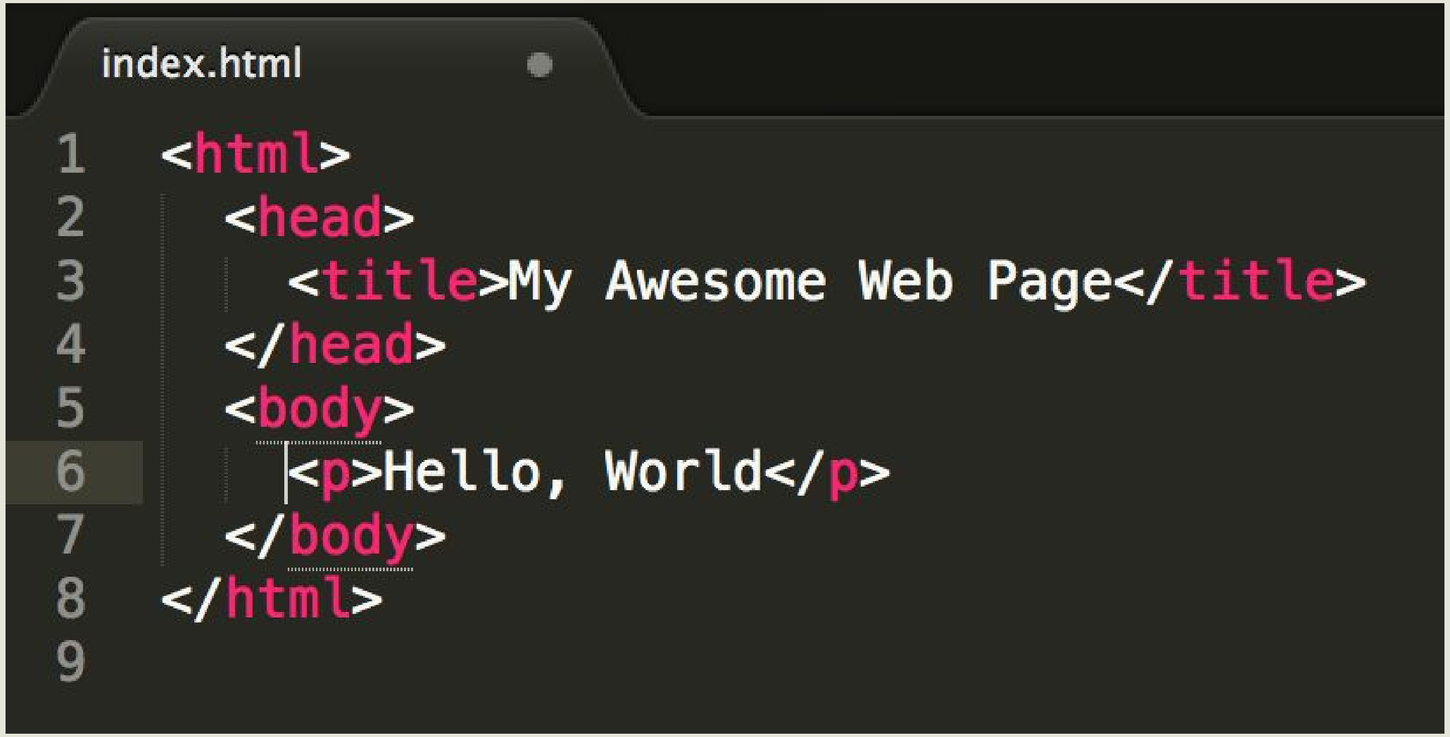
In your browser on GitHub you will see the file(s) that you pushed as well as the commit message.



DEPLOYMENT

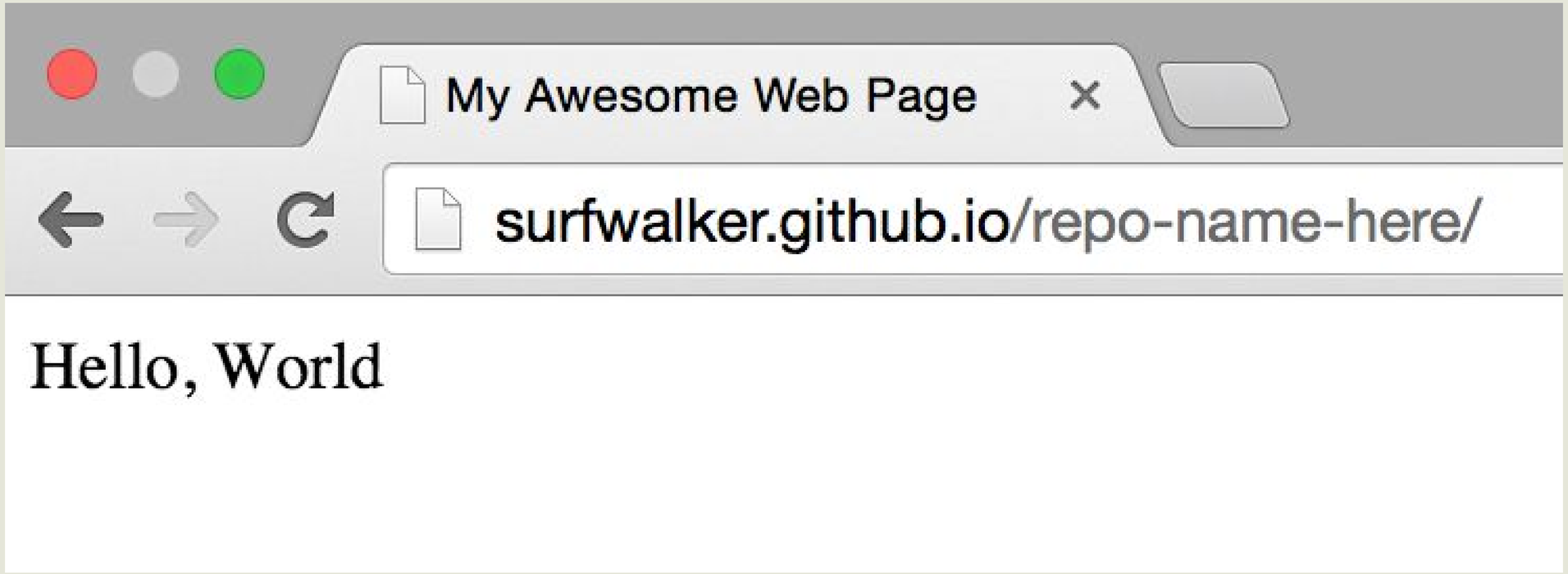
GITHUB PAGES

- GitHub will make one of your repos available for the world to see.
- That’s called “deployment.” It turns this...



GITHUB PAGES

- ...into this.
- The URL for your web page is *yourGitHubUserName.github.io/your-project-name*.

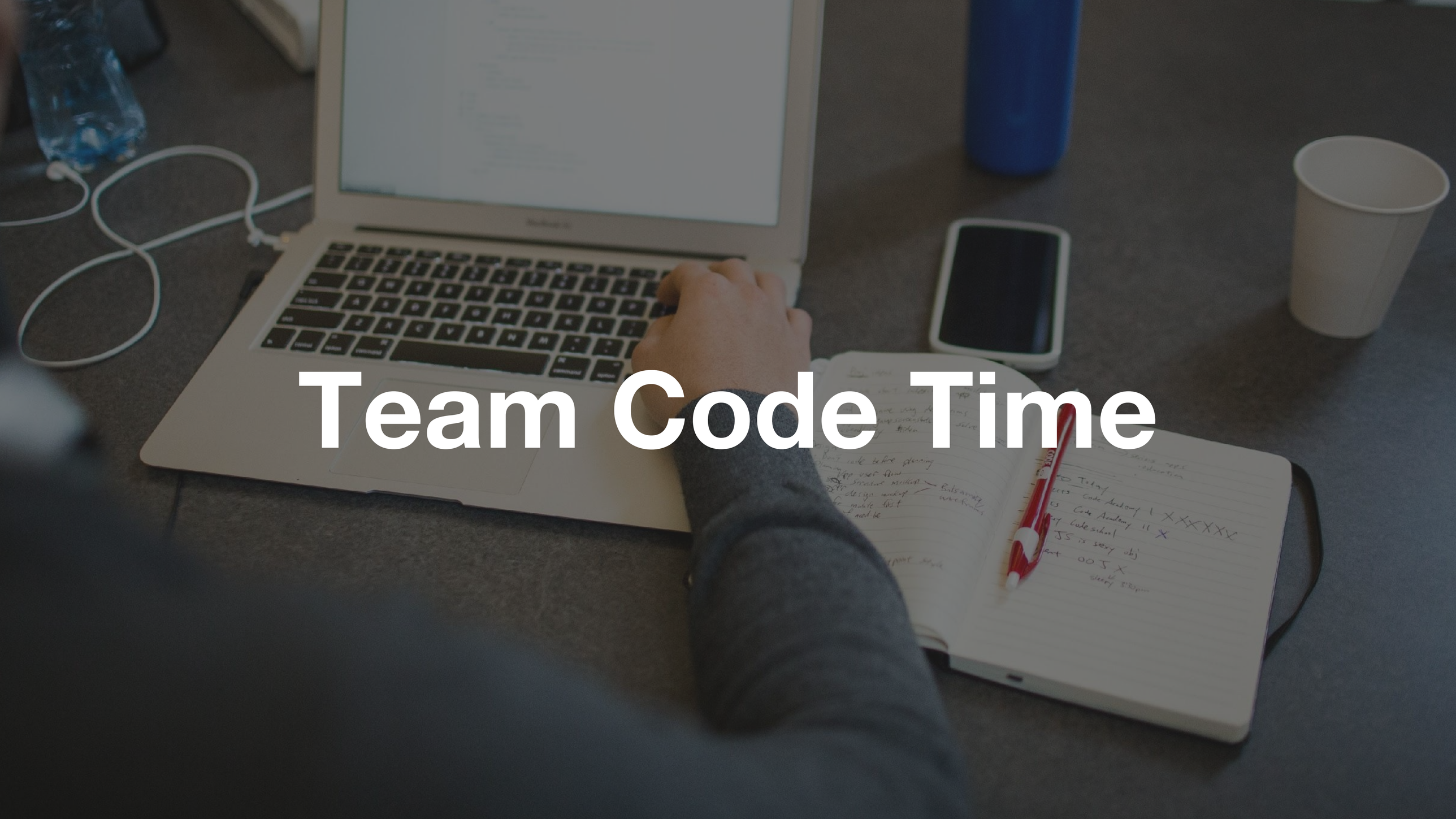


SHOW OFF!

Congratulations! You have successfully deployed your awesome work on the internet.

Now you can share that link with friends and family, and they can see what you did.

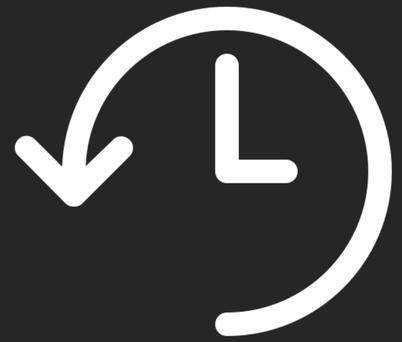
Give yourself a pat on the back!


Team Code Time

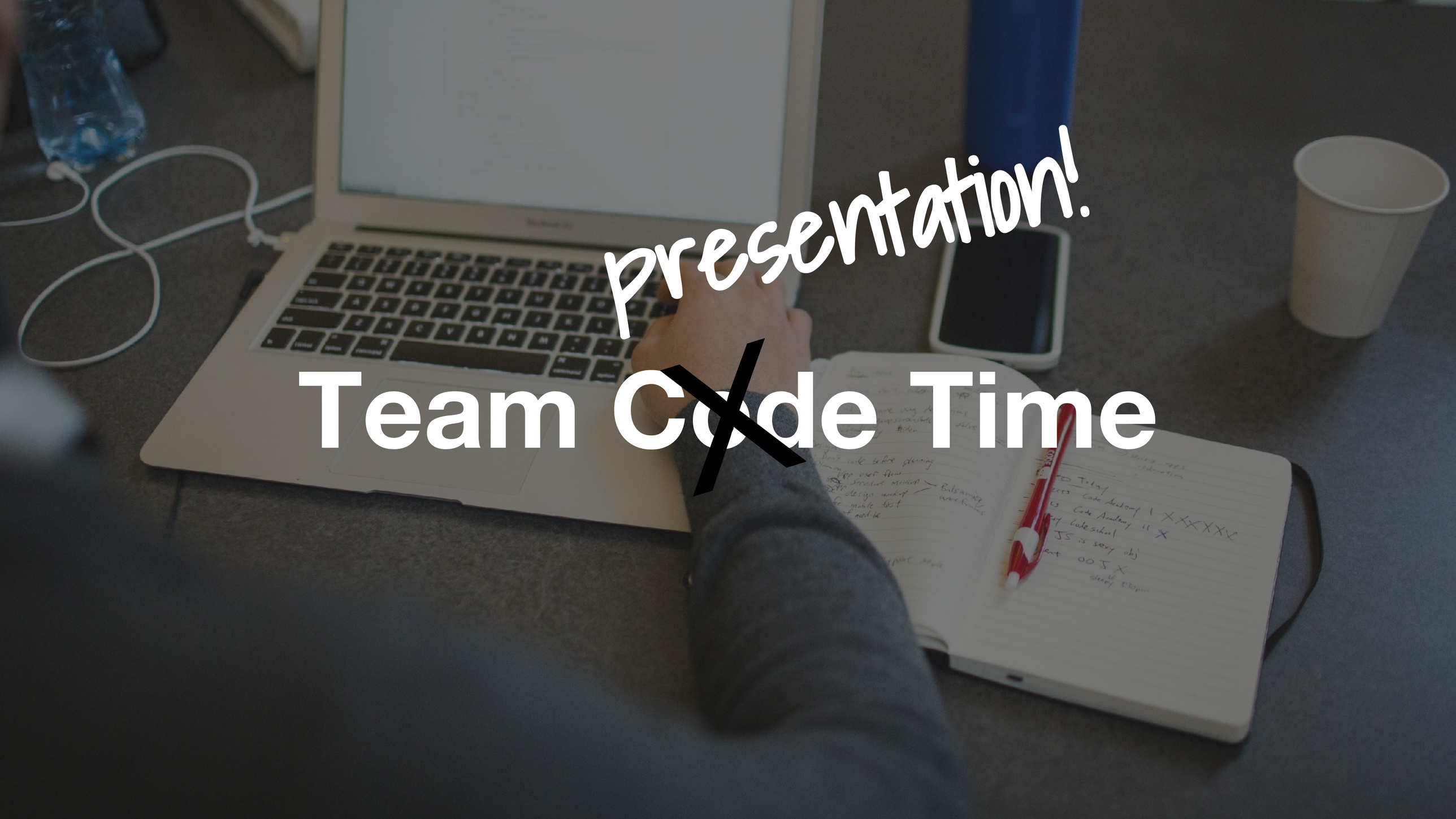
SHARE IT!

- Follow the “Uploading a File to GitHub” instructions to get your code onto GitHub
- Go to yourusername.github.io and see your code live on the web!

DINNER



SURVEY


presentation!
Team Code Time

NEXT STEPS

WHERE TO LEARN NEXT: CODE FELLOWS

If you liked this workshop, 201 is your next step.

- Daytime option (4 weeks)
- Nights & Weekends option (8 weeks)
- Classes start throughout the year

WHERE TO LEARN NEXT: ONLINE

Lots of great tutorials online for beginner-intermediate coders

- Codecademy
- Treehouse

Books:

- The Jon Duckett books, which are our textbooks for 201, are an easy-to-understand reference for HTML, CSS, and JavaScript.

WHERE TO LEARN NEXT: MEETUPS

There are a ton of tech-related meetups. Try searching for:

- Beginner-level workshops
- In Seattle: New Tech Seattle is a great meetup to absorb the energy of the startup community. Everyone with an interest in tech is welcome. Plus, food.
- In Portland: Code Oregon Labs

CELEBRATE!

REFLECT

- With your partner, spend a few minutes discussing:
  - Your favorite thing you learned today
  - What surprised you about today
  - Where you want to go next in your coding journey