coders
for causes

Workshops Materials

./Innovation with a mission

Coders for Causes

Copyright Coders for Causes

Table of contents

1. About 4
1.1 Coders for Causes Workshops 4
1.2 Contributions 5
1.2.1 Structure 5
1.2.2 Installation 5
1.2.3 Commands 5
1.2.4 Web Documentation Configuration 5
2. 2021/2022 Summer 6
2.1 Coders for Causes 2021/22 Summer Workshops 6
2.1.1 Project Technology 6
2.1.2 Workshop Recordings 6
2.2 Setup 7
2.2.1 Coders for Causes Project Team 7
2.2.2 Developer Tools 7
2.3 Project and Workshop Structure 10
2.3.1 Ways of Working 10
2.3.2 Workshop Schedule 10
2.3.3 Who are you? 12
2.4 Introduction to Web Development Space 13
2.4.1 Content 13
2.4.2 What and Why Web Development? 13
2.4.3 FAQs about Web Development 15
2.4.4 Basics of Web and Limitation 17
2.4.5 Server-side Applications (Backend) 19
2.4.6 Others 20
2.4.7 Word of Encouragement 27
2.5 Introduction to the Web Basics - HTML, CSS & JavaScript 28
2.5.1 Content 28
2.5.2 Key Web Technologies 28
2.5.3 HTML 29
2.5.4 CSS 33
2.5.5 JavaScript 36
2.5.6 Creating a Pokemon API Webapp 39
2.6 Practical Software Engineering Practices 40
2.6.1 How do software engineers work? 40

Copyright Coders for Causes

2.6.2 Why do Software Engineers Work like this? 42
2.6.3 What is Git? 43
2.6.4 GitHub and Code Repositories 46
2.6.5 INTERACTIVE WORKSHOP TIME !!!! 46
2.6.6 Other Resources 46
2.7 Introduction to Frontend Frameworks 47
2.7.1 Introduction to Frontend Frameworks 47
2.7.2 Introduction to React 61
2.7.3 Introduction to Vue 63
2.8 Welcome to the Djangol! 64
2.8.1 How does an application communicate with backend systems? 64
2.8.2 What are APIs and REST-APIs? 64
2.8.3 What is Django? 65
2.8.4 What is Django REST Framework (DRF)? 66
2.8.5 Features of Django and DRF 66
2.8.6 Packaging a "Django App" 67
2.8.7 Django Admin 67
2.8.8 Other Tools 68
2.8.9 Demo - Live Coding 71
3. 2021 Winter 72
3.1 Coders for Causes 2021 Winter Workshops 72
3.1.1 Project Technology 72
3.1.2 Where are the materials? 72

Copyright Coders for Causes

1. About

1.1 Coders for Causes Workshops

This is where you can see all the materials for workshops that are presented to the volunteers of the project for each year.

If you would to know more about us, please visit our website at codersforcauses.org

Copyright Coders for Causes

1.2 Contributions

Hi! We are happy that you thought of contributing! If you have any suggestions or issues, please raise it here. I would be happy if you could provide pull requests, if you know how to do it here.

1.2.1 Structure

Folder Structure

The structure of this repo is as follows:

1.2.2 Installation

Python

> Prerequisite

You need to have Python installed to be able to use `pip`. There are a few ways of installing Python. You can use a package distributor like Anaconda Or you can just install Python.

Once you have installed Python, install mkdocs requirements by opening a terminal and typing:

> Python Environments (Optional)

however, it is good practice to use different environments for different purposes, in which case, for Anaconda, you would open a terminal and type:

then enter:

Docker

Just run `docker-compose up`, it should show the web server running at localhost:8000

1.2.3 Commands

- `mkdocs new [dir-name]` - Create a new project.
- `mkdocs serve` - Start the live-reloading docs server. Very helpful when you want to take a look at the docs before deploying.
- `mkdocs build` - Build the documentation site.
- `mkdocs -h` - Print help message and exit.
- `mkdocs gh-deploy` - Deploy in github pages

1.2.4 Web Documentation Configuration

For full documentation visit:

- mkdocs.org for the generic MkDocs
- PyMdown Extensions for the different extensions that are installed
- MkDocs Material for the customisation of the web server documentation.

Copyright Coders for Causes

2. 2021/2022 Summer

2.1 Coders for Causes 2021/22 Summer Workshops

This project period continues the two main projects from the winter of 2020 :

- Foodbank
- WAIS

If you have not before seen the existing progress, see this video.

These two projects have their own corresponding technology stacks being used, hence will dictate the workshops that will be held.

2.1.1 Project Technology

Foodbank

Foodbank is mainly with frontend with React + NextJS + TypeScript + TailwindCSS with Firebase and Notion CMS.

WAIS

WAIS is a full-stack application with Vue and Django. It uses Docker containerisation for both development (and production in the future).

2.1.2 Workshop Recordings

The workshop recordings will be held on our youtube channel.

Copyright Coders for Causes

2.2 Setup

This contains everything you need to know about getting setup.

2.2.1 Coders for Causes Project Team

The following access you will need to have when working on the project

- Coders for Causes Official Project Organisation
  - This includes the project repositories and CFC related long-term materials
- Coders for Causes Learning Organisation
  - This includes the templates for learning as well as the demo workshops
- Discord Channel for Project and exclusive Workshops

2.2.2 Developer Tools

These are the following tools that you need:

- Code Editor: VS Code
- Version Control: Git
- Interpreter: Nodejs
- Custom Package Manager: Yarn
- Interpreter (for WAIS): Python
- Containerisation (for WAIS): Docker

Optional Tools

These are tools that you may like to use, but are not required:

- GUI for Git: Gitkraken / GitHub Desktop

After these installation, seek at the OS-specific tools.

Windows

These are tools specifically for Windows:

- Virtual Machine: Vbox
  - You need this if you have some trouble with windows

Warning for Windows Users

Copyright Coders for Causes

Legit, among all the OS, you will have the most frustrating time as a developer in windows (unless you're doing C#)

Wanna have a better developer experience

Copyright Coders for Causes

You have a couple of options:

- Using WSL
- Dual Booting
- Virtual Machines

The recommended OS to try is Ubuntu-based Linux. My personal favourite is PopOS.

Linux

These are installation specific to Linux:

- Docker Post Install
    - Lots of user forget this documentation

Copyright Coders for Causes

2.3 Project and Workshop Structure

A mission to empower the next-generation of software engineers and delivering value to the community!

2.3.1 Ways of Working

- (November 27 to February 24 excluding December 19 to January 4) = ~ 11 weeks for project
- 2 meetups every week (1 online mid-week, 1 on Saturday)
- Each Saturday meeting will have ~2-3 hour workshops. Some workshops will take the whole hour, while some workshops will take 15-30 minutes.
- The other time allocated is for working on project with supervision
- Most workshops will be introductory

2.3.2 Workshop Schedule

The following are rough guideline to the workshop schedules. All workshops are optional, you can attend any of them even if you are not on the specific team dedicated to it (be mindful that there may be time conflicts if it is with the other team).

Most Workshops are introductory

They will usually cover the following idea:

- why is it useful to learn it
- what are the different aspects of it that you will need to learn

Copyright Coders for Causes

You will still have to put in the effort to learn it thoroughly.

Date	Workshop Name	Recommended Team to attend	Duration
Saturday, 27 November 2021	Introduction to CFC + Web Development Space	All	~1 hour
Wednesday, 1 December 2021	Introduction to Web Basics	All	~1-1.5 hour
Saturday, 4 December 2021	Practical Software Engineering Practices	All	~1 hour
Saturday, 4 December 2021	Introduction to Frontend Frameworks	All	~45 minutes - 1 hour
Saturday, 4 December 2021	Introduction to React	Foodbank	~45 minutes - 1 hour
Saturday, 4 December 2021	Introduction to Vue	WAIS	~45 minutes - 1 hour
Saturday, 11 December 2021	Introduction to Django	WAIS	~1 - 1.5 hours
Saturday, 11 December 2021	Introduction to Typescript	Foodbank	~30 minutes
Saturday, 18 December 2021	MVC Codebase Structure / Frontend-Backend Integration	All	~1-1.5 hour
Saturday, 18 December 2021	Typical Codebase Structure	All	~30 minutes - 1 hour
Saturday, 18 December 2021	Package Manager - JavaScript and Python	All	~1 hour
Saturday, 8 January 2022	Introduction to Docker	All	~1 hour
Saturday, 8 January 2022	Introduction to Unit Testing and CI/CD	All	~1 hour

Workshops after January 8, 2022

The workshops right here are still being decided upon. If you have an idea for a workshop that you would like to attend, please let us know either on Discord or at Github.

The one that are on consideration are:

- Deployment on Heroku, and Vercel
- Introduction to End-to-End Testing with Cypress
- Gitkraken Workshop
- Introduction to Prototyping with Figma
- Introduction to Linux and Command-Line Scripting - Bash
- Increase your productivity in VsCode
- How to write good documentation
- Honing your detective skills with Browser Developer Tools

Copyright Coders for Causes

2.3.3 Who are you?

Before continuing further, answer the following:

- What’s your name?
- What’s your background?
- Why you’re here?

Who are the Coders for Causes

The Coders for Causes is an organisation that aims to empower the next-generation of software engineers while delivering value to the wider-community by helping charities and not-for-profit organisations.

Copyright Coders for Causes

2.4 Introduction to Web Development Space

Navigating the Deep Dark Space of Web Development

This workshop covers a brief overview of the most common tools and technologies used in web development.

2.4.1 Content

- What and Why Web Development?
- FAQs about Web Development
- Basics of Web and Limitation
- Server-side Applications (Backend)
- Others
    - Languages of the Web (the usual)
    - Hosting Stuff
    - CSS Frameworks
    - Developer Tools
    - TypeScript
    - Testing
    - Continuous Integration / Continuous Deployment
    - Virtualisation and Containerisation
    - Browsers
    - Firefox Developer Tools
    - Package Managers
    - Version Control
    - Linters and Formatters
    - Teamwork
    - Roadmaps
- Word of Encouragement

2.4.2 What and Why Web Development?

What is web development?

- Websites development
- Web applications (client-side and server-side) development

Why Web Development ?

- Accessibility and Portability
- Career and On-demand in job market
- Huge possibility to combine with other emerging technologies (e.g. IoT, Machine Learning) and industry (e.g. Health, Mining, O&G)

Career in Web Development

Source: Insights from Stack Overflow's 2016 survey of 50,000 developers

Copyright Coders for Causes

"Half of Developers are Web Developers"

Copyright Coders for Causes

2.4.3 FAQs about Web Development

Why code websites, why not use drag and drops like Wordpress, WIX?

- Content Management System (CMS)
- Limitations on theme/template used
- Difficult to extend
- Cybersecurity

More information

CMS are one of the application of web development, but there are plenty more such as - internet of things, custom software for a particular industrial application (eg. using Machine Learning)

CMS are usually limited to the template or plugin that you use. If those plugin don't exist, then it limits your productivity very much (difficulty to extend).

CMS are usually built to cater for non-technical users. This means that thye become the subject of hackers. Think about a scenario where a hacker was able to find a vulnerability in WordPress, now every other WordPress site will be vulnerable.

What is the best way to learn all these?

In summary, the best way to learn:

- Do personal projects (inspiration + motivation)
- Do team projects (get peer reviews and correct bad practices straight away)
- Watch Online Courses (to figure out what is available)

More Information

To be told that you have to learn "this, this, and that" before you could do things is tiresome.

Often times, we want to learn to be a developer so that we can create cool things like software where thousands of people can use the app. We don't tend to be a developer for the sake of us needing to watch endless videos on different things.

Why does CFC not do mobile development as much as web development?

- App stores has a developer cost
- Easier to deal/teach web technologies
- Accessibility (mobile, sensors, tablets, laptops and PCs)
- Bigger open-source community

Copyright Coders for Causes

If I already know a frontend framework, is it better to learn another frontend framework or to learn a backend framework?

- It is better to learn a backend framework

Reasoning

You want to build skills that complement one another rather than be an alternative.

It is much more valuable for you to learn a backend framework because that helps you build a **functional** app.

Copyright Coders for Causes

2.4.4 Basics of Web and Limitation

HTML

What is it?

- Hypertext Markup Language
- Describes the structure of a web page

Limitation

- Doesn’t handle repeated content well
- No variables or calculation

HTML Syntax

CSS

What is it?

- Cascading Style Sheets
- Describes the presentation of a web page

Limitation

- Most css is quite similar (Handled by CSS Libraries)
- Not very dynamic (Handled by CSS Frameworks)

CSS Syntax

JS

What is it?

- JavaScript
- Used to program complex features on a web page

Limitation

- Has the capability to modify the user interface, but becomes really tedious to modify interface (more about this in another workshop)

JS Syntax

Copyright Coders for Causes

Modern Frameworks

- Websites can be much more… they can be web applications
- “App” in a website (client-side rendering)

Modern Frameworks

React.js

- More mature and used more in industry

Vue.js

- Growing fast in popularity and use.

General Information

- Both are good to use and learn.
- Knowledge is transferable between the two frameworks.

Comparison between HTML and JSX

Copyright Coders for Causes

Highlighted portions are starting chunk of distinct code.

HTML

JSX

2.4.5 Server-side Applications (Backend)

How do devices communicate?

- HTTP Request - Hypertext Transfer Protocol

What do server applications do?

- Serve frontends (server-side rendering)
- Web API (Application Programming Interface)
    - Serve data (usually from a database)
    - Process Request (Sending emails or SMS, Machine Learning)

Databases

Place to store the data

Mongodb

Allows for database design to be modified without complex migration or data loss

SQL

Typically faster and better for large amounts of data or systems that need data consistency and reliability

Copyright Coders for Causes

2.4.6 Others

Languages of the Web (the usual)

- Python (Django, Flask)
- JavaScript (Node.js, Express)
- Ruby, Go, Rust, C

Hosting Stuff

Many ways - Own a server - Use a 3rd party platform

CSS Frameworks

Frontend (JS)

- Vue, React
- Nuxt.js, Next.js

Frontend (CSS)

- MaterialUI, Vuetify
- Bootstrap

Developer Tools

TYPESCRIPT

- Type checking is super useful for complex apps
- Allows for way better javascript developer tooling
- Can be annoying if you’re new at it

TESTING

- Selenium, Cypress
    - End to end automated testing tools
- Jest, Mocha, Pytest
    - Unit testing
- Testing is vital to software projects

CONTINUOUS INTEGRATION / CONTINUOUS DEPLOYMENT

- Automated Testing
- Event-driven scripts
- E.g. Github Action, Bitbucket Pipelines

VIRTUALISATION AND CONTAINERISATION

- Allows execution of services in a virtual environment
- eg. Docker (Containerisation), Vagrant (Virtualisation)

BROWSERS

- Standard browsers
    - Google Chrome, Firefox, Edge, etc.
- Backwards compatibility
    - Internet Explorer

Copyright Coders for Causes

- Other
    - Mobile - Responsive
    - Screen readers - Accessibility

FIREFOX DEVELOPER TOOLS

- Page Inspector
    - Visualise page aspects
    - Grid layout
- Web Console
    - `console.log("Hello World")`
- Responsive Design Mode
    - View from POV of different screen sizes such as mobile, tablets, etc.

Some more tools

- JavaScript Debugger
- Network Monitor
- Performance Tools
- Rulers
- Colour Pickers Learn more at: https://developer.mozilla.org/en-US/docs/Tools

PACKAGE MANAGERS

- Installs libraries that can be used
- Also has code shortcuts (e.g. npm run start)

(More about `package.json` and `poetry.toml` in the projects and Package Manager Workshop)

VERSION CONTROL

- Essential for developer teams and complex software development
- Git

LINTERS AND FORMATTERS

- Makes code formatting consistent (following standard)
- Useful with version controls to avoid pointless change

eg.ESLint, Prettier

TEAMWORK

- Many tools out there
- Used to stop teams from stepping on each others toes
- Github Issues + Pull Requests

Roadmaps

There's an open-source community that maintains a learning roadmap for developers. See https://roadmap.sh/

Frontend Developer

Copyright Coders for Causes

Copyright Coders for Causes

Backend Developer

Copyright Coders for Causes

Copyright Coders for Causes

Dev-Ops

Copyright Coders for Causes

Copyright Coders for Causes

2.4.7 Word of Encouragement

Encouragement from the Tech Lead

"I can admit that this journey of learning will be difficult, and can sometimes be overwhelming and demotivating. Please, if at any point of this project, you feel that you don't know enough, or you're feeling lost, please reach out! We are all in this journey together! Nobody is born talented, skills are honed with determination and willingness to learn."

"When I was a first year student entering on the CFC winter project, I didn't feel like I was good enough. I couldn't create a good looking interface, I didn't know how to use npm and all sorts of those things. I was just like many of you! if I gave up just because of all those things I didn't know, of all those self-doubts, then I wouldn't be here today. I admit that I was lucky because I was in CFC, I had connections where I can just ask questions instead of feeling lost of not knowing. So please do leverage that opportunity to reach out"

"You being in this project not only gives you the opportunity to raise your talents, but you also unlock one of the biggest factor of the growth of your career, and that is the connections with your fellow software engineers."

Copyright Coders for Causes

2.5 Introduction to the Web Basics - HTML, CSS & JavaScript

The tools of the web

This workshop will introduce you to the basics of HTML, CSS and JavaScript, the fundamental technologies of web development. This is a very introductory workshop, and there is still so much we can learn, but this will be a good place to start.

2.5.1 Content

- Key Web Technologies
- HTML
  - Elements
  - Attributes
  - Images
  - Anchors/Hyperlinks
  - Forms
  - Document Object Model
- CSS
  - CSS Format
  - Rules of Selection
  - Document and External Style Sheets
  - Flexboxes
- JavaScript
  - The Basics of JavaScript
  - Arrays
  - Objects
  - Functions
  - Loops
- Web App Tutorial

2.5.2 Key Web Technologies

HTML, CSS and JavaScript each have a different job when it comes to creating web pages.

What do they do?

- HTML: describes the content and structure of the web page
- CSS: describes the style and appearance of the web page
- JavaScript: provides funtionality to a web page

Copyright Coders for Causes

2.5.3 HTML

Hyper Text Markup Language (HTML) is used to structure the webpage. The general structure of a webpage can be seen below.

My First HTML Page

Elements

A webpage is made of elements, each with their own properties, that contain content to be displayed on the page. Elements are defined by tags, such as `h1`, `div` and `body`. Most elements have an open and close tab. The container and its content, together, are called an element.

The most notable, and probably the tag you will use most, is the `<div>` tag. It defines a division or section within the HTML document, and is used as a container that holds other elements.

The Seperator

We will go through the important ones during this workshop, but you can find the full of them here.

Attributes

Most tags have attributes that specify information or change the tag in some way. The most common being `class` and `id`.

MOST COMMON ATTRIBUTES

- `class` : used to specify one or more class names for a HTML element.
  - Classes are used to group certain elements in order to give them specific features through CSS and to allow many elements to be manipulated using JavaScript.
- `id` : used to specify a unique id for an element and must be unique
  - The `id` attribute is assigned to an element so that element can be exclusively specified in the style declaration and JavaScript manipulation

Below is an example of how attributes are assigned in your code.

Elements and their Attributes

Images

Images are added by using the `<img>` element tag. It can also be used to add gifs!

Example

Copyright Coders for Causes

Copyright Coders for Causes

Cute, right?

IMAGE ATTRIBUTES

- `src` : specifies the location of the image to be displayed
  - This can either be a url, as in the example above, or the relative path of an image within the site's directory
- `alt` : an optional attribute that contains a text description.
  - Is useful for accessibility or if the image does not load properly

Anchors and Hyperlinks

We can add links to other websites, or even to sections within the same page, using the `<a>` element, known as an anchor.

> Adding a hyperlink to an HTML page

They can be adding added by themself, or within text such as here.

ANCHOR ATTRIBUTES

- `href` : specifies the destination to link to
  - External site: simply include the url you wish to visit
  - Different page within the site: include the path to the new page, usually in the form of `"/.../index.html"`
  - Different section on same page: use `#name_of_section` where `name_of_section` is the `id` of the element you wish to go to
- `target` : specifies where to open the link
  - Setting the target attribute to `"_blank"` opens the link in another tab
- `download` : specifies that the linked resouces will be downloaded.
  - Only needs to be included in the declaration of the element
  - Optional: if value of download is set, that value will be the name of the file

Always be careul when clicking links. You never know when there is something you should not click.

IMAGES AND HYPERLINKS

Any image, or any element in general, can be turned into a link by enclosing the element within the anchor tabs.

> Hyperlinked image

Copyright Coders for Causes

Forms

Forms are used to collect user input. They are defined by the `<form>` tags and usually contain form elements such as `<input>`, `<textbox>`, etc.

Inputs can come in many different forms, such as textboxes, radio buttons, checkboxes and drop down menus. Each input field is given a `value`
Once a form is complete and filled out, we need to submit, or POST, it. This can be done through a special input of type `submit`.

The only required attribute of `<form>` is `action`. The `action` attribute speicifies the URL of the application that is to be called when the Submit button is pressed. If no action, then the attribute takes the value of an empty string and the current page is the destination.

Copyright Coders for Causes

Example

INPUT VALIDATION

Input validation is a very important aspect to consider when dealing with forms. Remember the famous acronym GIGO, Garbage-in Garbage-Out.
To assist you with input validation, HTML forms have in-built validation for different types of data, such as emails, numbers and dates. This can be achieved by changing the input type of a field to the respective data entry type.

Baseline input validation with HTML

Document Object Model

All the elements of a HTML page make up a document tree, called the Document Object Model (DOM).

The DOM is a platform and language-neutral interface that allows programs to dynamically access and update the content, structure and style of the HTML document. Each element in a HTML document is represented by a node on the tree.
We can then use things such as JavaScript to access and update the HTML document using the DOM. We will see more of this in the tutorial at the end.

2.5.4 CSS

Cascading Style Sheets (CSS) provides style to the web. It is used to specify the layout and style of markup languages. CSS tells the browser *how* to display the elements that are written in HTML.

We can write the CSS style rules into an element using inline CSS, where the `style` attribute of an element is modified directly in the HTML.

Inline style sheets

However, this gets hard to manage and maintain, especially when the number of elements in a document grows, and when we want to change many elements that have the same style.
We use *document-level* style sheets or *external* style sheets to combat this issue.

But first, we must understand how to create these style sheets.

Copyright Coders for Causes

CSS Format

- Selector
  - A value, or list of values, that specify the elements for which the following style will be applied to
  - Rules for specification will be discussed a little later on, see here
- Attribute
  - The attribute/property of an element you wish you change
  - Some example attributes include `background-color`, `font-size` and `width`
- Effect
  - The effect is the value you set each attribute to be
  - This includes setting `background-color` to "red", or `font-size` to "16px"

The following CSS specifies that all `img` elements are to be centered and have a width of 50 pixels.

My First CSS

How do I know what attributes exist, and what do I change to get my desired effect?

There are two simple answers to this question:
1. **Google it.** Chances are, someone has already tried to do something similar and your solution already exists.
2. **Play around**. Learn what attributes exist, and how changing thier effects changes their on screen appearance.

Rules of Selection

As mentioned earlier, selectors have a set of rules that allow you to apply styles to certain, or even very specific, groups of elements. These can range selecting elements based on the type of element they are, or what class they are in, to selecting all elements of a certain type that are immediately preceded by an element of another type.

Element

Element selectors simply apply the defined style to all elements of the same type.

Class

Class selectors apply the defined style to all elements belonging to that class. They are similar to element selectors, except that the class name is preceded by a `.` character.

We can also specify the type of element within a class we wish to apply the style to by including it before the `.` character.
The following code selects all `<p>` elements that are of the class `narrow`:

id

These selectors apply the style to an element based on its `id`. The `id` is preceded by `#` in the selector definition.

Psuedo Classes

Psuedo classes are styles that apply only when a certian action occurs or a condition is met, and not all the time. Some common selectors include `hover`, `focus` and `active`. They are included after the element, class or id has been specified, and are preceded with a colon `:`.

The full list of them can be found here.

Copyright Coders for Causes

Group

To specify a group of elements to apply a style to, simply list the elements, ids or classes delimitted by a comma (`,`).

Other

There exists other selectors that allow you to be more specific as to what elements are to be selected, such as contextual selectors, but we will leave those as *self-learning*.

Contextual Selectors

Contextual selectors group elements based on their position and surroundings in the document tree.

Further reading into the topic can be found through searching the web for platforms that teach you all about computer science, such as Geeks for Geeks.

Document and External Style Sheets

Now that we know what we are doing, let's create some style sheets! Document and External style sheets help us better organise and manage the styles of the document. They are located in one location, so you do not have to go far to find them and change parts of your document.

DOCUMENT STYLE SHEETS

Document style sheets are located within the `<head>` of a HTML document, under the `<style>` tag. They work in the same was as mentioned above.
See the below example of how document style sheets are implemented within a HTML page.

Document Style Sheets

EXTERNAL STYLE SHEETS

What happens when our site grows, and the number of HTML pages increases, and suddenly styles have to change, and we have to change every document, but they all have to follow the same styling as the others, and we have to manage that?!
Thankfully, we can store our styles in a seperate `.css` file and then simply reference the stylesheet in our HTML document.

styles.css

Once we have our css file completed, we can reference it in the HTML document, again in the `<head>`, so that it can apply the styles to our document.

Applying my CSS files to my HTML document

Copyright Coders for Causes

Flexboxes

Flexboxes are a great way to position your items within a container.

Learning is best done when having fun and getting your hands dirty (metaphorically). Flexbox Froggy is an interactive website that teaches you all about flexboxes, one step at a time. Your aim is to allign all the frogs to their corresponding lilypads. It is a much better learning tool than sitting and reading about some code I wrote.

2.5.5 JavaScript

JavaScript, also known as JS, gives a web page fucntionality and reactiveness. It allows the user to interact with the web page, and for us to make it to do things that we want it to do.
Similar to CSS, all your JS can be implemented into a HTML document by encapsulating it in the `<script>` tag within the `<head>` of the document, or even `<body>` in this case. However, we will be sticking to having our JS stored in external files for ease of managability.
See the below example of how to add a script file to your page. To add many files, simply add another `<script>` element and reference the other file.

> Adding JavaScript to my HTML document

The Basics of JavaScript

We shall discuss the basic syntax of JS, such as variables and functions, just to get you started. Feel free to do some of your own learning, too. There is a lot of cool things you can do and shortcuts you can use when you dive deeper into JavaScript, such as the *ternary operator*, but we will leave these for now.

VARIABLES

Variables are named memory locations that store data.
To define a variable, we can use three different keywords, each giving the variable special properties.

Variable Declaration

- `var`
  - allows the variable to be redeclared later on in the program
  - gives the variable a global scope, meaning they can be accessed anywhere within the file
- `let`
  - once a variable has been declared using `let`, it cannot be redeclared. It's value can still change, however.
  - gives the variable *block scope*, meaning it can only be accessed within the block of code that it has been declared in.
    For example, if I declare a variable using the `let` key word, I cannot access it outside of the function.
- `const`
  - Once the variable has been declared and assigned, it cannot be redeclared and the value never changes. It stays *constant*, sort of.
  - It, too, gives the variable block scope.

For a full explanation on variable declaration in JS, check out w3schools' page on it.

Variables can hold different data types, such as numbers, strings, objects, functions and arrays, but JS will cover the type identification for you.

Datatypes

- There are two groups of data types in JS: primitives and structural.

Copyright Coders for Causes

- A **primitive** is data that is not an object and has no methods.
  There are seven primitive data types:
    - String, Number, BigInt, undefined, null and symbol
- A **structural** data type is one where the data is in the form of an object, and that object has its own methods. The main structural data types are:
    - Objects and Functions

MATH AND LOGIC

Math and logic works similar in JS to other programming languages.

Math

- `+`, `-`: addition and subtraction
- `*`, `/`: multiplication and division, respectively.
- `%`: modulo operator. Returns the remainder left over after division.
    - For example, `8 % 3` returns 2.
- `**`: exponent (*x to the power of y*)
    - `base ** power`

Logic

- `&&`: AND operation
- `||`: OR operation
- `!`: NOT operation
- `>` / `>=`: greater than/greater than or equal to
- `<` / `<=`: less than/less than or equal to
- `==`: equal to
- `===`: exactly equal to
    - Works in a similar way to `==`, except it also checks that the datatype is the same

Difference between == and ===

Arrays

Arrays are an ordered list of values. They can hold values of many datatypes. Their index starts at 0.

Objects

Objects are variables that can hold more than one value. One can be seen in the previous example in Arrays.
The different values of an object are called keys. The keys can hold regular primitive values, such as numbers or strings, or can hold other objects, such as functions. Think of Objects as a list of key/value pairs.

To access a key's value within an object, you must first reference the object in question, then insert a `.` folllowed by the key you wish to get.

Objects in JavaScript

Functions

Functions are blocks of code designed to execute a particular task.
In JS, the syntax for defining a function is as follows:

Copyright Coders for Causes

Functions in JavaScript

Functions can be called or stored in variables.

Calling and Storing Functions

Functions can return a value (after calculation, etc) or simply perform work on existing data/variables. Functions that do not return anything are normally called *procedures*.

ARROW FUNCTIONS

Arrow functions are just a compact way of writing normal functions.
They work by removing the `function` key word, and even the `return` keyword in some cases.
Arrows functions lead with their parameters, usually enclosed in normal brackets. An arrow `=>` then follows, preceding the actual block of code to be executed.

Arrow Functions: Example 1

In single-lined functions, such as the one above, both the { braces } and the `return` can be omitted. However, when there are extra lines of processing, both must be included.

Arrow Functions: Example 2

Loops

For

For loops repeat until a condition is met. That condition is defined in the for loop. For loops have the following structure:

For Loops

For-in

For-in loops iterate over the indexes of data in an iterable object, such as an array.

For-in Loops

For-of

For-of loops iterate over the data in the iterable object.

For-of Loops

Copyright Coders for Causes

While

While loops iterate while a condition is true. They are called a "pre-test" loop, where the condition is tested before the loop can run. The condition is included in the brackets.

While Loops

Do-while

Do-while loops are similar to while loops, except that they let the block of code run once before testing the condition. They are known as "post-test" loops, and the loop is guaranteed to execute at least once.

Do-while Loops

Loops can be broken or stopped using the `break` or `continue` statements.

BREAKIN' OUT

- `break` : execution leaves the loop completely and continues on with the next lines of code
- `continue` : disregards the rest of the code in the loop block and moves on to the next item in the loop

2.5.6 Creating a Pokemon API Webapp

Now that we know a little bit about the tools of the web, let's build a simple web app that uses the skills we have learnt in this workshop, as well as some other skills we will learn along the way, to create an app that can do something cool.

For this tutorial, we will pay homage to the recent releases of Pokemon Brilliant Diamond and Shining Pearl, of which I have spent an embarrassing number of hours on since they came out about a week ago, and create a web app that uses the PokeAPI to display images and information about any Pokemon we want.

This tutorial will take place at 4:30pm AWST on the 31st of November, 2021 via the CFC Discord server. The recording of that tutorial will be here once it has finished.

Back to Top

Copyright Coders for Causes

2.6 Practical Software Engineering Practices

How software engineer works collaboratively!

2.6.1 How do software engineers work?

They use **issue/ticket management system** to track their work.

> Github Issues and Jira

The examples are Github Issues and Jira.

Copyright Coders for Causes

Listing of issues can be displayed in **kanban boards**.

Kanban Boards

Copyright Coders for Causes

2.6.2 Why do Software Engineers Work like this?

Scrums

Copyright Coders for Causes

One example of "agile" ways of working and it looks like this.

2.6.3 What is Git?

Git is a

- **version control system**
- tools for **coordinating work for different changes**

Main Parts of Git

Main Parts of git:

- Change Node (commit node)
- Branch

Copyright Coders for Causes

Git Different Versions

Software Versions

Copyright Coders for Causes

Copyright Code4 for Causes

Nodes - Representation of a change (commits + merge)

Main/Master: - Branch usually as production or deployed

Development Branch - Branch used for development Used for compiling features for a release

Feature Branch - Branch used for development for a particular issue

2.6.4 GitHub and Code Repositories

GitHub - Place to hold code repositories

Repositories - Place where codes are stored with different versions

2.6.5 INTERACTIVE WORKSHOP TIME !!!!

1. Do the following from codersforcauses/software-engineering-practices-demo: Demonstration of software engineering practices (github.com):
2. Look at the GitHub issues
3. Assign yourself an issue
4. Clone repo with `git clone https://github.com/codersforcauses/software-engineering-practices-demo-2021-summer.git`
5. Create a branch with the following format s{Issue Number}-{Issue Name}. `git checkout -b s{Issue Number}-{Issue Name}`
6. Create the change to satisfy or complete the github issue
7. Commit (package) the change `git commit -m "{insert message here}"` and push git push.
8. Create a pull request and request for a reviewer from CFC peeps

2.6.6 Other Resources

- Git and Github

Copyright Coders for Causes

2.7 Introduction to Frontend Frameworks

2.7.1 Introduction to Frontend Frameworks

An intuitive explanation of why frontend frameworks are used

Frontend Engineering

- Frontend Engineering, one of the most important aspect of web dev
- Feared by lots of developers (ehem ehem CSS)

Memes

Copyright Coders for Causes

Job Market and Trends

StackOverflow 2020

Can be seen that there is a huge portion that will needs frontend.

Note: Developers can identify as many types

Copyright Coders for Causes

Startups

Copyright Coders for Causes

GITHUB COPILOT AND DRAG AND DROPS

- Getting replaced by AIs, Drag and drops?
- Far from reality?

Github Copilot and Squarespace

Copyright Coders for Causes

Problems

CROSS-BROWSER SUPPORT

- Need to support multiple browsers

HTML IS WET!

Not DRY = WET

D – Don't

R – Repeat

Y - Yourself

Github Tags System

Copyright Coders for Causes

INEFFICIENCIES OF RELOADING

- Most parts of website don't change

Youtube

Copyright Coders for Causes

- What about with notifications and realtime updates?

notifications

Copyright Coders for Causes

Copyright Coders for Causes

STATE MANAGEMENT AND RERENDERING

- How do you handle changes in the state of the application?

Example "Spotimania"

Realtime Multiplayer Song Guessing Game https://spotimania.herokuapp.com/

Information

- Song picture
- Notifications when someone joined or scored
- Song title + artist

How do we handle when we "Submit Guess"?

HTML Format Submit Approach

1. Reloads = loses current state
2. How do you coordinate with players?

AJAX Calls

1. From frontend, use JS to do AJAX Calls
2. After receiving AJAX call, modify everything that needs the data (Find element by DOM, do some janky HTML inject)

Cross-site Scripting

Copyright Coders for Causes

Now, you have to implement your own strategy to do this.

When you try to do render by that approach:

- slower (rerendering things that should not render)
- tedious (find element and update)
- security
- code complexity
- hard to keep track of states (especially if different portion of the UI relies on a certain data)

How do we fix it?

CROSS-BROWSER SUPPORT

Babel: Transpiles / Translates New JavaScript to old JavaScript that is supported

Webpack: Bundles / Packs Bundles the JavaScript modules into a single minified file

Copyright Coders for Causes

SCSS Compilation

Translates concise and programatic version of CSS to something that can be understood by the browser

```
.example {
    display: grid;
    transition: all .5s;
    user-select: none;
    background: linear-gradient(to bottom, white,
black);
}
```

CSS

```
.example {
    display: -ms-grid;
    display: grid;
    -webkit-transition: all .5s;
    -o-transition: all .5s;
    transition: all .5s;
    -webkit-user-select: none;
       -moz-user-select: none;
        -ms-user-select: none;
            user-select: none;
    background: -webkit-gradient(linear, left top,
left bottom, from(white), to(black));
    background: -webkit-linear-gradient(top, white,
black);
    background: -o-linear-gradient(top, white,
black);
    background: linear-gradient(to bottom, white,
black);
}
```

Minification

Makes files smaller by removing unnecessary characters

Copyright Coders for Causes

```
var gapi=window.gapi=window.gapi||{};gapi._bs=new Date(
",")};return[encodeURIComponent(a.g).replace(/%2C/g,","
b[d].split("="),l=decodeURIComponent(g[0]),k=decodeURIC
ga=/\/cb=/g,fa=/\/\//g,la=function(){var a=F();if(!a)th
g&&Object.prototype.hasOwnProperty.call(c,e)&&g(c[e],a,
z){var v=c.callback;v&&(c.callback=function(){0==--z&&v
((D||{}).config||{}).update;d?d(e):e&&w(E,"cu",[]).push
I,k);k.push.apply(k,p);N("ml0",p,v);b.sync||f.___gapisy
gapi.load("",{callback:window["gapi_onload"],_c:{"jsl":
```

VERSUS

```
angular.module('memryMain')
  .directive('defPanel', ["defModel", "$timeout", funct
    'use strict';
    return {
      replace: true,
      restrict: 'E',
      templateUrl: 'views/def-panel.html',
      scope: true,
      link: function (scope, element) {
```

WHAT IF HTML IS NOT WET? WHAT IF HTML IS A PROGRAMMING LANGUAGE?

Memes about HTML as a programming language

Copyright Coders for Causes

Copyright Coders for Causes

What if HTML has variables?

- perform computations to be displayed in the interface (solves **State Management and Rerendering**)

What if HTML has loops?

- Eliminate repetitive code (solves **wet HTML**)

What if HTML has functions?

- Divide interface into modular components (solves **wet HTML**)
    - load only components that are not loaded (solves **Inefficiencies of Reloading**)
- interface determined by parameters
- Maybe we can write a function that will automatically update HTML when data changes? (solves **State Management and Rerendering**)

What if HTML has condition?

- Conditional rendering (solves Reloading to show different view)

What's next?

The next two workshop content are:

- React
- Vue

Copyright Coders for Causes

2.7.2 Introduction to React

Sparking the REACTion of User Interface and Experience

Decomposing the HTML User Interface

Complete code

Relevant Code for Discussion

1. Navigation Bar
2. Some Text
3. Carousel

HTML STORED IN VARIABLES

HTML STORED IN FUNCTIONS

Carousel Decomposition

The highlighted ones are the key-distinct data driven elements.

1. Carousel Indicators
2. Carousel Items

Copyright Coders for Causes

Now, what if we have a variable called `dogsData` an array of objects (may come from `fetch`).

ASSEMBLE THE COMPONENTS

or with JSX

Intro to React

Complete Modularised Code

Getting Started with React

Getting Started with the app

To get started with react, use the following commands

`npx create-react-app my-app` where my-app is the name of the folder for where the react app is going to be.

`cd my-app`

Notice that it created a couple of folders. In a couple of sentences:

`src/index.js` - injects the user interface in the public/index.html (the front part of the build)

`public` - folder containing the static assets as well as the index.html

`package.json` - dependencies and scripts for the app

type `yarn start` to run the development version of React (it will run `react-scripts` start within the node_modules … so running `react-scripts` start will not exactly be the same). This will serve up React in a port

To be left as a live coding exercise. Live coding demonstration:

Demo

- Passing Array props
- show “rafce”
- useState when adding new item

Copyright Coders for Causes

2.7.3 Introduction to Vue

A Vuetiful Introduction to VueJS

Due to time constraints, we had to reuse the previous Winter Materials.

As always, see the youtube video playlist

Copyright Coders for Causes

2.8 Welcome to the Djangol!

We've got puns 'n' data, We've got everything you request

2.8.1 How does an application communicate with backend systems?

JavaScript Object Notation

JSON

Link

Query this endpoint using JavaScript

Other ways

JSON is not the only way. There are things like SOAP, RPC, but nevermind them for now as you will most likely have to deal with JSON for the most parts.

2.8.2 What are APIs and REST-APIs?

Application Programming Interface

Analogy

Who interacts with the user interface? - the user Who interacts with the application programming interface ? - the application program (eg. the browser)

Representational State Transfer Application Programming Interface

- backend architectural pattern that follows the GET/POST/PUT/PATCH/DELETE
- Can be represented in Swagger/ Open API specification

Copyright Coders for Causes

Swagger/ Open API specification

2.8.3 What is Django?

Django

- Python web framework for creating server-side application

Copyright Coders for Causes

Follows MVC:

- Model - database
- View – Interface (API or User Interface)
- Controller – URLs + routes

See Documentation

2.8.4 What is Django REST Framework (DRF)?

- library for creating REST-API
- just makes it easier develop REST-API

In:

- Authentication + Permission
- Generic API Views
- Serialisers (payload validation and format)

See Documentation

2.8.5 Features of Django and DRF

Serializers

- converter to and from Python Data types to JSON/XML
- can be used to do validation

More info here

Models

Models

- representation of database connection or schema Example: Todo Django + React

MODELS WITH SERIALIZERS

- Model already defined the structure of the data

Configuration file `settings.py`

One of the most important file - determines the app configurations and default

Example of Items you can configure

- timezones
- start of URL
- installed Django Apps
- middlewares
- usually becomes the "default" configs after environment variables
- database configuration

Route with URLs

Determines the URL route of the API

- starts with the `urls.py` in the config (settings.py)

Copyright Coders for Causes

2.8.6 Packaging a "Django App"

"Django Apps" are plugin system of Django. Django codebase prefers packaging each endpoint feature to a "Django App"

> Authenticate with Microsoft

For example: django-saml2-auth-lw (used for Microsoft SAML SSO)

More documentation can be seen here

You can start creating your own Django app by following the steps:

1. `python manage.py startapp <app_name>`. This will bootstrap commonly used files for any "app".
2. Add the app in the `INSTALLED_APPS` in the settings.py
3. `python manage.py makemigrations` to create the migration scripts
4. `python manage.py migrate` to run the migration script

Views

The interface at which any application program interacts with to get something done in the backend.

> Use DRF instead of Base Django

This is where it gets a little tricky. Django accomodates both frontend and backend, but where it shines is with DRF. So always refer to documentation regarding write it from the documentation.

If you plan to use Django as an entire full-stack option, then you may not need DRF at all. Just use the generic views.

FUNCTION-BASED VIEWS AND CLASS-BASED VIEWS

> Codes

Summary: In DRF, there's a lot more advantage to use class-based views. However, if you want the flexibility and writing things on your own, then you might want to use function-based view.

> Function-Based View

Link:

> Class-Based View

To be seen in in the live demos

2.8.7 Django Admin

Django comes with a built-in user-interface for managing "Django Apps" with models.py By default it in the path `/admin`

> Django Admin

Copyright Coders for Causes

2.8.8 Other Tools

DRF Frontend for Consuming API

When you use DRF for developing API, and use the generic API views and serializers, it will automatically create this for you.

DRF Frontend

Copyright Coders for Causes

Django REST framework v2.0.0

User List

OPTIONS

API endpoint that represents a list of users.

```
GET /users/
```

```
HTTP 200 OK
Vary: Accept
Content-Type: text/html
Allow: GET, POST, HEAD, OPTIONS

{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "email": "admin@example.com",
            "groups": [],
            "url": "http://127.0.0.1:8000/users/1/",
            "username": "admin"
        },
        {
            "email": "tom@example.com",
            "groups": [],
            "url": "http://127.0.0.1:8000/users/2/",
            "username": "tom"
        }
    ]
}
```

Username

Email

Groups

POST

Postman

Development tool for interacting with APIs:

- environment variables
- collection of possible endpoints
- send different JSON body

Postman

Copyright Coders for Causes

Python Debugger (pdb)

When you run the django `python manage.py runserver`, you can add `breakpoint()` to anywhere piece of your code. This will stop the terminal line to the breakpoint if it hits it. From here you can do debugging.

Copyright Coders for Causes

2.8.9 Demo - Live Coding

Demonstrate how to create a backend for a Todo app.

1. Setup Python environment `python -m venv venv` then activate with `source venv/bin/activate`
2. Install Django `pip install Django`
3. Setup django codebase `django-admin startproject todo`
4. Install DRF
5. Freeze requirements `pip freeze > requirements.txt`
6. Do a python sqlite migration `python manage.py migrate` to initialise the Django database for the apps
7. Create a super user `python manage.py createsuperuser`
8. Run the django app with `python manage.py runserver`
9. Login with the super user and show Django Admin
10. Setup django app `python manage.py startapp api`
11. Create a model `models.py`
12. Do a python sqlite migration `python manage.py makemigrations` and `python manage.py migrate`
13. Show sqlite
14. Create serialiser `serializers.py`
15. Create Class-based view `views.py` and link back to the `urls.py`
16. Manually test with DRF Frontend or postman

Final code can be seen here.

Copyright Coders for Causes

3. 2021 Winter

3.1 Coders for Causes 2021 Winter Workshops

This project period there are two main projects:

- Foodbank
- WAIS

If you have not before seen the existing progress, see this video.

These two projects have their own corresponding technology stacks being used, hence will dictate the workshops that will be held.

3.1.1 Project Technology

Foodbank

Foodbank is mainly with frontend with React + TailwindCSS with Firebase and Notion CMS.

WAIS

WAIS is a full-stack application with Vue and Django. It uses Docker containerisation for both development (and production in the future).

3.1.2 Where are the materials?

This website has only been created prior to 2021/22 Summer workshops. However, you can find the videos in our Youtube channel, and workshops slides in google drive.

Copyright Coders for Causes

https://github.com/codersforcauses/workshops/

Copyright Coders for Causes