Curriculum Braaains Taste Test

Greetings! Thanks so much for your interest in my curriculum. In the following pages you'll find the curriculum documents for four different lessons pulled from four of the seven different modules currently available. The four lessons here are

- **ADTL-01: Manipulating Undead Files** – Covers: *How to work with Files and Folders*
- **01-02: Zombie Headings Will Roll and Listing the Undead** – Covers: *<h1> through <h6>, <ol>, <ul>, <li>, <dl>, <dt>, and <dd>*
- **02-01: Zombie Lipstick, Rules of CSS Warfare, Stopping the Spread of the Contagion, Selecting Your Target, Stay Classy, Zombies, and IDing Your Patient Hero** – Covers: *CSS rules and properties, Applying CSS to HTML and CSS selectors*
- **06-02: Suit Up for Your First Zombie Fight and Flexible Layouts—A Vice around a Zombie's Head** – Covers: *Viewport meta tag, overflow-x property and flexible layouts*

Slides for each lesson were also included.

Each of the seven modules (except the ADTL module) is based on a book. The modules available are

- ADTL — additional understanding and/or addresses baseline concepts that aren't directly covered in the books (using files, the client/server model, domain names, hosting, using code editors/CodePen, HTML and CSS debugging, browser developer tools, and a paper craft take on rubber duck programming).
- 01 — *A Beginner's Guide to Learning HTML (and Smacking Zombies Upside the Web Development)*
- 02 — *Beginner CSS: Like Putting Lipstick on a Zombie*
- 03 — *Beginner Usability: A Novice's Guide to Zombie Proofing Your Website*
- 04 — *HTML5 Forms & Interactive Element (Or How to Poke a Zombie in the Eye)*
- 05 — *Intermediate CSS: Zombie in a Cocktail Dress*
- 06 — *Responsive Design: An Undead Introduction to Mobile Web Development*

If you have questions, comments, snide remarks, or constructive criticism, don't hesitate to send me a note at feedback@undead.institute. My goal is to make a teacher's life easier, more fun *and* less zombie filled. Please let me know if there's anything I can do to further those goals.

Thanks again and let me know how it goes.

At your service,

John C. Rhea
Chief Zombie Smacker, The Undead Institute

ADTL-01: Manipulating Undead Files

In this lesson, students will learn how files and folders work on their computers.

Question of the Day

How do you manipulate files?

Learning Objectives

Students can

- Identify files and file types
- Identify folders

Pacing

20 minutes Instruction

Vocabulary

Operating system: The software you interact with to run programs (e.g., Windows, Mac OSX, Linux, etc.)
File: A unit of data in a computer, designated by a file name and file extension. This could be a picture, document, piece of audio, piece of video, etc.
Folders: Kind of like a bucket that you can put files and other folders into

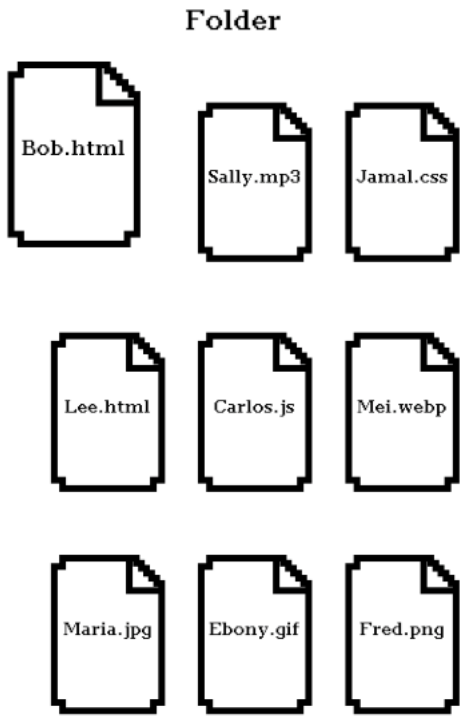
Materials and Preparation

ADTL-01 slides

Section	Description
Launch	**Terrible Zombie Joke** Name a zombie's favorite cooking show. *What is Jeopardy!?*
Instruction	Discuss what files and folders are. **A Horde of Files** Before we talk about code, let's talk about files. You can think of a computer file as a single zombie. That zombie has some information, like which horde he's in and perhaps a poem or two if he's a more artistic zombie. Each zombie is part of a zombie nest (i.e., the group of zombies that shuffles after humans together). A nest is kind of like a computer folder. It's a way of structuring and gathering zombies together to make them easy to find. For example, if you're Bob the supreme zombie commander in charge of shuffleboard tournaments and defeating humanity, you need to know where you can find Jamal, Fred, and Maria. If you know which nest they are in, it's much easier to narrow your search to that particular nest. You'll search through twelve or fifteen zombies for Jamal, Fred, and Maria, rather than searching through 12.5 million zombies for Jamal, Fred, and Maria (plus you're less likely to find the same or similarly named zombies in the nest). Now, if you had a powerful zombie search tool that could quickly and easily pull up Jamal, Fred, or Maria out of the 12.5 million zombies, you wouldn't need to narrow your search to a nest or folder. While these types of search tools make life much easier for Bob (and for you when you're trying to

find a file), in web development, we don't usually have the luxury of searching like that on a page-by-page level.

A web page itself is like a zombie nest commander who can only find the zombies (files) in his nest (folder). If the zombie isn't a part of his nest, he can't find it. The files of a web page have the same limitation. They can easily find other files in their same folder, and they can even look for files (zombies) in a different folder (nest), as long as you tell them which one to look in, but unless you localize the search to a particular folder (nest), they won't know how to find it.



01-02: Zombie Headings Will Roll and Listing the Undead

In this lesson, students will learn how headings and lists are used to format text in HTML to make the output easier to read.

Question of the Day

What are ways we can structure content?

Learning Objectives

Students can

- Write and manipulate h1–h6 elements
- Write and manipulate ol, ul, and li elements
- Create description lists

Pacing

5 minutes Launch
20 minutes Instruction
20 minutes Activity
5 minutes Exit Ticket

Vocabulary

Headings <h1>, <h2>, <h3>, <h4, <h5>, and <h6>
Ordered list
List Item
Unordered list
Description List <dl>
Description Term <dt>
Description Details <dd>

Materials and Preparation

A Beginner's Guide to Learning HTML (and Smacking Zombies Upside the Web Development) (PDF), pages 14–21

01-02 slides

Use Your Head
CodePen
File

Things that Will and Won't Kill Zombies
CodePen
File

Things that Need New Definitions
CodePen
File

Section	Description
Launch	**Terrible Zombie Joke of the Day** What do you call a man who keeps all the zombies in his basement? *A horde-er.* **Journal Scenario** A zombie horde is headed toward your house. Make a list of three to five things you would grab before you evacuate. Put the most important thing at the top and the least important thing at the

	bottom. Make a second list of three to five things, in no particular order, that you'd want to leave behind when you evacuate. No matter how annoying they might be, you may not put siblings on the list. Lastly, create a list of things that you might need to describe after the apocalypse (e.g., the types of zombies that exist and how to avoid and/or fight them). Give each list a title.
Instruction	Discuss the concept and hierarchy of headings and subheadings in documents. Just like in a Word document or other paper, each heading should describe the content below it, and subheadings should break up the content of the higher heading level. Explain HTML's six heading levels and the default formatting and relationship between them (bold and each heading level is smaller than the last, with h1 being much larger than body text and h6 being a similar size than body text). Discuss what a list is and the difference between ordered and unordered lists. Show them the ol, ul, and li tags and how lists items are shared between ordered and unordered lists. Show them how an ordered list's numbers always appear, even if you move items around. Show them an unordered list and the bullets the browser automatically applies. Go over creating a description list and what each of the three tags (dl, dt, and dd) do. dl is the list itself, dt, or description term, is the thing being described, and dd is the description details or the words used to describe the term.
Activity	**Student or Teacher and Student Activities:** *Use Your Head* CodePen File *Things that Will and Won't Kill Zombies* CodePen File *Things that Need New Definitions* CodePen File **Student Activities:** Students will code their lists from the "Launch" section, including a heading for the titles they gave to each list.
Exit Ticket	**Think-Pair-Share** • How important are headings to content online? • Where do you see lists used online? • What role might headings and lists play in making web pages easy to read?

02-01: Zombie Lipstick, Rules of CSS Warfare, Stopping the Spread of the Contagion, Selecting Your Target, Stay Classy, Zombies, and IDing Your Patient Hero

In this lesson, students will learn what CSS is, how the syntax works, and how to select elements from a web page.

Question of the Day

How can we make visual changes to an HTML page?

Learning Objectives

Students can

- Recognize the parts of a CSS rule
- Understand how CSS is applied to HTML elements
- Select elements using a class
- Select elements using an ID

Pacing

5 minutes Launch
20 minutes Instruction
20 minutes Activity
5 minutes Exit Ticket

Vocabulary

The cascade
Cascading style sheets (CSS)
Rule
Selector
Declaration
Class attribute
Id attribute

Materials and Preparation

Beginner CSS: Like Putting Lipstick on a Zombie (PDF), pages 8–17

02-01 slides

Rules of Engagement
CodePen
File

Section	Description
Launch	**Terrible Zombie Joke** What's a zombie's favorite sport? *Shuffleboard.* Show a few of the examples on the http://www.csszengarden.com site. Ask the students how they think these different sites were accomplished? Finally, reveal that all of these pages have the exact same HTML. The only difference is the CSS file used.
Instruction	Give an overview of what CSS is and discuss the separation of concerns (i.e., HTML is the structure, CSS is the look and feel, and JavaScript is the logic and interactivity). **Note:** As the web moves forward, these lines are blurrier than they used to be, but they still largely hold true.

	Say you want to choose part of an HTML page to turn a sickly yellow-green color (you're hoping to gross out the zombies). How might you tell the browser which parts you wanted to change the color of? Discuss CSS rules and the two major pieces of a rule: the selector and the declaration block. The declaration block can be further broken into a series of properties and values (e.g., color: red). Discuss the three major ways of associating CSS with HTML: 1. Direct injection: style attribute directly on an HTML element 2. Page injection: style element within the <head> element (to reduce the number of files involved in examples, we'll mostly use this method) 3. Site injection: CSS file connected with a <link> element Be sure to also mention the differences. For instance, let's say you had a horde (say thirty or more) of p elements that had a style attribute with color: green. It would be great till your teacher asked you to change the color of the text to red. Then you'd have to go into more than thirty p elements to update each and every one. If, however, you had a style element with a CSS rule that selected p elements and set their color to green, a change to red would involve you updating a single line. Similarly, if you had more than thirty pages, each with more than thirty p elements, you'd have to go into each page and update that line. If, however, each page referenced a single CSS file, you could make the change in one place, and it would cascade through every page to every p element. Discuss class attributes and how they signify a class or grouping of elements. They do not even need to be the same type of element. Discuss ids and how they must be unique on each page. Discuss how to tell the browser that it's dealing with a class versus an id in the CSS: a period for classes; a hashtag for ids. Last, a CSS comment begins with /* and ends with */. The browser will not display anything between these two endpoints. You'll see these used extensively in the activities to give you directions and help you play with the code.

Activity	**Student or Teacher and Student Activities:**
	
	Rules of Engagement
	CodePen
	File
	
	Student Activities:
	
	Pull up the answers from *Assembling the Undead Monster* (from 01-07) or any other web page you've built (or use this *Assembling the Undead Monster* answer key: CodePen HTML file). Add a style element in the head (*Note:* the style element is not needed if using CodePen, just add CSS to the CSS box) with a CSS rule such as:
	
	`<style>`
	`p {`
	`  color: darkmagenta;`
	`}`
	`</style>`
	
	Change the selector used, try any other element in your HTML page as a selector (as long as that

	element has text in it; elements in the head won't change anything if you select them.) You can also try changing the color. Most common colors work (e.g., red, green, blue, white, purple, yellow, etc.). Within the style element, try adding a second CSS rule with a different selector and a different color value. Add example class and id attributes, and change colors using class and id selectors.
Exit Ticket	**Think-Pair-Share** • Consider your school's website. What things about an HTML page might you be able to change with CSS? • How might you want to change your school's site?

06-02: Suit Up for Your First Zombie Fight and Flexible Layouts—A Vice around a Zombie's Head

In this lesson, students will learn to prepare their sites for responsive design and create flexible layouts.

Question of the Day

How do we prepare our sites for responsive design?

Learning Objectives

Students can

- Add a viewport meta tag
- Set overflow-x on body and html elements
- Build flexible layouts

Pacing

5 minutes Launch
20 minutes Instruction
20 minutes Activity
5 minutes Exit Ticket

Vocabulary

Meta element
Overflow-x: Like the overflow property, it handles how content too big for an element is displayed, but only affects the horizontal/x direction (overflow-y handles the y direction)

Materials and Preparation

Responsive Design: An Undead Introduction to Mobile Web Development (PDF) pages 14–19

06-02 slides

Hiding the Zombie Overflow
CodePen
File

Flex Those Zombie-Fighting Muscles
CodePen
File

Weird Flex, Zombie, but OK
CodePen
File

Section	Description
Launch	**Terrible Zombie Joke** How can you tell if the dealer at your weekly poker game is a zombie? *If he shuffles with his feet.* Which is easier to read? (Show images of a zoomed-out page as well as one that's a regular size; i.e., with a meta viewport tag and without. To prepare these it may be easiest to take a screenshot of the site in question at desktop size and then just shrink it down.)
Instruction	Discuss meta viewport tags, overflow-x: hidden, and creating flexible layouts. By default, a website viewed in Safari on an iPhone will zoom out so that the entire website is visible, and then you can zoom in to the part of the website you want to see. This was a game changer when

	it first came out, as it made websites almost usable on a phone. (Before that, it was a mix of broken mishmash on what we called a "feature" phone.) However, when responsive design came along, we no longer wanted to zoom out, as we optimize the site to show at its natural size rather than zoomed out and so tiny as to be unreadable. To fix this and tell the browser not to zoom out, we have the viewport meta tag. <meta name="viewport" description="width=device-width, initial-scale=1"> **Flexible Layouts** Another thing to look out for is any element wider than the size of the device window. If you've ever tried scrolling on a touch screen by flicking when the site could horizontally scroll, you know how annoying this can be. One way to make this far less likely to happen is the overflow-x property. You can set it to hide anything that overflows the body element like this: body { overflow-x: hidden; } For greatest compatibility add it to the html element too html { overflow-x: hidden; } Having flexible layouts is essential to good responsive design. At the dawn of the Internet epoch, text expanded to the width of the page (mostly because we didn't know what we were doing). As screens grew and layout became more sophisticated, we began to make rigid designs using tables that fit the screen size du jour. (When you've got three total sizes, that's not a big deal. When you've got three hundred, or even just thirty, it's a much bigger deal.) Setting an exact width on any layout is unlikely to appear correctly on most devices. Thus, setting a flexible layout that expands or contracts based on the size of the browser or device can be super helpful and prevent you from having to constantly readjust your layout. All that's required is to use percentages in your widths. So, instead of setting a width to 800px or 50em, you would just set it to 100% or if you had two elements that should appear there, their percentage widths added together should be 100 percent (this assumes box-sizing: border-box;). *Note:* It's possible that you wouldn't want text to go full width (e.g., when the width of the containing element is over 1000 pixels). To prevent this, you can add a max width on the element and set it to the maximum width you're okay with. The element should stop growing at that width and should be flexible when the browser or device is thinner than that width.
Activity	**Student or Teacher and Student Activities:** *Hiding the Zombie Overflow* CodePen Files demo *Flex Those Zombie-Fighting Muscles* CodePen File **Student Activities:** *Weird Flex, Zombie, but OK* CodePen File

Exit Ticket	Show the finished product of *Weird Flex, Zombie, but OK* at 300 pixels wide