Lessons!

LESSON 1

Technological Typography

WHAT IS A WEBSITE?

Everyone's a little bit of a graphic designer. That includes you, even if you don't know it. At some point, you've probably formatted a document, and you had to make some decisions. You decided what font to use, how big to make it, whether to set it left, right, or center aligned, and so on. While you were making these decisions, you were probably focused on making your document look pretty, or professional, or cool. And furthermore, you probably didn't have to try very hard to make these decisions. It's easy to digitally edit a text document, right? Just click a few buttons, select a couple of options, and you're good to go.

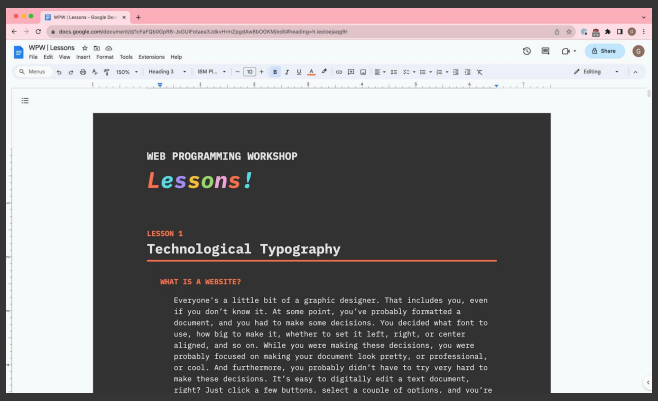

FIGURE 1.1

I've made many design decisions while writing this document in Google Docs.

When you finish designing your document, you might print it. You might save it as a PDF, or as an image. But once it's done, it's done. If everything goes right, the document's layout won't change ever again.

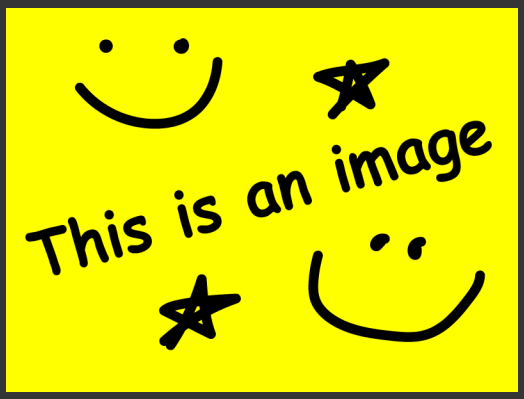

FIGURE 1.2

This is an image. It's always going to look like this, as unforunate as that may be.

A lot of the time, websites look like the documents that you're familiar with. Sometimes, they're even designed to emulate the experience of a printed document.

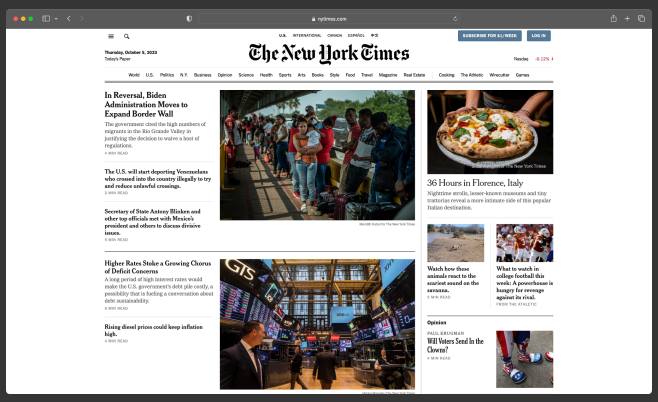

FIGURE 1.3

The homepage for The New York Times borrows its design from the newspaper's print edition. Visit the website

However, websites face a little conundrum that printed documents don't face. And when I say little, I actually mean earth-shattering.

FIGURE 1.4

The same homepage for The New York Times, but in a smaller browser window.

Websites don't have the luxury of just being one thing. While a PDF or image is set in stone, a website might be viewed on a phone, tablet, or desktop. Even worse, there isn't one set size for each of those devices.

For a long time, we didn't care much about this problem. Home computing was still in its infancy, and smartphones were a long way off. Web design from that era reflects this — web developers used what limited tools were available to make websites that just worked.

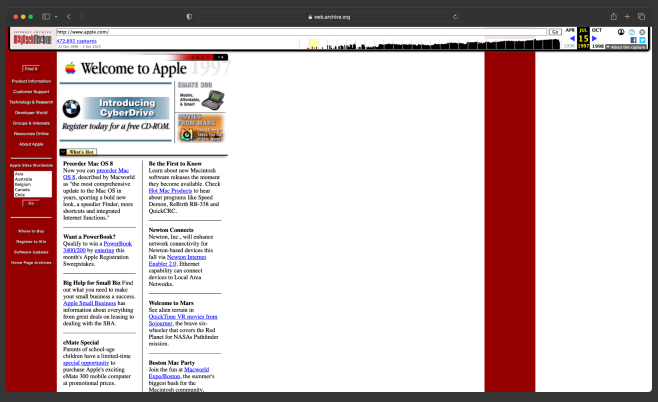

FIGURE 1.5

Apple's homepage from July 15, 1997. View on archive.org

We live in a different era. Today's websites are fluid, adapting for multiple screen sizes, resolutions, aspect ratios, and types of input. We have way more people to account for, and they're all using radically different devices. At the same time, we have much better tools available to us to create these websites.

It’s easy to gloss over the fact that websites are an invented medium. If you haven’t worked with code before, it might seem that programming languages are some sort of natural phenomena — they just seem to exist. In fact, programming languages are designed, by humans, in the same way that you might design a document. Let’s say you need your document to look fancy. What if your text editor has no calligraphic, swirly fonts? If enough people have that problem, then there might be a new version of your text editor that features fancy fonts.

Websites started as a place to simply communicate information.

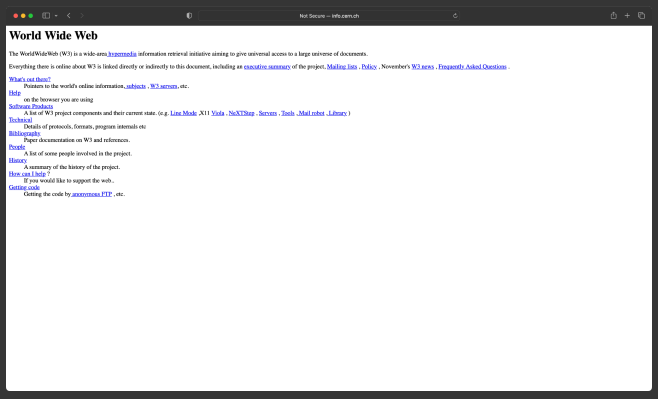

FIGURE 1.6

The first website ever created. Visit the website

Now, as you’re well aware of, they’re a place to do so much more.

BABY STEPS

That was all fine and dandy, but we didn’t really talk about what a website is, did we?

This is a website.

DEMO 1.1: A website

It’s not much. In fact, it is the bare minimum of what we might call a “website.”

A website has to contain at least one file. This file has to be an HTML file, which ends with the extension of “.html”. **HTML**, aka HyperText Markup Language (you don’t need to remember that), is a programming language. It’s the main programming language for websites, and the only one that’s required.

That first demo was an HTML file. I can prove it! Open up the demo again, and download it. Then, drag that file into your browser.

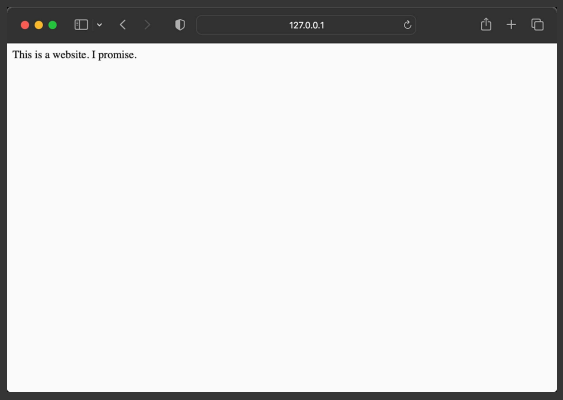

FIGURE 1.7

Web browsers read and display HTML files.

An HTML file is a file that contains HTML code. Code is just text, but it's text without any formatting. This means that the file doesn't contain any information about the text's font, color, alignment, etc. Instead, it's what we call **plain text**.

DEMO 1.2: Let's talk about plain text

Notice that in our code editor, we have the plain text on the left and a preview of the website on the right. This is how websites work. You write code, and then it gets interpreted by a web browser and displayed properly.

DEMO 1.3: Let the web browser do the work

Well, displayed properly only if your code is written properly.

DEMO 1.4: Let the web browser, uh, do the work?

This is what we would call the **syntax** of a programming language. Computers, despite their marvels, can't think. They take instructions, and then they spit something out. Every programming language has its own syntax, or its own way of writing instructions for a computer to understand. Most of the time, if the computer spits out something wrong, then it's, uh, your fault for not following this syntax.

Let's do a classic exercise to see what it means to write code.

EXERCISE 1.1: Tell me a picture

Instead of saying that writing code is hard, it feels more appropriate to say that writing code is tricky. The hardest part about learning your first programming language is learning both how easy it is to break your own code, and how easy it is to fix it.

When we're coding in HTML, we are usually creating things called **HTML elements**. This is the syntax for telling the web browser that a heading is different from a paragraph, which is different from another paragraph, and so on. Most HTML elements use the following format:

- An **opening tag** indicates the start of an element. This is some text enclosed in angle/alligator brackets (< and >) that indicates the type of the element. For example, the opening tag for a paragraph element is .
- A **closing tag** indicates the end of an element. This is the same as the opening tag, except with a forward slash (/) added right after the first angle bracket. For example, the closing tag for a paragraph element is .

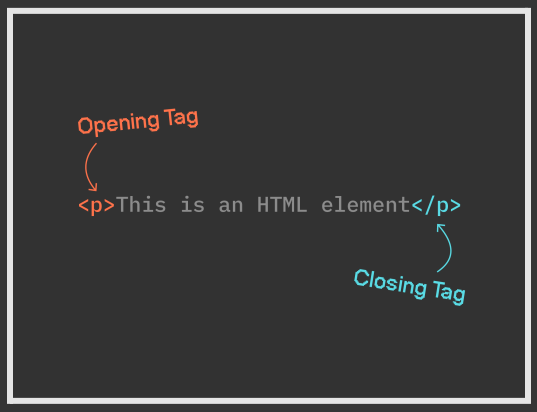

FIGURE 1.8

The standard syntax for creating an HTML element. This one is a paragraph!

HTML elements can be tricky because even a single typo can break your code. Luckily, we have systems for tracking down typos.

For instance, what happens if I forget an angle bracket?

DEMO 1.5: Missing angle bracket

Or, what happens if I name the closing tag wrong?

DEMO 1.6: Wrong closing tag

Or, what happens if I forget a closing tag entirely?

DEMO 1.7: Missing closing tag

Or, what happens if you intersect HTML tags (instead of nesting them)?

DEMO 1.8: Intersecting tags

Code editors are built to help you find and correct typos. Editors will help you out by color-coding your code, showing you where things break your programming language's syntax. This is partially while file extensions (like .html) are so important — they tell your code editor what language to check your syntax against.

Speaking of other languages, there is another one we should talk about.

MAKE IT PRETTY

HTML is one of the three programming languages that web browsers understand. The other two are CSS and JavaScript. JavaScript will be useful much later on, but CSS is almost as fundamental as HTML is.

CSS, which stands for Cascading Style Sheets, is the language that lets us make HTML pretty. CSS is different from HTML because it follows a completely different syntax.

So, what does CSS do?

DEMO 1.9: Make it bold

This is still HTML, but there's CSS hiding in the background that's telling our web browser to bold some text. CSS controls every single design decision on a website, including font weight (which determines if text is bold or not).

DEMO 1.10: Make it bold, but different

We can manually define an element's styles using inline CSS. **Inline CSS** lets us define CSS properties for a specific HTML element.

We also just discovered the **"span"** element, which is a sort of generic version of the "strong" and "em" elements we saw in previous demos. To bold text we use **"strong"**, to italicize text we use **"em"** (short for emphasized), and to do whatever we want to text, including bolding it or italicizing it, we use "span".

Let's look at another example.

DEMO 1.11: Make it pretty

We can combine multiple CSS properties to make more complicated designs. CSS follows a different syntax from HTML:

- First, we type the CSS property we want to set. A **CSS property** is any part of an element's design, like font weight, font size, font family, and so on. CSS properties can't contain spaces, so font weight would be "font-weight". There's a huge set of properties we can define, and you'll gradually learn more as we keep going.
- This property name is followed by a **colon** (:).
- Then, we specify the property's value. This depends on what property we're setting.
- If we want multiple properties, we separate them by a semicolon (;).

FIGURE 1.9

There is no limit to the number of CSS properties on a given element. Don't get your colons (:) and semicolons (;) mixed up!

We can embed CSS into HTML via the "style" attribute. **HTML attributes** live in the opening tag of an HTML element and follow a specific format:

- Start with the name of the attribute. For inline CSS, we use the "style" attribute.
- Follow that with an equals (=) sign.
- Then add either a single (') or double (") quote to start adding content to the attribute.
- Lastly, close the attribute with a single (') or double (") quote to match the first quote.

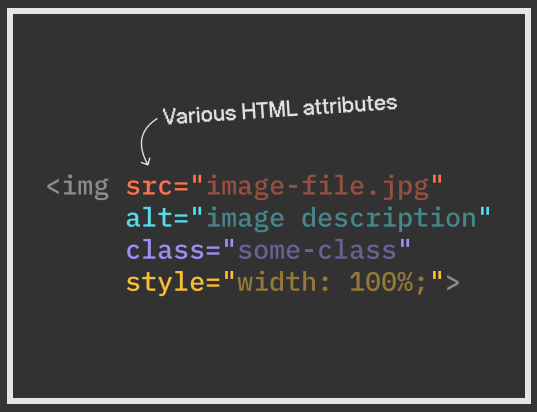

FIGURE 1.10

There are many useful HTML attributes, but we'll just be working with the "style" attribute for now.

EXERCISE 1.2: Programmatic poetry

MAKE IT MAKE SENSE

____.

- Different kinds of elements
- Block elements take up full width
- Inline elements dont
- Span tags are inline elements that let us apply styles inside of block elements
- But dont intersect elements
- **Programmatic poetry exercise (geronimo stilton example)**

- Semantics section
- Look at whole slew of html elements
- See how they function

- Introduce the div element
- Exercise: typeset a wikipedia article

- Next lesson
- Talk about internal css vs inline css
- First, apply css to all elements of a type
- Then, be more specific with a css class
- Even more specific just to show it
- Css class + inline css
- Color codes?
- Box model
    - Margin, padding, border
- Absolute positioning
    - Talk about absolute positioning inside of absolute positioning
    - Units
- Self portrait exercise

FIRST HW ASSIGNMENT:

- Create a visual narrative using only HTML elements (NO CSS ALLOWED)
- Research + analysis

DEMO 1.1: A website

When

DEMO 1.5: Nesting HTML elements

We'll often place HTML elements inside of other HTML elements, either as a method for organizing information, or styling specific bits of information.

DEMO 1.5: Nesting HTML elements

When we do this, it's important to make sure we don't intersect HTML elements. Otherwise, we end up with broken code.

DEMO 1.6: Nesting HTML elements

Simple enough, right? Well, notice that I said "most" HTML elements follow this format. Just like spoken languages, programming languages have exceptions to their many rules.

Let's take a look at a variety of commonly-used HTML elements and see how they differ.

DEMO 1.5: Commonly-used HTML elements

It's obvious what some elements do, and less obvious what others are for. Headings are available in six levels, ranging from most important to least important (think subheadings). Paragraphs are paragraphs.

But "strong" elements are for... bolded text? And "em" elements are for italicized text? Do "span" elements do anything at all? What about "div" elements? And what is this weird formatting used for "img" elements? (And why don't "img" and "br" elements have closing tags?!)

We're gonna approach these questions one at a time. But to get started, we should introduce CSS.

What is a website?

Oh, that's easy. This is a website.

DEMO 1.1: A website

From one perspective, websites are just files full of text. We call this text "plain text" because the file doesn't contain any formatting information for its content. This means that the text is just text — it doesn't have a font, color, weight, size, or anything of the sort. It is just text, and that text is code.

For websites, that code is specifically HTML code.

What is HTML?

DEMO 1.2: HTML

HTML is the main language that websites use to display content. It's different from human languages because it's intended for web browsers, not humans. A web browser can't interpret language based on context clues like we do, so that means that HTML has to be extremely precise. This precision appears in the form of HTML tags.

DEMO 1.3: "strong" (bolded) text

HTML tags split up plain text into various distinct elements. In the previous demo, we witnessed our first HTML tag — the "strong" tag. This tag tells the browser that the text contained within it is supposed to be bold.

What if we wanted our text to be italic?

DEMO 1.4: "em" (italicized) text

Easy enough! What if we wanted our text to be bold AND italic?

DEMO 1.5: Bold AND italic

HTML elements are (usually) marked by an opening and closing tag. The closing tag features the same name as the opening tag, with an extra backslash (/) to indicate that it isn't starting a new element. We have to be a little careful to not mix up or overlap our HTML tags, but we can place an HTML tag inside of another one.

It seems like we have enough information about HTML to start laying out a document. Let's try it!

DEMO 1.6: A failure of a document

Well, that didn't work. HTML code doesn't care about line breaks. Let's give that one more go.

DEMO 1.7: Less of a failure of a document

We found a solution! The "br" tag provides the line break we were looking for. It's also one of those funny elements that doesn't require a closing tag.

Now that we have some tools for controlling structure, what if I wanted to do something more stylistic?

DEMO 1.8: Paint it red

Wahoo! We made some text red!

We made that happen through another programming language that web browsers can understand. This language is called CSS and works with HTML to style elements. Think of it this way: HTML is about structuring content, and CSS is about making that content look pretty.

We can embed CSS into HTML tags through something called the "style" attribute. This lives inside of the opening tag and styles any elements within the relevant HTML element. When we apply CSS this way, we call it inline CSS. In the previous demo, we used the "span" tag to let us apply some inline CSS without messing up the structure.

Let's take a look at another example of inline CSS.

DEMO 1.9: Another way to bold

In this demo, notice how our strong tag became unbolded. This is because "font-weight" is just a CSS property, and any CSS property can be manually defined or overridden.

Alright, you're doing great. Let's try making something pretty with code.

EXERCISE 1.1: Programmatic Poetry

SEMANTICS

There's one big caveat to what we've learned so far. Even though we can make anything look like anything with CSS, we should still keep our code organized with HTML tags. This means that headings should be marked with heading elements ("h1" to "h6"), paragraphs should be paragraph elements ("p"), and bolded or italic text should be contained within the "strong" and "em" tags from before. We call these semantic HTML elements because they clearly describe the type of content inside of them.

Let's take a look at how we might structure an article using semantic HTML tags.

DEMO 1.10

This structure is looking solid. We're even seeing some new elements, like ordered and unordered lists. And, we're starting to see that there are default CSS styles set for different elements. Headings are bigger and bolder than paragraphs, and lists are marked with bullet points.

But now I have a big design request — what if I wanted everything to be set in a sans serif font?

Well, one way is to add inline CSS to every single element.

DEMO 1.11

That is a huge pain though, because now I want all of my text to be green. Oh, and I might want it to be larger later on. I'm not sure yet; I want to see it both ways before making a decision.

Luckily, we've got another tool up our sleeve that'll make our lives way, way, WAAAAY easier.

DEMO 1.12

When you apply CSS to a main element (a.k.a. parent), all of its sub elements (a.k.a children) will inherit the same styles. This is actually where CSS gets its name, Cascading Style Sheets, because of this cascading-down effect.

The "div" tag is a sort of universal "anything" tag for structuring content. You can put any other HTML elements into it, including other div tags. Anything contained inside a div tag (the child elements) will inherit the CSS applied to the div tag (the parent element).

Div tags are also useful because they let us tell the web browser what content is contained where. We can indicate that through the “id” attribute.

DEMO 1.13

The “a” element (which gets its name from the word “anchor”) is how we link to content in our website. In this case, we are using it as an anchor link, jumping us to a section contained within the same web page. We can similarly use it to link to somewhere else.

DEMO 1.14

Let’s put all of that knowledge into practice!

EXERCISE 1.2: Wikipedia, By You

IN CONCLUSION

In this lesson, we explored the basics of HTML and how to incorporate inline CSS to start styling elements. In the next lesson, we’ll delve deeper into CSS by working with classes.

LESSON 2

A Class on Classes

REDUCE, REUSE, RECYCLE

No matter what you're coding, you're going to be juggling large amounts of code at once. "Complicated" code can be mathematically and logically complex, but it can also just be hard to keep track of.

When CSS gets complicated, it's usually because there's too much to keep track of, not because anything individually is overly complex.

So far, we've been using inline CSS. This lets us embed CSS directly into HTML elements, but it also means that we have to keep track of CSS styles per element. That's super annoying. Luckily, we have CSS classes!

DEMO 2.1

CSS classes are groups of CSS styles that can be applied to multiple HTML elements simultaneously. There are a few reasons why we'd want to use classes instead of inline CSS:

1. CSS classes are centralized. You don't have to search through your code to find out why something looks a certain way. Instead, just look at its class!
2. CSS classes are reusable. If you want multiple elements to have the same styles, then just give them the same class!

Unlike inline CSS, classes have to be defined in a new HTML tag called the "style" tag. This is an invisible HTML element that lets you write CSS code directly in an HTML document. (We'll look at CSS files in a later lesson.)

The "style" tag is also special because it has to live in another invisible HTML element — the "head" tag. This lives at the top of your HTML document and defines your site's metadata.

There's a whole slew of other metadata we can include in the head tag.

DEMO 2.2

Now that we know what classes are, let's put them into practice.

EXERCISE 2.1: Wikipedia, By You, Too

SPECIFICITY

When you use CSS, you might either accidentally or on purpose define the same property twice for one element. Maybe, you say that element should be red, and then you say that it should be blue. Which color will the element end up as?

DEMO 2.3

The answer to this question is CSS specificity. When an element has conflicting styles, the most specific style will override any other styles.

This means that if an element has inline CSS and a class, its inline CSS will override any styles from that class. Simple enough, right?

Let's pretend it is that simple for now. Now, let's look at how we might want to actually combine classes with inline CSS.

So far, we've looked at how to style text using CSS. What if we wanted to create shapes instead?

DEMO 2.4

Remember the "div" element? Since it isn't a semantic HTML element, it can technically be anything. We just need to give it some CSS and make it shine.

We can combine our shape's class with some inline CSS to make a variety of forms.

DEMO 2.5

THE BOX MODEL

We're now verging on a big concept — the CSS box model.

The box model determines the size and shape of all HTML elements. Here's what you need to know:

- An element's "margin" is outside of it.
- An element's "padding" is inside of it.
- An element's "border" is between its margin and padding.
- An element has a "width" and "height", which are often determined by its parent element.
- All of these properties are defined using various units.

There's a lot to keep track of, but luckily these rules apply to virtually all HTML elements. Learn it once and you won't forget it!

DEMO 2.6

But, this only determines the shape of an element. What about where that element goes?

In almost all cases, we actually want that element's position to be automatically defined by the parent element, or by the web page itself.

That being said, if we're making art we might want to create something more deliberate.

EXERCISE 2.2: CSS Self-portrait

COMBINATORS

OK, so specificity is not always simple. I lied before. Sorry.

As I mentioned before, parent elements have a lot to say about what their child elements are doing. One way that happens is through class combinators.

Combinators let us extend a class so that it affects the children of an element, not just the element itself.

DEMO 2.7

In this case, only the paragraphs inside the "div" are red.

This is too complicated for now so I'm going to end this lesson here.

IN CONCLUSION

In this lesson, we used CSS classes to make our code more efficient and maintainable. We also found ways to use classes with inline CSS, explored the CSS box model, and got more specific about what our classes should apply to. In the next lesson, we'll learn how to take these ideas off of this website and into the real world.

LESSON 3

Coding on Your Own (Device)

FILES AND FILES AND FILES

PROJECT 3.1: Small Sites; Big Stories, pt. 1

IN CONCLUSION

LESSON 4

Going Live

SECTION

____

LESSON 5

The Click of a Mouse

SECTION

____

LESSON 6

Making Your Code Dance

SECTION

____

LESSON 7

No Screen Too Small

SECTION

____

LESSON 8

Grid Systems in (Web) Design

SECTION

LESSON 9

Flexible Rows and Columns

SECTION

____

LESSON 10

Inspecting the Web

SECTION

____

LESSON 11

If This, Then That

SECTION

____

LESSON 12

One Fewer Click

SECTION

____

LESSON 13

On an (Set) Interval

SECTION

____

LESSON 14

End the Loop

SECTION

____

LESSON 15

A Set of Stuff

SECTION

____

LESSON 16

Coding with Code

SECTION

____

LESSON 17

A World of Code

SECTION

____