2.6.1 Debugging Your Code

As a great coder once said, "Why isn't my code working?!"

At some point, nearly every coder has uttered these exact words. Coding is far from the effortless work of writing line after brilliant line of new features and visual effects. It is often a slow and cumbersome process—one that will resist your every effort to make your code work as you expect. You should brace yourself for the reality of spending hours (if not days) chasing down missing commas, misplaced parentheses, and erroneous capitalization.

Despite what you may think, this will be your reality throughout your career. Every developer's role includes error-catching. The faster you catch and resolve bugs, the sooner you can get back to creating applications and building out your clients' latest requests. Therefore, your career will partly depend on your ability to not only generate ideas and code them out but also fix things when they fail to work.

How to Debug

Fixing broken code is called **debugging**. Just as with all problems in life, there are right ways to deal with coding issues, and there are wrong ways.

In this course you'll develop a fleet of debugging strategies, but in this lesson, you'll get a small taste of what it's like to work through perplexing code errors.

Tip #1: Start Small

Let's start by discussing the wrong way to deal with coding issues. When new developers experience an issue with their code, they often assume that they did everything wrong and immediately start over, or they give up altogether. This isn't a great way to learn or get code working.

A better approach is to focus your attention on the lines of code closest to the feature that's broken. For example, if clicking a button doesn't trigger a pop-up like you expected, look closely at the code associated with the button. Check your syntax carefully. Check how you've named things. Check your parentheses. Check your capitalization.

In other words, start small and center your attention on what's broken. Then move outward to the rest of the code. You might discover that the button isn't broken after all but the pop-up is. Or maybe you forgot to save your file. Perhaps you aren't even looking at the right file. Don't assume there's a huge problem with your code; sometimes it's the smallest thing that keeps you from a working solution.

Tip #2: Reference Working Code

Many new developers think they're cheating if they look back at old code. This couldn't be further from the truth! As a developer, you should think yourself as a curator of references. As you work with many examples of code, you continually build a bank of good code. Whenever you face a task or an issue, you can reference this Bank of Good Code and remember how you solved a previous problem.

Professional developers constantly search through their own code and that of others to tackle the issue is at hand. Certain bits of code will become second nature to you, but you'll need time and many hours of practice to reach that point. Don't be hard on yourself if you aren't there yet.

But let's get back to debugging. Say we have the following instance of a broken button:

Broken Code: Up Button

```
$(".upButton").on(click, function(){
    $(".cartooncharacter").animate({top:"-=200px"}, "normal");
});
```

The preceding code was written to move an image of a cartoon character when the button is clicked, but the code isn't working. The cartoon character just sits there.

We can proceed in one of two ways:

- Hunt and peck through the code at random.
- Pull an example of working code, like the following snippet.

Working Code: Down Button

```
$(".downButton").on("click", function(){
    $(".cartooncharacter").animate({top:"+=200px"}, "normal");
});
```

Take a moment to compare the code between these two snippets. Can you spot the error? (Really try here!)

Hopefully, you noticed the missing quotation marks around the word "click." Even if you've never been exposed to JavaScript, having access to the working code can help you spot the issue.

This is the value of good code references.

Tip #3: Keep Your Code Clean

The third key to error-free coding is to keep your code clean. The concept of clean code will make more sense as you progress through the class, but in the meantime, look at these two blocks of code:

Sloppy Code

```
$("#randomButton").on("click", function()
{for (var i=0; i<3; i++){alert(Math.floor(Math.random() * 25) + 1);}})
```

Clean Code

```
// When randomButton is clicked...
$("#randomButton").on("click", function(){

    // Loop through 3 times...
    for (var i=0; i<3; i++){

        // And create an alert with a new random number between 1 and
         alert(Math.floor(Math.random() * 25) + 1);
    }

});
```

Believe it or not, both these blocks function exactly the same. However, for a developer who needs to maintain or improve code, the latter is obviously preferable.

You'll find in this course that you can often get by with sloppy code, but it will come back to haunt you later on. Disorganized code is more difficult to read and debug; clean code makes errors easier to spot.

Look at the following example of broken (but very realistic) code:

Sloppy Broken Code

```
for (var i=0; i<=5; i++)
    {if(i===5)
{console.log("Yay! My favorite #5!!")}
else{console.log("I don't like this number very much.")
}
```

It's hard to spot, but the code was missing a `}` toward the end of the else statement.

Now take a look at the same code with the same error, but structured with good indentation:

Clean Broken Code

```
// Loop through a set of numbers
for (var i = 0; i <= 5; i++) {

    // If the number is 5
    if (i === 5) {
        // Print it out.
        console.log("Yay! My favorite #5!!");
    }

    // If the number is not 5
    else {
        // Print it out
        console.log("I don't like this number very much.");

}
```

Assuming you know that every `{` must end with a `}`, this format makes the error much easier to spot!

Tip #4: Read the Debug Error

As you proceed through the course, you'll be introduced to browser-based debugging tools like **Google Debugger (https://developers.google.com/web/tools/chrome-devtools/debug/?hl=en)** .

These tools will help flag troublesome lines of code that are difficult to spot.

If you're relieved to hear about these tools because you found the last exercises difficult, you're right! These debugging tools are incredibly powerful. However, they're only as powerful as the developer who uses them.

Take, for instance, the following broken code and the resulting output provided by the Google Debugger:

Broken Code

```
var myName = "Sam";
var favFood = "Green Eggs and Ham";
alert(myName + " loves " + favfood);
```

Google Debugger Output

```
quick.html:3 Uncaught ReferenceError: favfood is not defined
```

For someone new to coding, the error message may be just as indecipherable as the original code, but let's try to dissect it together.

For one, we can see that it specifies a line number of 3, which means that it thinks the error is on line 3: `alert...` (We say "think" because sometimes the debugger gets tripped up by complex code.)

Next, it tells us that the error has something to do with the entry `favfood` being "not defined." We'll get into concepts like variables and definitions later, but for now, consider the difference between the `myName` data and the `favFood` data. If you look closely, you might notice that `favFood` is capitalized in one instance as `favFood` and isn't capitalized in another as `favfood`.

Google Debugger picked up and flagged this minor oversight, and with a little bit of sleuthing, you can easily resolve it. Again, don't be afraid of error messages! They provide fantastic clues to your solution.

Tip #5: Test Often

This is a highly effective preventive strategy for keeping bugs out of your code.

When new developers are assigned a task, they often attempt to write all of the code in one sitting. For instance, to create a game of tic-tac-toe, they might try to create the layout, logic, button events, rules for determining who wins, and many other features without testing a single piece of functional code.

This approach will result in a hopeless labyrinth of bugs. Instead, be as minimalistic as possible. Get into the habit of making modest changes, saving those changes, and then immediately testing them in the browser. This way, you can isolate the location and the number of bugs to only the block of code on which you're working. If you try to bite off too much, errors will creep into numerous places, and each one will have a cascading effect.

Again, this will make more sense as you start coding, but keep this advice in the back of your mind!

Tip #6: Get Help

Despite the stereotypes, coding is actually a very collaborative line of work. In professional settings, coders communicate constantly with one another and with online developer communities. As you begin your web development journey, remember that you can always ask for help. There's no shame at all in asking for a second pair of eyes. Sometimes you only need a fresh perspective to make a breakthrough!

Tip #7: Practice, Practice, Practice

Last but definitely not least, the best tip of all: always be coding!

The single best way to become a faster debugger and better coder is to code a lot. At first, many of those hours you put in will involve mindless hunting, but don't write those off as wasted hours. Remember that every hour you spend debugging an issue increases your knowledge and skill set. The more time you spend studying spots where errors exist and where they don't, the better equipped you'll be to solve any problem you come across.

Time to Get Coding (and Debugging)!

It bears repeating: errors are and will always be a fact of life. But don't let errors discourage you or make you quit. Instead, relish every time you fix something that's broken, and celebrate the fact that you've entered the fray as a professional developer.

Now it's time to get to work! Complete the following activity to try your hand at debugging some HTML and JavaScript.

PS It's okay that you don't completely understand HTML or JavaScript yet! That's the point.

Resources

- **The Art of Debugging (https://remysharp.com/2015/10/14/the-art-of-debugging)**
- **Do You Spend More Time Coding or Debugging (https://news.ycombinator.com/item?id=11729806)**
- **Getting Started with Google Debugger (https://developers.google.com/web/tools/chrome-devtools/debug/?hl=en)**
- **Stack Overflow (http://stackoverflow.com/)**