**Debugging**

Debugging is the process of removing errors/bugs from code. Some bugs produce errors which break the script and give you warnings as to what went wrong. The browser console is very good at doing this. Some bugs don’t break the code at all however you find that your code doesn’t do what you think it should do.

**How to get good at Debugging**

Bugs are a fact of life for programmers. You can become better at debugging or even prevent writing bugs at all if you follow certain guidelines:

1. Write neat code. **Indent your code properly** to show where brackets or parentheses are opened and closed. This makes your code infinitely easier to read.
2. **Comment** your code. Sometimes writing out pseudocode helps you to understand problems better.
3. **Understand** the **scope** of the language and the **flow of code**. Accessing variables that aren’t defined or aren’t in the same scope leads to many mistakes. Also, errors might be triggered at one point however they manifest in another area.
4. Read and **understand** what **the debugger** is telling you – in JavaScript’s case the console will alert you to errors.
5. Use **debugger;** in your JavaScript code.
6. Use **console.log()** to keep track of your values in JavaScript. For weakly typed languages like JavaScript make use of **typeof** to keep tabs on data types.
7. **Test your code often**. And don’t just test your code when the project is done. Test iteratively so that you can be confident of your code at each step.

**Understanding Errors**

If a JavaScript statement generates an error, it throws an exception and the interpreter will stop and begin looking outward to find exception-handling code. If it does not find this, the interpreter will jump out of the current scope (for example jumping outside of a function if the error began there) and will continue searching. Take note that where a function is called and where it is defined can be too different locations. It continues to do this until it gets to the global context where the script stops and an Error object is created.

An Error object contains 4 elements that help you to diagnose the problem. You will see this object in the browser’s dev console outlining the error the interpreter encountered. Clinking on the link will bring you to the error trigger in the source.



Text

Description automatically generated

**const error = {**

**name : "The type of error that has occurred",**

**message : "Description of the error that has occurred",**

**fileNumber : "Name of the file/JavaScript file where the error has occurred",**

**lineNumber : "Line number in the code where the error occurred"**

**}**

**Types of errors**

There are 6 types of errors in JavaScript however most people will be familiar with 3 kinds due to their frequency of coming up.

|  |  |
| --- | --- |
| **SyntaxError**  *Often the result of a simple typo, these errors are derived from incorrect use of the language – hanging braces/parenthesis, forgetting to close off quotation marks etc.* | Mismatching or Unclosed Quotes  **alert( "Hello’ );**  SyntaxError: Unexpected EOF  Missing Closing Bracket  **alert( "Hello"**  SyntaxError: Expected token ‘)’  Missing Comma in Array  **const carArray = ["VW", "Toyota" "Ford"];**  SyntaxError: Expected token ‘]’ |
| **ReferenceError** *Caused by trying to access/use a variable or function that is not declared or is out of scope.* | Variable is Undeclared  **const width = 100;**  **const area = width \* height;**  ReferenceError: Can’t find variable: height  Named Function is Undefined  **function sayName() { alert( "James" ) }**  **sayMyName();**  ReferenceError: Can’t find variable: sayMyName() |
| **TypeError**  *Caused by trying to use an object or method that does not exist. Often a syntax error (spelling mistake) when using JavaScript methods or attempting to reassign value to a const variable.* | Incorrect Syntax for console.log() Method  **console.lg( "Hello World" );**  TypeError: console.lg is not a function  Incorrect Case for document Object  **Document.write( "Mistake" );**  TypeError: Document.write() is not a function  Attempting to reassign value to a const  **const num = 1;**  **num = 2;**  TypeError: Assignment to constant variable |
| **RangeError**  *Caused by calling function using numbers outside of its range* |  |
| **URIError**  *Not correctly escaping characters in URIs* |  |
| **EvalError**  *Incorrect use of* ***eval()*** *function* |  |

**debugger;**

You can add the **debugger** keyword into your JavaScript code. When the interpreter hits **debugger;**, it pauses executing the script and the browser source window pops up. From here you can look at local and global variables and see what their current values are at the exact moment **debugger** was reached. You can add multiple **debuggers** in your code to check what is happening at the different points. At any time, you can move the interpreter to the next **debugger** and let it flow to the end of the script.

Graphical user interface, text, application

Description automatically generated

**Logical Errors/Bugs**

Bugs do not always produce errors. You can write code that contains none of the above mistakes however if your code doesn’t act the way it should, you might have a logical error. Logical errors arise when your syntax is correct however your code doesn’t work the way you intend it to. These can be far more detrimental because:

1. They are hard to fix. Uncovering where the problem is in your code can be a difficult task.
2. They are hard to detect. In fact, they may go unnoticed until a user of your program encounters them.

Since logical errors don’t produce bugs, the only way to avoid them is to **test your code frequently and often,** supply different inputs (a diverse selection of parameters or values) and check to make sure that the resulting outputs are correct.

**Examples of Testing**

|  |  |  |
| --- | --- | --- |
| Inputs | Outputs | Expected Outputs |
| -**1**- + -**1**- | -**2**- | **2** PASSED |
| -**3**- + -**5**- | -**8**- | **8** PASSED |
| -**4**- + **“6”** | **“46”** | **10** FAILED |

Below are examples of logical errors.

* This code will always console.log LARGE because the conditional doesn’t check for true but rather assigns true.

**let isLarge = false;**

**if ( isLarge = true ) {**

**console.log( "LARGE" );**

**}**

* This **addNumbers** function will correctly add numbers together so long as it receives numbers as parameters. If a string is provided by mistake, the function will concatenate the “numbers” together.

**function addNumbers( n1, n2 ) {**

**return n1 + n2;**

**}**

**addNumbers( 1, 2 );** // returns 3

**addNumbers( 1, "2" );** // returns "12"

* This code is supposed to **console.log** the sum of an array. As you can see in the **for** loop’s initialization, **let i = 1;** as opposed to **let i = 0;**.Therefore, the first element in the array is always skipped and the correct **total** is not calculated.

**const arr = [ 1, 2, 3 ];**

**let total = 0;**

**for ( let i = 1; i < arr.length; i += 1 ) {**

**total += arr[ i ];**

**}**

**console.log( "Array total is " + total );** // Array total is 5