Week 05 Notes:

Testing and Debugging

Errors, exceptions, and warnings

- System error – There's a problem with the system or external devices
- Programming error- the program contains incorrect syntax or faulty logic
- User error – the user has entered data incorrectly which the program can't handle
- Exceptions – An exception is an error that produces a return value that the program can then use to deal with the error.
    - unicorn();
    - << ReferenceError: unicorn is not defined
- Stack Traces
    - An exception can also produce a stack trace. This is a sequence of functions or methods that lead to the point where the error occurred. Usually, it is, not just one function that causes the error.

```
function three(){ unicorn(); }function two(){ three(); }function one(){ two(); }one();
<< index.html:13 Uncaught ReferenceError: unicorn is not defined at three (index.html:13)    at two (index.html:17)    at one (index.html:21)    at index.html:24`
```

- Warnings:
    - Will not cause the program to crash but can cause it to run incorrectly.

The importance of testing and debugging

- JS is a forgiving language. It would not alert you it would just fail in the background. /
- The program that you write should fail loudly in development so you can catch any errors before it goes to production.

Strict mode

- It will catch even poor style as actual errors.
- Can be applied globally or in a single method.

```
'use strict';

e = 2.718;<< ReferenceError: e is not defined
```

- Linting Tools: JS Lint, JS Hint, ES Lint. It will help with formatting or forcing a style guide.

- Passing a lint test does not guarantee there are no bugs, but it will keep you consistent and less likely to have problems.
- Feature detection: You can use if statements to catch potential issues that you could predict. So even if it doesn't exist but could end up existing you are planning ahead.
-

Debugging in the browser

- Debugging is the process of finding out where bugs occur in the code and then dealing with them. In many cases, the point at which an error occurs is not always where it originated, so you'll need to run through the program to see what's happening at different stages of its execution
- alert() stops a program from running until OK is clicked, it allows us to effectively put breakpoints in the code that let us check the value of variables at that point to see if they're what we expect them to be

```
function amIOldEnough(age){    if (age = 12) {        alert(age);
return 'No, sorry.';    } else if (age < 18) {        return 'Only if
you are accompanied by an adult.';    }    else {        return 'Yep,
come on in!';    }}
```

- Most modern JavaScript environments have a console object that provides several methods for logging information and debugging

Error objects

- EvalError is not used in the current ECMAScript specification and is only retained for backward compatibility. It was used to identify errors when using the global eval() function.

- RangeError is thrown when a number is outside an allowable range of values.

- ReferenceError is thrown when a reference is made to an item that doesn't exist. For example, calling a function that hasn't been defined.

- SyntaxError is thrown when there's an error in the code's syntax.

- TypeError is thrown when there's an error in the type of value used; for example, a string is used when a number is expected.

- URIError is thrown when there's a problem encoding or decoding the URI.

- InternalError is a non-standard error that is thrown when an error occurs in the JavaScript engine. A common cause of this is too much recursion.

- Throwing exceptions

Exception handling

- The throw statement can be applied to any JavaScript expression, causing the execution of the program to stop.
- When an exception occurs, the program terminates with an error message.
- Try, catch, and finally
- The code inside the catch block will only run if an exception is thrown inside the try block.
- A finally block can be added after a catch block. This will always be executed after the try or catch block, regardless of whether an exception occurred or not.

Testing frameworks

- Testing is an important part of programming that can often be overlooked. Writing good tests means your code will be less brittle as it develops, and any errors will be identified early on.
- Test-driven development (TDD) is the process of writing tests before any actual code
- The code is continually tested at each stage to make sure it continues to work. This process should be followed in small piecemeal chunks every time a new feature is implemented, resulting in the following workflow:
    - Write tests (that initially fail)
    - Write code to pass the tests
    - Refactor the code
    - Test refactored code
    - Write more tests for new features
- Testing frameworks provide a structure to write meaningful tests and then run them. There are a large number of frameworks available for JavaScript, but we'll be focusing on the Jest framework.
- Jest:
    - Jest is a TDD framework, created by Facebook
    - You install it `npm install -g jest`

```
function squareRoot(number) {    'use strict';    if (number < 0) {
throw new RangeError("You can't find the square root of negative
numbers")    }    return Math.sqrt(number);};

test('square root of 4 is 2', () => {expect(squareRoot(4)).toBe(2);});
```