Week 5 Notes
Testing and Debugging-Chapter 10

Errors, Exceptions, and Warnings

System Error—There is a problem with the system or external devices that is being interfaced.
Programmer Error—The program contains incorrect syntax or faulty logic; --could be a typo.
User Error—the user has entered data incorrectly which the program is unable to handle.

We should try to limit user errors by good programming and interactons…the program should be deisgned so that Users can't make mistakes.

Exceptions
An exception is an error that produces a return value that can then be used by the program to deal with the error…such as "undefined".

Stack Traces
They are sequences of functions or methods calls that lead to the point where the error occurred…the work backwards from one point to where the error occurred.

Warnings
Occurs if there are errors in the code that isn't enough to cause the program to crash.

Warnings and exceptions are presented differently depending the environment or on the browser.

The Importance of Testing and Debugging

Failing silently makes errors more difficult to spot and track down…so you want them to fail loudly….or at least loud and graceful.

Strict Mode
Produces more excpetions and warnings and prohibits the use of some deprecated features…helps eliminate poor style. It will throw exceptions If sloppy code is used.

Strict mode is often referred to as "sloppy mode".

To initate it….'use strict';

--you can put it in a function—per function basis or at the beginning of a file.

Linting tools test the quality of JS code beyond the strict mode.
--enforce programming style guides….there are plug ins, online, software tools for linting.

Feature Detection

You can't always rely on users having up-to-date browsers. So they may not be able to use all features in a program. Detect and prevent problems with feature detection methods.

--if statement to check for an object.

```
if (window.holoDeck) {
   virtualReality.activate();
}
```

Libraries also can help detect—Modernizr and "Can I Use?"…browser sniffing was the old way.

"alert method" was often used to help debug…but now consoles are better.

Using the Console

Consoles log information that is helpful to debugging….console.log();

--console.trace() method will log an interactive stack trace in the console. This will show the functions that were called In the lead up to an exception.

Debugging Tools

Breakpoints in code will pause it at certain points.

The main Browsers have documentation for how to track bugs.

The "debugger" keyword will create a breakpoint in the code that will pause the execution of the code and allow you to see where the program is currently up to. Debuig tool will automatically kick in and you'll be able to see the value of the variable by hoovering over it.

Error Objects

An object error can be created by the programmer by the host environment or a constructor function.

EvalError is not used in the current ECMAScript specification and only retained for backwards compatibility. It was used to identify errors when using the globaleval()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 too much recursion.

Throwing Exceptions
Use a throw statement to make your own exceptions….the program will then stop. It is best to through an error object.

Can use if statements:

```
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);
}
```

Exception Handling
Handle exceptions gracefully by catching them and keep them hidden from users. Use "try, catch and finally".

If there is an error the code will catch an error to be queried later. You can then continue with the final statement so the error in code doesn't stop everything.

```
function imaginarySquareRoot(number) {
   'use strict';
   let answer;
   try {
      answer = String(squareRoot(number));
   } catch(error) {
      answer = squareRoot(-number)+"i";
   } finally {
      return `+ or - ${answer}`;
   }
}
```

How often are custom error objects used if since there are already built-in ones?

Tests

Good tests mean your code will be less brittle and errors and problems can be easily identified.

TDD-test-driven development. The process of writing tests before any actual code….then write code to make it pass the test. Considered best practice but coders don't always use it.

Testing Framework
Use a framework to test code

Jest—Is this still the best framework to use for testing?.. industry standard?

Refractor?

Jest method link:
https://jestjs.io/docs/expect

Exception object –has methods called **matchers**

What is the it() method?