Chapter 10, Testing and Debugging

\* Remember that errors and bugs will always be there.

**Errors, Exceptions, and Warnings**

**System Error** – there’s a problem with the system or external devices with which the program is interacting.

**Programmer error** ― the program contains incorrect syntax or faulty logic; it could even be as simple as a typo.

**User error** ― the user has entered data incorrectly, which the program is unable to handle.

**Exceptions** – an error that produces a return value that can then be used by the program to deal with the error.

* unicorn();
* << ReferenceError: unicorn is not defined

Stack Traces: a sequence of functions or method calls that lead to the point where the error occurred.

Strict Mode: produces more exceptions and warnings and prohibits the use of some deprecated features

Debugging in the browser:

The most basic form of debugging is to the use the alert() method

* 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!';
* }
* }

Error Objects:

An error object can be created by the host environment when an exception occurs, or it can be created in the code suing a constructor function:

* Const error = new Error(“Oops, something went wrong”);
* **EvalError** is not used in the current ECMAScript specification and only retained for backwards 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 too much recursion.

Exception Handling:

When an exception occurs, the program terminates with an error message.

SqaureRoot() function:

* function imaginarySquareRoot(number) {
* 'use strict';
* try {
* return String(squareRoot(number));
* } catch(error) {
* return squareRoot(-number)+'i';
* }
* }

Sandbox, Chapter 8, 8.2 The locked box:

* const box = {
* locked: true,
* unlock() { this.locked = false; },
* lock() { this.locked = true; },
* \_content: [],
* get content() {
* if (this.locked) throw new Error("Locked!");
* return this.\_content;
* }
* };
* function withBoxUnlocked(body) {
* let locked = box.locked;
* if (!locked) {
* return body();
* }
* box.unlock();
* try {
* return body();
* } finally {
* box.lock();
* }
* }
* withBoxUnlocked(function() {
* box.content.push("gold piece");
* });
* try {
* withBoxUnlocked(function() {
* throw new Error("Pirates on the horizon! Abort!");
* });
* } catch (e) {
* console.log("Error raised:", e);
* }
* console.log(box.locked);
* // → true

Output:

Error raised: Error: Pirates on the horizon! Abort!

true