Writing better code

Common Errors

- Exception and Error are synonomous, so are raise and throw
- ReferenceError: when one references a var or function that doesn't exist

```
a;    // Referencing a variable that doesn't exist results in a ReferenceError.
a();  // The same is true of attempting to call a function that doesn't exist.
```

- TypeError: when accessing a property/value that has none, i.e. `null`, or calling something that isn't a Function

```
var a;      // a is declared but is empty, as it has not been set to a value.
typeof(a);  // "undefined"

a.name;     // TypeError: Cannot read property 'name' of undefined

a = 1;
a();        // TypeError: Property 'a' is not a function
```

- SyntaxError: happens immediately after loading a program, e.g. `function({) //// SyntaxError: Unexpected token (`
- SyntaxError can also be raised at runtime: e.g. `JSON.parse('not really JSON'); // SyntaxError: Unexpected token i in JSON at position 0`
- Other errors: RangeError, URIError

Preventing Errors

- Using guard clause to prevent unexpected scenarios

```
function lowerInitial(word) {
  return word[0].toLowerCase();
}

// which doesn't work when there's an empty string
lowerInitial('');       // TypeError: Cannot read property 'toLowerCase' of undefined
```

- The above could be avoided by using a guard clause

```
function lowerInitial(word) {
  if (word.length === 0) {       // If word contains an empty String,
    return '-';                  // return a dash instead of proceeding.
  }

  return word[0].toLowerCase();  // word is guaranteed to have at least
}                                // 1 character, so word[0] never evaluates
                                 // as undefined.
```

- This can be used when a function can't trust its arguments are valid, which can have invalid types, structures, values, properties
- These "untrustworthy moments" are edge cases, e.g. in `lowerInitial`, shortest possible String ('') is an edge case

- e.g. if an argument is expected to be numeric, will the program still work if the argument is zero/negative, fractions? If a string is expected, what if its' empty, or starts/finish with spaces/special characters? What if a combo of values can create issues?
- Planning the code means writing out possible permutation of arguments (common user cases)

Catching errors

- Some functions can throw errors, but how to predict or avoid those errors?
- Eg. `JSON.parse` takes a single String and converts it to an object, and if what's passed isn't a JSON object

```
var data = 'not valid JSON';

JSON.parse(data);  // throws SyntaxError: Unexpected token i in JSON at position 0
```

- Using `try/catch/finally` to handle `JSON.parse` errors

```
function parseJSON(data) {
  var result;

  try {
    result = JSON.parse(data);  // Throws an Error if "data" is invalid
  } catch (e) {
    // We run this code if JSON.parse throws an Error
    // "e" contains an Error object that we can inspect and use.
    console.log('There was a', e.name, 'parsing JSON data:', e.message);
    result = null;
  } finally {
    // This code runs whether `JSON.parse` succeeds or fails.
    console.log('Finished parsing data.');
  }

  return result;
}

var data = 'not valid JSON';

parseJSON(data);    // Logs "There was a SyntaxError parsing JSON data:
                    //       Unexpected token i in JSON at position 0"
                    // Logs "Finished parsing data."
                    // Returns null
```

- Use this block, if 1. built-in JS fucntion/method can throw an Error and need to handle/prevent the Error, 2. simple guard clause is impossible or impractical to prevent Error
- More error handling

Study Guide:

- primitive values, types and type conversions
- variable scopes and hoisting
- function declarations, expressions and scopes
- object properties and mutation
- assignments and comparison

- pure functions and side effects