dotnotes

JAVASCRIPT ES6 SYNTAX

// The Big Picture // ES6 adds new declarative syntax, which shifts our focus away from imperative implementation details & allows us to write higher-level, more interpretable code.

TEMPLATE STRINGS

Easily embed variables & expressions:

```
`The cactus is named ${name}.`
```

- You can also use line breaks within backticks to create multi-line strings. No more `\n`!

ARROW FUNCTIONS

A concise syntax for creating functions.

- There are many syntactic variations.

```
let sum = function(a,b) {
    return a+b;
};
```

```
let sum = (a,b) => a+b;
```

- Arrow functions do not bind their own `this`; rather they inherit one from their Parent => "lexical scoping".
- Just because you can use shorthand doesn't mean you should. Readability first.

SCOPING

To declare variables that will be used in a wider context.
→ "Contain this variable within the immediate function scope."

let vs. Var

To encorce block scoping, rather than relying on stylistic conventions.
→ "Contain this variable within this block scope."

Const

To create a block-scoped constant variable that cannot be re-assigned.
→ "Signal intent: this variable will not change."

- This does not mean that the value of the variable is immutable.
- Must be initialized with a value.

Both let and Const are {BLOCK} SCOPED VARIABLES
↳ variables only exist within their enclosing block, denoted by {}.

"REST / SPREAD" OPERATOR

The behavior of `...` depends on the context:

① In an assignment context, `...` transforms an indefinite list of arguments into a proper array =>

```
function foo (...args)
```
→ `args` is an array of all parameters passed into the function.

```
let [s, ...m] = [1,2,3]
```
→ `m` = `[2,3]` ↑ It gathers the "rest" of the elements.

② Otherwise, `...` expands an iterable:

```
foo(...args)
```
→ Pass the elements of an array into a function.

```
let a = [1,2,3]
let b = [0,...a]
```
→ `b` = `[0,1,2,3]` Succint array creation!

DESTRUCTURING

Extract data from arrays & objects:

```
let ["first","second"] = myArr
```
→ Extract 1st & 2nd elements from `myArr`.

```
let {a,b} = myObj
```
→ Extract properties `a` & `b` from `myObj`.

This is just the tip of the iceberg when it comes to destructuring...

DEFAULT VALUES

↳ if no value (or `undefined`) is passed:

```
function spam(n=1)
```
← Default value assigned in function head.

```
let [a=42, b, c] = ham
```
← A default value can also be passed when destructuring.

CONCISE PROPERTIES & METHODS

Eliminate duplicative code in objects.

```
var obj={
    a,
    b(){}
}
```

`a,` ← when the property name = value name.
`b(){}` ← Shorthand for method definitions.