JS FORMAL INTRO

https://forms.gle/H72nCVb9hMBRs8U28

WRITING JS

IMPORTANT CONCEPTS

- Primitive Types
- OOP and FOP
- Hoisting
- Closures
- The DOM and events

PRIMITIVE TYPES

JavaScript is dynamically and strongly typed

- number
- bigint
- string
- boolean
- null
- undefined
- symbol

- `var`, `let`, `const`
- `typeof`

Everything else is type Object

OBJECTS (DATA STRUCTURE)

```
const obj = {
    key: 'value',
    another: 1
};
```

```
console.log(obj.key);       // 'value'
console.log(obj['key']);    // 'value'

const someVar = 'another';
console.log(obj[someVar]); // 1
```

FUNCTIONS

Function Declaration

```
function name(param1, param2) { /* code */ }
```

Function Expression

```
const x = function name(param1, param2) { /* code */ }
```

Anonymous Function Expression

```
const x = function(param1, param2) { /* code */ }
```

Arrow Function (ES2015/ES6)

```
const x = (param1, param2) => { /* code */ };
```

HOISTING

- Declarations with `var` and function declarations, **not assignments**, are hoisted to the top of the function they are in
- This means they can be accessed anywhere within that function (even before they're declared)
- If they aren't in a function, it becomes globally scoped

```
console.log(x); // undefined (but not a ReferenceError)
if (true) {
    var x = 2;
}
console.log(x); // 2
```

- The second `console.log` would log `undefined` if the boolean was `false` since it's still declared, but never assigned.

CLOSURES

Functions can remember the context in which they were created

```
const outer = (str) => {
    const inner = () => {
        return str;
    };
    return inner;
};
const x = outer('x');
const y = outer('y');
const z = outer('z');
console.log(x(), y(), z()); // 'x', 'y', 'z'
```

Would anything change if `str` was globally scoped?

ADDITIONAL RESOURCES

- JS Documentation
- JS for Absolute Beginners
- JS and the DOM (1 of 4)
- Vanilla JS Playlist
- Some Nice JS Videos