Advanced JavaScript Concepts

JavaScript is weird but awesome!

Important concepts in JS

- First-class Functions
- EventDriven

First-class Functions

- A programming language is said to support first-class functions (or function literal) if it treats functions as first-class objects.

- Specifically, this means that the language supports constructing new functions during the execution of a program, storing them in data structures, passing them as arguments to other functions, and returning them as the values of other functions.

More concisely…

- A function is an instance of the Object type
- A function can have properties and has a link back to its constructor method
- You can store the function in a variable
- You can pass the function as a parameter to another function
- You can return the function from a function

We will come back to functions in a minute…

Event Driven Environment

- In computer programming, event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads.

- JavaScript code sits in the memory and does nothing but monitoring until summoned.

- Creates its own eco system, results in closure, context and scope.

- That's what people mean mainly when they say JS is fast… especially NodeJS

Closure

- So… since the code only run once, what happens to the guys we actually want to keep around, say those which involve in some callback functions??

- The technique of language that the language retains state and scope after execution. Still stored in memory as long as the listener (reference) exists

- But what about memory leak! O_O (Didn't know that could happen in JavaScript huh?)

- JS Garbage collection comes in place after. Trivial in small programs.

Scope&Context

- Scope === variable access (what variables do you have access to when you run certain piece of code.

- Context === the value of this.

Scope: children and parents

- Children can access parents' pocket.
- Parents cannot access children's pocket.
- Reversed access has to specify the scope.
- Scope pollution & "use strict"

Context

- Context === this
- Root context: window, Var context: var name
- This can change when called differently
- Call, apply and bind
- Examples …