Dealing with Context Loss

Method Losing Context when Taken Out of Object

- If you remove a method from its containing object and execute it, it loses its context
- You can mitigate this by passing an extra parameter (the context) and using `call` or `apply` to explicitly set that context wherever you need to
- Some of JS's built-in methods like `forEach` allow an optional `thisArg` argument as the context object(value of `this`) that the function should use
- If you can't update the function or supply a context you need to hard bind with `bind`

Internal Function Losing Method Context

- JS has a really nasty gotcha where the context does not propogate to its internal functions
- An object with a property that is a function, with an internal function loses it's context/access to the object and its other properties
- Solutions
    - Preserve context with a local variable in the lexical scope
    - `var self = this` or `var that = this`
        - Save's `this` in a variable before calling the function, then reference that variable in function
    - Pass the context to internal functions
    - Use `call` or `apply` to pass the context object to the function
    - Bind the context with a function expression
    - You can use `bind` when you define a function to provide a permanent context (wont work with function declaration, only function expressions)
    - Advantage of using bind is you only need to do it once and then call it as often as you want without needing to provide a context every time

Function as Argument Losing Surrounding Context

- Functions that take functions as arguments have same problems as internal functions listed above (and same solutions)
    - Use a local variable in the lexical scope to store `this`
    - `bind` the argument function with the surrounding context
    - Use the optional `thisArg` argument