Implicit and Explicit Function Execution Contexts

Function Execution Context

- Whenever function is executed, it has access to an object called the execution context of that function, through keyword `this`
- Which object `this` refers to depends on how the function was invoked

`this`

- The mechanics of how `this` binding works is complicated, but important
- Some difficulty comes from tendency to apply lexical scoping rules, but `this` has different rules based on how a function is invoked

Two Main Types of Execution Contexts

- Implicit: Execution context implicitly set by JS
- Explicit: Execution context explicitly set by programmer

Implicit Function Execution Context

- AKA Implicit Binding for functions
- The context for a functin you invoke without supplying an explicit context
- JS binds such functions to the global object (assigned as one of the global object's properties)
- Binding a function to a context occurs **when you execute the function- not when you defined it**

Implicit Method Execution Context

- The implicit context for a method (function referenced as an object property) invoked without an explicit context provided
- JS implicitly binds methods invoked to the owning or calling object

Explicit Function Execution Context

- We can use `call` and `apply` methods to change a function's execution context at execution time
- `call` can pass arguments to a function (list arguments after the context object) Ex: `someObject.someMethod.call(context, arg1, arg2)`
- `apply` method is identical to `call` except that it uses an array to pass arguments Ex: `someObject.someMethod.call(context, [arg1, arg2])`
- Mnemonic for remembering which is which
    - **C**all: **Count** the **Commas**
    - **A**pply: **Apply** as **Array**