Benjamin Zhuang - Week 2 Day 1

---

1. What are the different types of scopes?
    a. Block scope
    b. Function scope
    c. Global scope
2. What is hoisting?
    a. All variables' declarations are executed before any other code executes. Thus, you can actually initialize a variable before declaring it.
3. Explain the differences between var, let, & const.
    a. Var is globally scoped, while let & const are block scoped
    b. All types of variables are hoisted but var are initialized to undefined by default.
    c. Var can be re-declared while others cannot.
    d. Const can't be reassigned but if it's reference type, then the referenced object's properties can still be changed.
    e. Const needs an initial value.
4. What is an execution context? How does it relate to the call stack?
    a. An execution context is the information needed to execute some code. We have one global execution context for top-level code and an execution context for each function invocation.
    b. A call stack contains all the execution context for the current executing code. It pops each execution context to make sure that the code is executed in the correct order.
5. What is the scope chain? How does lexical scoping work?
    a. The scope chain is a mechanism that JavaScript uses to lookup a variable. It will first look at the current scope of the variable, then the outer scope, and it will extend all the way to the global scope or until it finds the variable.
    b. Lexical scope is like the origin scope of a variable. Therefore, code inside a function may access a variable defined outside because of how the scope chain works. However, you cannot access a variable inside of a function when the execution context is outside.
6. What is a closure? Can you provide an example use-case in words?
    a. A closure is basically accessing variables from the outer function inside of an inner function scope. When the inner function returns, the outer function references are returned along with the inner function, and we call this a closure.
    b. It's useful for simulating private variables by lexical scopes. Basically, you can define a functional class with some properties and some getter and setter

functions. You could use the setter function to change the properties of this class because it has the closure within the functional class. However, there is no way to access the properties of the functional class because of lexical scopes.

7. What is an IIFE? When would you use it?
    a. IIFE stands for immediately invoked function expression. It's a syntax for immediately invoking a function after it's defined.
    b. Avoid polluting the global variables if we are just doing some initialization code.
    c. Alternatively, you can use IIFE to create public/private variables and methods.
8. What does 'this' refer to in the cases that were discussed in lecture?
    a. When creating an object with the **new** keyword, "this" refers to the created object.
    b. For keyword functions, "this" keyword changes based on where it's invoked. If it were invoked at top-level, inside a keyword/arrow function, it would refer to the **global object**, which is oftentimes the "window" object. If it were invoked, and it's an object method, "this" keyword would refer to the **parent object** instead. And if the keyword function is invoked by some event handler, then "this" would refer to the DOM element receiving the event.
    c. If "use strict" mode is invoked, "this" will refer to **undefined** for keyword functions.
    d. For arrow functions, they don't provide their own "this" binding. Instead, "this" would refer to the "this" in its **lexical scope**.
9. What are the differences between call, apply & bind?
    a. Call() & apply() are almost identical except that call receives an argument list, while apply() receives an single array of arguments.
    b. Bind() creates a new function and binds the object as the new "this" keyword. For example, if you do something like func.bind(obj) at top-level for some keyword function, rather than having a "this" of either **undefined or the global object**, "this" will refer to **obj** instead because now **obj** is binded to the func.
10. Can you name the new ES6 features?
    a. Block scoped variables including let & const
    b. Arrow functions
    c. Template literals
    d. Map & Set
    e. Array & Object destructuring
    f. Spread operator
    g. Enhanced object literals
    h. Promises, async/await
    i. Default function parameters

    j. Generators
    k. Classes
    l. Modules
11. What is ‘use strict’? What are the major effects that it has?
    a. Invokes the strict mode, which makes changes to the normal JavasScript semantics.
    b. Strict mode captures silent errors that would happen in normal JS. For example, you can’t use undeclared variables, functions, or objects. You also can’t have duplicate argument names.
12. What is currying?
    a. Currying is the mechanism of breaking down a function that takes multiple arguments into a nested function, such that each nested function will only take a single argument.
13. What are ES6 modules? Why are they useful? How do we make these modules available in other JS files?
    a. ES6 modules are a way to split JavaScript programs into parts such that each module would contain related logic or use-cases. Then, these modules can be imported when needed.
    b. To make these modules available in other js files, we use the “import” keyword.
14. What is Object-Oriented Programming (OOP)?
    a. It’s pretty much a programming paradigm based on classes and objects. There are four main features in OOP,
        i. Abstraction. A way to abstract away or hide the actual implementation of the objects/methods and only show the functionality.
        ii. Encapsulation. A way to only show a subset of properties to the outside world (i.e. outside of the class and object) and hiding other critical information to the class, such as the method implementation and some other private data.
        iii. Inheritance. A way to inherit properties from the parent object, thus helping the reusability of a class.
        iv. Polymorphism. A way to implement the same method in many different ways.
15. What is prototype-based OOP in JS?
    a. A group of properties and methods that every object inherits from the given prototype.
    b. So all prototypes have a property “prototype”; all classes have a property; and all objects have a property “__proto__” that inherits from the class prototype.
16. What is the prototype chain?

    a. It refers to the cycle of finding the current object's prototype, then its parent's prototype until finding the property or reaching null. Everything in JS extends from Object.prototype and Object.prototype.__proto__ === null.
17. How do you implement inheritance in JavaScript before ES6 and with ES6?
    a. Before ES6, you would use constructor functions and prototypes to implement inheritance.
    b. With ES6, you would use classes, extend and super keywords to implement inheritance..