Lexical Scope

Lexical scope means that scope is defined by author-time decisions of where functions are declared. The lexing phase of compilation is essentially able to know where and how all identifiers are declared, and thus predict how they will be looked-up during execution.

eval(..) can "cheat" lexical scope. It can modify existing lexical scope (at runtime) by evaluating a string of "code" which has one or more declarations in it.

new scoping rules:

Functions

catch block

curly braces with let keyword (es6)

Prototype

Every object in Javascript has a prototype. When a messages reaches an object, JavaScript will attempt to find a property in that object first, if it cannot find it then the message will be sent to the object's prototype and so on

Closure

Closure is when a function remembers its lexical scope, even when the function is executed outside that lexical scope.

When the function is transported outside of its lexical scope, a closure is created where that function still has access to its lexical scope

Hoisting

Hoisting is the mechanism of moving the variables and functions declaration to the top of the function scope (or global scope if outside any function). function expressions are not hoisted

Function declaration: function bar(){}

Function expression: var foo = function bar() {}

this

In JavaScript this always refers to the "owner" of the function we're executing, or rather, to the object that a function is a method of. Every function while executing has a reference to its current execution context called this

In arrow functions, this is set lexically, i.e. it's set to the value of the enclosing execution context's this. In global code, it will be set to the global object:

Promises

A promise represents the eventual result of an asynchronous operation. It is a placeholder into which the successful result value or reason for failure will materialize.