Scope

Introduction

- Scope is the suite of variables and functions that is accessible at any given point in the code

```
function outer() {
  var x = 1;
  ... // <!-- is y accessible here?
  function inner() {
    var y = 2;
    ... // <!-- is outer accessible here?
  }
}
```

Hoisting

- The JS engine loads your variable and function declarations into memory during compilation

```
hello();

function hello() {
  console.log('Hello');
}
```

Global & Local Scope

- A variable or function declared outside of any other function is globally scoped

- Global variables are properties of the `window` object in the browser but need not be prefixed as such (it is assumed)

- A variable or function declared inside a function is locally scoped, as are function parameters

The Scope Chain

- Each function has a theoretical link to its parent scope, thereby enabling access to the parent scope's variables and functions

```
window: <---------------|
globalVar               |
outer(): <--------------|---|
    outerVar            |   |
    parentScope --|     |   |
    inner():            |   |
        innerVar        |   |
        parentScope --|
```

Block Scope

- Variables declared inside a function are scoped to that function; they are not accessible outside of the function

- Assuming `var` the same is not true of a conditional or iterative block, i.e. variables declared inside an if statement or for loop block can and do escape into the surrounding scope

- Variables declared using `let` are block scoped

Lexical Scope

- A variable declared in a function, f, is accessible to its inner functions even after f has been popped off the stack; said variable is lexically scoped, e.g.:

```
function f() {
  var lex = 'Hello';
  return function() { console.log(lex); }
}
```

- Lexically scoped variables facilitate the creation of functions with state, i.e. functions that maintain information between invocations

Closures

- A closure is a function and its lexical environment (local and lexical variables and functions)

```
function f() {
  var lex = 'Hello';
  return function() { console.log(lex); }
}
```

- The function returned by f is a closure

- NB: a closure captures references to the lexically scoped variables when the closure is created, but the values are assigned when the closure is invoked

IIFEs

- An IIFE is an immediately invoked functional expression, i.e. a function that is invoked at the point at which it is declared, e.g.:

```
(function() {
  console.log('I invoke myself');
})();
```

- IIFEs are commonly used to encapsulate data (see the revealing module pattern)

Summary

- Scope is the suite of variables & functions accessible at a point in the code
- Hoisting is the loading of declarations at compile-time
- The scope chain is the set of theoretical links from function to parent scope
- Variables declared with `let` are block scoped
- A lexically scoped variable is maintained in memory even after its declaring function is popped off the stack
- A closure is a function and its lexical environment
- An IFFE is an immediately invoked functional expression and is commonly used to encapsulate data