JS Scope & Hoisting

What is Scope?

- Scope defines everything you have access to
- Scope is like a pyramid
    - Lower scopes can access those above them, but not below
- The top level is the Global Scope
- Essentially, scoping is name resolution
    - Where can you access JavaScript identifiers in your code?

Lexical Scoping

- JavaScript uses Lexical Scoping
- Meaning scope is defined by the position in source code
    - Based on where the code is!

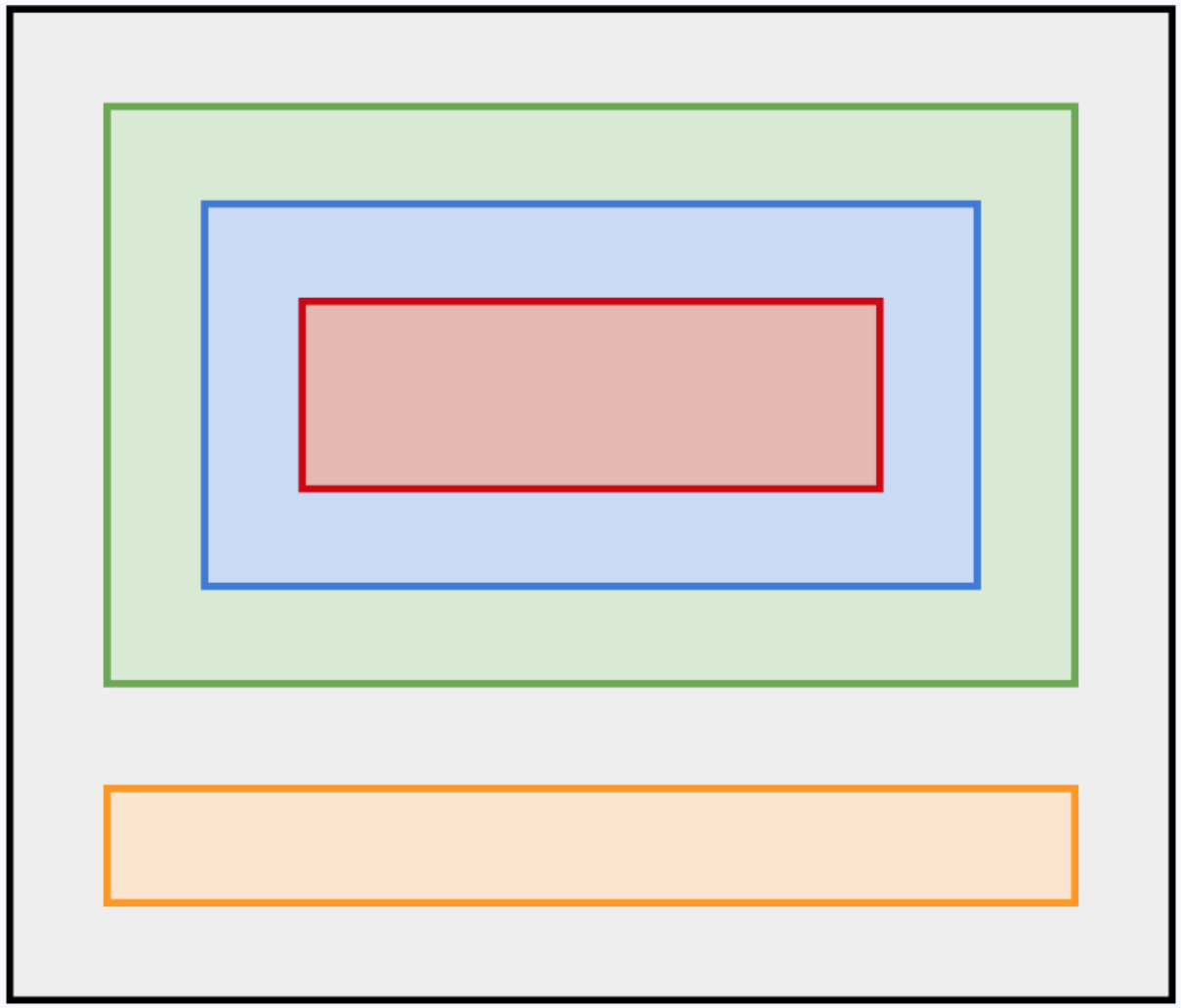
Function Scope

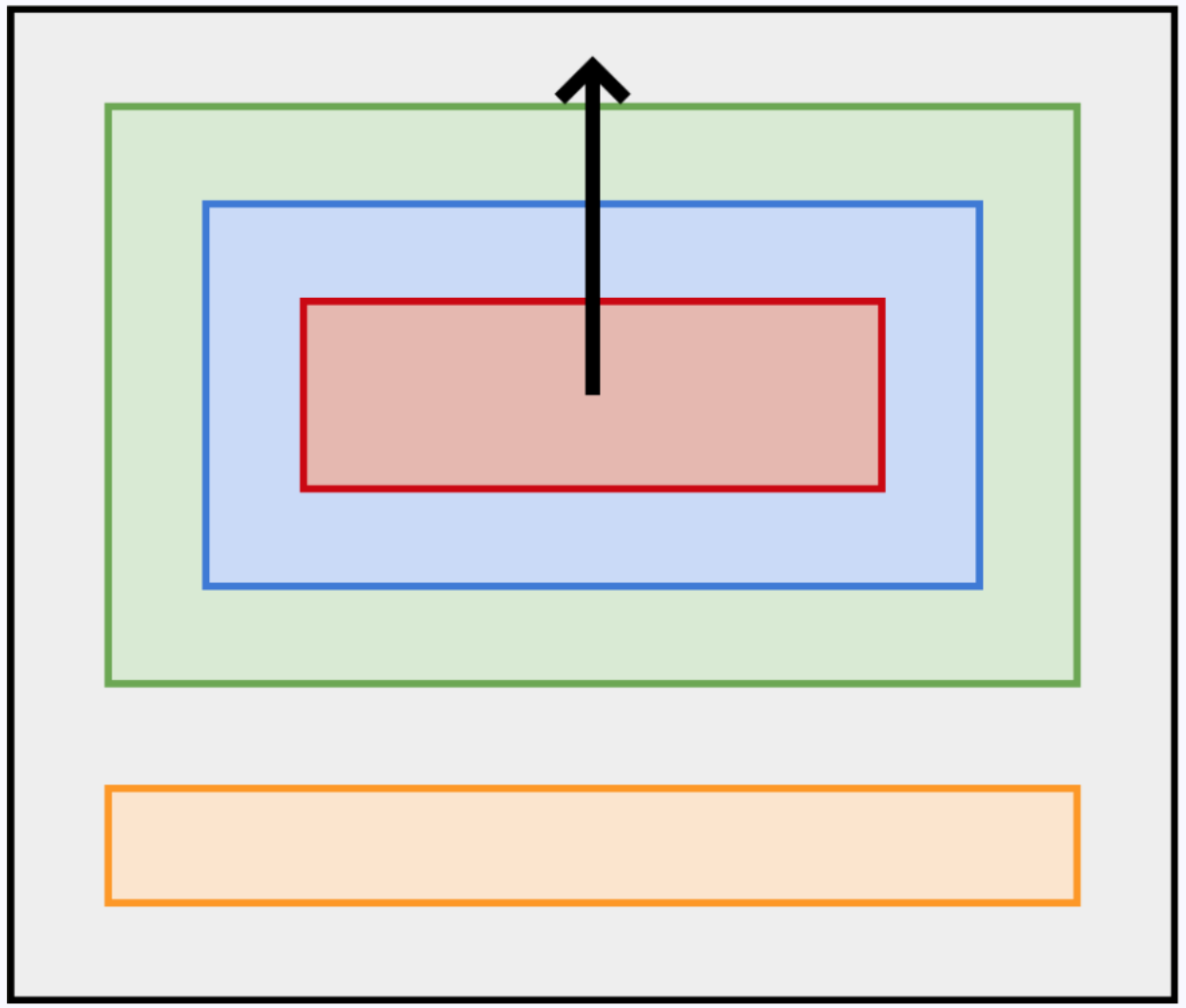
- Function Scoping is the scope that is created when a function is called
- The scope a function creates is called **Local Scope**

```
var global = "Global Scope";

function someFunction() {
  var local = "Local Scope";
}

console.log(global); // => "Global Scope"
console.log(local);  // => ReferenceError
```

Function Scope

```
var global = "Global Scope";

function someFunction() {
    var innerScope = "Inner Scope";

    function someInnerFunction() {
        var innerInnerScope = "InnerInner Scope";
        // 3. What can we access from here?
    }

    someInnerFunction();
    // 2. What can we access from here?
}

someFunction();
// 1. What can we access from here?
```





What is Hoisting?

- It is a way to explain the way that *execution contexts* work
- One way to think of it...
    - Variable and function declarations get moved to the top of the scope
- ...But really, they get put in memory during the compile phase

Exercise

Go through the quiz found here

Closures

What are Closures?

A fancy name for a function that has access to an outer scope's variables etc.

Why would you use them?

- Useful for securing your web applications
- You can create private data and functions
- You can create utility functions easily

What are Closures?

```
function createGame() {
  var score = 0;
  return function scoreGoal() {
    score += 10;
    return score;
  }
}

var scoreGoal = createGame();

console.log( scoreGoal() );
console.log( scoreGoal() );
```

What are Closures?

```
function createGame() {
  var score = 0;
  return {
    gainPoints: function() {
      return score += 10;
    },
    losePoints: function() {
      return score -= 10;
    },
    getScore: function() {
      console.log( score );
    }
  };
}

var player = createGame();
player.gainPoints();
```

IIFE

Immediately Invoked Function Expressions

Useful for creating a new scope!

A function that runs straight away

Bonus Exercise!

```
function printNumbers() {
  for (var i = 0; i < 10; i += 1) {
    setTimeout(function() {
      console.log(i);
    }, i * 100);
  }
}

printNumbers();

// The timer is broken!
// It prints out 10, 10 times
// Make it work properly!
// Hint: You may need to create a new scope
// Try and explain why it happens
```