SCOPE

WHAT IS SCOPE?

- Scope is the set of variables you can access from a given point of code
- Inside functions, the local scope is defined as parameters given to the function, “var” variable statements and function declarations.
- Inside a function is the only place that a new scope is created.

DOES THIS WORK?

```
1  function addTen (x) {
2    var num = 10;
3    return x + num;
4  }
5
6  console.log(addTen(4)) // -> 14
```

DOES THIS WORK AS WELL?

```
1    var num = 10;
2
3    function addTen (x,num) {
4      return x + num;
5    }
6
7    console.log(addTen(4, num))
```

WHAT ABOUT?

```
1   var num = 10;
2
3   function addTen (x) {
4     return x + num;
5   }
6
7   console.log(addTen(4))
```

DOES THIS WORK AS WELL?

```
1   function createTen() {
2     var num = 10;
3   }
4
5   function addTen(n) {
6     retrun n + num;
7   }
8
9   console.log(addTen(4));
```

WHAT ABOUT?

```
1   function createTen() {
2     var num = 10;
3   }
4
5   createTen();
6   console.log(num);
```

```
);
```

WHAT ABOUT?

```
1   function createNum(){
2     return 5;
3   }
4
5   var num = createNum();
6
7   console.log('num', num);
```

FUNCTIONS CREATE NEW SCOPE

- Imagine the brackets surrounding a function body are like one-way glass. Code enclosed by the brackets can see out, but the code outside the brackets cannot see in.

```
1   var num1 = 10
2
3   function printNum() {
4     var num2 = 3;
5     console.log(num1,num2);
6   }
7
8   printNum(); // 10 3
9   console.log(num1) // 10
10  console.log(num2) //ReferenceError
```

GLOBAL SCOPE

- This is the Danger Zone!!!
- Variables declared without a var keyword are attached to the global scope

DON'T EVER DO THIS

```
1   function helloWorld () {
2     sayHello = "Hello World"
3     return sayHello
4   }
```

sayHello is hoisted to the global scope

WHY NOT?

- You might overwrite something you don’t want to

```
1   var doYouTrustMe = true;
2
3   function believeInMe() {
4     return doYouTrustMe === true;
5   }
6
7   function dontLikeMe() {
8     doYouTrustMe = false // changing a global
      variable!
9   }
0
1   dontLikeMe();
2
3   if(believeInMe() === true) {
4     console.log("Everything is going to be
      alright")
5   } else {
6     throw new Error("An error occured and its
      is Bad News Bears!!!");
7   }
```

FUNCTION IN A FUNCTION: VARIABLES

```
1   var favoriteFood = 'pizza';
2
3   function outer() {
4     var favoriteFood = 'sushi';
5
6     function inner() {
7       console.log('my favorite variety of
        food is', favoriteFood);
8     }
9     inner();
10  }
11
12  outer(); // Which favorite will be logged?
```

FUNCTION IN A FUNCTION: PARAMETERS

```
1   var favoriteFood = 'pizza';
2
3   function outer(favoriteFood) {
4
5     function inner() {
6       console.log('my favorite variety of
        food is', favoriteFood);
7     }
8     inner();
9   }
10
11  outer('sushi'); // Which favorite will be
    logged?
```

HOISTING

- The JavaScript interpreter makes two passes over your code: variables are created on the first pass, and they're assigned their values on the second.

```
86    console.log(x);
```

// ReferenceError

```
86    console.log(x);
87
88    var x = 2;
```

// undefined
Even though x has not been assigned its value yet, the variable still exists

- The difference between functions declarations and function expressions is how they're hoisted.
- In the case of function expression (starts with the word 'var'), the variable is hoisted without its value. Hence the TypeError.
- In the case of function declarations (starting with the 'function'), the variable is hoisted along with its value(the function). Hence the successful call.

```
92   sayHi();  // TypeError: sayHi is not a function
93
94   var sayHi = function() {
95       console.log('Hi!');
96   }
```

```
85   sayHi();  // 'Hi!'
86
87   function sayHi() {
88       console.log('Hi!');
89   }
```