Scope

Overview

```
/*
   - What is scope?

   - Types of scope
     - Global
     - Functional (local)
     - Block

   - Scope best practices
*/
```

What is scope?

```
1   /* Scope refers to which variables can be accessed by your code at a
2      specific location in your code. */
3
4   let wow = 'wow';
5   console.log('I can access wow because it is in scope:', wow);
6
7
8
9
10
11
12
13
14
```

outer ear
inner ear

Global scope

```
1   /* JS is 'lexically scoped', which means the location at which a variable
2      is declared determines its scope. */
3
4   /* A variable that is declared outside of a function is globally scoped;
5      it can be referenced from any line of code throughout the file */
6   let global = 'ear';
7
8   console.log('outer', global);
9
10  function funFunction() {
11    console.log('inner', global);
12  }
13
14  funFunction();
```

undefined
here I am

Global scope

```
1   /* Even though a global variable can be referenced from anywhere in your
2      code, the value assigned to the variable cannot be accessed until after
3      the assignment operation occurs. */
4
5   console.log(waitForIt);
6
7   let waitForIt = 'here I am';
8
9   console.log(waitForIt);
10
11
12
13
14
```

Functional scope

```
I am so happy!
ReferenceError: message is not defined
```

```
1   /* Variables declared inside of a function are 'locally-scoped'. */
2
3   /* They cannot be referenced outside of the function. */
4
5   function happyFunction() {
6     let message = 'I am so happy!';
7     console.log(message);
8   }
9
10  happyFunction();
11  console.log(message);
12
13
14
```

Functional scope

```
1   /* What if a variable is defined locally and globally? */
2
3   let message = 'think globally';
4
5   function logAMessage() {
6     let message = 'act locally';
7
8     // JS will look for message locally, first
9     console.log(message);
10  }
11
12  logAMessage();
13
14
```

Functional scope

```
1   /* What if a variable is defined locally and globally? */
2
3   let message = 'think globally';
4
5   function logAMessage() {
6     let msg = 'act locally';
7
8     /* if it can't find it locally, JS will look at the scope outside the
9        function, this case, the global scope */
10    console.log(message);
11  }
12
13  logAMessage();
14
```

Functional scope

```
1   /* parameters are also locally scoped */
2
3   let message = 'think globally';
4
5   function logAMessage(message) {
6     console.log(message);
7   }
8
9   logAMessage('act locally');
10
11
12
13
14
```

global outer inner

Functional scope

```
1   /* consider nested functions */
2   let globalVar = 'global';
3
4   function outer() {
5     let outerVar = 'outer';
6
7     function inner() {
8       let innerVar = 'inner';
9       console.log(globalVar, outerVar, innerVar);
10    }
11    inner();
12  }
13
14  outer();
```

inner

Functional scope

```
1   /* consider nested functions */
2   let collision = 'global';
3
4   function outer(collision) {
5
6     function inner() {
7       let collision = 'inner';
8       console.log(collision);
9     }
10
11    inner();
12  }
13
14  outer('outer');
```

outer

Functional scope

```
1   /* consider nested functions */
2   let collision = 'global';
3
4   function outer(collision) {
5
6     function inner() {
7       console.log(collision);
8     }
9     inner();
10  }
11
12  outer('outer');
13
14
```

Functional scope

```
1   /* consider nested functions */
2   let collision = 'global';
3
4   function outer() {
5
6     function inner() {
7       console.log(collision);
8     }
9     inner();
10  }
11
12  outer('outer');
13
14
```

Functional scope

```
outer inner
ReferenceError: innerVar is not defined
```

```
1   /* note the inner function can access the scope of the outer function,
2      but the opposite is not true */
3   function outer() {
4     let outerVar = 'outer';
5
6     function inner() {
7       let innerVar = 'inner';
8       console.log(outerVar, innerVar);
9     }
10    inner();
11    console.log(innerVar);
12  }
13
14  outer();
```

undefined inner

Functional scope

```
1   /* the inner function still looks for a local declaration of the variable
2      name before looking at the next level of scope */
3
4   function outer() {
5     let outerVar = 'outer';
6
7     function inner(outerVar) {
8       let innerVar = 'inner';
9       console.log(outerVar, innerVar);
10    }
11    inner();
12  }
13
14  outer();
```

Block scope

```
1   /* Any block of code (code inside of curly brackets) creates its own
2      scope, too */
3
4   if (true) {
5     let block = 'Jenny from the';
6     console.log(block, 'block');
7   }
8
9   console.log(block);
10
11
12
13
14
```

Jenny from the block
ReferenceError: block is not defined

Jenny from the block
Jenny from the

Block scope

```
1   /* the pre-ES6 var keyword ignores block scope */
2
3   if (true) {
4     var block = 'Jenny from the';
5     console.log(block, 'block');
6   }
7
8   console.log(block);
9
10
11
12
13
14
```

Error: everything is broken

Scope best practices

```
1   /* functions generally should not change globally scoped variables */
2   let alwaysTrue = true;
3
4   function dontMindMe() {
5     alwaysTrue = false; // danger! changing global variable!
6   }
7
8   dontMindMe();
9
10  if (alwaysTrue) {
11    console.log('all is well');
12  } else {
13    throw new Error('everything is broken');
14  }
```

Scope best practices

```
1   /* functions generally should not change globally scoped variables */
2   let alwaysTrue = true;
3
4   function dontMindMe() {
5     let alwaysTrue = false; // this is ok, just creating a local variable
6   }
7
8   dontMindMe();
9
10  if (alwaysTrue) {
11    console.log('all is well');
12  } else {
13    throw new Error('everything is broken');
14  }
```

Scope best practices

```
1   /* avoid cluttering the global namespace with lots of variables */
2
3   /* only declare variables globally if they need to be accessed globally */
4
5   /* otherwise, it's safer to declare variables in functions or blocks so
6      you don't overwrite variables accidentally, or access the wrong
7      variable by mistake (both likely sources of bugs) */
8
9
10
11
12
13
14
```

Recap

```
/*
   - What is scope?

   - Types of scope
     - Global
     - Functional (local)
     - Block

   - Scope best practices
*/
```