Scope



Overview

```
- What is scope?
    - Types of scope
      - Global
     - Functional (local)
      - Block
     - Scope best practices
10
   */
```

What is scope?

```
/* Scope refers to which variables can be accessed by your code at a
   specific location in your code. */
let wow = 'wow';
console.log('I can access wow because it is in scope:', wow);
```



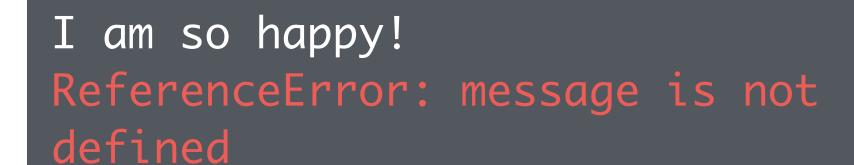
Global scope

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

Global scope

```
/* Even though a global variable can be referenced from anywhere in your
      code, the value assigned to the variable cannot be accessed until after
      the assignment operation occurs. */
   console.log(waitForIt);
   let waitForIt = 'here I am';
   console.log(waitForIt);
10
```





```
/* Variables declared inside of a function are 'locally-scoped'. */
/* They cannot be referenced outside of the function. */
function happyFunction() {
  let message = 'I am so happy!';
  console.log(message);
happyFunction();
console.log(message);
```



```
/* What if a variable is defined locally and globally? */
   let message = 'think globally';
   function logAMessage() {
     let message = 'act locally';
     // JS will look for message locally, first
     console.log(message);
10 }
   logAMessage();
```

```
/* What if a variable is defined locally and globally? */
   let message = 'think globally';
   function logAMessage() {
     let msg = 'act locally';
     /* if it can't find it locally, JS will look at the scope outside the
        function, this case, the global scope */
     console.log(message);
13 logAMessage();
```



```
/* parameters are also locally scoped */
   let message = 'think globally';
   function logAMessage(message) {
     console.log(message);
   logAMessage('act locally');
10
```



```
/* consider nested functions */
   let globalVar = 'global';
   function outer() {
     let outerVar = 'outer';
     function inner() {
       let innerVar = 'inner';
       console.log(globalVar, outerVar, innerVar);
     inner();
14 outer();
```



```
/* consider nested functions */
   let collision = 'global';
   function outer(collision) {
     function inner() {
       let collision = 'inner';
       console.log(collision);
     inner();
14 outer('outer');
```



```
/* consider nested functions */
let collision = 'global';
function outer(collision) {
  function inner() {
    console.log(collision);
  inner();
outer('outer');
```



```
/* consider nested functions */
let collision = 'global';
function outer() {
  function inner() {
    console.log(collision);
  inner();
outer('outer');
```





outer inner

```
/* note the inner function can access the scope of the outer function,
      but the opposite is not true */
   function outer() {
     let outerVar = 'outer';
     function inner() {
       let innerVar = 'inner';
       console.log(outerVar, innerVar);
     inner();
     console.log(innerVar);
14 outer();
```



```
/* the inner function still looks for a local declaration of the variable
      name before looking at the next level of scope */
   function outer() {
     let outerVar = 'outer';
     function inner(outerVar) {
       let innerVar = 'inner';
       console.log(outerVar, innerVar);
     inner();
14 outer();
```





Jenny from the block ReferenceError: block is not defined

```
/* Any block of code (code inside of curly brackets) creates its own
      scope, too */
   if (true) {
     let block = 'Jenny from the';
     console.log(block, 'block');
   console.log(block);
10
```

Block scope

```
/* the pre-ES6 var keyword ignores block scope */
   if (true) {
     var block = 'Jenny from the';
     console.log(block, 'block');
   console.log(block);
10
```



```
/* functions generally should not change globally scoped variables */
   let alwaysTrue = true;
   function dontMindMe() {
     alwaysTrue = false; // danger! changing global variable!
   dontMindMe();
   if (alwaysTrue) {
     console.log('all is well');
12 } else {
     throw new Error('everything is broken');
```

Scope best practices

```
/* functions generally should not change globally scoped variables */
   let alwaysTrue = true;
   function dontMindMe() {
     let alwaysTrue = false; // this is ok, just creating a local variable
   dontMindMe();
   if (alwaysTrue) {
     console.log('all is well');
12 } else {
     throw new Error('everything is broken');
```



```
/* avoid cluttering the global namespace with lots of variables */
/* only declare variables globally if they need to be accessed globally */
/* otherwise, it's safer to declare variables in functions or blocks so
   you don't overwrite variables accidentally, or access the wrong
   variable by mistake (both likely sources of bugs) */
```



Recap

```
- What is scope?
    - Types of scope
      - Global
   - Functional (local)
      - Block
    - Scope best practices
10
```