Chapter 17: Declarations and Assignments

Section 17.1: Modifying constants

Declaring a variable `const` only prevents its value from being *replaced* by a new value. `const` does not put any restrictions on the internal state of an object. The following example shows that a value of a property of a `const` object can be changed, and even new properties can be added, because the object that is assigned to `person` is modified, but not *replaced*.

```
const person = {
    name: "John"
};
console.log('The name of the person is', person.name);

person.name = "Steve";
console.log('The name of the person is', person.name);

person.surname = "Fox";
console.log('The name of the person is', person.name, 'and the surname is', person.surname);
```

Result:

```
The name of the person is John
The name of the person is Steve
The name of the person is Steve and the surname is Fox
```

In this example we've created constant object called `person` and we've reassigned `person.name` property and created new `person.surname` property.

Section 17.2: Declaring and initializing constants

You can initialize a constant by using the `const` keyword.

```
const foo = 100;
const bar = false;
const person = { name: "John" };
const fun = function () { /* ... */ };
const arrowFun = () => /* ... */ ;
```

Important
You must declare and initialize a constant in the same statement.

Section 17.3: Declaration

There are four principle ways to declare a variable in JavaScript: using the `var`, `let` or `const` keywords, or without a keyword at all ("bare" declaration). The method used determines the resulting scope of the variable, or reassignability in the case of `const`.

- The `var` keyword creates a function-scope variable.
- The `let` keyword creates a block-scope variable.
- The `const` keyword creates a block-scope variable that cannot be reassigned.
- A bare declaration creates a global variable.

```
var a = 'foo';    // Function-scope
```

```
let b = 'foo';    // Block-scope
const c = 'foo';  // Block-scope & immutable reference
```

Keep in mind that you can't declare constants without initializing them at the same time.

```
const foo; // "Uncaught SyntaxError: Missing initializer in const declaration"
```

(An example of keyword-less variable declaration is not included above for technical reasons. Continue reading to see an example.)

Section 17.4: Undefined

Declared variable without a value will have the value `undefined`

```
var a;

console.log(a); // logs: undefined
```

Trying to retrieve the value of undeclared variables results in a ReferenceError. However, both the type of undeclared and unitialized variables is "undefined":

```
var a;
console.log(typeof a === "undefined"); // logs: true
console.log(typeof variableDoesNotExist === "undefined"); // logs: true
```

Section 17.5: Data Types

JavaScript variables can hold many data types: numbers, strings, arrays, objects and more:

```
// Number
var length = 16;

// String
var message = "Hello, World!";

// Array
var carNames = ['Chevrolet', 'Nissan', 'BMW'];

// Object
var person = {
    firstName: "John",
    lastName: "Doe"
};
```

JavaScript has dynamic types. This means that the same variable can be used as different types:

```
var a;              // a is undefined
var a = 5;          // a is a Number
var a = "John";     // a is a String
```

Section 17.6: Mathematic operations and assignment

Increment by

```
var a = 9,
b = 3;
```

```
b += a;
```

b will now be 12

This is functionally the same as

```
b = b + a;
```

Decrement by

```
var a = 9,
b = 3;
b -= a;
```

b will now be 6

This is functionally the same as

```
b = b - a;
```

Multiply by

```
var a = 5,
b = 3;
b *= a;
```

b will now be 15

This is functionally the same as

```
b = b * a;
```

Divide by

```
var a = 3,
b = 15;
b /= a;
```

b will now be 5

This is functionally the same as

```
b = b / a;
```

Version ≥ 7

Raised to the power of

```
var a = 3,
b = 15;
b **= a;
```

b will now be 3375

This is functionally the same as

```
b = b ** a;
```

Section 17.7: Assignment

To assign a value to a previously declared variable, use the assignment operator, `=`:

```
a = 6;
b = "Foo";
```

As an alternative to independent declaration and assignment, it is possible to perform both steps in one statement:

```
var a = 6;
let b = "Foo";
```

It is in this syntax that global variables may be declared without a keyword; if one were to declare a bare variable without an assignment immediately afterword, the interpreter would not be able to differentiate global declarations `a;` from references to variables `a;`.

```
c = 5;
c = "Now the value is a String.";
myNewGlobal;    // ReferenceError
```

Note, however, that the above syntax is generally discouraged and is not strict-mode compliant. This is to avoid the scenario in which a programmer inadvertently drops a **`let`** or **`var`** keyword from their statement, accidentally creating a variable in the global namespace without realizing it. This can pollute the global namespace and conflict with libraries and the proper functioning of a script. Therefore global variables should be declared and initialized using the **`var`** keyword in the context of the window object, instead, so that the intent is explicitly stated.

Additionally, variables may be declared several at a time by separating each declaration (and optional value assignment) with a comma. Using this syntax, the var and let keywords need only be used once at the beginning of each statement.

```
globalA = "1", globalB = "2";
let x, y = 5;
var person = 'John Doe',
    foo,
    age = 14,
    date = new Date();
```

Notice in the preceding code snippet that the order in which declaration and assignment expressions occur (**`var`** `a, b, c = 2, d;`) does not matter. You may freely intermix the two.

Function declaration effectively creates variables, as well.