JavaScript const: Declaring Constants in ES6

Summary: in this tutorial, you'll learn how to define constants by using the JavaScript `const` keyword.

Introduction to the JavaScript const keyword

ES6 provides a new way of declaring a constant by using the `const` keyword. The `const` keyword creates a read-only reference to a value.

```
const CONSTANT_NAME = value;
```

By convention, the constant identifiers are in uppercase.

Like the let keyword, the `const` keyword declares blocked-scope variables. However, the block-scoped variables declared by the const keyword can't be **reassigned**.

The variables declared by the `let` keyword are mutable. It means that you can change their values anytime you want as shown in the following example:

```
let a = 10;
a = 20;
a = a + 5;
console.log(a); // 25
```

However, variables created by the `const` keyword are "immutable". In other words, you can't reassign them to different values.

If you attempt to reassign a variable declared by the `const` keyword, you'll get a `TypeError` like this:

```
const RATE = 0.1;
RATE = 0.2; // TypeError
```

Unlike the let keyword, you need to initialize the value to the variable declared by the const keyword.

The following example causes a `SyntaxError` due to missing the initializer in the `const` variable declaration:

```
const RED; // SyntaxError
```

JavaScript const and Objects

The `const` keyword ensures that the variable it creates is read-only. However, it doesn't mean that the actual value to which the `const` variable reference is immutable. For example:

```
const person = { age: 20 };
person.age = 30; // OK
console.log(person.age); // 30
```

Even though the `person` variable is a constant, you can change the value of its property.

However, you cannot reassign a different value to the `person` constant like this:

```
person = { age: 40 }; // TypeError
```

If you want the value of the `person` object to be immutable, you have to freeze it by using the `Object.freeze()` method:

```
const person = Object.freeze({age: 20});
person.age = 30; // TypeError
```

Note that `Object.freeze()` is shallow, meaning that it can freeze the properties of the object, not the objects referenced by the properties.

For example, the `company` object is constant and frozen.

```
const company = Object.freeze({
    name: 'ABC corp',
    address: {
        street: 'North 1st street',
        city: 'San Jose',
        state: 'CA',
        zipcode: 95134
    }
});
```

But the `company.address` object is not immutable, you can add a new property to the `company.address` object as follows:

```
company.address.country = 'USA'; // OK
```

JavaScript const and Arrays

Consider the following example:

```
const colors = ['red'];
colors.push('green');
console.log(colors); // ["red", "green"]

colors.pop();
colors.pop();
console.log(colors); // []

colors = []; // TypeError
```

In this example, we declare an array `colors` that has one element using the `const` keyword. Then, we can change the array's elements by adding the `green` color. However, we cannot reassign the array `colors` to another array.

JavaScript const in a for loop

ES6 provides a new construct called `for...of` that allows you to create a loop iterating over iterable objects such as arrays, maps, and sets.

```
let scores = [75, 80, 95];

for (let score of scores) {
        console.log(score);
}
```

If you don't intend to modify the `score` variable inside the loop, you can use the `const` keyword instead:

```
let scores = [75, 80, 95];
for (const score of scores) {
    console.log(score);
}
```

In this example, the `for...of` creates a new binding for the `const` keyword in each loop iteration. In other words, a new `score` constant is created in each iteration.

Notice that the `const` will not work in an imperative for loop. Trying to use the `const` keyword to declare a variable in the imperative `for` loop will result in a `TypeError`:

```
for (const i = 0; i < scores.length; i++) { // TypeError
    console.log(scores[i]);
}
```

The reason is that the declaration is only evaluated once before the loop body starts.

Summary

- The `const` keyword creates a read-only reference to a value. The readonly reference cannot be reassigned but the value can be changed.

- The variables declared by the const keyword are blocked-scope and cannot be redeclared.