JavaScript Numbers

Summary: in this tutorial, you'll learn about the JavaScript number types and how to use them effectively.

Introduction to the JavaScript Number

JavaScript uses the `number` type to represent both integers and floating-point values. Technically, the JavaScript `number` type uses the IEEE-754 format.

ES2020 introduced a new primitive type `bigint` representing big integer numbers with values larger than $2^{53} – 1$.

To support various types of numbers, JavaScript uses different number literal formats.

Integer numbers

The following shows how to declare a variable that holds a decimal integer:

```
let counter = 100;
```

Integers can be represented in the following formats:

- Octal (base 8)
- Hexadecimal (based 16)

When you use the octal and hexadecimal numbers in arithmetic operations, JavaScript treats them as decimal numbers.

Octal numbers

An octal literal number starts with the digit zero (0) followed by a sequence of octal digits (numbers from 0 through 7). For example:

```
let num = 071;
console.log(num);
```

Output:

```
57
```

If an octal number contains a number not in the range from 0 to 7, the JavaScript engine ignores the 0 and treats the number as a decimal. For example:

```
let num = 080;
console.log(num);
```

Output:

```
80
```

This implicit behavior might cause issues. Therefore, ES6 introduced a new octal literal that starts with the `0o` followed by a sequence of octal digits (from 0 to 7). For example:

```
let num = 0o71;
console.log(num);
```

Output:

```
57
```

If you have an invalid number after `0o` , JavaScript will issue a syntax error like this:

```
let num = 0o80;
console.log(num);
```

Output:

```
let num = 0o80;
          ^^
SyntaxError: Invalid or unexpected token
```

Hexadecimal numbers

Hexadecimal numbers start with 0x or 0X followed by any number of hexadecimal digits (0 through 9, and a through f). For example:

```
let num = 0x1a;
console.log(num);
```

Output:

```
26
```

Floating-point numbers

To define a floating-point literal number, you include a decimal point and at least one number after that. For example:

```
let price = 9.99;
let tax = 0.08;
let discount = .05; // valid but not recommeded
```

When you have a very big number, you can use e-notation. E-notation indicates a number should be multiplied by 10 raised to a given power. For example:

```
let amount = 3.14e7;
console.log(amount);
```

Output:

```
31400000
```

The notation `3.14e7` means that take `3.14` and multiply it by `10`^{7} .

Likewise, you can use the E-notation to represent a very small number. For example:

```
let amount = 5e-7;
console.log(amount);
```

Output:

```
0.0000005
```

The 5e-7 notation means that take 5 and divide it by 10,000,000.

Also, JavaScript automatically converts any floating-point number with at least six zeros after the decimal point into e-notation. For example:

```
let amount = 0.0000005;
console.log(amount);
```

Output:

```
5e-7
```

Floating-point numbers are accurate up to 17 decimal places. When you perform arithmetic operations on floating-point numbers, you often get the approximate result. For example:

```
let amount = 0.2 + 0.1;
console.log(amount);
```

Output:

```
0.30000000000000004
```

Big Integers

JavaScript introduced the `bigint` type starting in ES2022. The `bigint` type stores whole numbers whose values are greater than $2^{53} – 1$.

A big integer literal has the `n` character at the end of an integer literal like this:

```
let pageView = 9007199254740991n;
```

Quiz

Summary

- JavaScript Number type represents both integer and floating-point numbers.