JS Expressions, Operators, and Control Structures

Asst.Prof. Dr. Umaporn Supasitthimethee
ผศ.ดร.อุมาพร สุภสิทธิเมธี

JavaScript | MDN (mozilla.org)

JavaScript: The Definitive Guide, Seventh Edition, by David Flanagan

JavaScript Operators

Operator precedence and associativity specify the order in which operations are performed in a complex expression.

Precedence (High)

(Low)

- Optional Chaining
  ?. — 1
- Increment and Decrement
  ++ -- — 2
- Invert Boolean value
  ! — 3
- Type of operand
  `typeof`
- Arithmetic operators
  - * / % — 4
  - + - — 5
- Relational operators
  < <= > >= — 6
- Equality operators
  == != (non strict equality)
  === !== (strict equality) — 7
- Logical operators
  && — 8
- Logical operators
  || — 9
- Nullish Coalescing
  ?? — 9
- Assignment operators
  += -= *= /= %= — 10
- Conditional Operator
  ?: — 11

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

```
//Arithmetic operators
console.log(5 + 2) // => 7: addition
console.log(5 - 2) // => 3: subtraction
console.log(5 * 2) // => 10: multiplication
console.log(5 / 2) // => 2.5: division

// JavaScript defines some shorthand arithmetic operators
let count = 0 // Define a variable
count++ // Increment the variable
count-- // Decrement the variable
count += 3 // Add 3: same as count = count + 3
count *= 2 // Multiply by 2: same as count = count * 2
console.log(`count = ${count}`) // => 6: variable names are expressions

//conditional operator
let result = count > 5 ? 'count > 5' : ' count<=5'
console.log(`result = ${result}"`)

//== and != non-strict equality
//If the two operands are different types, interpreter attempts to convert them to suitable type.
console.log(`15 == '15' ${15 == '15'}`) //true

//=== and!=== strict equality without type conversion
console.log(`15 === '15' ${15 === '15'}`) //false

//logical operators
// && (and), || (or), ! (not)

console.log(`5 < '10' && '1' > 5 is  ${5 < '10' && '1' > 5}`) //false
console.log(`5 < '10' || '1' > 5 is  ${5 < '10' || '1' > 5}`) //true
console.log(`!(0) is ${!0}`) //true
```

Conditional Operator (?:)

- The first operand is evaluated and interpreted as a Boolean.
- If the value of the first operand is true, then the second operand is evaluated, and its values is returned.
- Otherwise, if the first operand is false, then the third operand is evaluated, and its value is returned.

boolean condition? *true action*: *false action*

```
let age = 40
const greeting = age > 40 ? 'Good Morning' : 'Hello'
console.log(greeting)
```

Optional chaining (?.)

- ES2020 adds two new kinds of property access expressions

```
Expression ?.identifier
Expression ?.[expression]
```

- In JavaScript, the values `null` and `undefined` are the only two values that do not have properties.
- In a regular properly access expression using `.` or `[]`, you get a type error if the expression on the left evaluates to `null` or `undefined`.
- You can use `?.` and `?.[]` syntax to guard against errors of this type.
- `?.` is short-circuiting, if the subexpression to the left of `?.` evaluates to `null` or `undefined`, then the entire expression immediately evaluates to `undefined` without any further property access attempts.

Optional chaining (?.)

```
// array
let index = 0
let data
const firstItem = data?.[0]
console.log(firstItem)

// object
let student
console.log(student?.firstName)

```

Nullish Coalescing (??)

- ?? Operator is defined by ES2020.
- If its left operand is not `null` and not `undefined`, it returns that value.
- Otherwise, it returns the value of the right operand.
- It equals to: *(a !==null && a!== undefined) ? a: b*

```
// nullish coalescing
let items = null
items = items ?? []
console.log(items)

let options = { delay: '5ms' }
options.title = options.title ?? 'untitled'
console.log(options)
```

Math

- Math is a built-in object that has properties and methods for mathematical constants and functions. All properties and methods of Math are static.
- Static properties
  - Math.PI - ratio of a circle's circumference to its diameter; approximately 3.14159.
- Static methods
  - Math.random() returns a pseudo-random number between 0 and 1.
  - Math.ceil(*x*) returns the smallest integer greater than or equal to x.
  - Math.floor(*x*) returns the largest integer less than or equal to x.
  - Math.pow(*x*, *y)* returns base x to the exponent power y (that is, x^y).

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

Conversions and Equality

- JavaScript has two operators that test whether two values are equal.
- The “strict equality operator,” ===, does not consider its operands to be equal if they are not of the same type.
- But because JavaScript is so flexible with type conversions, it also defines the == operator with a flexible definition of equality.

```
null == undefined // true: These two values are treated as equal.
"0" == 0          // true: String converts to a number before comparing.
0 == false        // true: Boolean converts to number before comparing.
"0" == false      // true: Both operands convert to 0 before comparing.


//if change from  == to strict equality  === , the results are all FALSE!
```

Equality with type conversion

- The equality operator == is like the strict equality operator, but it is less strict. If the values of the two operands are not the same type, it attempts some type conversions and tries the comparison again:
  - If the **two values have the same type**, test them for strict equality as described previously. **If they are strictly equal, they are equal**. **If they are not strictly equal, they are not equal.**
  - If the **two values do not have the same type**, **the == operator may still consider them equal**. It uses the following rules and type conversions to check for equality:
    - If **one value is null and the other is undefined**, they are **equal.**
    - If **one value is a number and the other is a string, convert the string to a number** and try the comparison again, using the converted value.
    - **If either value is true, convert it to 1** and try the comparison again. **If either value is false, convert it to 0** and try the comparison again.
    - Any other combinations of values are not equal
  - If **one of the operands is an object** and **the other is a number or a string, try to convert the object to a primitive using the object's valueOf() and toString() methods**.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Equality

JavaScript String

- Strings can be compared with the standard === equality and !== inequality operators
- two strings are equal if and only if they consist of exactly the same sequence of 16-bit values.
- Strings can also be compared with the <, <=, >, and >= operators. String comparison is done simply by comparing the 16-bit values.
- To determine the length of a string—the number of 16-bit values it contains—use the length property of the string: `str.length`

```
let str1 = 'Hello'
let str2 = 'hello'
console.log(`str1 === str2 is ${str1 === str2}`)
console.log(`str1 < str2 is ${str1 < str2}`)
console.log(`str1 > str2 is ${str1 > str2}`)
console.log(`str1.length = ${str1.length}`)
console.log(
  `str1.toLowerCase === str2.toLowerCase is ${
    str1.toLowerCase === str2.toLowerCase
  }`
)
console.log(`str1.charAt(str1.length-1) = ${str1.charAt(str1.length -
 1)}`)
```

```
//output
str1 === str2 is false
str1 < str2 is true
str1 > str2 is false
str1.length = 5
str1.toLowerCase === str2.toLowerCase is true
str1.charAt(str1.length-1) = o
```

Comparing Primitives vs Objects

- **Primitives are also compared by value**: two values are the same only if they have the same value.
- **Objects are not compared by value**: two distinct objects are not equal even if they have the same properties and values.
- **Objects** are sometimes called **reference types** to distinguish them from JavaScript's primitive types
- we say that objects are compared by reference: **two object values are the same if and only if they refer to the same underlying object.**

Comparing Objects

```
let myObj = {
  id: 1,
  task: 'grading exam'
}

let myObj2 = {
  id: 1,
  task: 'grading exam'
}

newObj = myObj
console.log(`newObj === myObj is ${newObj === myObj}`)
console.log(`myObj1 === myObj2 is ${myObj === myObj2}`)
```

```
//output
newObj === myObj is true
myObj1 === myObj2 is false
```

Comparing Array

Two distinct arrays are not equal even if they have the same elements in the same order:

```
let a = []
let b = a
b[0] = 1
let c = [1]
console.log(`a === b is ${a === b}`)
console.log(`b == c is ${b == c}`)
```

```
//output
a === b is true
b == c is false
```

Conditionals – *if/else*

use a statement block { } to combine multiple statements into one

```
if (expression)
  statement
```

```
if (expression)
    statement1
else
    statement2
```

```
if (expression1) {
    // Execute code block #1
}
else if (expression2) {
    // Execute code block #2
}
else if (expression3) {
    // Execute code block #3
}
else {
    // If all else fails, execute block #4
}
```

Conditionals – *switch*

```
switch(expression){
    statements
}
```

```
switch(n) {
   case 1: // Start here if n === 1
   // Execute code block #1.
   break // Stop here
   case 2: // Start here if n === 2
   // Execute code block #2.
   break // Stop here
   case 3:
   // Start here if n === 3 // Execute code block #3.
   break // Stop here
   default:
   // If all else fails... // Execute code block #4.
   break // Stop here
}
```

The matching case is determined using the === identity operator, not the == equality operator, so the expressions must match without any type conversion.

Loop- *while/do-while*

```
while (expression)
   statement
```

```
let count = 0
while(count < 10) {
   console.log(count)
   count++
}
```

```
do
   statement
while (expression)
```

```
let count = 0;
do {
  console.log(count)
  count++
} while (count < 10)
```

Loop- *for*

The **for** statement simplifies loops that follow a common pattern.

```
for(initialize ; test ; increment)
   statement
```

```
for(let i = 0, len = data.length; i < len; i++)
   console.log(data[i])
```

The **for/of** loop works with *iterable* objects including arrays, strings, sets, and maps are iterable:

```
for(variable of iterableObject)
   statement
```

```
let data = [1, 2, 3, 4, 5, 6, 7, 8, 9]
let sum = 0
for(let element of data) {
    sum += element
})
console.log(`sum = ${sum}`) //sum=45
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of

The **for/in** statement loops through the property names of a specified object

```
for (variable in object)
     statement
```

```
for(let property in object) {
   console.log(property) //print property name
   console.log (object[property]) //print value of each
property
}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in

forEach

- **`forEach()`** method iterates through an array, invoking a function you specify for each element.

```
let data = [1, 2, 3, 4, 5, 6, 7, 8, 9]
let sum = 0
data.forEach((num) => (sum += num))
console.log(`sum=${sum}`) //sum=45
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

- **`reduce()`** combine the elements of an array, using the function you specify, to produce a single value.

```
const sum2 = data.reduce((total, currentValue) => total + currentValue)
console.log(`sum2=${sum2}`) //sum=45
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce