Notes on Operators

Increment/Decrement Operator in JS:

In JavaScript (JS), there are two types of increment/decrement operators: post-increment/decrement and pre-increment/decrement.

Post-increment/decrement operator

The post-increment/decrement operator (++) and (--) first use the current value of a variable and then increment/decrement it. **For example:**

```
let x = 1;
console.log(x++); // Output: 1
console.log(x); // Output: 2
```

In the above example, x++ returns the value of x (which is 1) and then increments x by 1 (so x becomes 2).

Pre-increment/decrement operator

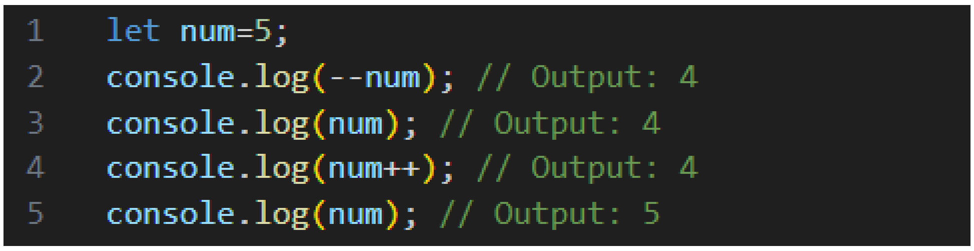
The pre-increment/decrement operator (++x) and (--x) first increment/decrement the value of a variable and then use it**. For example:**

```
let y = 1;
console.log(++y); // Output: 2
console.log(y); // Output: 2
```

In the above example, ++y increments y by 1 (so y becomes 2) and then returns the new value of y (which is also 2).

Both types of operators can be used with any variable that holds a numeric value, including variables that are defined using the var, let, and const keywords.

The example given below demonstrates the behavior of pre- and post-decrement operators and how they can affect the value of a variable in JavaScript.



```
1   let num=5;
2   console.log(--num); // Output: 4
3   console.log(num); // Output: 4
4   console.log(num++); // Output: 4
5   console.log(num); // Output: 5
```

Explanation:

Line 1: The initial value of the variable num is set to 5.

Line 2: The console.log(--num); statement uses the pre-decrement operator to decrement num by 1 before printing its value to the console. The output is 4.

Line 3: The console.log(num); statement prints the value of num, which is 4 because it was previously decremented.

Line 4: The console.log(num++); statement uses the post-decrement operator to print the value of num (which is still 4), and then decrement num by 1. The output is 4.

Line 5: The console.log(num); statement prints the value of num, which is now 5 because it was incremented after the third console.log statement.

String Comparison and Unicode in JavaScript

In JavaScript (JS), strings are compared using Unicode values, which is a standard for assigning unique numeric values to different characters from various writing systems around the world.

Unicode Values

- Unicode represents characters as numeric values, which is important for string manipulation and comparison operations.

- Each character in the Unicode character set is assigned a unique number, known as a code point. Code points are represented in JavaScript as Unicode escape sequences, which are denoted by a backslash followed by "u" and a sequence of four hexadecimal digits.

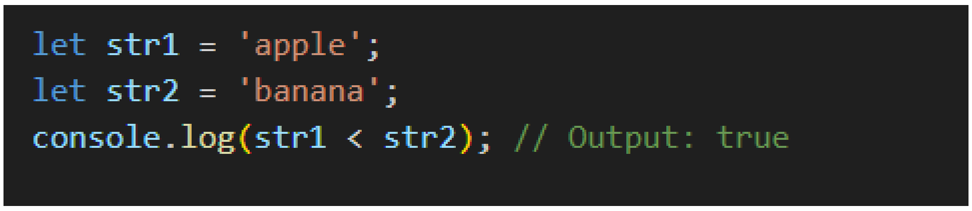
- **For example,** the Unicode escape sequence "\u0041" represents the character "A", while the escape sequence "\u20AC" represents the euro sign "€".

- Unicode values play a crucial role in string comparison, as JavaScript compares strings character-by-character using their Unicode values. Unicode also provides a standardized way to represent characters from different languages and scripts, making it easier to create multilingual applications.

String Comparison

When comparing two strings, JS compares each character's Unicode value at the same index in each string. If the Unicode values are the same, it moves to the next character's Unicode value until it finds a difference or reaches the end of both strings.

For example, consider the following code snippet:



```
let str1 = 'apple';
let str2 = 'banana';
console.log(str1 < str2); // Output: true
```

In the above example, the str1 is compared to str2 using Unicode values. The Unicode value of the first character in str1 (which is 'a') is less than the Unicode value of the first character in str2 (which is 'b'). Therefore, str1 is less than str2, and the output is true.

- It's important to note that when comparing strings with different character lengths, JS compares each character up to the length of the shorter string. If all the characters are the same up to the length of the shorter string, then the longer string is considered greater.

- Unicode values also play a role in string-related operations, such as string concatenation and manipulation. Therefore, understanding Unicode is crucial in developing robust and reliable JS programs that work with strings.