**Difference Between var, let and const**

|  |
| --- |
|  |
|  | Scope | Redeclare | Reassign | Hoisted | Binds this |
| var | No | Yes | Yes | Yes | Yes |
| let | Yes | No | Yes | No | No |
| const | Yes | No | No | No | No |

## What is Good?

let and const have **block scope**.

let and const can not be **redeclared**.

let and const must be **declared** before use.

let and const does **not bind** to this.

let and const are **not hoisted**.

## What is Not Good?

var does not have to be declared.

var is hoisted.

var binds to this.

## Hoisting:

## You can use the variable before it is declared

* Variables defined with var are **hoisted** to the top and can be initialized at any time.

Meaning: You can use the variable before it is declared:

Ex: carName = "Volvo";  
 var carName;

* Variables defined with let are also hoisted to the top of the block, but not initialized.

Meaning: Using a let variable before it is declared will result in a ReferenceError

Ex: carName = "Saab";  
 let carName = "Volvo";

* Variables defined with const are also hoisted to the top, but not initialized.

Meaning: Using a const variable before it is declared will result in a ReferenceError

Ex: alert (carName);  
 const carName = "Volvo";

**Constant Objects and Arrays**

The keyword const is a little misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

* Reassign a constant value
* Reassign a constant array
* Reassign a constant object

But you CAN:

* Change the elements of constant array
* Change the properties of constant object

# JavaScript Operators

Javascript operators are used to perform different types of mathematical and logical computations.

### Examples:

The **Assignment Operator** **=** assigns values

The **Addition Operator** **+** adds values

The **Multiplication Operator** **\*** multiplies values

The **Comparison Operator** **>** compares values

**Types of JavaScript Operators**

There are different types of JavaScript operators:

* Arithmetic Operators

**Operator Description**

+ Addition

- Subtraction

\* Multiplication

\*\* Exponentiation

/ Division

% Modulus

++ Increment

-- Decrement

* Assignment Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| \*\*= | x \*\*= y | x = x \*\* y |

* Comparison Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | equal to |
| === | equal value and equal type |
| != | not equal |
| !== | not equal value or not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? | ternary operator |

* String Operators

All the comparison operators above can also be used on strings:

Ex: let text1 = "A";  
 let text2 = "B";  
 let result = text1 < text2;

Note that strings are compared alphabetically:

Ex: let text1 = "20";  
 let text2 = "5";  
 let result = text1 < text2;

## Note

## When used on strings, the + operator is called the concatenation operator

If you add a number and a string, the result will be a string!

* Logical Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | logical and |
| || | logical or |
| ! | logical not |

* Type Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| typeof | Returns the type of a variable |
| instanceof | Returns true if an object is an instance of an object type |

* Ternary Operators
* Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Same as** | **Result** | **Decimal** |
| & | AND | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | OR | 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | NOT | ~ 5 | ~0101 | 1010 | 10 |
| ^ | XOR | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | left shift | 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | right shift | 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| >>> | unsigned right shift | 5 >>> 1 | 0101 >>> 1 | 0010 | 2 |