**MILESTONE 3**

Operators

● Unary Operators

● Bitwise Operators

● Boolean Operators

● Multiplicative Operators

● Additive Operators

● Relational Operators

● Equality Operators

● Conditional Operator

● Assignment Operators

● Comma Operator

1. Unary operator

* It is used to perform numerical operations on variables.
* +operand 🡪 If the operand is a string, it will be converted into number when we use this operator.
* EX: let str1 = "12"

let **num = +str1;**

console.log(num); // 12

console.log(typeof (num)) //number

* -operand 🡪is used to convert its operand to a negative number if it is a positive number already.
* EX: let str1 = "12";

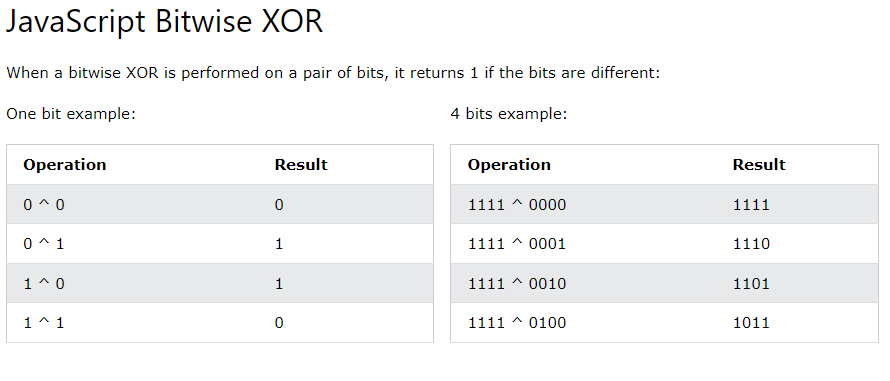
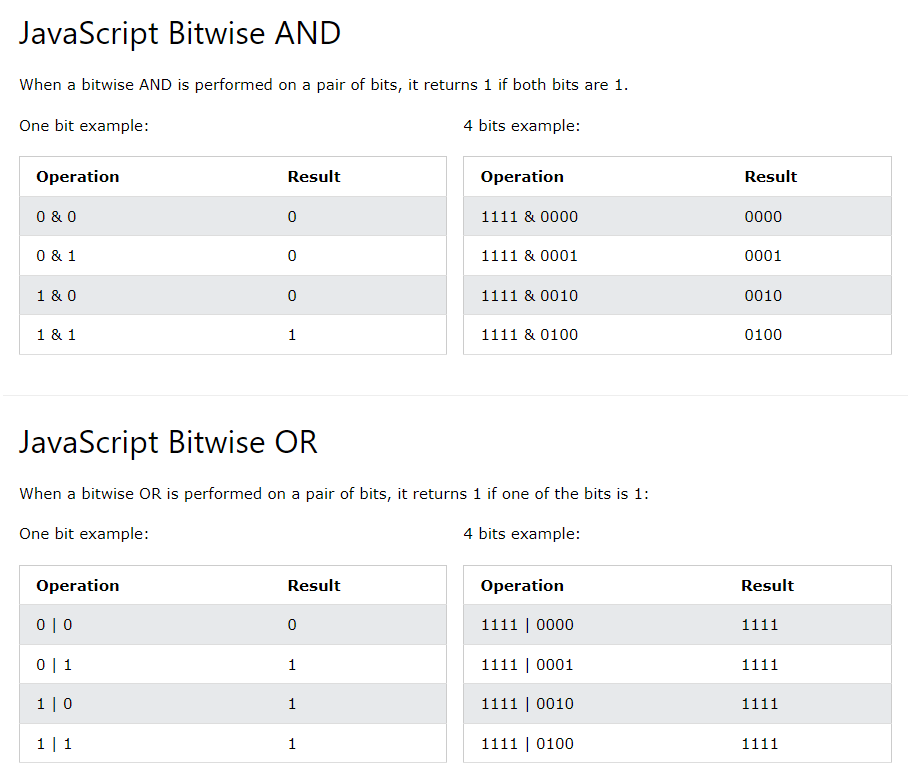
let num = -str1;

console.log(num); //-12

* =x/ -x doesn’t have any effect if given in-front of a number.
* ++x 🡪this operator used to increase the value and assign it to the variable.
* --x 🡪this operator is used to decrease the value and assign it to the variable
* x++ 🡪this operator assigns value to variable and then increases the value by 1.
* x-- 🡪this operator assigns value to the variable and then decreases the value by 1.
* Logical NOT (!). it negates the boolean value of an operand, converting false to true and true to false.
* Bitwise NOT (~). It inverts all bits. ie., 0 to 1, 1 to 0
* Void operator 🡪 evaluates an expression and returns undefined. It ensures that the expression has no return value.
* Delete operator 🡪 removes a property of an object.

1. Bitwise Operator

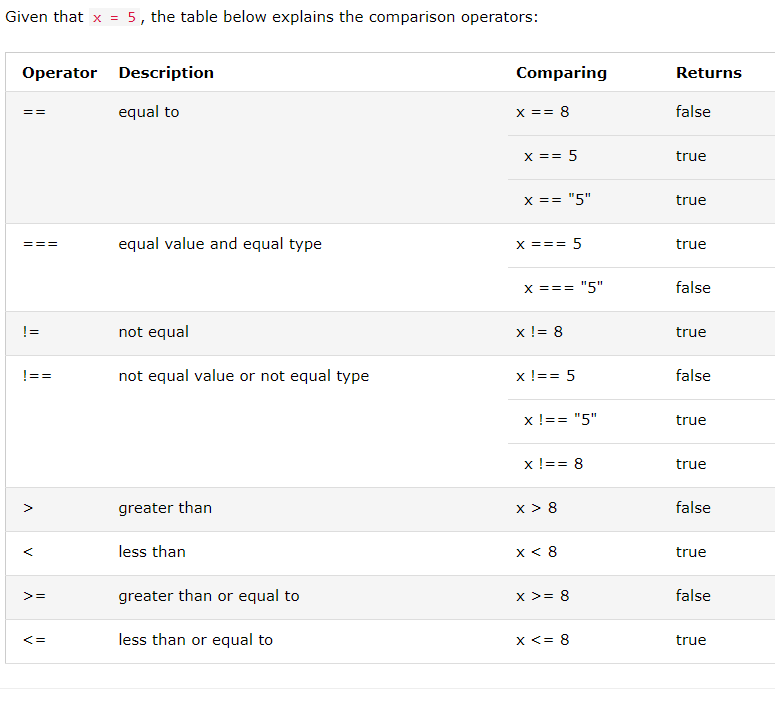
* & - AND 0🡪 sets each bits to 1, if both the bits are 1.
* | - OR 🡪 sets each bit to 1, if one of the 2 bits is 1.
* ^ - XOR 🡪sets each bit to 1 if only one of two bits is 1.
* ~ - NOT 🡪Inverts all the bits.
* << 🡪Zero fill left shift, It shifts to left by pushing zeros in from the right and let the leftmost bits fall off.
* >> 🡪Signed right shift, shifts to right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off.
* >>> 🡪 Zero fill right shift, shifts right by pushing zeros in from the left, and let the rightmost bits fall off.



1. Boolean operator

* Logical AND - && 🡪 returns true if both operands are true. Or it returns false.
* Logical OR - || 🡪 returns true if atleast one of the operands is true. Returns false only if both are false.
* NOT - ! 🡪returns true if false, returns false if it is true.

1. Comparison Operator:



1. Multiplicative operators:

* Asterisk (\*) is used to represent multiplication.
* It is an arithmetic multiplication operator used to find the product of operands.
* If 2\* “12” = 24 is answer.
* 2n \* 2n; // 4n 🡪 bigint multiplication operation
* 2n \* 2 🡪 throws an error.

1. Additive operators:

* + 🡪 Addition, used to add 2/ more operands
* - 🡪 Subtraction, used to subtract 2/ more operands.
* \* 🡪 Multiplication, used to multiply 2/ more operands.
* \*\* 🡪 Exponential/ power, used to raise the first operand to the power of 2nd operand.
* Math.pow(x,y) === x\*\*y
* / 🡪Division, divides two operands.
* % 🡪 Modulus, used to get remainder of the values.
* ++ 🡪 Increment
* -- 🡪 Decrement.
* Precedence of the operators
  + \*, /, +,-
  + When using parenthesis, operations inside the parenthesis are computed first.
  + When many operations have same precedence, they are computed from left to right.

1. Relational Operators:

* In operator, instanceOf operator, <, <=, >, >=, ==, ===, !=, !==
* In-operator checks if a specified property exists in an object if an element exists in array. It returns a Boolean value.
* EX: let languages = ["HTML", "CSS", "JavaScript"];

console.log(1 in languages); // true – as index 1 is present in languages.

console.log(3 in languages); // false – as index 3 is not present in languages.

* instanceOf operator tests if an object is an instance of a particular class or constructor. It also returns a Boolean value.
* EX: SYNTAX - let gfg = objectName instanceof objectType

1. Equality Operators:

* ==, ===, !=, !==
* == and != is loosely equal semantic
* === and !== is strictly equal semantic.
* EX: "3" === 3; // false
* EX: "3" == 3; // true

1. Conditional (Ternary) Operator:

* SYNTAX: variablename = (condition) ? value1:value2
* EX: let voteable = (age < 18) ? "Too young":"Old enough";
* When we compare string with a number, JS will convert string to number and then compare.
* ?? operator returns first argument if it is not null/ undefined. Else it will return 2nd argument. 🡪 nullish coalescing operator
* EX: let name = null;

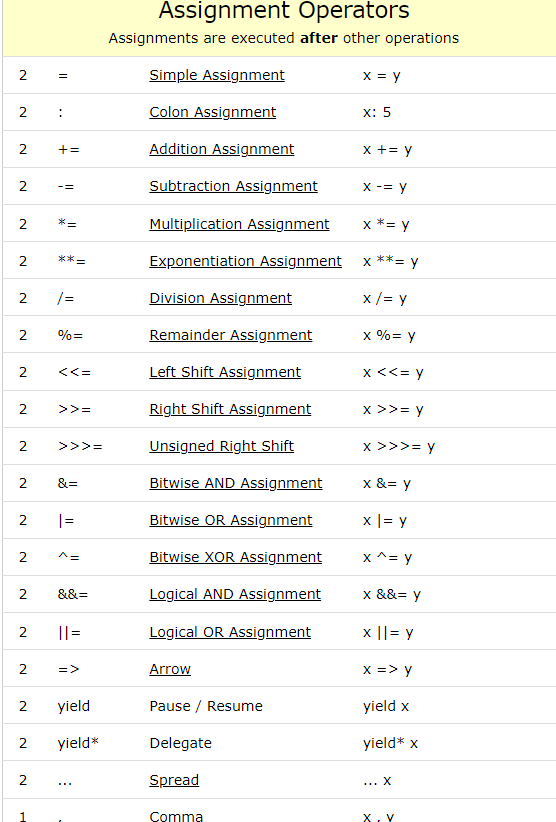
let text = "missing";

let result = name ?? text; // result=missing

* Optional chaining operator (?.) 🡪 returns undefined if an object is null/ undefined instead of throwing an error.

1. Assignment Operator:

* Assignments are executed after other operations.
* =, :, +=, -=, \*=, \*\*=, /=, %=, <<=, >>=, >>>=, &=, !=, ^=, &&=, ||=, =>



1. Comma Operator:

* You can use the comma operator when you want to include multiple expressions in a location that requires a single expression.
* It has the lowest precedence of all operators.
* Mostly used in for loops.
* The comma operator allows you to put multiple expressions in a place where one expression is expected.
* When the comma operator is used, each expression is evaluated from left to right, but only the result of the final expression is returned.
* Semicolons separate statements, each of which is evaluated independently. The comma operator allows multiple expressions to be evaluated within a single statement, with only the last expression’s result being returned
* EX: for (let a = 0, b =5; a <= 5; a++, b--) {

console.log(a, b);

}

1. Spread operator:

* The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.
* The spread operator is often used in combination with destructuring.
* const numbers = [1, 2, 3, 4, 5, 6];

const [one, two, ...rest] = numbers; //one-1, two-2, rest-3,4,5,6

* Spread syntax "expands" an array into its elements, while rest syntax collects multiple elements and "condenses" them into a single element.
* function sum(...theArgs) {

let total = 0;

for (const arg of theArgs) {

total += arg; }

return total;}

console.log(sum(1, 2, 3)); // Expected output: 6

console.log(sum(1, 2, 3, 4)); // Expected output: 10