JavaScript Math Review

Working with the Number Data Type

- **Definition**: JavaScript's Number type includes integers, floating-point numbers, Infinity and NaN. Floating-point numbers are numbers with a decimal point. Positive Infinity is a number greater than any other number while -Infinity is a number smaller than any other number. NaN (Not a Number) represents an invalid numeric value like the string "Jessica".

Common Arithmetic Operations

- **Addition Operator**: This operator (+) is used to calculate the sum of two or more numbers.
- **Subtraction Operator**: This operator (-) is used to calculate the difference between two numbers.
- **Multiplication Operator**: This operator (*) is used to calculate the product of two or more numbers.
- **Division Operator**: This operator (/) is used to calculate the quotient between two numbers
- **Division By Zero**: If you try to divide by zero, JavaScript will return Infinity.
- **Remainder Operator**: This operator(%) returns the remainder of a division.
- **Exponentiation Operator**: This operator (**) raises one number to the power of another.

Calculations with Numbers and Strings

- **Explanation**: When you use the + operator with a number and a string, JavaScript will coerce the number into a string and concatenate the two values. When you use the -, * or / operators with a string and number, JavaScript will coerce the string into a number and the result will be a number. For null and undefined, JavaScript treats null as 0 and undefined as NaN in mathematical operations.

Example Code

```
const result = 5 + '10';
```

```
console.log(result); // 510
console.log(typeof result); // string

const subtractionResult = '10' - 5;
console.log(subtractionResult); // 5
console.log(typeof subtractionResult); // number

const multiplicationResult = '10' * 2;
console.log(multiplicationResult); // 20
console.log(typeof multiplicationResult); // number

const divisionResult = '20' / 2;
console.log(divisionResult); // 10
console.log(typeof divisionResult); // number

const result1 = null + 5;
console.log(result1); // 5
console.log(typeof result1); // number

const result2 = undefined + 5;
console.log(result2); // NaN
console.log(typeof result2); // number
```

Operator Precedence

- **Definition**: Operator precedence determines the order in which operations are evaluated in an expression. Operators with higher precedence are evaluated before those with lower precedence. Values inside the parenthesis will be evaluated first and multiplication/division will have higher precedence than addition/subtraction. If the operators have the same precedence, then JavaScript will use associativity.

Example Code

```
const result = (2 + 3) * 4;


console.log(result); // 20


const result2 = 10 - 2 + 3;


console.log(result2); // 11


const result3 = 2 ** 3 ** 2;


console.log(result3); // 512
```

- **Definition**: Associativity informs us the direction in which an expression is evaluated when multiple operators of the same type exist. It defines whether the expression should be evaluated from left-to-right (left-associative) or right-to-left (right-associative). For example, the exponent operator is also right to left associative:

Example Code

```
const result4 = 5 ** 4 ** 1; // 625


console.log(result4);
```

Increment and Decrement Operators

- **Increment Operator**: This operator is used to increase the value by one. The prefix notation ++num increases the value of the variable first, then returns a new value. The postfix notation num++ returns the current value of the variable first, then increases it.

Example Code

```
let x = 5;

console.log(++x); // 6
console.log(x); // 6



let y = 5;

console.log(y++); // 5
console.log(y); // 6
```

- **Decrement Operator**: This operator is used to decrease the value by one. The prefix notation and postfix notation work the same way as earlier with the increment operator.

Example Code

```
let num = 5;

console.log(--num); // 4
console.log(num--); // 4
console.log(num); // 3
```

Compound Assignment Operators

- **Addition Assignment (+=) Operator**: This operator performs addition on the values and assigns the result to the variable.
- **Subtraction Assignment (-=) Operator**: This operator performs subtraction on the values and assigns the result to the variable.
- **Multiplication Assignment (*=) Operator**: This operator performs multiplication on the values and assigns the result to the variable.

- **Division Assignment (/=) Operator**: This operator performs division on the values and assigns the result to the variable.
- **Remainder Assignment (%=) Operator**: This operator divides a variable by the specified number and assigns the remainder to the variable.
- **Exponentiation Assignment (**=) Operator**: This operator raises a variable to the power of the specified number and reassigns the result to the variable.

Booleans and Equality

- **Boolean Definition**: A boolean is a data type that can only have two values: true or false.
- **Equality (==) Operator**: This operator uses type coercion before checking if the values are equal.

Example Code

```
console.log(5 == '5'); // true
```

- **Strict Equality (===) Operator**: This operator does not perform type coercion and checks if both the types and values are equal.

Example Code

```
console.log(5 === '5'); // false
```

- **Inequality (!=) Operator**: This operator uses type coercion before checking if the values are not equal.
- **Strict Inequality (!==) Operator**: This operator does not perform type coercion and checks if both the types and values are not equal.

Comparison Operators

- **Greater Than (>) Operator**: This operator checks if the value on the left is greater than the one on the right.
- **Greater Than (>=) or Equal Operator**: This operator checks if the value on the left is greater than or equal to the one on the right.
- **Less Than (<) Operator**: This operator checks if the value on the left is less than the one on the right.
- **Less Than (<=) or Equal Operator**: This operator checks if the value on the left is less than or equal to the one on the right.

Unary Operators

- **Unary Plus Operator**: This operator converts its operand into a number. If the operand is already a number, it remains unchanged.

Example Code

```
const str = '42';

const num = +str;


console.log(num); // 42

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

- **Unary Negation (-) Operator**: This operator negates the operand.

Example Code

```
const num = 4;

console.log(-num); // -4
```

- **Logical NOT (!) Operator**: This operator flips the boolean value of its operand. So, if the operand is true, it becomes false, and if it's false, it becomes true.

Bitwise Operators

- **Bitwise AND (&) Operator**: This operator returns a 1 in each bit position for which the corresponding bits of both operands are 1.
- **Bitwise AND Assignment (&=) Operator**: This operator performs a bitwise AND operation with the specified number and reassigns the result to the variable.
- **Bitwise OR (|) Operator**: This operator returns a 1 in each bit position for which the corresponding bits of either or both operands are 1.
- **Bitwise OR Assignment (|=) Operator**: This operator performs a bitwise OR operation with the specified number and reassigns the result to the variable.
- **Bitwise XOR (^) Operator**: This operator returns a 1 in each bit position for which the corresponding bits of either, but not both, operands are 1.
- **Bitwise NOT (~) Operator**: This operator inverts the binary representation of a number.

- **Left Shift (<<) Operator**: This operator shifts all bits to the left by a specified number of positions.
- **Right Shift (>>) Operator**: This operator shifts all bits to the right.

Conditional Statements, Truthy Values, Falsy Values and the Ternary Operator

- **if/else if/else**: An if statement takes a condition and runs a block of code if that condition is truthy. If the condition is false, then it moves to the else if block. If none of those conditions are true, then it will execute the else clause. Truthy values are any values that result in true when evaluated in a Boolean context like an if statement. Falsy values are values that evaluate to false in a Boolean context.

Example Code

```
const score = 87;


if (score >= 90) {

 console.log('You got an A');

} else if (score >= 80) {

 console.log('You got a B'); // You got an B

} else if (score >= 70) {

 console.log('You got a C');

} else {

 console.log('You failed! You need to study more!');

}
```

- **Ternary Operator**: This operator is often used as a shorter way to write if else statements.

Example Code

```
const temperature = 30;

const weather = temperature > 25 ? 'sunny' : 'cool';


console.log(`It's a ${weather} day!`); // It's a sunny day!
```

Binary Logical Operators

- **Logical AND (&&) Operator**: This operator checks if both operands are true. If both are true, then it will return the second value. If either operand is falsy, then it will return the falsy value. If both operands are falsy, it will return the first falsy value.

Example Code

```
const result = true && 'hello';


console.log(result); // hello
```

- **Logical OR (||) Operator**: This operator checks if at least one of the operands is truthy.
- **Nullish Coalescing (??) Operator**: This operator will return a value only if the first one is null or undefined.

Example Code

```
const userSettings = {
 theme: null,
 volume: 0,
 notifications: false,
};


let theme = userSettings.theme ?? 'light';
console.log(theme); // light
```

The Math Object

- **The Math.random() Method**: This method generates a random floating-point number between 0 (inclusive) and 1 (exclusive). This means the possible output can be 0, but it will never actually reach 1.
- **The Math.max() Method**: This method takes a set of numbers and returns the maximum value.

- **The Math.min() Method**: This method takes a set of numbers and returns the minimum value.
- **The Math.ceil() Method**: This method rounds a value up to the nearest whole integer.
- **The Math.floor() Method**: This method rounds a value down to the nearest whole integer.
- **The Math.round() Method**: This method rounds a value to the nearest whole integer.

Example Code

```
console.log(Math.round(2.3)); // 2

console.log(Math.round(4.5)); // 5

console.log(Math.round(4.8)); // 5
```

- **The Math.trunc() Method**: This method removes the decimal part of a number, returning only the integer portion, without rounding.
- **The Math.sqrt() Method**: This method will return the square root of a number.
- **The Math.cbrt() Method**: This method will return the cube root of a number.
- **The Math.abs() Method**: This method will return the absolute value of a number.
- **The Math.pow() Method**: This method takes two numbers and raises the first to the power of the second.

Common Number Methods

- **isNaN()**: NaN stands for "Not-a-Number". It's a special value that represents an unrepresentable or undefined numerical result. The isNaN() function property is used to determine whether a value is NaN or not. Number.isNaN() provides a more reliable way to check for NaN values, especially in cases where type coercion might lead to unexpected results with the global isNaN() function.

Example Code

```
console.log(isNaN(NaN));       // true

console.log(isNaN(undefined)); // true

console.log(isNaN({}));        // true
console.log(isNaN(true));     // false

console.log(isNaN(null));     // false

console.log(isNaN(37));       // false



console.log(Number.isNaN(NaN));       // true

console.log(Number.isNaN(Number.NaN)); // true

console.log(Number.isNaN(0 / 0));     // true


console.log(Number.isNaN("NaN"));     // false

console.log(Number.isNaN(undefined));  // false
```

- **The parseFloat() Method**: This method parses a string argument and returns a floating-point number. It's designed to extract a number from the beginning of a string, even if the string contains non-numeric characters later on.
- **The parseInt() Method**: This method parses a string argument and returns an integer. parseInt() stops parsing at the first non-digit it encounters. For floating-point numbers, it returns only the integer part. If it can't find a valid integer at the start of the string, it returns NaN.
- **The toFixed() Method**: This method is called on a number and takes one optional argument, which is the number of digits to appear after the decimal point. It returns a string representation of the number with the specified number of decimal places.