Agenda :

1. JavaScript Operators
2. Strings
3. String Properties
4. String Operations
5. Inbuilt String functions
6. String Methods

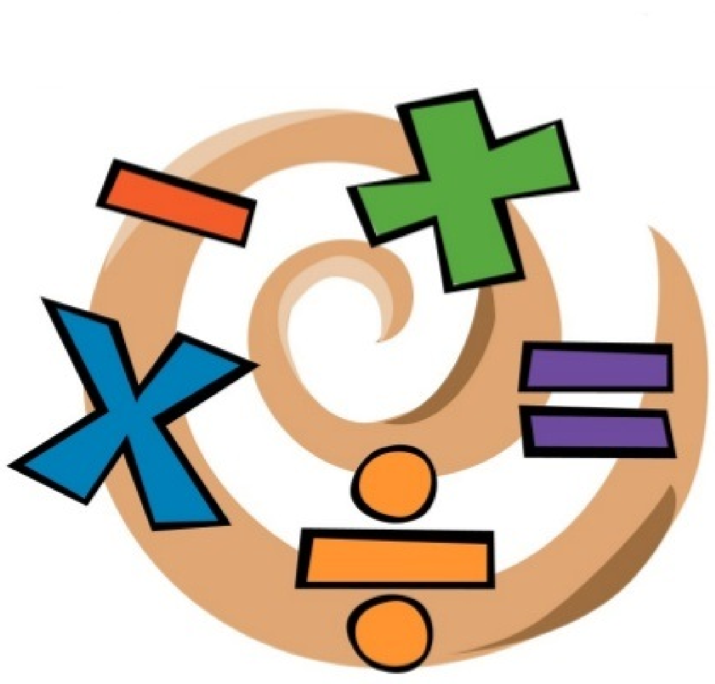
JavaScript Operators:

What is an Operator?

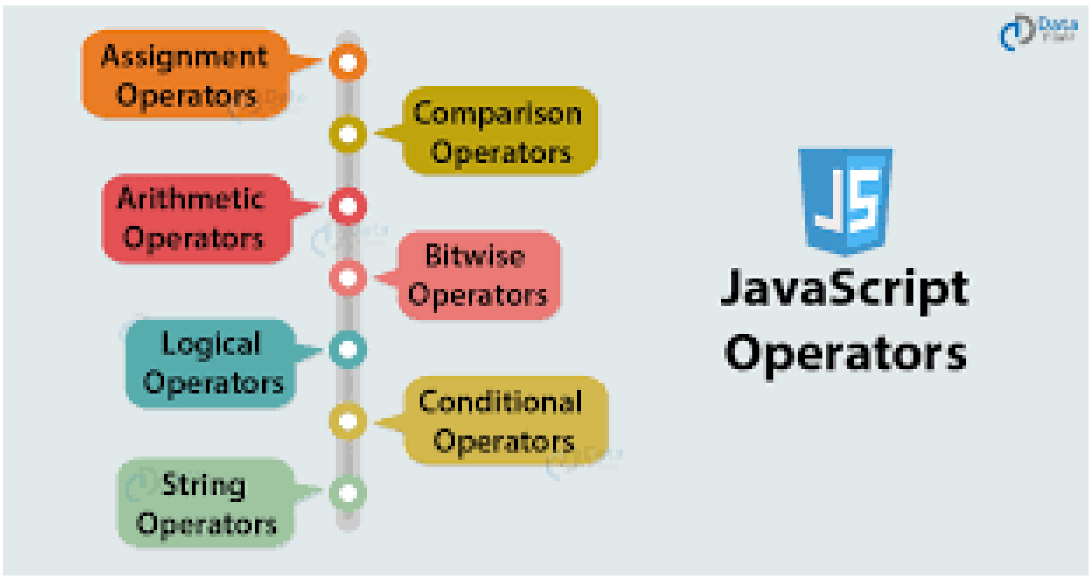
In JavaScript, an operator is a special symbol used to perform operations on operands (values and variables). For example,

```
2 + 3; // 5
```

Here `+` is an operator that performs addition, and `2` and `3` are operands.



- In JavaScript, Operators are used to perform a **number of operations.**
- Operators can be used to assign values, compare values & perform arithmetic operations
- The commonly used JavaScript Operators are classified into different categories :



JavaScript Operator Types

Here is a list of different operators you will learn in this tutorial.

- Assignment Operators
- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- String Operators

- Other Operators

JavaScript Assignment Operators

Assignment operators are used to **assign** values to variables. For example,

```
const x = 5;
```

Here, the `=` operator is used to assign value `5` to variable `x`.

Here's a list of commonly used assignment operators :

Operator	Name	Example
=	Assignment operator	a = 7; // 7
+=	Addition assignment	a += 5; // a = a + 5
-=	Subtraction Assignment	a -= 2; // a = a - 2
*=	Multiplication Assignment	a *= 3; // a = a * 3
/=	Division Assignment	a /= 2; // a = a / 2
%=	Remainder Assignment	a %= 2; // a = a % 2
**=	Exponentiation Assignment	a **= 2; // a = square of a

> Note : The commonly used assignment operator is =. You will understand other assignment operators such as +=, -=, *= etc. once we learn arithmetic operators.

JavaScript Arithmetic Operators

Arithmetic operators are used to perform **arithmetic calculations**. For example,

```
const number = 3 + 5; // 8
```

Here, the `+` operator is used to add two operands.

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Remainder	x % y
++	Increment (increments by 1)	++x or x++
--	Decrement (decrements by 1)	--x or x--
**	Exponentiation (Power)	x ** y

Example 1: Arithmetic operators in JavaScript

```
let x = 5;
let y = 3;

// addition
console.log('x + y = ', x + y);  // 8

// subtraction
```

```
console.log('x - y = ', x - y);  // 2

// multiplication
console.log('x * y = ', x * y);  // 15

// division
console.log('x / y = ', x / y);  // 1.6666666666666667

// remainder
console.log('x % y = ', x % y);   // 2

// increment
console.log('++x = ', ++x); // x is now 6
console.log('x++ = ', x++); // prints 6 and then increased to 7
console.log('x = ', x);     // 7

// decrement
console.log('--x = ', --x); // x is now 6
console.log('x-- = ', x--); // prints 6 and then decreased to 5
console.log('x = ', x);     // 5

//exponentiation
console.log('x ** y =', x ** y);
```

Increment ++ and Decrement -- Operator as Prefix and Postfix

In programming (Java, C, C++, JavaScript etc.), the increment operator `++` increases the value of a variable by 1. Similarly, the decrement operator `-` decreases the value of a variable by 1.

```
a = 5
++a;          // a becomes 6
a++;          // a becomes 7
--a;          // a becomes 6
a--;          // a becomes 5
```

Simple enough till now. However, there is an important difference when these two operators are used as a prefix and a postfix.

- If you use the `++` operator as a prefix like `++var` , the value of `var` is incremented by 1; then it returns the value.
- If you use the `++` operator as a postfix like `var++` , the original value of `var` is returned first; then var is incremented by 1.

> The `--` operator works in a similar way to the `++` operator except `--` decreases the value by 1.

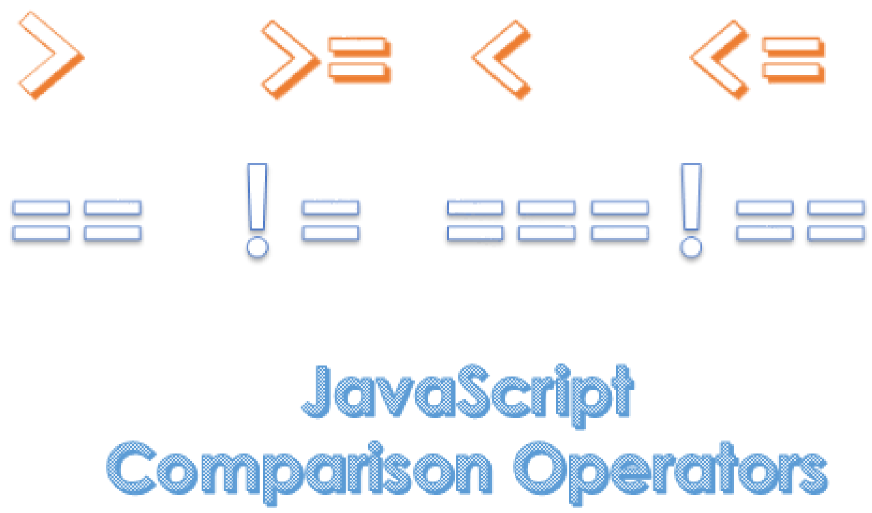
Example :

```
let var1 = 5, var2 = 5;

// 5 is displayed
// Then, var1 is increased to 6
console.log(var1++)

// var2 is increased to 6
// Then, var2 is displayed
console.log(++var2)
```

Output :

```
5
6
```

JavaScript Comparison Operators



Comparison operators **compare** two values and return a Boolean value, either `true` or `false` . For example,

```
const a = 3, b = 2;
console.log(a > b); // true
```

Here, the comparison operator >is used to compare whether a is greater than b.

Operator	Description	Example
==	**Equal to** : returns `true` if the operands are equal	x == y
!=	**Not equal to** : returns `true` if the operands are not equal	x != y
===	**Strict equal to** : `true` if the operands are equal and of the same type	x === y
!==	**Strict not equal to** : `true` if the operands are equal but of different type or not equal at all	x !== y
>	**Greater than** : `true` if left operand is greater than the right operand	x > y
>=	**Greater than or equal to** : `true` if left operand is greater than or equal to the right operand	x >= y
<	**Less than** : `true` if the left operand is less than the right operand	x < y
<=	**Less than or equal to** : `true` if the left operand is less than or equal to the right operand	x <= y

Example 2: Comparison operators in JavaScript

```
// equal operator
console.log(2 == 2); // true
console.log(2 == '2'); // true

// not equal operator
console.log(3 != 2); // true
console.log('hello' != 'Hello'); // true

// strict equal operator
console.log(2 === 2); // true
console.log(2 === '2'); // false

// strict not equal operator
console.log(2 !== '2'); // true
console.log(2 !== 2); // false
```

- Comparison operators are used in decision-making and loops.

JavaScript Logical Operators

Logical operators perform logical operations and return a Boolean value, either true or false. For example,

```
const x = 5, y = 3;
(x < 6) && (y < 5); // true
```

Here, &&is the logical operator AND. Since both x < 6and y < 5are true, the result is true.

Operators	Description	Example
&&	Logical AND: `true` if both the operands are `true` ,else returns `false`	x && y
||	Logical OR: `true` if either of the operands is `true` ; returns false if both are `false`	x || y
!	Logical NOT: `true` if the operand is `false` and vice-versa.	!x

Example 3: Logical Operators in JavaScript

```
// logical AND
console.log(true && true); // true
console.log(true && false); // false

// logical OR
console.log(true || false); // true

// logical NOT
console.log(!true); // false
```

Output:

```
true
false
true
false
```

- Logical operators are used in decision making and loops.

JavaScript Bitwise Operators

Bitwise operators perform operations on binary representations of numbers.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left shift
>>	Sign-propagating right shift
>>>	Zero-fill right shift

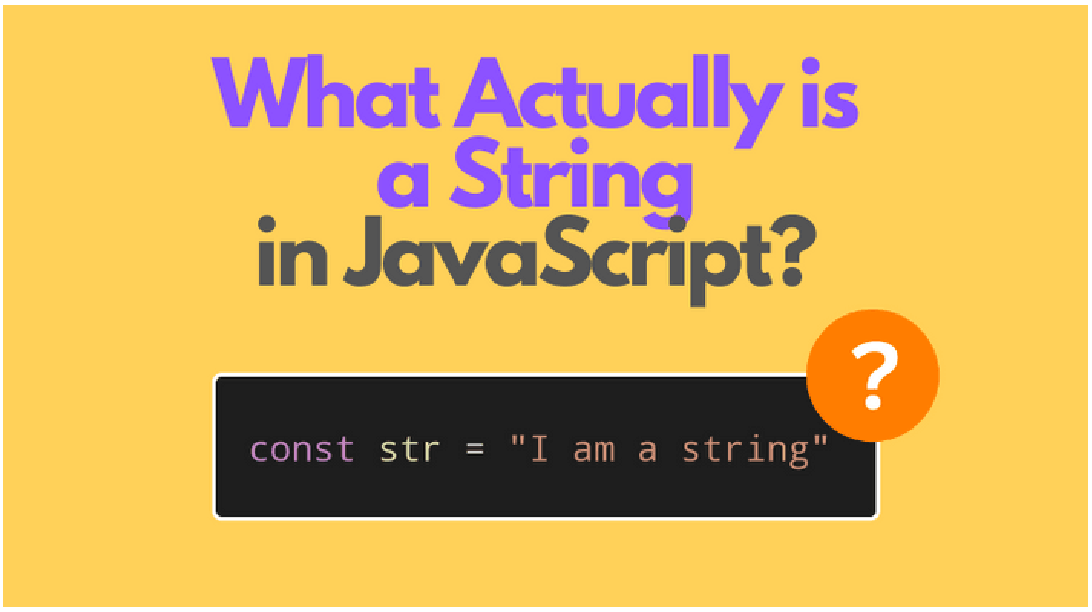
- Bitwise operators are rarely used in everyday programming.

Other JavaScript Operators

Here's a list of other operators available in JavaScript

Operator	Description	Example
,	evaluates multiple operands and returns the value of the last operand.	let a = (1, 3 , 4); // 4
?:	returns value based on the condition	(5 > 3) ? 'success' : 'error'; // "success"
delete	deletes an object's property, or an element of an array	delete x
typeof	returns a string indicating the data type	typeof 3; // "number"
void	discards the expression's return value	void(x)
in	returns `true` if the specified property is in the object	prop in object
instanceof	returns `true` if the specified object is of of the specified object type	object instanceof object_type

JavaScript String


What Actually is a String in JavaScript?
const str = "I am a string"
?

- JavaScript string contains a sequence of UTF-16 units
- JavaScript string is a primitive data type that is used to work with texts. For example,

```
const name = 'John';
```

** Create JavaScript Strings **

In JavaScript, strings are created by surrounding them with quotes. There are three ways you can use quotes.

- **Single quotes:** `'Hello'`
- **Double quotes:** `"Hello"`
- **Backticks:** `Hello`

For example,

```
//strings example
const name = 'Peter';
const name1 = "Jack";
const result = `The names are ${name} and ${name1}`;
```

Single quotes and double quotes are practically the same and you can use either of them.

Backticks are generally used when you need to include variables or expressions into a string. This is done by wrapping variables or expressior with `${variable or expression}` as shown above.

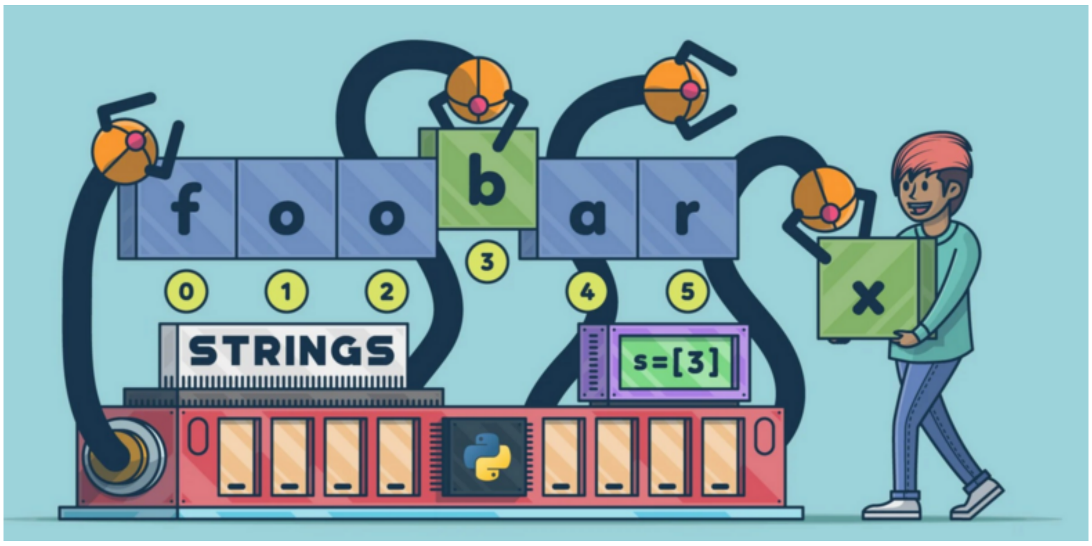
You can also write a quote inside another quote. For example,

```
const name = 'My name is "Peter".';
```

However, the quote should not match the surrounding quotes. For example,

```
const name = 'My name is 'Peter'.'; // error
```

Access String Characters



You can access the characters in a string in two ways.

One way is to treat strings as an array. For example,

```
const a = 'hello';
console.log(a[1]); // "e"
```

Another way is to use the method charAt(). For example,

```
const a = 'hello';
console.log(a.charAt(1)); // "e"
```

JavaScript Strings are immutable

In JavaScript, strings are immutable. That means the characters of a string cannot be changed. For example,

```
let a = 'hello';
a[0] = 'H';
console.log(a); // "hello"
```

However, you can assign the variable name to a new string. For example,

```
let a = 'hello';
a = 'Hello';
console.log(a); // "Hello"
```

JavaScript is Case-Sensitive

JavaScript is case-sensitive. That means in JavaScript, the lowercase and uppercase letters are treated as different values. For example,

```
const a = 'a';
const b = 'A'
console.log(a === b); // false
```

In JavaScript, `a` and `A` are treated as different values.

JavaScript Multiline Strings

To use a multiline string, you can either use the `+` operator or the `\` operator. For example,

```
// using the + operator
const message1 = 'This is a long message ' +
    'that spans across multiple lines' +
    'in the code.'

// using the \ operator
const message2 = 'This is a long message \
that spans across multiple lines \
in the code.'
```

JavaScript String Length

To find the length of a string, you can use built-in `length` property. For example,

```
const a = 'hello';
console.log(a.length); // 5
```

JavaScript String Objects

Below is the list of the properties of String object and their description :

Property	Description
Constructor	Returns a reference to the String function that created the object
Length	Returns the length of the string
Prototype	Allows you to add properties and methods to an object

You can also create strings using the new keyword. For example,

```
const a = 'hello';
const b = new String('hello');

console.log(a); // "hello"
console.log(b); // "hello"

console.log(typeof a); // "string"
console.log(typeof b); // "object"
```

> Note : It is recommended to avoid using string objects. Using string objects slows down the program.

JavaScript String Methods

Here are the commonly used JavaScript String methods:

Method	Description
charAt(index)	returns the character at the specified index
concat()	joins two or more strings
replace()	replaces a string with another string
split()	converts the string to an array of strings
substr(start, length)	returns a part of a string
substring(start,end)	returns a part of a string
slice(start, end)	returns a part of a string
toLowerCase()	returns the passed string in lower case
toUpperCase()	returns the passed string in upper case
trim()	removes whitespace from the strings
includes()	searches for a string and returns a Boolean value
search()	searches for a string and returns a position of a match

Example: JavaScript String Methods

```
const text1 = 'hello';
const text2 = 'world';
const text3 = '     JavaScript    ';

// concatenating two strings
const result1 = text1.concat(' ', text2);
console.log(result1); // "hello world"

// converting the text to uppercase
const result2 = text1.toUpperCase();
console.log(result2); // HELLO

// removing whitespace from the string
const result3 = text3.trim();
console.log(result3); // JavaScript

// converting the string to an array
const result4 = text1.split();
console.log(result4); // ["hello"]

// slicing the string
const result5= text1.slice(1, 3);
console.log(result5); // "el"
```

JavaScript String() Function

The `String()` function is used to convert various data types to strings. For example,

```
const a = 225; // number
const b = true; // boolean

//converting to string
const result1 = String(a);
const result2 = String(b);
```

```
console.log(result1); // "225"
console.log(result2); // "true"
```

Escape Character

You can use the backslash escape character \ to include special characters in a string. For example,

```
const name = 'My name is \'Peter\'.';
console.log(name);
```

Output:

```
My name is 'Peter'.
```

In the above program, the same quote is included using `\`.

Here are other ways that you can use `\`:

Code	Output
\"	include double quote
\\\\	include backslash
\n	new line
\r	carriage return
\v	vertical tab
\t	horizontal tab
\b	backspace
\f	form feed

JavaScript String Operators

In JavaScript, you can also use the `+` operator to concatenate (join) two or more strings.

Example 4: String operators in JavaScript

```
// concatenation operator
console.log('hello' + 'world');

let a = 'JavaScript';

a += ' tutorial';  // a = a + ' tutorial';
console.log(a);
```

Output:

```
helloworld
JavaScript tutorial
```

> Note : When + is used with strings, it performs concatenation. However, when + is used with numbers, it performs addition.

Interview Questions

> Given a string, reverse each word in the sentence

```
var string = "Welcome to this Javascript Guide!";

// Output becomes !ediuG tpircsavaJ siht ot emocleW
var reverseEntireSentence = reverseBySeparator(string, "");

// Output becomes emocleW ot siht tpircsavaJ !ediuG
var reverseEachWord = reverseBySeparator(reverseEntireSentence, " ");

function reverseBySeparator(string, separator) {
  return string.split(separator).reverse().join(separator);
}
```

> What's the spread operator?

The spread operator is also indicated by the `...` operator. It'll spread an object's property into another object and spread the array entries into anoth
array.

For example, if we have:

```
const foo = [1, 2, 3];
const bar = [...foo];
console.log(bar);
```

Then we get `[1, 2, 3]` as the value of `bar` since we made a copy of `foo` and assigned it to `bar` with the spread operator.

It's also useful for merging arrays. For instance, if we have:

```
const foo = [1, 2, 3];
const bar = [3, 4, 5];
const baz = [...foo, ...bar];
console.log(baz);
```

Then `baz` would be `[1, 2, 3, 3, 4, 5]` since we combined the entries of the `foo` and `bar` arrays into the `baz` array.