WELCOME TO
JS MASTERY COURSE : THE ULTIMATE GUIDE !

HTML

CSS

JAVASCRIPT

LEVEL 1

What is Javascript ?

JavaScript is a high-level, versatile, and interpreted programming language primarily used for creating interactive and dynamic content on websites. It is often employed to enhance user experience by enabling client-side scripts that run in web browsers. Developed by Netscape, JavaScript has become one of the core technologies for building web applications.

JavaScript is an essential component of web development alongside HTML and CSS. It allows developers to manipulate the Document Object Model (DOM) of a web page, enabling dynamic updates and changes to the content displayed to users. JavaScript can be used for various purposes, including form validation, interactive user interfaces, asynchronous communication with servers (Ajax), and more.

It is important to note that JavaScript is distinct from Java, despite the similarity in their names. JavaScript is mainly used on the client side, while Java is typically used for server-side development. Additionally, JavaScript has evolved over the years, and with the introduction of technologies like Node.js, it can now be used on the server side as well.

WHAT IS A PROGRAMMING LANGUAGE ?

A programming language is a formal system designed to instruct a computer or machine. It consists of a set of rules and syntax that allows programmers to write instructions for the computer to execute specific tasks. These instructions, known as code, can range from simple operations to complex algorithms.

Programming languages serve as a bridge between human understanding and machine execution. They provide a way for programmers to communicate their intentions to computers in a structured and understandable manner. Different programming languages have varying levels of abstraction, capabilities, and purposes, making them suitable for different types of tasks and applications.

Some common programming languages include Python, Java, C++, JavaScript, and many others. Each language has its own strengths and weaknesses, and the choice of a programming language often depends on the requirements of a particular project.

WHAT IS A SYNTAX ?

In programming, syntax refers to the set of rules that dictate how programs written in a specific programming language are structured. It defines the combinations of symbols, keywords, and other elements that are considered valid in that language. Proper syntax is crucial for writing code that the computer can understand and execute.

INTERNAL AND EXTERNAL JS

Certainly! When we talk about "internal JS" and "external JS," we are referring to the placement of JavaScript code within an HTML document.

Internal JavaScript: Internal JavaScript refers to including JavaScript code directly within the HTML document. This is typically done within the <script> tag, either in the <head> section or at the end of the <body> section. Here's an example:

```
<script>
// Your JavaScript code goes here
function myFunction() {
alert("Hello, World!");
}
</script>
```

External JavaScript: External JavaScript involves placing the JavaScript code in a separate file with a .js extension, and then linking to that file from the HTML document. Here's an example:

```
<script  src="external.js"></script>
```

```
external.js:
// Your JavaScript code goes here
function myFunction() {
  alert("Hello,  World!");
}
```

OUR FIRST JS CODE

Create a script.js file and type this example below and run this code on terminal.

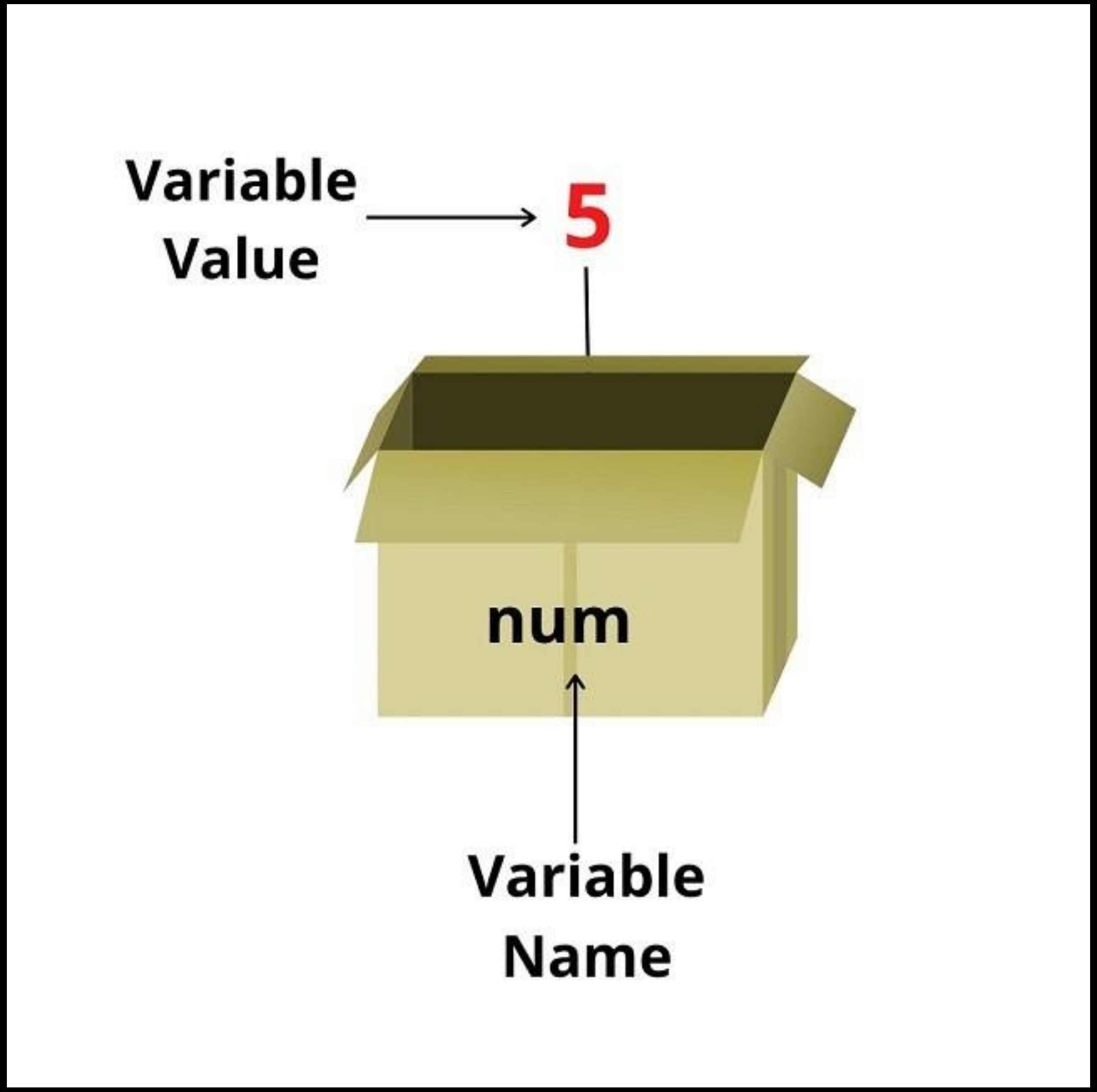
```
Console.log(“hello   world”)
```

HURRAY ! YOU RUN YOUR FIRST PROGRAM

LEVEL 2

VARIABLES IN JAVASCRIPT

In JavaScript, variables are used to store and manage data in a program. Here are some key points about variables in JavaScript:



```
let num = 5;
```

RULES OF VARIABLES

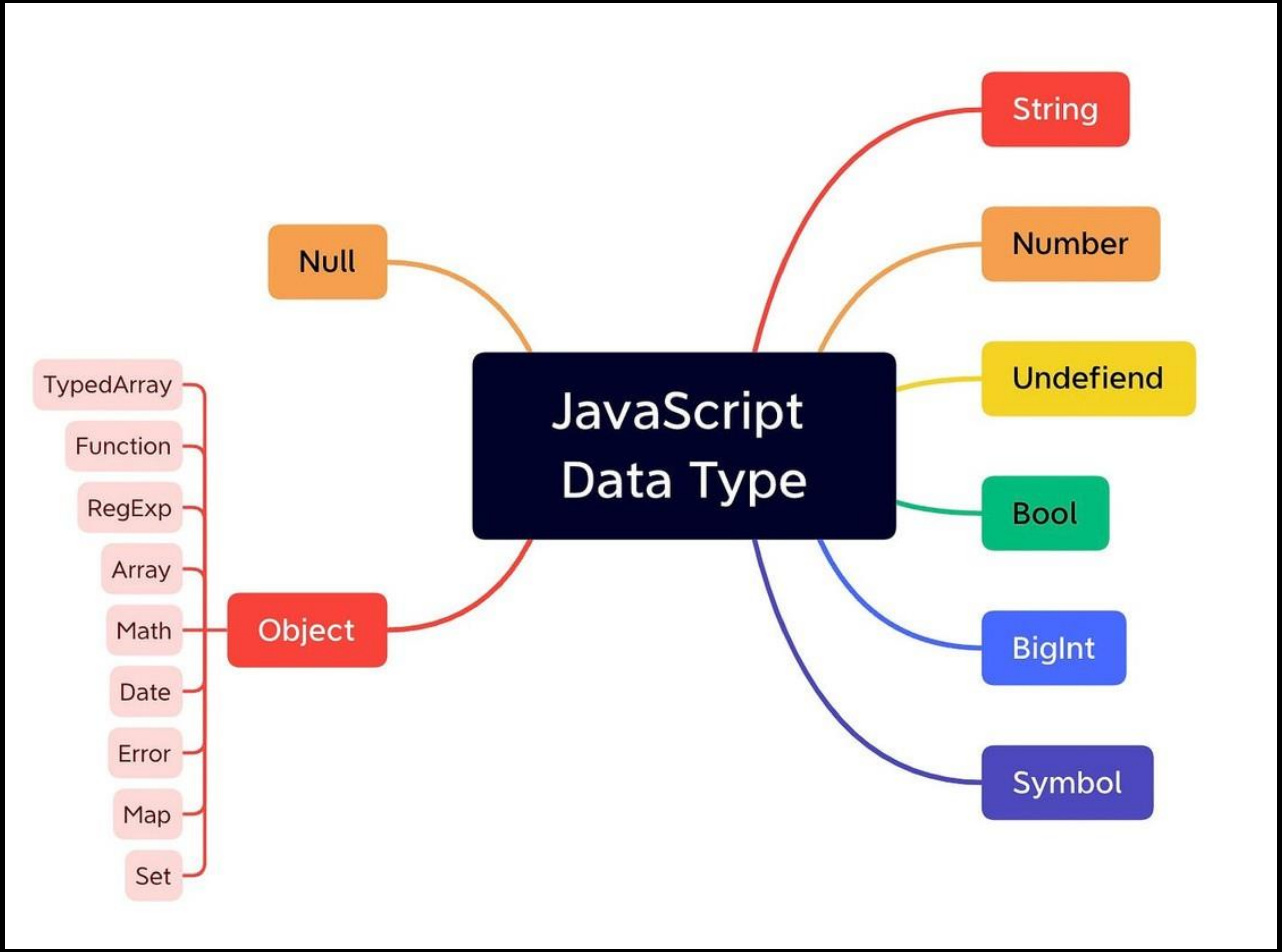
- Variable names must start with a letter, an underscore (_) or a dollar sign ($).
- Subsequent characters can also be digits (0-9).
- Variable names cannot contain spaces.
- Variables cannot be the same as reserved keywords such as if or const .
- By convention, JavaScript variable names are written in camelCase.

VAR, LET & CONST

	global scoped	function scoped	block scoped	reassignable	redeclarable	can be hoisted
var	+	+	-	+	+	+
let	-	+	+	+	-	-
const	-	+	+	-	-	-

DATA TYPES IN JS

In JavaScript, there are several data types that can be used to store and manipulate different kinds of values. Here are the main data types in JavaScript:



Primitive Data Types:

- String: Represents textual data, enclosed in single or double quotes.
- Number: Represents numeric values, including integers and floating-point numbers.
- Boolean: Represents either true or false.
- Undefined: Represents a variable that has been declared but not assigned a value.
- Null: Represents the intentional absence of any object value.

Object Data Type:

- Object: Represents a collection of key-value pairs. Objects can be used to store and organize complex data structures.

Special Data Types:

- Symbol: Introduced in ECMAScript 6 (ES6), symbols are unique and immutable values that can be used as object keys.

- BigInt: Also introduced in ES6, BigInt is used to represent integers of arbitrary precision.

Derived Data Types:

- Function: A type of object that can be invoked.

COMMENT IN JS

In JavaScript (JS), a comment is a piece of text in the code that is not executed or interpreted by the JavaScript engine. Comments are used for adding explanatory notes or information within the code to make it more readable and understandable for developers. They are ignored by the browser or JavaScript runtime and are only meant for human readers.

Single-line comments: These comments start with "//" (double slashes) and continue until the end of the line.

Multi-line comments: These comments start with "/*" and end with "*/" and can span multiple lines.

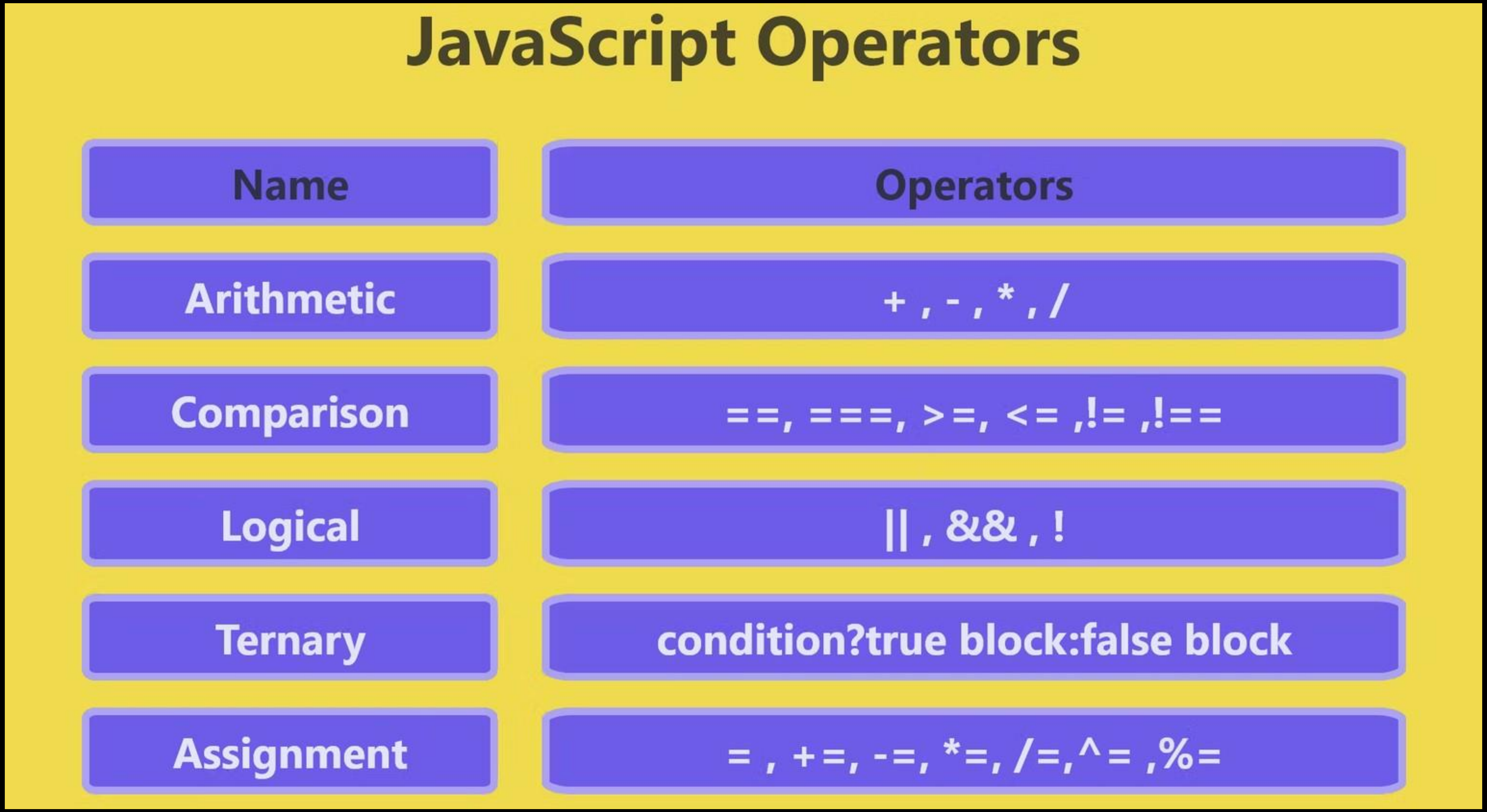
```
// This is a single-line comment
var x = 10; // Assigning a value to the variable x
```

```
/*
This is a multi-line comment.
It can span multiple lines without the need for
"//" on each line.
*/
var y = 20; // Another variable assignment
```

LEVEL 3

OPERATORS IN JS

In JavaScript, an "operator" refers to a special symbol or keyword that is used to perform operations on operands. Operands can be variables, values, or expressions. JavaScript operators are categorized into various types based on the kind of operation they perform. Here are some common types of operators in JavaScript:



JavaScript Operators

Name	Operators
Arithmetic	+ , - , * , /
Comparison	==, ===, >=, <= ,!= ,!==
Logical	|| , && , !
Ternary	condition?true block:false block
Assignment	= , +=, -=, *=, /=,^= ,%=

ARITHMETIC OPERATORS

Arithmetic Operators: Perform mathematical operations.

- **Addition: +**
- **Subtraction: -**
- **Multiplication: ***
- **Division: /**
- **Modulus: %**

ASSIGNMENT OPERATORS

Assignment Operators: Assign values to variables.

- Assignment: =
- Addition assignment: +=
- Subtraction assignment: -=
- Multiplication assignment: *=
- Division assignment: /=

COMPARISON OPERATORS

Comparison Operators: Compare values and return a Boolean result.

- Equal to: == or === (strict equality)
- Not equal to: != or !== (strict inequality)
- Greater than: >
- Less than: <
- Greater than or equal to: >=
- Less than or equal to: <=

LOGICAL OPERATORS

Logical Operators: Perform logical operations and return a Boolean result.

- AND: &&
- OR: ||
- NOT: !

UNARY OPERATOR

Unary Operators: Operate on a single operand.

- Increment: ++
- Decrement: --

LEVEL 4

CONDITIONAL STATEMENTS

In JavaScript, conditional statements are used to make decisions in your code based on certain conditions. The most common conditional statements are if, else if, and else. Here's a brief overview:

if statement :

```
if (condition) {
  // code to be executed if the condition is true
}
```

if-else statement

```
if (condition) {
// code to be executed if the condition is true
} else {
// code to be executed if the condition is false
}
```

if-else if-else statement:

```
if  (condition1) {
// code to be executed if condition1 is true
} else if  (condition2) {
// code to be executed if condition2 is true
} else {
// code to be executed if none of the conditions are true
}
```

TERNARY OPERATOR

Ternary Operator (Conditional Operator): A shorthand for an if-else statement.

- Syntax: condition ? expression_if_true : expression_if_false

SWITCH STATEMENT

Certainly! In JavaScript, a switch statement is a control flow statement that allows you to compare a variable against multiple values and execute code blocks based on the matching value. It provides a more concise way to write multiple if...else statements.

```
switch (expression) {
case value1:
// code to be executed if expression matches value1
break;
case value2:
// code to be executed if expression matches value2
break;
// additional cases as needed
default:
// code to be executed if no case matches expression
}
```

Here's a brief explanation of how it works:

- The switch statement evaluates the expression.
- If the expression matches the value in a case, the code block following that case is executed.
- If no case matches, the code block following the default keyword is executed (optional).

```
let day = "Monday";

switch (day) {
case "Monday":
console.log("It's the start of the week");
break;
case "Friday":
console.log("It's almost the weekend");
break;
default:
console.log("It's a regular day");
}
```

In this example, since the value of day is "Monday," the code block under the case "Monday": will be executed, and the output will be "It's the start of the week."

ALERT, CONFIRM OR PROMPT

Certainly! In JavaScript, an alert is a method that displays a dialog box with a specified message and an OK button.

A confirm is another dialog box that displays a message along with OK and Cancel buttons.

A prompt is similar to confirm but also allows the user to input text.

Alert:

- Syntax: alert("Your message");

```
alert("This is an alert box!");
```

Confirm:

- Syntax: confirm("Your message");

```
if (confirm("Do you want to proceed?")) {
// Code to execute if OK is pressed
} else {
// Code to execute if Cancel is pressed
}
```

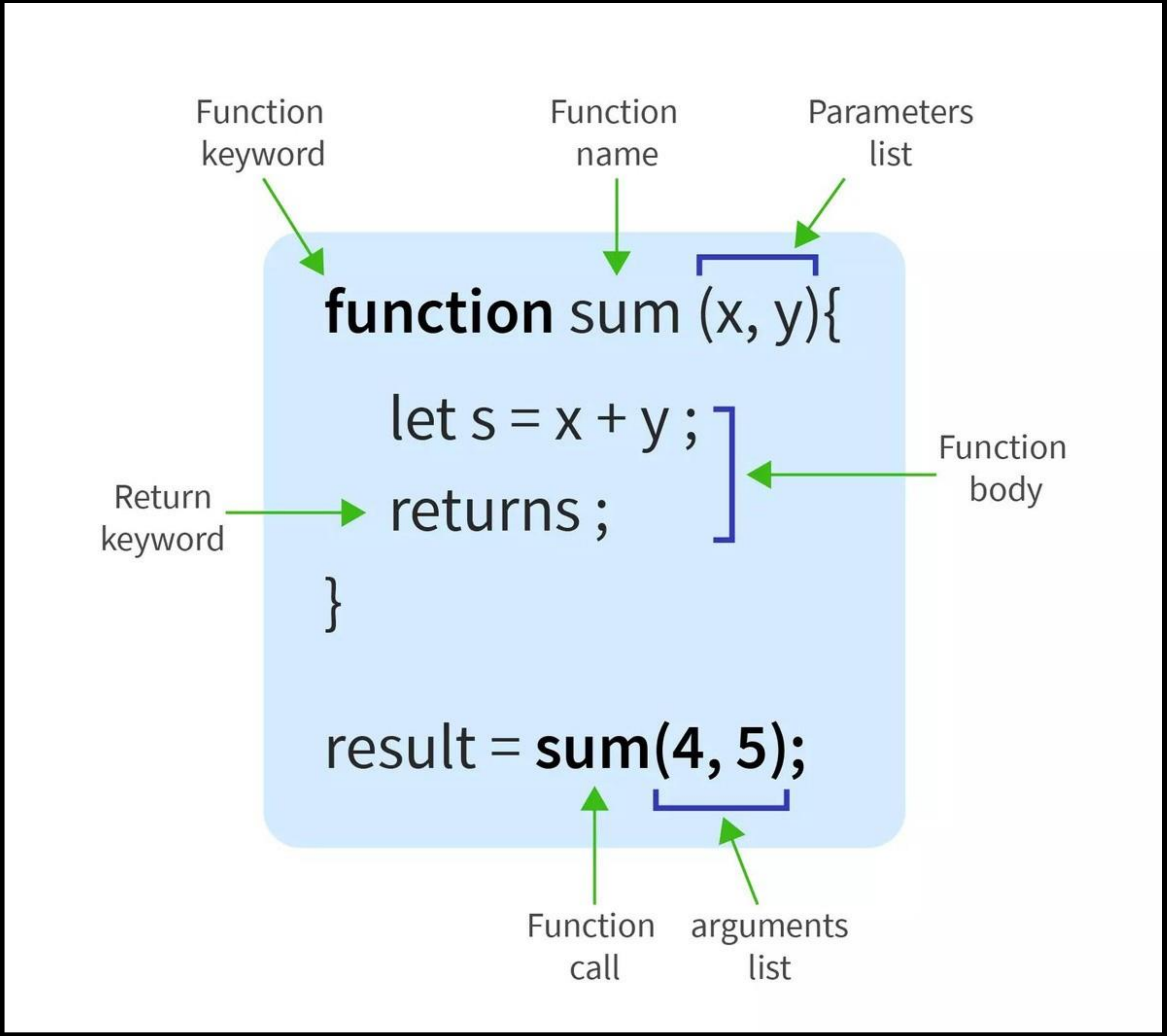
Prompt:

- **Syntax: prompt("Your message", "Default value");**

```
let userInput = prompt("Enter your name:", "John Doe");
if (userInput !== null) {
// Code to execute with user input
} else {
// Code to execute if Cancel is pressed
}
```

LEVEL 5

FUNCTIONS IN JS



```
function sum (x, y){
    let s = x + y ;
    returns ;
}

result = sum(4, 5);
```

In JavaScript, functions are blocks of reusable code that perform a specific task. They are a fundamental building block of the language and are used to organize and structure code. Functions can be defined using the function keyword.

FUNCTION WITH PARAMETER

In JavaScript, functions with parameters allow you to pass values into the function when it is called. Parameters are variables that act as placeholders for the values you provide when invoking the function. Here's a simple example:

```
// Function with parameters
function greet(name) {
console.log("Hello, " + name + "!");
}

// Calling the function with an argument
greet("John"); // Output: Hello, John!

// Calling the function with a different argument
greet("Alice"); // Output: Hello, Alice!
```

In this example, the greet function has one parameter, name. When you call the function and provide an argument (e.g., "John" or "Alice"), that argument is substituted for the name parameter within the function's code.

You can define multiple parameters in a function, and when calling the function, you need to provide values in the same order as the parameters are defined:

```
// Function with multiple parameters
function addNumbers(a, b) {
return a + b;
}

// Calling the function with arguments
let result = addNumbers(5, 7);
console.log(result); // Output: 12
```

Here, the addNumbers function takes two parameters (a and b) and returns their sum. When calling the function with addNumbers(5, 7), it adds 5 and 7, resulting in 12.

FUNCTION WITH RETURN STATEMENT

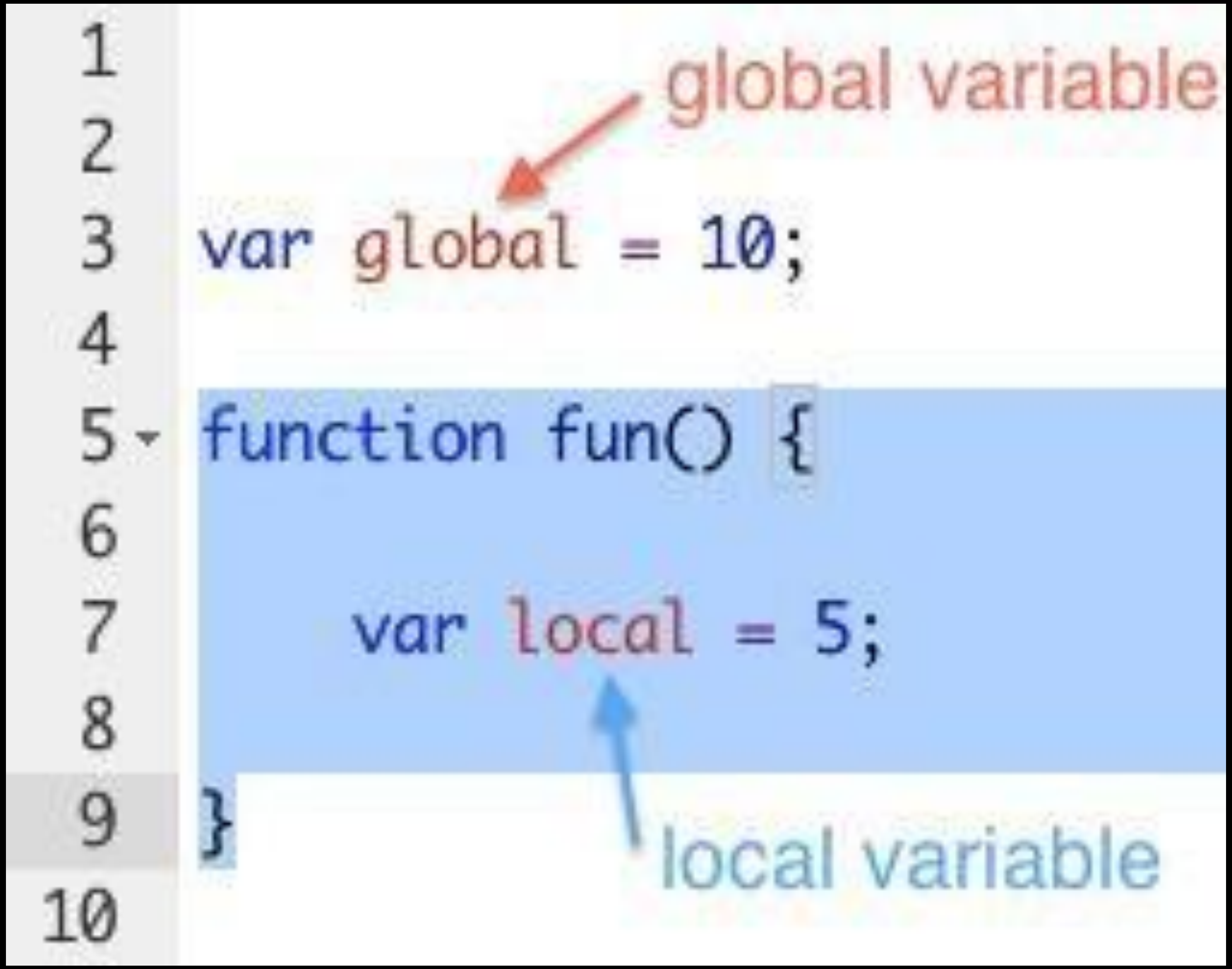
In JavaScript, functions with a return statement are used to perform a specific task and then return a value to the calling code. The return statement is used to send a value back to the code that called the function. This allows you to use the result of the function in your program.

```
function addNumbers(a, b) {
return a + b;
}

// Calling the function and storing the result in a variable
let sum = addNumbers(5, 3);

// Printing the result
console.log(sum); // Output: 8
```

In this example, the addNumbers function takes two parameters (a and b), adds them together using the + operator, and then returns the result using the return statement.

GLOBAL AND LOCAL VARIABLE

Certainly! In JavaScript, variables can be categorized into two main types: global variables and local variables.



```

var global = 10;

function fun() {

    var local = 5;

}

```

Global Variables:

- A global variable is declared outside of any function or block.
- It is accessible throughout the entire code, both inside and outside functions.
- Declaring a variable without the var, let, or const keyword automatically makes it a global variable.

```
// Global variable
var globalVar = "I am global";

function exampleFunction() {
console.log(globalVar);
// Accessible here
}
```

Local Variables:

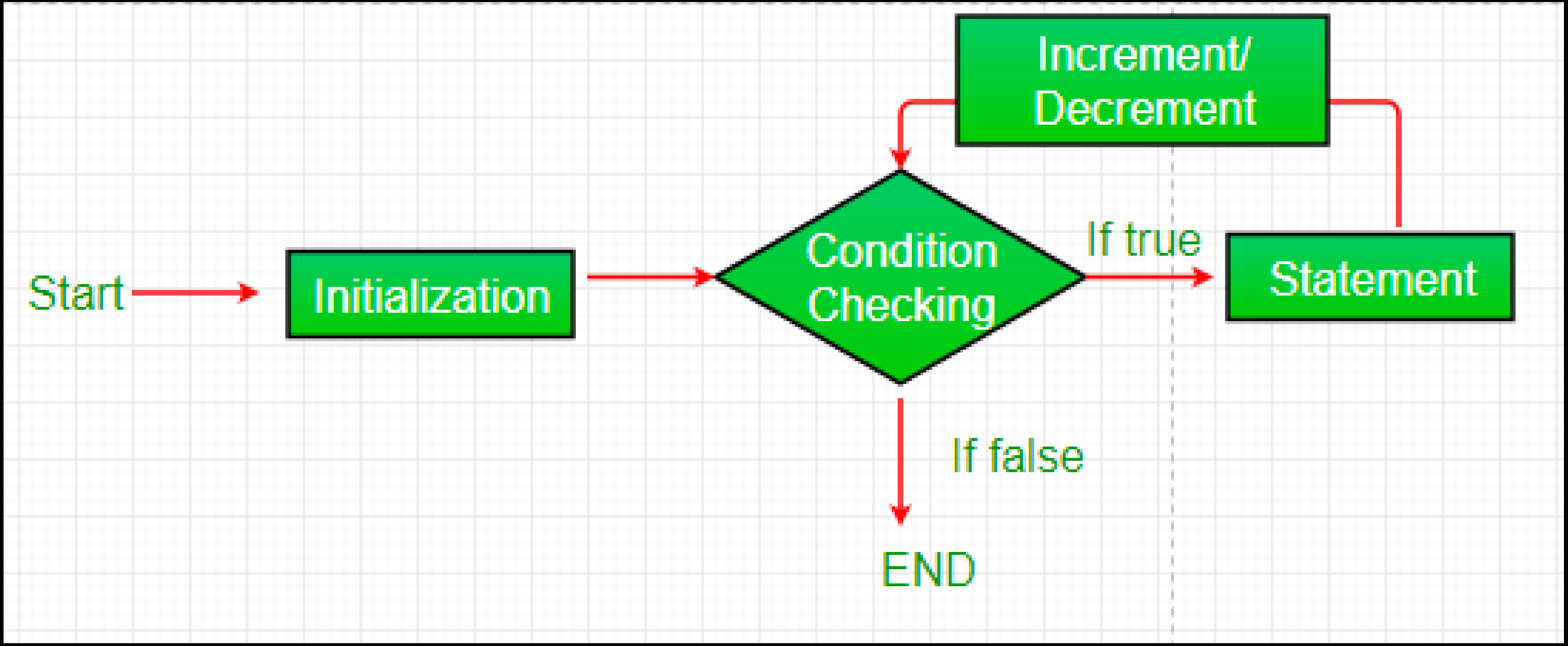
- A local variable is declared within a function or a block using the var, let, or const keyword.
- It is only accessible within the function or block where it is declared.

```
function exampleFunction() {
// Local variable
var localVar = "I am local";
console.log(localVar); // Accessible here
}

// Trying to access localVar outside the function will result in an error
// console.log(localVar); // This will throw an error
```

LEVEL 6

LOOPS IN JS

In JavaScript, loops are used to repeatedly execute a block of code as long as a specified condition is true. There are different types of loops, but two common ones are the for loop and the while loop.



For Loop:

Here's the syntax:

```
for (initialization; condition; increment/decrement) {
// code to be executed
}
```

```
for (let i = 0; i < 5; i++) {
console.log("Iteration " + i);
}
```

This for loop initializes i to 0, runs as long as i is less than 5, and increments i by 1 in each iteration.

While Loop:

Here's the syntax:

```
while  (condition)  {
 // code to be executed
}
```

```
let i = 0;
while (i < 5) {
console.log("Iteration " + i);
i++;
}
```

This while loop runs as long as i is less than 5, and increments i by 1 in each iteration.

Do-while LOOP

Certainly! The do-while loop is another type of loop in JavaScript. It is similar to the while loop, but with a key difference: the do-while loop always executes the code block at least once, and then it checks the specified condition. Here's the syntax for the do-while loop:

```
do {
// code to be executed
} while (condition);
```

```
let i = 0;

do {
console.log("Iteration " + i);
i++;
} while (i < 5);
```

In this example, the loop will execute at least once because the condition i < 5 is checked after the code block is executed. As i increments, the loop will continue to execute until the condition becomes false.

BREAK AND CONTINUE STATEMENT

In JavaScript, break and continue are control flow statements used in loops to modify the default flow of execution.

break:

- The break statement is used to exit a loop prematurely, before its normal termination condition is met.
- It can be used in for, while, and do...while loops.

```
for (let i = 0; i < 5; i++) {
    if (i === 3) {
        break;
    }
    console.log(i);
}
// Output: 0 1 2
```

continue:

- The continue statement is used to skip the rest of the code inside a loop for the current iteration and move to the next iteration.
- It allows you to skip certain iterations without terminating the entire loop.

```
for (let i = 0; i < 5; i++) {
if (i === 2) {
continue;
}
console.log(i);
}
// Output: 0 1 3 4
```

NESTED LOOP

Certainly! In JavaScript, a nested loop is a loop inside another loop. This is often used when you need to iterate over a set of data multiple times, typically with an outer loop controlling the number of iterations and an inner loop handling the individual elements.

```
for (let i = 0; i < 3; i++) {
console.log("Outer loop iteration: " + i);

for (let j = 0; j < 2; j++) {
console.log("  Inner loop iteration: " + j);
}
}
```

In this example, the outer loop runs three times, and for each iteration of the outer loop, the inner loop runs twice. The console.log statements inside both loops demonstrate the iterations.

Nested loops are commonly used when working with 2D arrays or matrices, among other scenarios where you need to perform operations on each element in a structured manner.

LEVEL 7

ARRAY IN JS

In JavaScript, an array is a data structure that allows you to store multiple values in a single variable. Arrays can hold various types of data, including numbers, strings, objects, and even other arrays. Here are some basic operations and information related to arrays in JavaScript:

Declaration: You can create an array using the following syntax:

```
let myArray = [1, 2, 3, 'four', true];
```

Accessing Elements: Array elements are accessed using their index, starting from 0. For example:

```
let firstElement = myArray[0]; // Access the first element
```

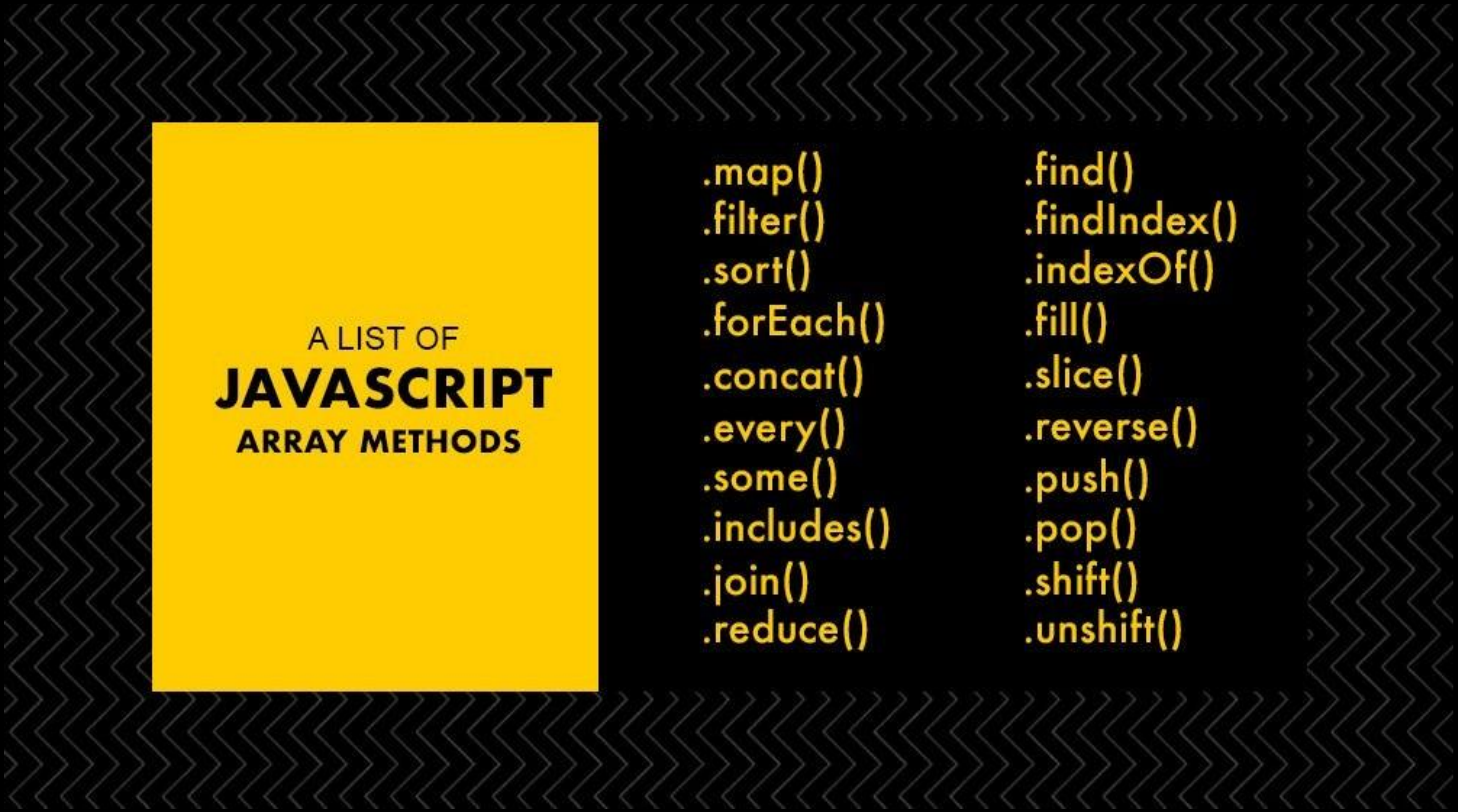
Length: You can find the length of an array using the length property:

```
let arrayLength = myArray.length;
```

Array.isArray: You can check if a variable is an array using the Array.isArray method:

```
let isArray = Array.isArray(myArray);
```

ARRAY METHODS



A LIST OF
JAVASCRIPT
ARRAY METHODS

.map()	.find()
.filter()	.findIndex()
.sort()	.indexOf()
.forEach()	.fill()
.concat()	.slice()
.every()	.reverse()
.some()	.push()
.includes()	.pop()
.join()	.shift()
.reduce()	.unshift()

Certainly! In JavaScript, array methods are functions that can be called on arrays to perform various operations. Here are some commonly used array methods in JavaScript:

push(): Adds one or more elements to the end of an array.

```
let fruits = ['apple', 'banana'];
fruits.push('orange');
// fruits is now ['apple', 'banana', 'orange']
```

pop(): Removes the last element from an array.

```
let fruits = ['apple', 'banana', 'orange'];
fruits.pop();
// fruits is now ['apple', 'banana']
```

shift(): Removes the first element from an array.

```
let fruits = ['apple', 'banana', 'orange'];
fruits.shift();
// fruits is now ['banana', 'orange']
```

unshift(): Adds one or more elements to the beginning of an array.

```
let fruits = ['banana', 'orange'];
fruits.unshift('apple');
// fruits is now ['apple', 'banana', 'orange']
```

indexOf(): Returns the index of the first occurrence of a specified element in an array.

```
let fruits = ['apple', 'banana', 'orange'];
let index = fruits.indexOf('banana');
// index is 1
```

slice(): Returns a shallow copy of a portion of an array.

```
let fruits = ['apple', 'banana', 'orange', 'grape'];
let citrus = fruits.slice(2, 4);
// citrus is ['orange', 'grape']
```

splice(): Changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.

```
let fruits = ['apple', 'banana', 'orange', 'grape'];
fruits.splice(2, 1, 'kiwi');
// fruits is now ['apple', 'banana', 'kiwi', 'grape']
```

FOR EACH LOOP

Certainly! In JavaScript, the "for each" loop is not a standalone construct like in some other programming languages. Instead, JavaScript provides the forEach method for arrays. It allows you to iterate over each element in an array and execute a provided function for each element.

```
const array = [1, 2, 3, 4, 5];

array.forEach(function(element) {
console.log(element);
});
```

In this example, the forEach method is called on the array, and the provided function is executed for each element in the array. The function takes the current element as its argument, and you can perform any desired operations within the function.

MULTIDIMENSIONAL ARRAYS

In JavaScript, a multidimensional array is essentially an array of arrays. You can create and work with multidimensional arrays to represent tables or matrices in your code. Here's an example of how you can create a 2D array:

```
// Creating a 2D array
let twoDArray = [
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]
];

// Accessing elements in a 2D array
console.log(twoDArray[0][1]); // Output: 2

// Modifying elements in a 2D array
twoDArray[1][2] = 10;
console.log(twoDArray);
// Output: [[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```

LEVEL 8

OBJECTS IN JS

In JavaScript, "objects" refer to a fundamental data type that allows you to store and organize data using key-value pairs. Objects in JavaScript can be thought of as collections of properties, where each property has a key (also known as a name or identifier) and a corresponding value.

```
let person = {
firstName: "John",
lastName: "Doe",
age: 30,
isStudent: false
};
```

In this example, person is an object with four properties: firstName, lastName, age, and isStudent. The keys are on the left side of the colons, and the values are on the right side. You can access the values of these properties using dot notation or square bracket notation, like this:

```
console.log(person.firstName); // Outputs: John
console.log(person['lastName']); // Outputs: Doe
```

Creating Objects: You can create an object using object literal notation {} or with the Object constructor.

```
// Using object literal notation
let myObject = {};

// Using Object constructor
let anotherObject = new Object();
```

Adding Properties: You can add properties to an object by assigning values to keys.

```
myObject.name = "John";
myObject.age = 25;
```

Accessing Properties: You can access the values of object properties using dot notation or square bracket notation.

```
console.log(myObject.name); // Output: John
console.log(myObject["age"]); // Output: 25
```

Nested Objects: Objects can contain other objects as values.

```
myObject.address = {
city: "New York",
zip: "10001"
};
```

Methods: Functions can be added to objects as methods.

```
myObject.sayHello = function() {
console.log("Hello!");
};
```

Object Methods: JavaScript provides built-in methods for objects, such as Object.keys(), Object.values(), and Object.entries().

```
console.log(Object.keys(myObject));
// Output: ['name', 'age', 'address', 'sayHello']
```

FOR IN LOOP

Certainly! In JavaScript, the for...in loop is used to iterate over the properties of an object. Here's a basic syntax:

```
for (variable in object) {
  // code to be executed
}
```

- variable: A variable that will be assigned the property name on each iteration.
- object: The object whose enumerable properties will be iterated.

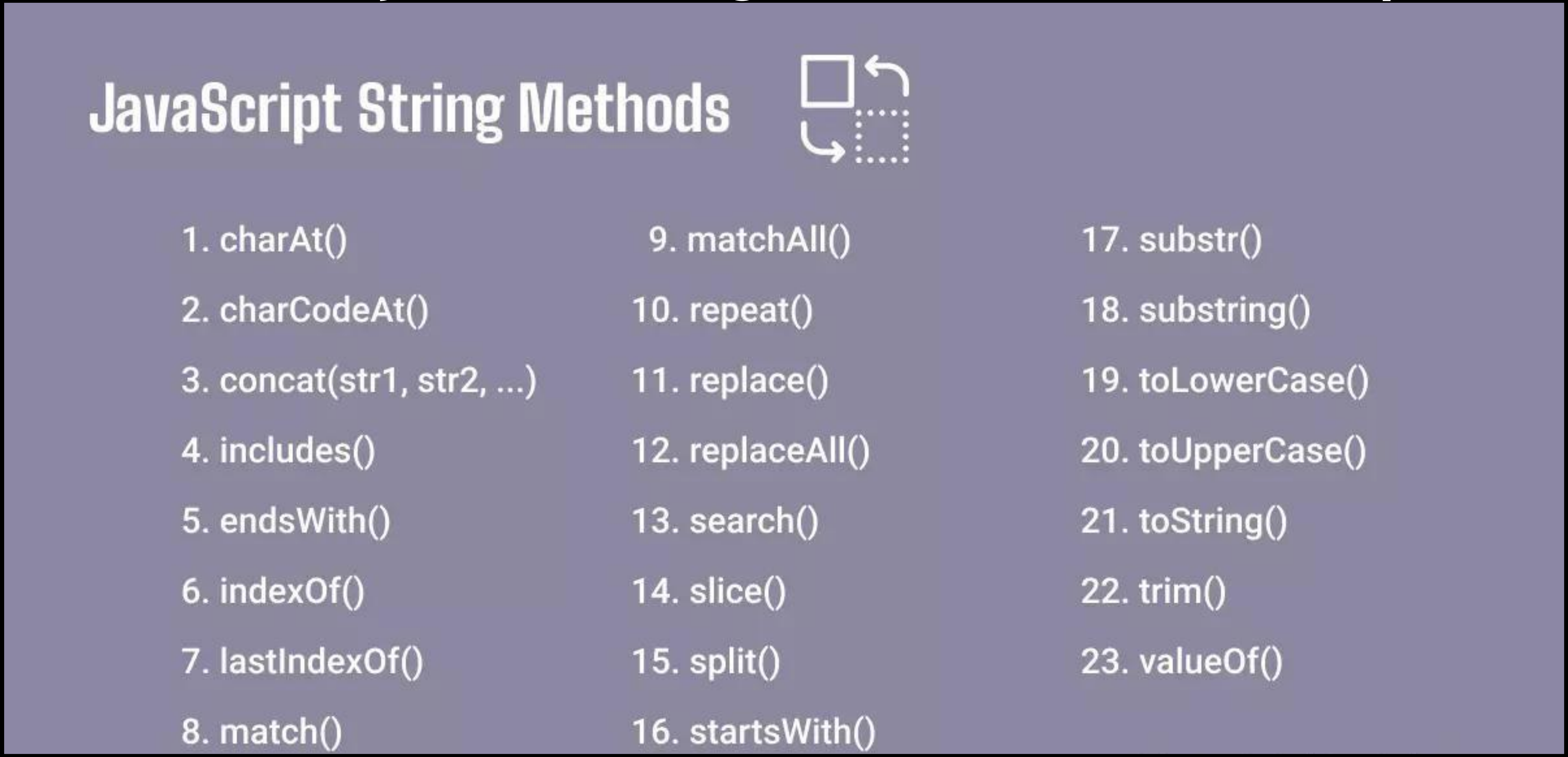
```
const myObject = { a: 1, b: 2, c: 3 };

for (let key in myObject) {
console.log(key, myObject[key]);
}
```

This loop will iterate over the properties (a, b, c) of the myObject and log both the property name and its corresponding value.

LEVEL 9

STRING METHODS IN JS

In JavaScript, string methods are functions that can be applied to strings to perform various operations or manipulations. Here are some commonly used string methods in JavaScript:



JavaScript String Methods

1. charAt()
2. charCodeAt()
3. concat(str1, str2, ...)
4. includes()
5. endsWith()
6. indexOf()
7. lastIndexOf()
8. match()
9. matchAll()
10. repeat()
11. replace()
12. replaceAll()
13. search()
14. slice()
15. split()
16. startsWith()
17. substr()
18. substring()
19. toLowerCase()
20. toUpperCase()
21. toString()
22. trim()
23. valueOf()

charAt(index): Returns the character at the specified index in a string.

```
var str = "Hello";
console.log(str.charAt(1)); // Outputs 'e'
```

concat(str1, str2, ...): Combines two or more strings.

```
var str1 = "Hello";
var str2 = " World";
console.log(str1.concat(str2)); // Outputs 'Hello World'
```

indexOf(searchValue, startIndex): Returns the index of the first occurrence of the specified value in a string.

```
var str = "Hello World";
console.log(str.indexOf("World")); // Outputs 6
```

toUpperCase(): Converts a string to uppercase.

```
var str = "hello";
console.log(str.toUpperCase()); // Outputs 'HELLO'
```

toLowerCase(): Converts a string to lowercase.

```
var str = "WORLD";
console.log(str.toLowerCase()); // Outputs 'world'
```

slice(startIndex, endIndex): Extracts a portion of a string.

```
var str = "Hello World";
console.log(str.slice(0, 5)); // Outputs 'Hello'
```

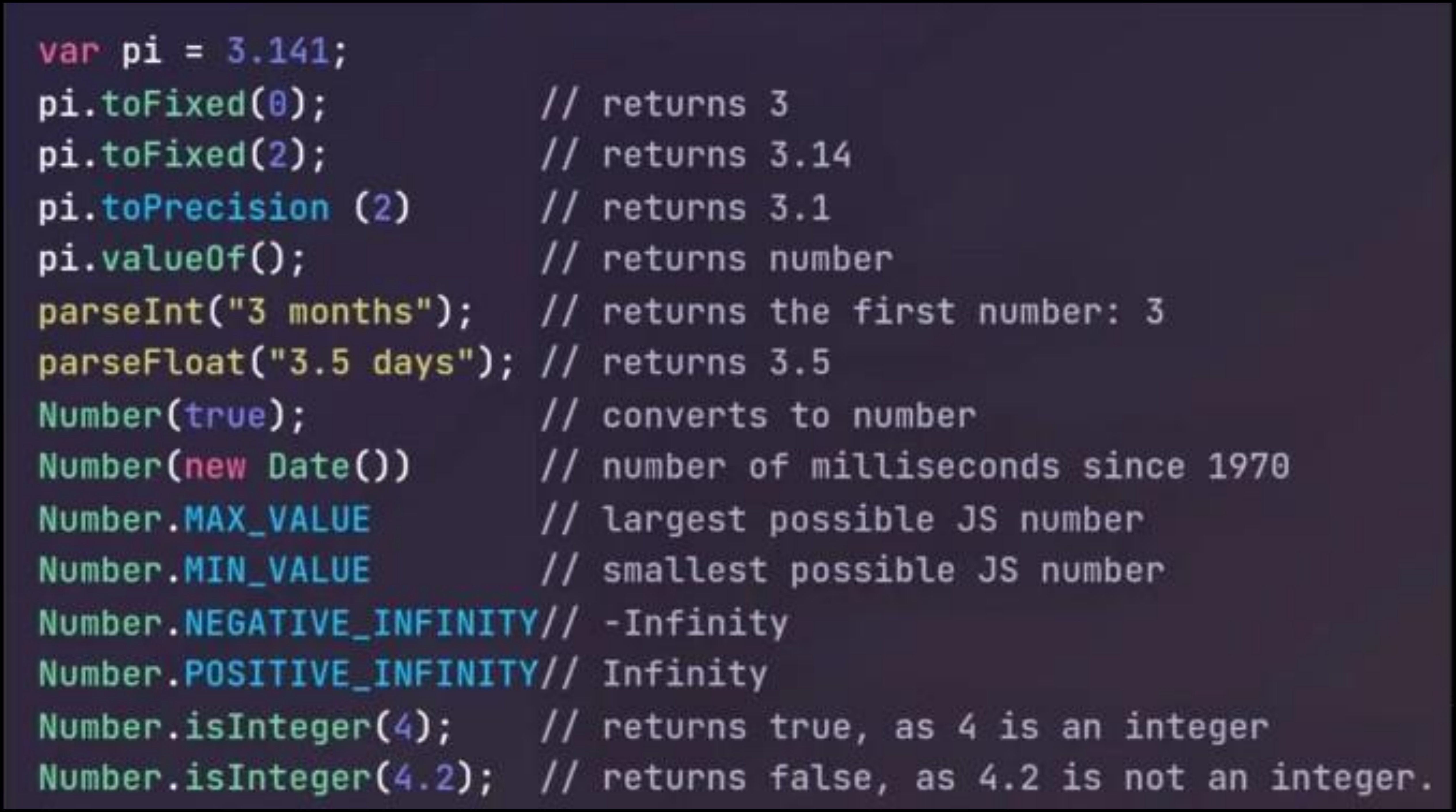
replace(oldValue, newValue): Replaces a specified value with another value in a string.

```
var str = "Hello World";
console.log(str.replace("World", "Universe")); // Outputs
'Hello Universe'
```

split(separator): Splits a string into an array of substrings based on a specified separator.

```
var str = "apple,orange,banana";
console.log(str.split(',')); // Outputs ['apple', 'orange',
'banana']
```

NUMBER METHODS IN JS



```
var pi = 3.141;
pi.toFixed(0);            // returns 3
pi.toFixed(2);            // returns 3.14
pi.toPrecision (2)        // returns 3.1
pi.valueOf();             // returns number
parseInt("3 months");     // returns the first number: 3
parseFloat("3.5 days");   // returns 3.5
Number(true);             // converts to number
Number(new Date())        // number of milliseconds since 1970
Number.MAX_VALUE          // largest possible JS number
Number.MIN_VALUE          // smallest possible JS number
Number.NEGATIVE_INFINITY// -Infinity
Number.POSITIVE_INFINITY// Infinity
Number.isInteger(4);      // returns true, as 4 is an integer
Number.isInteger(4.2);    // returns false, as 4.2 is not an integer.
```

In JavaScript, there are several methods and properties related to numbers that you can use for various operations. Here are some commonly used ones:

toFixed(): Converts a number to a string, rounding to a specified number of decimal places.

```
let num = 3.14159;
let rounded = num.toFixed(2); // Result: "3.14"
```

parseInt(): Parses a string and returns an integer.

```
let str = "123";
let num = parseInt(str); // Result: 123
```

parseFloat(): Parses a string and returns a floating-point number.

```
let str = "3.14";
let num = parseFloat(str); // Result: 3.14
```

isNaN(): Checks if a value is NaN (Not-a-Number).

```
isNaN("Hello"); // Result: true
isNaN(123); // Result: false
```

isFinite(): Checks if a value is a finite number.

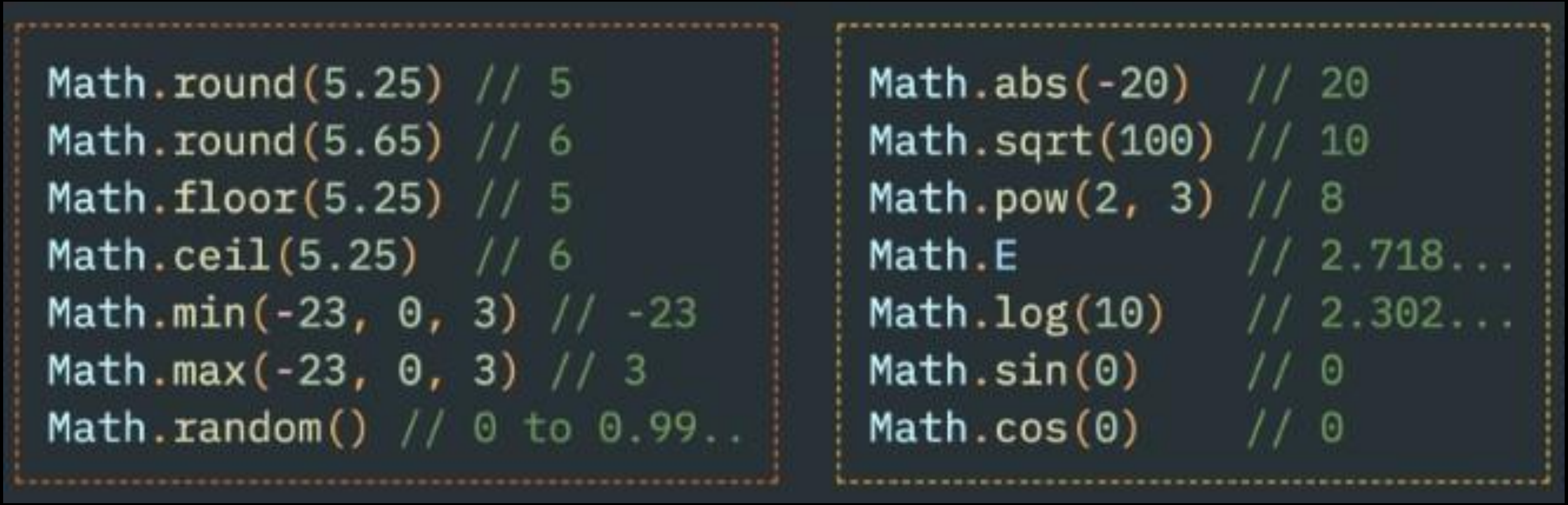
```
isFinite(42); // Result: true
isFinite(Infinity); // Result: false
```

Number(): Converts a value to a number.

```
let str = "42";
let num = Number(str); // Result: 42
```

MATH METHODS IN JS

In JavaScript, you can perform various mathematical operations using built-in Math methods. Here are some commonly used Math methods in JavaScript:



```
Math.round(5.25) // 5
Math.round(5.65) // 6
Math.floor(5.25) // 5
Math.ceil(5.25)  // 6
Math.min(-23, 0, 3) // -23
Math.max(-23, 0, 3) // 3
Math.random() // 0 to 0.99..
```

```
Math.abs(-20)   // 20
Math.sqrt(100) // 10
Math.pow(2, 3) // 8
Math.E          // 2.718...
Math.log(10)    // 2.302...
Math.sin(0)     // 0
Math.cos(0)     // 0
```

Math.abs(x): Returns the absolute value of a number.

```
let absoluteValue = Math.abs(-5); // Result: 5
```

Math.ceil(x): Returns the smallest integer greater than or equal to a given number.

```
let ceilValue = Math.ceil(4.3); // Result: 5
```

Math.floor(x): Returns the largest integer less than or equal to a given number.

```
let floorValue = Math.floor(4.9); // Result: 4
```

Math.round(x): Rounds a number to the nearest integer.

```
let roundValue = Math.round(4.6); // Result: 5
```

Math.random(): Generates a pseudo-random number between 0 (inclusive) and 1 (exclusive).

```
let randomValue = Math.random(); // Result: Random value between 0 and 1
```

Math.max(x, y, ...): Returns the largest of the given numbers.

```
let maxValue = Math.max(10, 5, 8); // Result: 10
```

Math.min(x, y, ...): Returns the smallest of the given numbers.

```
let minValue = Math.min(10, 5, 8); // Result: 5
```

Math.pow(x, y): Returns the result of raising x to the power of y.

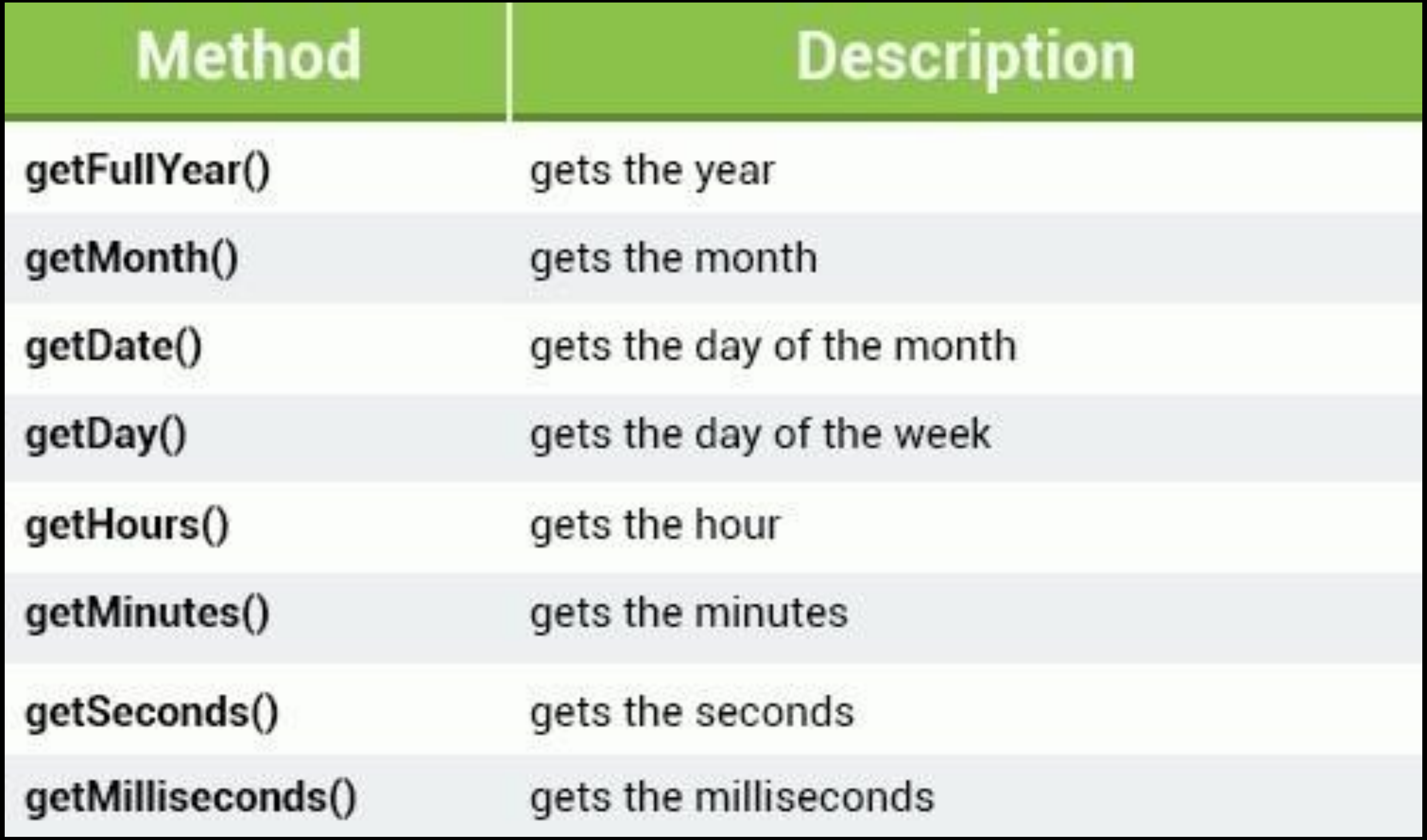
```
let powerResult = Math.pow(2, 3); // Result: 8
```

Math.sqrt(x): Returns the square root of a number.

```
let squareRoot = Math.sqrt(25); // Result: 5
```

DATE METHODS IN JS

Certainly! In JavaScript, the Date object is used to work with dates and times. Here are some commonly used methods associated with the Date object:



Method	Description
getFullYear()	gets the year
getMonth()	gets the month
getDate()	gets the day of the month
getDay()	gets the day of the week
getHours()	gets the hour
getMinutes()	gets the minutes
getSeconds()	gets the seconds
getMilliseconds()	gets the milliseconds

new Date(): Creates a new Date object representing the current date and time.

```
let currentDate = new Date();
```

getDate(): Returns the day of the month (1-31) for the specified date.

```
let dayOfMonth = currentDate.getDate();
```

getMonth(): Returns the month (0-11) for the specified date.

```
let month = currentDate.getMonth();
```

Note: Months are zero-based, so January is 0 and December is 11.

getFullYear(): Returns the year (4 digits for 4-digit years) for the specified date.

```
let year = currentDate.getFullYear();
```

getHours(), getMinutes(), getSeconds(): Return the hour, minute, and second of the specified date, respectively.

```
let hours = currentDate.getHours();
let minutes = currentDate.getMinutes();
let seconds = currentDate.getSeconds();
```

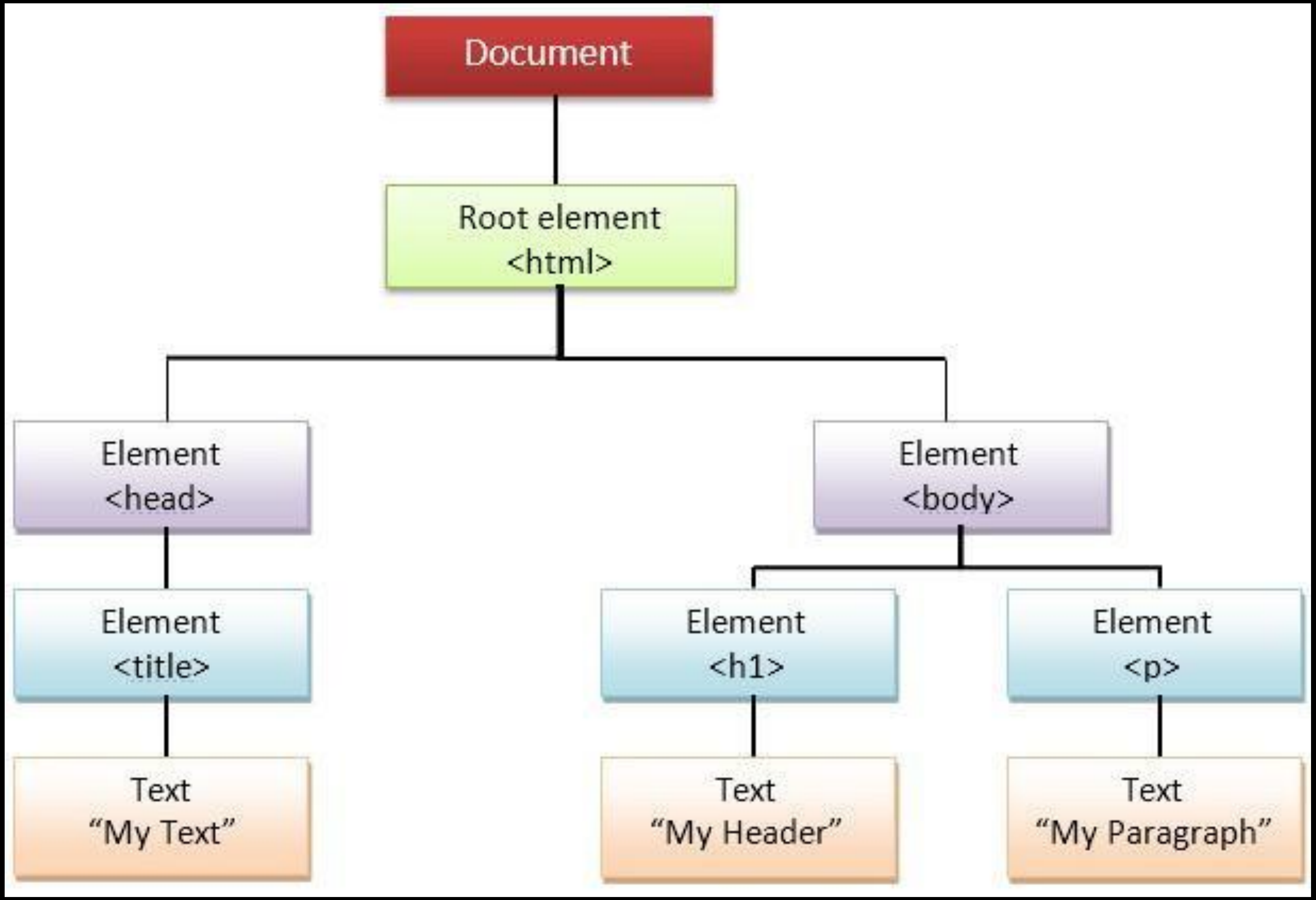
toLocaleDateString(): Returns a string representing the date portion of the Date object in a localized format.

```
let formattedDate = currentDate.toLocaleDateString();
```

LEVEL 10

DOM (DOCUMENT OBJECT MODEL)

In JavaScript, the Document Object Model (DOM) is a programming interface that represents the structure of a document as a tree of objects. It allows scripts to dynamically access and manipulate the content, structure, and style of a document.



Here's a basic overview of how to use the DOM in JavaScript:

Accessing Elements:

To access elements in the DOM, you can use various methods. The most common one is getElementById to get an element by its ID:

```
let element =
document.getElementById("yourElementId");
```

- **You can also use getElementsByClassName, getElementsByTagName, or querySelector to select elements by class, tag name, or CSS selector.**

getElementsByClassName:

```
var elementsByClassExample =
document.getElementsByClassName('yourClassName');
```

getElementsByTagName:

```
var elementsByTagExample =
document.getElementsByTagName('yourTagName');
```

querySelector: querySelector returns the first element that matches a specified CSS selector

```
var queryExample =
document.querySelector('yourCSSSelector');
```

querySelectorAll: querySelectorAll returns a NodeList containing all elements that match a specified CSS selector.

```
var queryAllExample =
document.querySelectorAll('yourCSSSelector');
```

INNER HTML & INNER TEXT

Certainly! In JavaScript, both innerHTML and innerText are properties used to manipulate the content of HTML elements, but they work in slightly different ways.

innerHTML:

- innerHTML is a property that allows you to get or set the HTML content inside an element.
- When used to get the content, it returns a string representing the HTML content of the element, including any HTML tags.
- When used to set the content, it replaces the existing content with the new HTML content provided.

```
// Get HTML content
var elementContent =
document.getElementById("exampleElement").innerHTML;

// Set HTML content
document.getElementById("exampleElement").innerHTML = "<p>New
content</p>";
```

innerText:

- innerText is a property that allows you to get or set the text content inside an element.
- When used to get the content, it returns a string representing the text content, excluding any HTML tags.
- When used to set the content, it replaces the existing content with the new text content provided.

```
// Get text content
var elementText =
document.getElementById("exampleElement").innerText;

// Set text content
document.getElementById("exampleElement").innerText = "New text
content";
```

Manipulating Elements:

- You can modify elements by changing their properties.

For example, to change the text content:

```
element.textContent = "New Text";
```

To change an attribute:

```
element.setAttribute("attributeName",   "attributeValue");
```

Creating and Appending Elements:

- You can create new elements using document.createElement and append them to the document using appendChild:

```
let newElement = document.createElement("div");
document.body.appendChild(newElement);
```

Event Handling:

- DOM allows you to handle events like clicks, keypress, etc. You can use addEventListener for this:

```
element.addEventListener("click", function() {
// Your code here
});
```

Traversing the DOM:

- You can navigate through the DOM using properties like parentNode, childNodes, nextSibling, and previousSibling:

```
let  parent  =  element.parentNode;
let   siblings   =   element.parentNode.childNodes;
```

Modifying Styles:

- You can change the styles of an element using the style property:

```
element.style.color = "red";
```

Manipulating Classes:

- Adding, removing, or toggling classes can be done using classList:

```
element.classList.add("newClass");
element.classList.remove("oldClass");
```

EVENTS IN JS

In JavaScript, events are actions or occurrences that happen in the browser, such as a user clicking a button, resizing a window, or submitting a form. Handling events is a crucial aspect of web development for creating interactive and dynamic user interfaces. Here are some key points about events in JavaScript:

Event Types:

- Mouse Events: These events occur when a user interacts with the mouse, such as clicking or moving it.
- Keyboard Events: Events related to keyboard interactions, such as key presses.
- Form Events: Events like form submission.
- Document/Window Events: Events related to the overall document or window, like loading or resizing.

Example Code:

- Adding an event listener for a button click:

```
const myButton = document.getElementById('myButton');

myButton.addEventListener('click', function(event) {
// Your code to handle the click event goes here
console.log('Button clicked!');
});
```

LEVEL 11

TEMPLATE STRINGS

In JavaScript, a template string is a way to create strings that allows for embedded expressions and multiline strings. Template strings are enclosed by backticks (` `) instead of single or double quotes. They can contain placeholders, indicated by the ${} syntax, where expressions or variables can be embedded.

```
const name = "John";
const age = 30;

// Using template string
const greeting = `Hello, my name is ${name} and I am ${age} years old.`;

console.log(greeting);
```

In the above example, the ${name} and ${age} inside the template string are placeholders that get replaced with the values of the name and age variables.

Template strings also support multiline strings without the need for concatenation:

```
const multilineString = `
This is a
multiline
string.
`;

console.log(multilineString);
```

ARROW FUNCTIONS

Certainly! In JavaScript, an arrow function is a concise way to write anonymous functions. Arrow functions were introduced in ECMAScript 6 (ES6) and provide a more concise syntax compared to traditional function expressions.

```
const add = (a, b) => {
return a + b;
};
```

REST AND SPREAD OPERATORS

Certainly! In JavaScript, the rest and spread operators are powerful features introduced in ECMAScript 6 (ES6) to work with arrays and objects.

Rest Operator (...):

- Used in function parameters to represent an indefinite number of arguments as an array.
- It collects all the remaining arguments into a single array variable.

```
function sum(...numbers) {
return numbers.reduce((acc, num) => acc + num, 0);
}

console.log(sum(1, 2, 3, 4)); // Output: 10
```

Spread Operator (...):

- Used to split array elements or object properties.
- Useful for spreading the elements of an array or the properties of an object into another array or object

Example with arrays:

```
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5];

console.log(arr2); // Output: [1, 2, 3, 4, 5]
```

Spread Operator (...):

- Used to split array elements or object properties.
- Useful for spreading the elements of an array or the properties of an object into another array or object

Example with Objects:

```
const obj1 = { a: 1, b: 2 };
const obj2 = { ...obj1, c: 3 };

console.log(obj2); // Output: { a: 1, b: 2, c: 3 }
```

OBJECT LITERALS

In JavaScript, object literals are a way to create objects using a concise syntax. They allow you to define and create an object and its properties in a single declaration. Here's a basic example:

```
let person = {
firstName: 'John',
lastName: 'Doe',
age: 30,
isStudent: false,
greet: function() {
console.log('Hello, ' + this.firstName + ' ' + this.lastName + '!');
}
};
```

In this example, person is an object literal with properties like firstName, lastName, age, isStudent, and a method greet. You can access and modify these properties using dot notation, like person.firstName or person.age, and invoke the greet method using person.greet().

DESTRUCTURING ARRAY AND OBJECTS

Certainly! In JavaScript, destructuring is a powerful feature that allows you to extract values from arrays or properties from objects and assign them to variables in a concise and readable way.

Destructuring Arrays:

Basic Array Destructuring:

```
const numbers = [1, 2, 3];
const [a, b, c] = numbers;

console.log(a);  // 1
console.log(b);  // 2
console.log(c);  // 3
```

Skipping Elements:

```
const numbers = [1, 2, 3, 4, 5];
const [a, , b] = numbers;

console.log(a);  //  1
console.log(b);  //  3
```

Rest Parameter:

```
const numbers = [1, 2, 3, 4, 5];
const [a, ...rest] = numbers;

console.log(a);    // 1
console.log(rest); // [2, 3, 4, 5]
```

Destructuring Objects:

Basic Object Destructuring:

```
const person = { name: 'John', age: 30 };
const { name, age } = person;

console.log(name); // John
console.log(age);  // 30
```

Renaming Variables:

```
const person = { name: 'John', age: 30 };
const { name: personName, age: personAge } = person;

console.log(personName);  //  John
console.log(personAge);  // 30
```

Default Values:

```
const person = { name: 'John' };
const { name, age = 25 } = person;

console.log(name);  //  John
console.log(age);  // 25 (default value)
```

Thaannkkk
you!