JavaScript – Part 1: Basics

Presented By

Anuradha Kumari Singh
(Research Scholar)
Department of Computer Science

TOPICS

1. Overview of JavaScript
2. Features of JavaScript
3. Syntax and Comments
4. Enabling JavaScript in HTML
5. Placement of Script (Head vs Body)
6. Variables (var, let, const)
7. Data Types
8. Operators (Arithmetic, Comparison, Logical)
9. Conditional Statements (if, else if, switch)
10. Loop Statements (For, While, Do-while)
11. Loop Control (break, continue)
12. Functions (Definition, Parameters, Return)

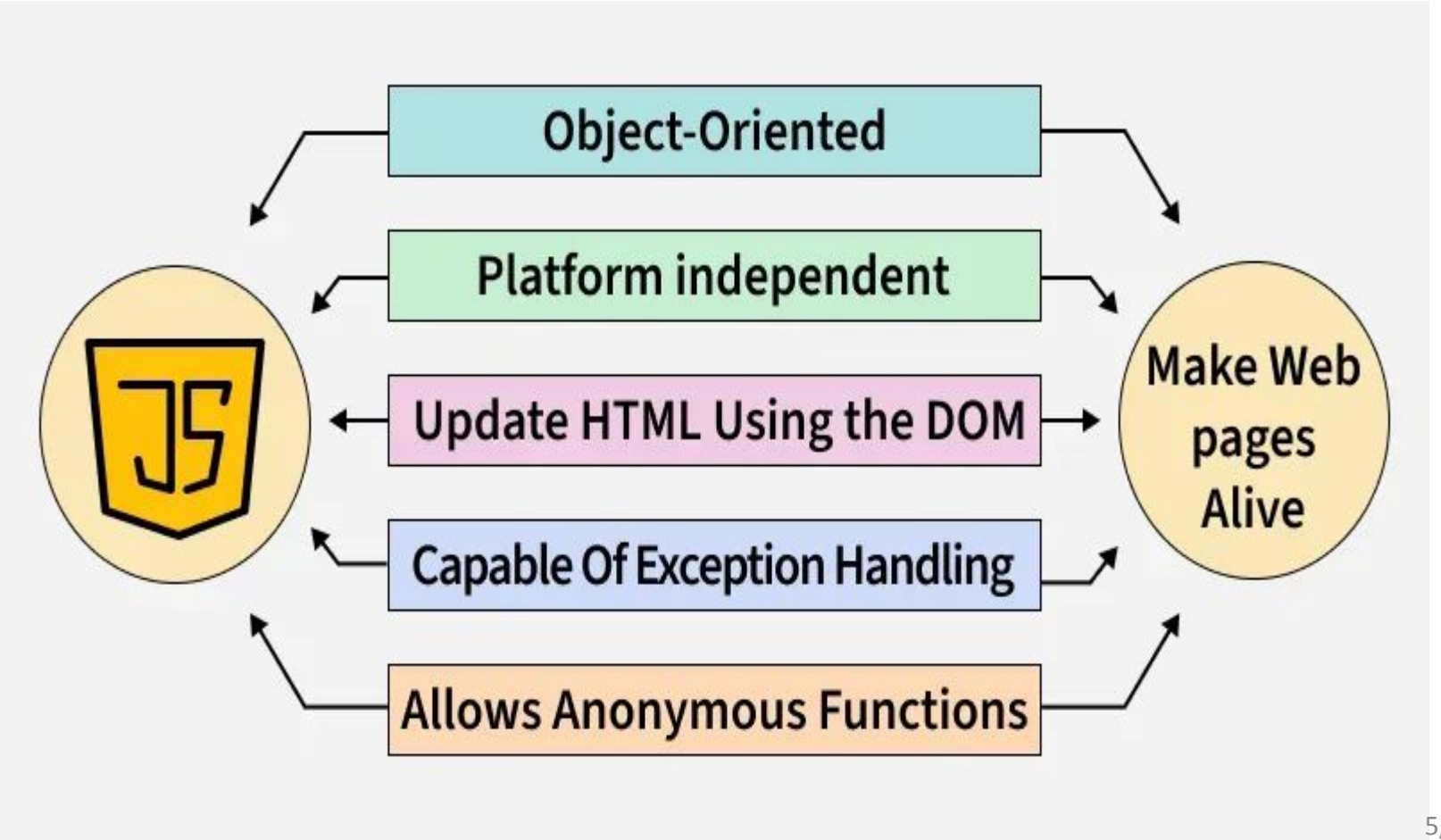
JavaScript is one of the 3 languages all web developers must learn:

1. HTML to define the content of web pages
2. CSS to specify the layout of web pages
3. JavaScript to program the behavior of web pages

Overview

- JavaScript is a lightweight, interpreted programming language.
- Used to make web pages dynamic and interactive.
- Developed by Brendan Eich in 1995, standardized as ECMAScript.

JavaScript is a versatile, dynamically typed programming language that brings life to web pages by making them interactive. It is used for building interactive web applications, supports both client-side and server-side development, and integrates seamlessly with HTML, CSS, and a rich standard library.



- JavaScript is a single-threaded language that executes one task at a time.
- It is an interpreted language which means it executes the code line by line.
- The data type of the variable is decided at run-time in JavaScript, which is why it is called dynamically typed.

Example

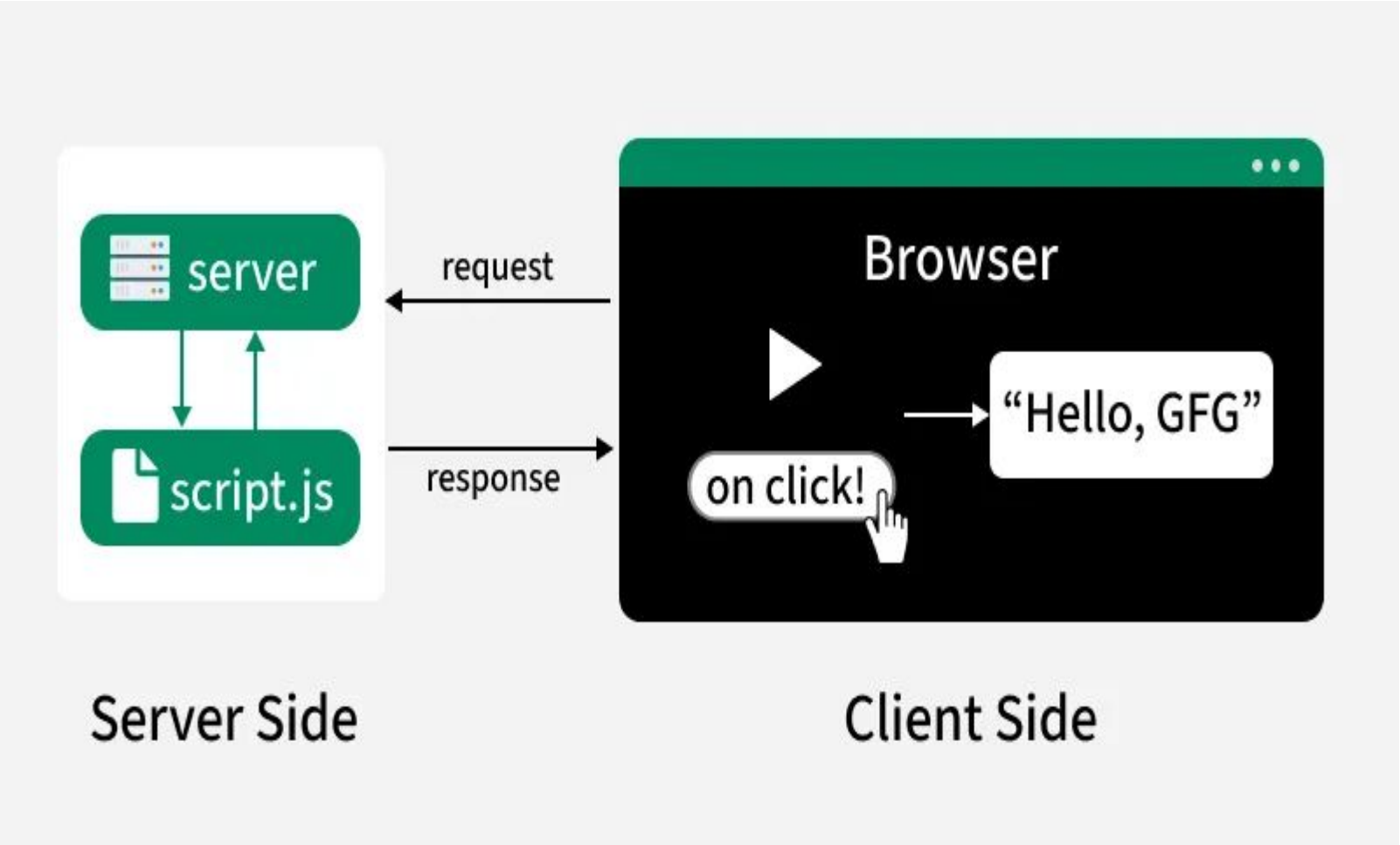
```
<html>
<head></head>
<body>
    <h1>Check the console for the message!</h1>
    <script>
        // This is our first JavaScript program
        console.log("Hello, World!");
    </script>
</body>
</html>
```

- The **<script> tag** is used to include JavaScript code inside an HTML document.
- **console.log()** prints messages to the browser's developer console. Open the browser console to see the "Hello, World!" message.

Key Features of JavaScript

- **Client-Side Scripting:** JavaScript runs on the user's browser, so has a faster response time without needing to communicate with the server.
- **Versatile:** Can be used for a wide range of tasks, from simple calculations to complex server-side applications.
- **Event-Driven:** Responds to user actions (clicks, keystrokes) in real-time.
- **Asynchronous:** It can handle tasks like fetching data from servers without freezing the user interface.
- **Rich Ecosystem:** There are numerous libraries and frameworks built on JavaScript, such as React, Angular, and Vue.js, which make development faster and more efficient.

Client Side and Server Side nature of JavaScript



Client-Side:

- Involves controlling the browser and its DOM (Document Object Model).
- Handles user events like clicks and form inputs.
- Common libraries include AngularJS, ReactJS, and VueJS.

Server-Side:

- Involves interacting with databases, manipulating files, and generating responses.
- Node.js and frameworks like Express.js are widely used for server-side JavaScript, enabling full-stack development.

Limitations of JavaScript

- **Security Risks** : Can be used for attacks like Cross-Site Scripting (XSS), where malicious scripts are injected into a website to steal data by exploiting elements like <img>, <object>, or <script> tags.
- **Performance** : Slower than traditional languages for complex tasks, but for simple tasks in a browser, performance is usually not a major issue.
- **Complexity** : To write advanced JavaScript, programmers need to understand core programming concepts, objects, and both client- and server-side scripting, which can be challenging.
- **Weak Error Handling and Type Checking** : Weakly typed, meaning variables don't require explicit types. This can lead to issues as type checking is not strictly enforced.

Syntax

- JavaScript is case-sensitive and uses Unicode.
- Statements usually end with a semicolon (;).
- Single-line comment:

  ⇒ // example
- Multi-line comment:

  ⇒ /* comment */

#Example1

```
// How to Declare variables:
let x = 5;
let y = 6;

// How to Compute values:
let z = x + y;
```

Placement of Script

```
<Html>
   <Head>
          <Title>  </Title>
           <Script>  JAVASCRIPT  </Script>
   </Head>
<Body>
        <Script>  JAVASCRIPT  </Script>
</Body>

</Html>
```

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Statements</h1>

<p>A JavaScript <b>program</b> is a list of <b>statements</b> to be executed by a computer.</p>

<p id="demo"></p>

<script>
let x, y, z;  // Statement 1
x = 5;        // Statement 2
y = 6;        // Statement 3
z = x + y;    // Statement 4

document.getElementById("demo").innerHTML = "The value of z is " + z;
</script>

</body>
</html>
```

Output1

JavaScript Statements

A JavaScript **program** is a list of **statements** to be executed by a computer.

The value of z is 11

#Example2

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Statements</h1>

<p>In HTML, JavaScript statements are executed by the browser.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello Dolly.";
</script>

</body>
</html>
```

Output2


JavaScript Statements
In HTML, JavaScript statements are executed by the browser.
Hello Dolly.

#Example3

```
<!DOCTYPE html>
<html>
<body>
<h1>My First JavaScript</h1>

<button type="button"
onclick="document.getElementById('demo').innerHTML =
Date()">
Click me to display Date and Time.</button>

<p id="demo"></p>

</body>
</html>
```

Output3


My First JavaScript
Click me to display Date and Time.
Thu Nov 13 2025 12:54:01 GMT+0530 (India Standard Time)

Variables

The JavaScript syntax defines two types of values:

- **Literals (Fixed values)**
- **Variables (Variable values**

- **A variable is like a container that holds data that can be reused or updated later in the program.**
- In JavaScript, variables are declared using the keywords var, let, or const.

- **Var:** The var keyword is used to declare a variable. It has a function-scoped or globally-scoped behaviour.

- **Let:** The let is introduced in ES6, has block scope and cannot be re-declared in the same scope.

- **const:** The const declares variables that cannot be reassigned. It's block-scoped as well.

#Example4

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Variables</h1>

<p>The underscore is treated as a letter in JavaScript names.</p>

<p id="demo"></p>

<script>
let _x = 2;
let $m  = 5;
let n = 3

document.getElementById("demo").innerHTML = _x + $m + n;
</script>

</body>
</html>
```

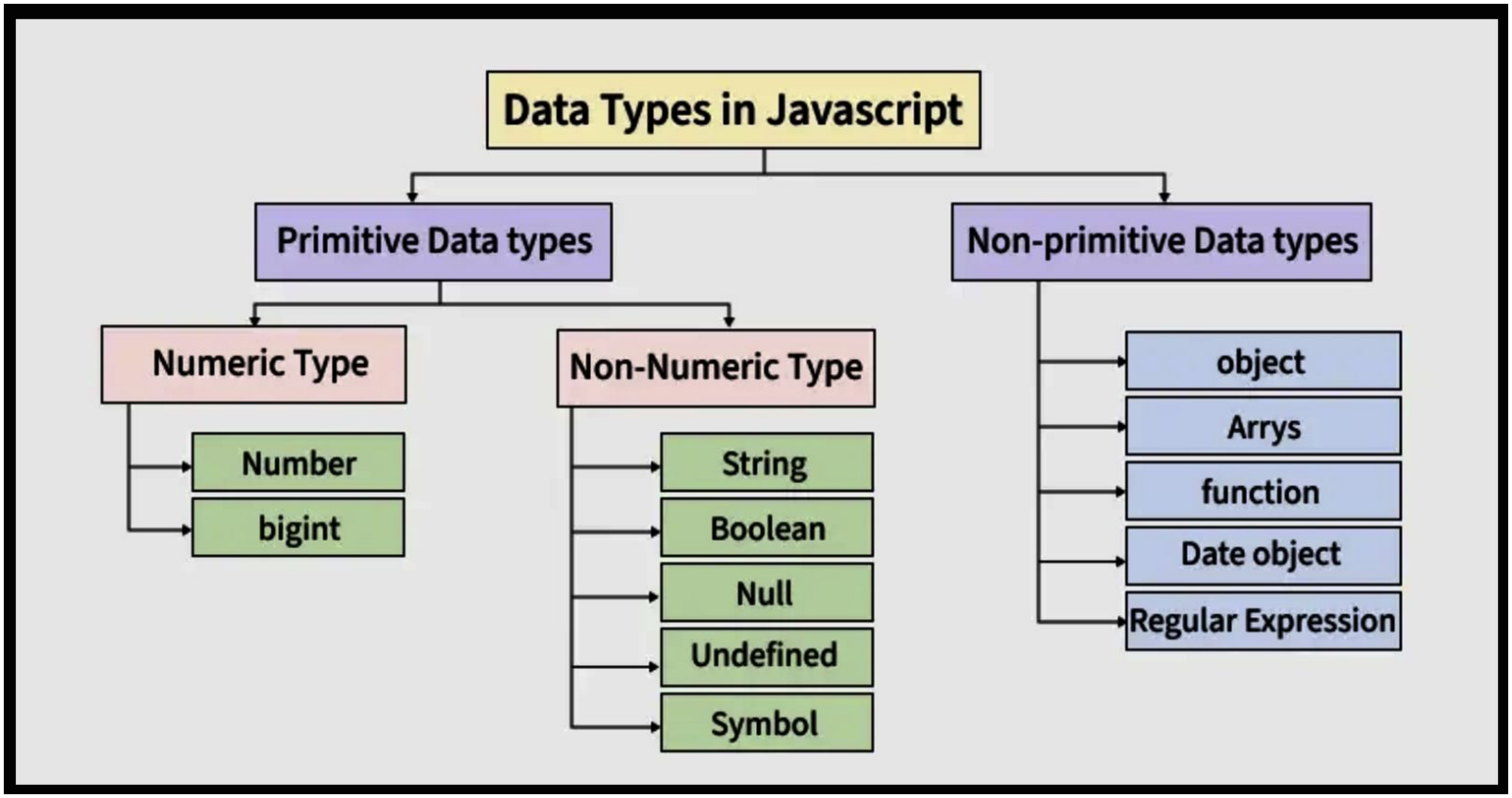
Output4

JavaScript Variables

The underscore is treated as a letter in JavaScript names.

7

Data Types


Data Types in Javascript
Primitive Data types
Non-primitive Data types
Numeric Type
Non-Numeric Type
Number
bigint
String
Boolean
Null
Undefined
Symbol
object
Arrys
function
Date object
Regular Expression

Type	Description
String	A text of characters enclosed in quotes
Number	A number representing a mathematical value
Bigint	A number representing a large integer
Boolean	A data type representing true or false
Object	A collection of key-value pairs of data
Undefined	A primitive variable with no assigned value
Null	A primitive value representing object absence
Symbol	A unique and primitive identifier

// **String**

```
let color = "Yellow";

let lastName = "Johnson";
```

// **Number**

```
let length = 16;

let weight = 7.5;
```

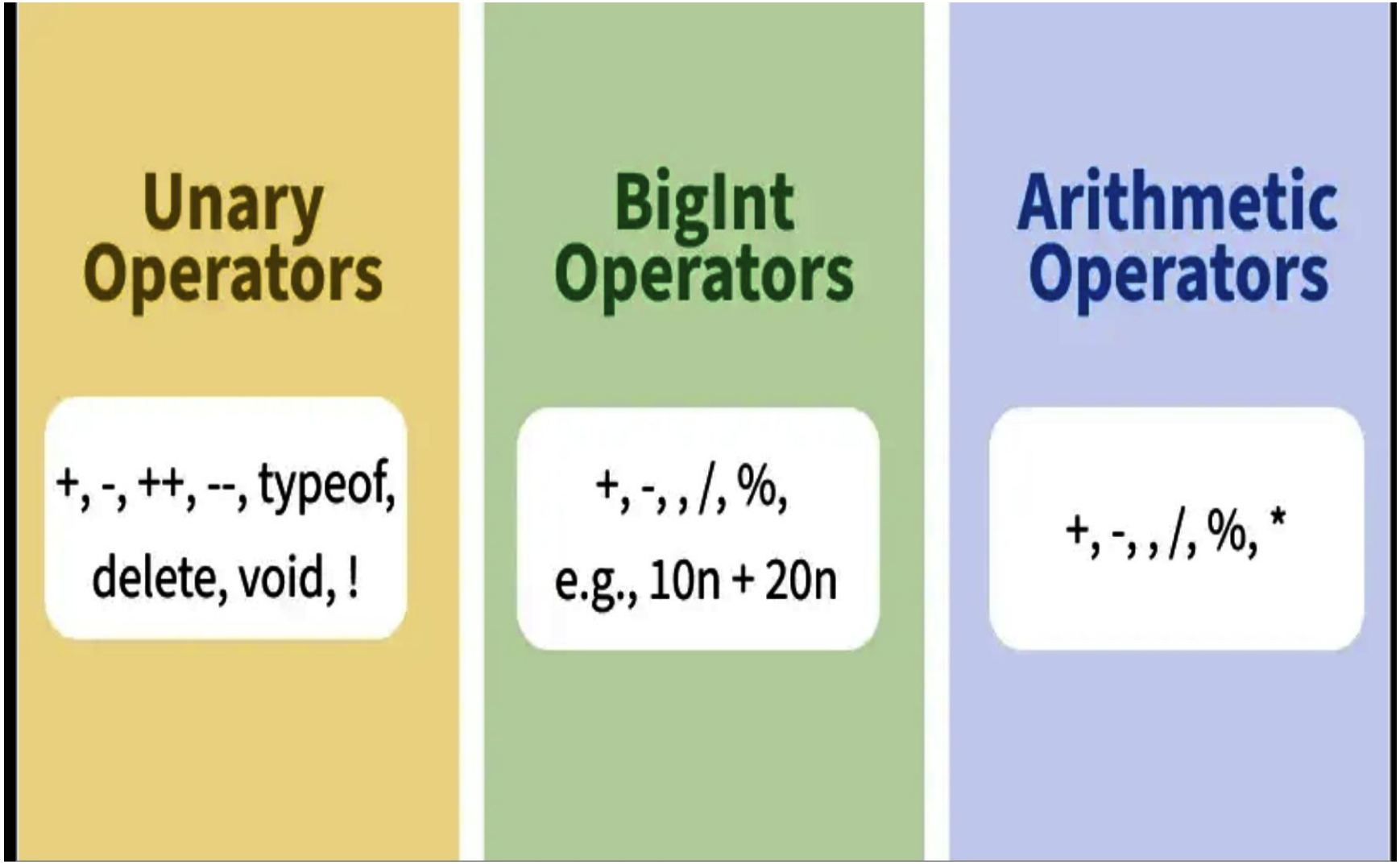
// **BigInt**

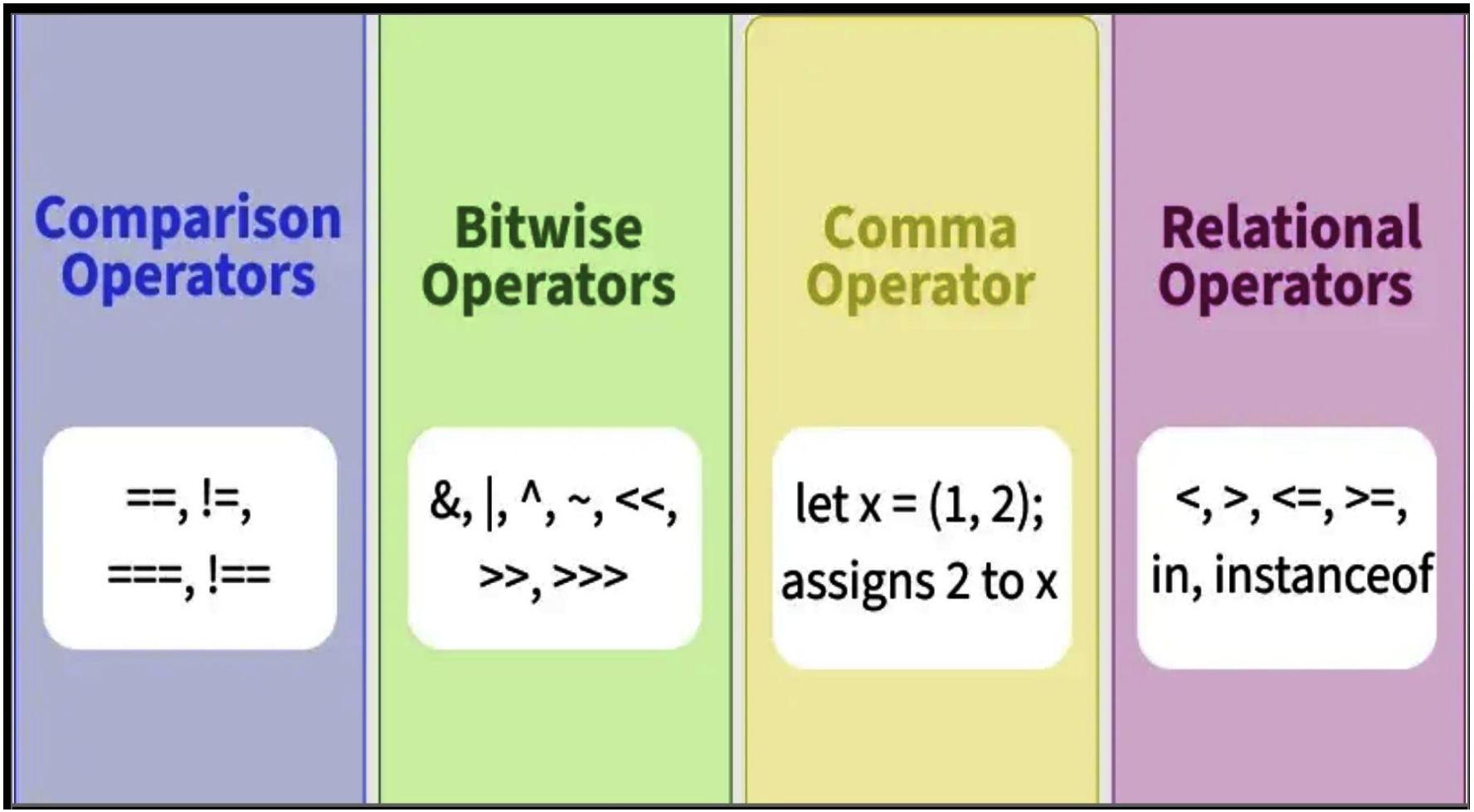
```
let x = 123456789012345678901234 5n;

let y = BigInt(123456789012345678901234 5)
```

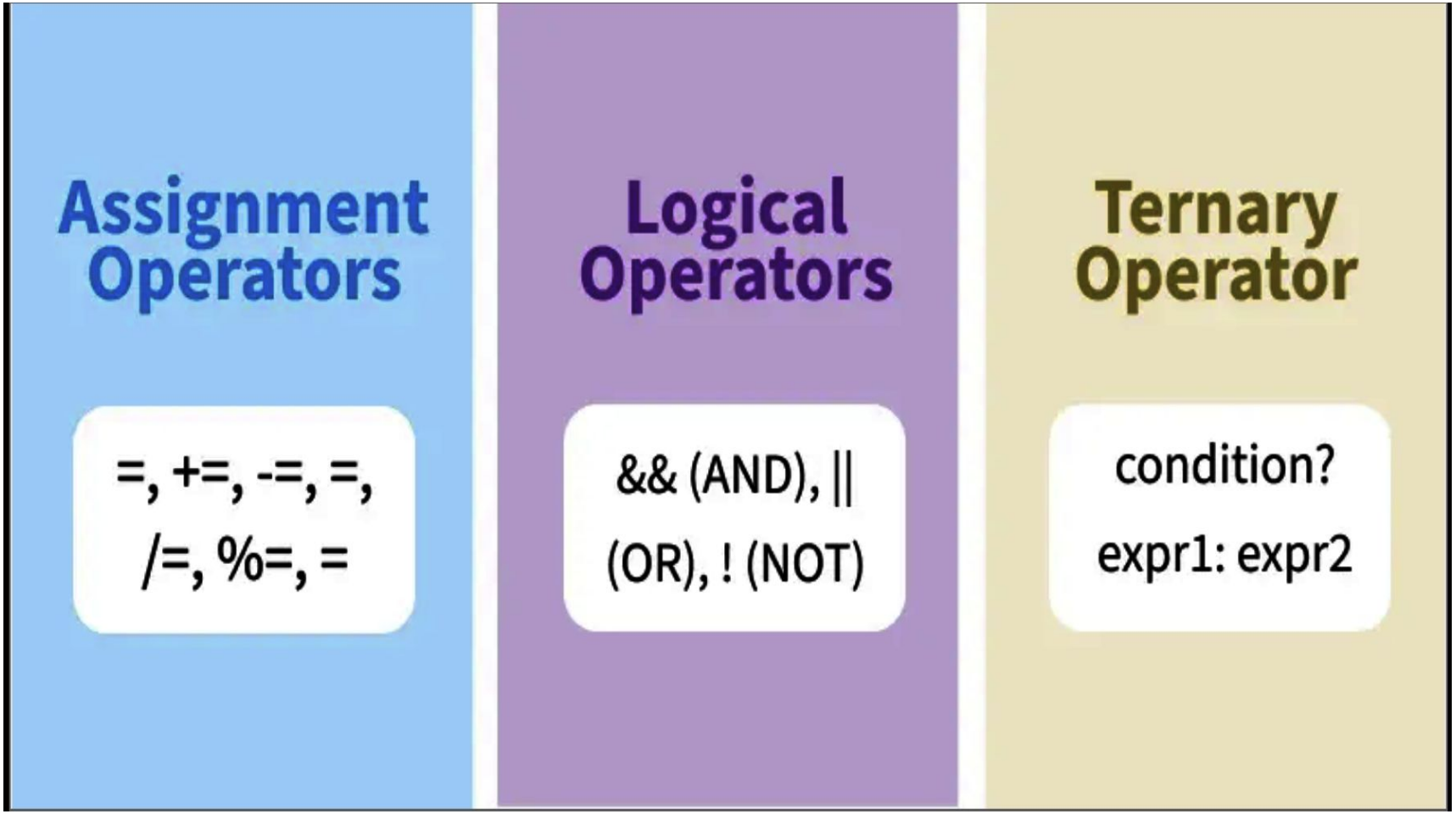
```
// Boolean

let x = true;

let y = false;

// Object

const person = {firstName:"John", lastName:"Doe"};

// Array object

const cars = ["Saab", "Volvo", "BMW"];

// Date object

const date = new Date("2022-03-25");
```

```
// Undefined

let x;

let y;

// Null

let x = null;

let y = null;

// Symbol

const x = Symbol();

const y = Symbol();
```

Operators


Unary Operators
+, -, ++, --, typeof, delete, void, !
BigInt Operators
+, -, , /, %, e.g., 10n + 20n
Arithmetic Operators
+, -, , /, %, *


Comparison Operators
==, !=,
===, !==
Bitwise Operators
&, |, ^, ~, <<,
>>, >>>
Comma Operator
let x = (1, 2);
assigns 2 to x
Relational Operators
<, >, <=, >=,
in, instanceof


Assignment Operators
=, +=, -=, =, /=, %=, =
Logical Operators
&& (AND), || (OR), ! (NOT)
Ternary Operator
condition? expr1: expr2

1. Arithmetic Operators:

perform mathematical calculations like addition, subtraction, multiplication, etc.

Example:

```
const sum = 5 + 3;        // Addition
const diff = 10 - 2;      // Subtraction
const p = 4 * 2;          // Multiplication
const q = 8 / 2;          // Division
console.log(sum, diff, p, q);
```

⇒ **8**
8
8
4

2. Assignment Operators:

Assignment operators are used to assign values to variables. They can also perform operations like addition or multiplication while assigning the value.

- = assigns a value to a variable.
- += adds and assigns the result to the variable.
- *= multiplies and assigns the result to the variable.

Example:

```
let n = 10;
n += 5;
n *= 2;
console.log(n);
```

⇒ **30**

3. Comparison Operators

Comparison operators compare two values and return a boolean (true or false). They are useful for making decisions in conditional statements.

- > checks if the left value is greater than the right.
- === checks for strict equality (both type and value).
- Other operators include <, <=, >=, and !==.

Example:

```
console.log(10 > 5);
console.log(10 === "10");
```

⇒ **true**

false

4. Logical Operators

Logical operators are mainly used to perform the logical operations that determine the equality or difference between the values.

- && returns true if both operands are true.
- || returns true if at least one operand is true.
- ! negates the boolean value.

Example:

```
const a = true, b = false;
console.log(a && b); // Logical AND
console.log(a || b);
```

⇒ **false**
true

5. Bitwise Operators

Bitwise operators perform operations on binary representations of numbers.

- & performs a bitwise AND.
- | performs a bitwise OR.
- ^ performs a bitwise XOR.
- ~ performs a bitwise NOT.

Example:

```
const res = 5 & 1; // Bitwise AND
console.log(res);
```

⇒ **1**

6.Ternary Operator

The ternary operator is a shorthand for conditional statements. It takes three operands.

Syntax: condition ? expression1 : expression2

evaluates expression1 if the condition is true, otherwise evaluates expression2.

Example:

```
const age = 18;
const status = age >= 18 ? "Adult" : "Minor";
console.log(status);
```

⇒ **Adult**

7. Comma Operator

Comma Operator (,) mainly evaluates its operands from left to right sequentially and returns the value of the rightmost operand.

- Each expression is evaluated from left to right.
- The final result of the expression is the rightmost value.

Example:

```
let n1, n2
const res = (n1 = 1, n2 = 2, n1 + n2);
console.log(res);
```

⇒ **3**

8. Unary Operators

Unary operators operate on a single operand (e.g., increment, decrement).

- ++ increments the value by 1.
- -- decrements the value by 1.
- typeof returns the type of a variable.

Example:

```
let x = 5;
console.log(++x); // Pre-increment
console.log(x--);
```

⇒ **6**

6

9. Relational Operators

JavaScript **Relational operators** are used to compare its operands and determine the relationship between them. They return a Boolean value (true or false) based on the comparison result.

- **in** checks if a property exists in an object.
- **instanceof** checks if an object is an instance of a constructor.

Example:

```
const obj = { length: 10 };
console.log("length" in obj);
console.log([] instanceof Array);
```

⇒ **true**
true

10. BigInt Operators:

BigInt operators allow calculations with numbers beyond the safe integer range.

- Operations like addition, subtraction, and multiplication work with BigInt.
- Use **n** suffix to denote BigInt literals.

Example:

```
const big1 = 12345678901234567890123456789n;
const big2 = 98765432109876543210987654321n;
console.log(big1 + big2);
```

⇒ **111111111011111111011111111110**

11. String Operators

JavaScript string operators include concatenation (+) and concatenation assignment (+=), used to join strings or combine strings with other data types.

- + concatenates strings.
- += appends to an existing string.

Example:

```
const s = "Hello" + " " + "World";

console.log(s);
```

⇒ **Hello World**

12. Chaining Operator (?.)

The optional chaining operator allows safe access to deeply nested properties without throwing errors if the property doesn't exist.

- **?.** safely accesses a property or method.
- Returns undefined if the property doesn't exist.

Example:

```
const obj = { name: "Aman", address: { city: "Delhi" } };

console.log(obj.address?.city);

console.log(obj.contact?.phone);
```

⇒ `Delhi`

`undefined`

Control Statements in JavaScript

JavaScript **control statement** is used to control the execution of a program based on a specific condition. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.

1. If Statement
2. If-else statement
3. switch statement
4. Conditional operator

1. If Statement

In this approach, we are using an if statement to check a specific condition, the code block gets executed when the given condition is satisfied.

Syntax

```
if ( condition_is_given_here ) {
   // If the condition is met,
   //the code  will get executed.
}
```

```
const num = 5;

if (num > 0) {
   console.log("The number is positive.");
};
```

2. If-else Statement

The if-else statement will perform some action for a specific condition. If the condition meets then a particular code of action will be executed otherwise it will execute another code of action that satisfies that particular condition.

Syntax

```
if (condition1) {
  // Executes when condition1 is true
  if (condition2) {
    // Executes when condition2 is true
  }
}
```

```
let num = -10;

if (num > 0)
   console.log("The number is positive.");
else
   console.log("The number is negative");
```

3. switch Statement

The switch case statement in JavaScript is also used for decision-making purposes. In some cases, using the switch case statement is seen to be more convenient than if-else statements.

Syntax

```
switch (expression) {
  case value1:
    statement1;
    break;
  case value2:
    statement2;
    break;
  case valueN:
    statementN;
    break;
  default:
    statementDefault;
}
```

```
let num = 5;

switch (num) {
   case 0:
      console.log("Number is zero.");
      break;
   case 1:
      console.log("Nuber is one.");
      break;
   case 2:
      console.log("Number is two.");
      break;
   default:
      console.log("Number is greater than 2.");
};
```

4. Conditional operator

The conditional operator, also referred to as the ternary operator (?:), is a shortcut for expressing conditional statements in JavaScript.

Syntax

condition ? value if true : value if false

```
let num = 10;

let result = num >= 0 ? "Positive" : "Negative";

console.log(`The number is ${result}.`);
```

LOOP Statements

1. For loop
2. While loop
3. Do while loop

1. For loop

In this approach, we are using for loop in which the execution of a set of instructions repeatedly until some condition evaluates and becomes false

Syntax

```
for (statement 1; statement 2; statement 3) {
   // Code here . . .
}
```

Example

```
for (let i = 0; i <= 10; i++) {
  if (i % 2 === 0) {
    console.log(i);
  }
};
```

⇒ **0 2 4 6 8 10**

2. While loop

The while loop repeats a block of code as long as a specified condition is true.

Syntax

```
while (condition) {
  // code block
}
```

Example

```
let i = 1;

while (i <= 5) {
    console.log(i);
    i++;
}
```

⇒**1 2 3 4 5**

3. Do-While loop

The do-while loop is similar to the while loop, except that the condition is evaluated after the execution of the loop's body. This means the code block will execute at least once, even if the condition is false.

Syntax

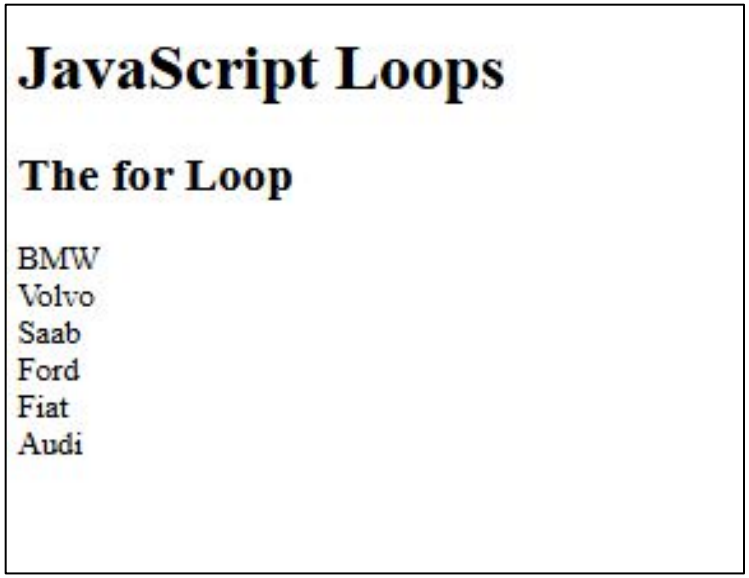
```
do {
   // code block
} while (condition);
```

Example

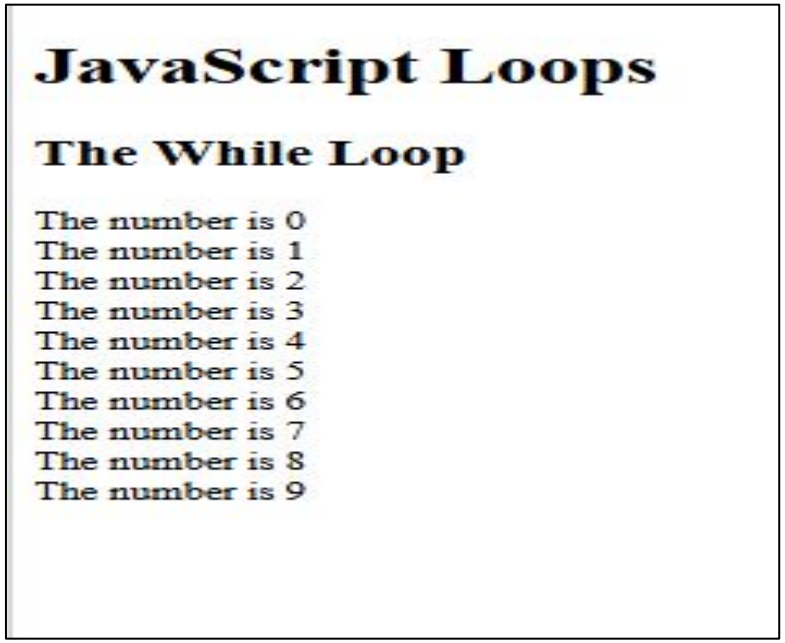
```
let i = 1;
do {
    console.log(i);
    i++;
} while (i <= 5);
```

⇒ **1 2 3 4 5**

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Loops</h1>
<h2>The for Loop</h2>

<p id="demo"></p>

<script>
const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat",
"Audi"];

let text = "";
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```



```
 <!DOCTYPE html>
<html>
<body>
<h1>JavaScript Loops</h1>
<h2>The While Loop</h2>

<p id="demo"></p>

<script>
let text = "";

let i = 0;
while (i < 10) {
  text += "The number is " + i + "<br>";
  i++;
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```



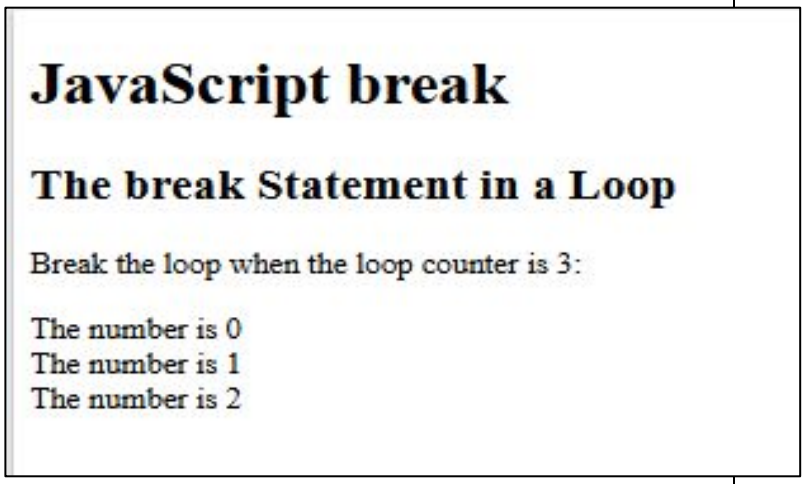
Break in Loops

- When break is encountered in a loop, the loop terminates immediately.
- The program control is transfered to the statements following the loop.
- No more loop iterations are executed.

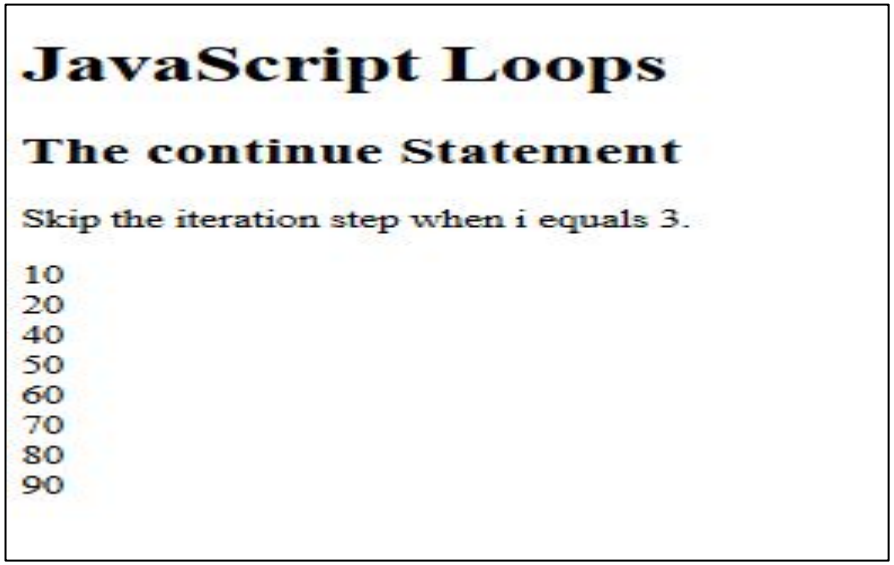
The Continue Statement

- The continue statement skips the current iteration in a loop.
- The remaining code in the iteration is skipped and processing moves to the next iteration.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript break</h1>
<h2>The break Statement in a Loop</h2>

<p>Break the loop when the loop counter is 3:</p>

<p id="demo"></p>

<script>
let text = "";

for (let i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```



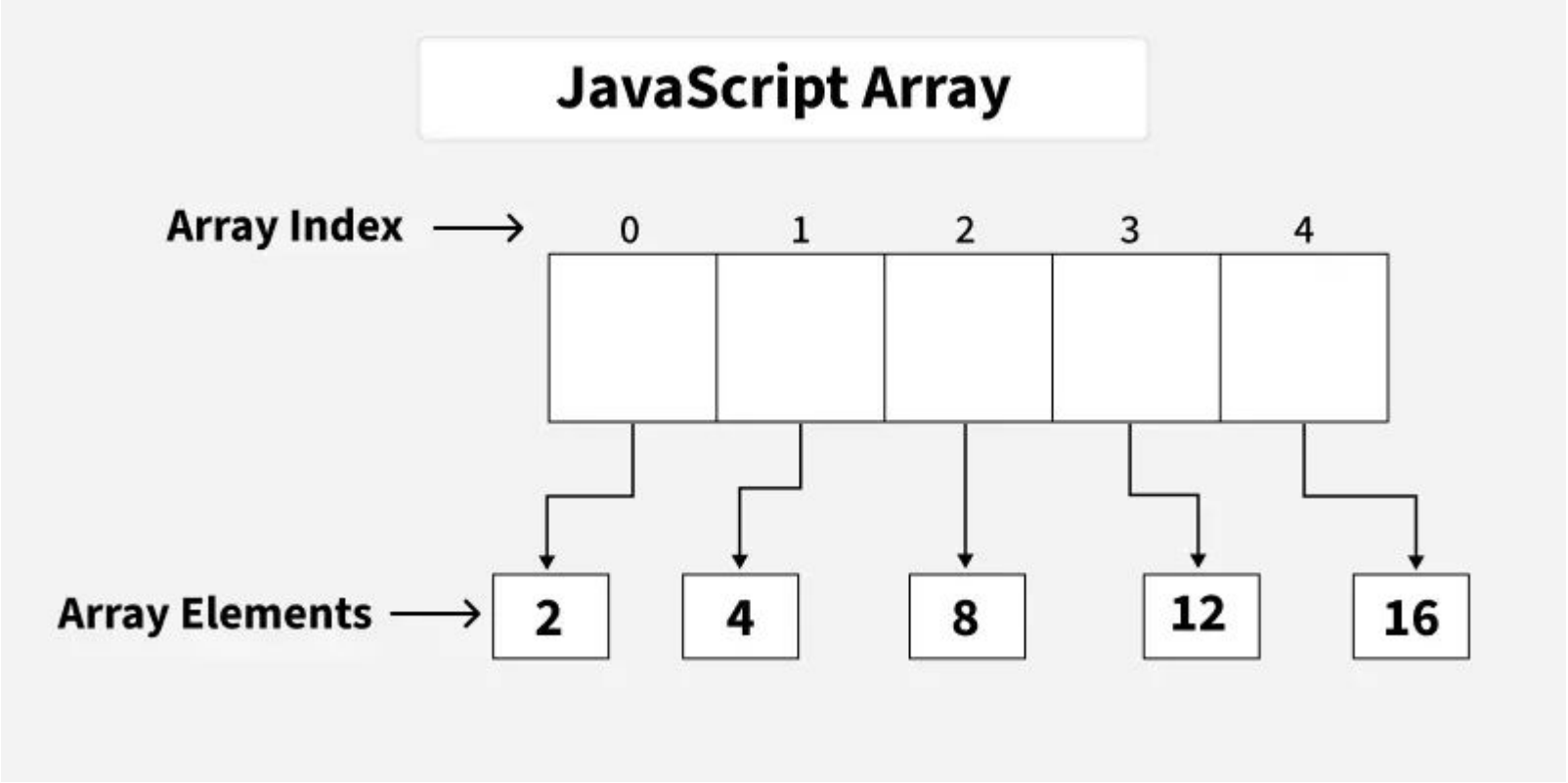
```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Loops</h1>
<h2>The continue Statement</h2>
<p>Skip the iteration step when i equals 3.</p>
<p id="demo"></p>
<script>
let text = "";
for (let i = 1; i < 10; i++) {
  if (i === 3) { continue; }
  text += i*10 + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```



JavaScript Arrays

In JavaScript, an array is an ordered list of values. Each value, known as an element, is assigned a numeric position in the array called its index. The indexing starts at 0, so the first element is at position 0, the second at position 1, and so on.

Arrays can hold any type of data-such as numbers, strings, objects, or even other arrays—making them a flexible and essential part of JavaScript programming.



1. Create Array using Literal

Creating an array using array literal involves using square brackets [] to define and initialize the array.

Example:

```
// Creating an Empty Array
let a = [];
console.log(a);

// Creating an Array and Initializing with Values
let b = [10, 20, 30];
console.log(b);
```

⇒ `[]`
`[ 10, 20, 30 ]`

2. Create using new Keyword (Constructor)

The "**Array Constructor**" refers to a method of creating arrays by invoking the Array constructor function.

Example:

```
// Creating and Initializing an array with values
let a = new Array(10, 20, 30);

console.log(a);
```

⇒ `[ 10, 20, 30 ]`

Functions

What are Functions?

- Functions are fundamental building blocks in all programming.
- Functions are reusable block of code designed to perform a particular task.
- Functions are executed when they are "called" or "invoked".

JavaScript Function Syntax

```
function name( p1, p2, ... ) {

  // code to be executed

  return

}
```

Functions are defined with the `function` keyword:

- followed by the function name
- followed by parentheses ( )
- followed by brackets { }

- The function name follows the naming rules for variables.
- Optional parameters are listed inside parentheses: (*p1, p2, ...*)
- Code to be executed is listed inside curly brackets: { }
- Functions can return an optional value back to the caller.

Function Invocation ()

The code inside the function will execute when "something" invokes (calls) the function:

- When it is invoked (called) from JavaScript code
- When an event occurs (a user clicks a button)
- Automatically (self invoked)

The () operator invokes a the function.

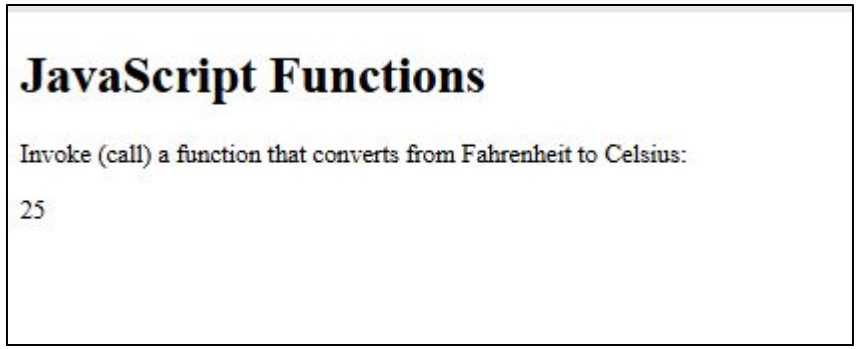
Example

toCelsius() invokes the toCelsius function:

```
// Convert Fahrenheit to Celsius:
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}


// Call the toCelcius() function
let value = toCelsius(77);
```

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Functions</h1>

<p>Invoke (call) a function that converts from Fahrenheit to Celsius:</p>

<p id="demo"></p>

<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}

let value = toCelsius(77);
document.getElementById("demo").innerHTML = value;
</script>

</body>
</html>
```



Note

In the examples above:

`toCelsius` refers to the function object.

`toCelsius()` refers to the function result.

Local Variables

- Variables declared within a JavaScript function, become LOCAL to the function.
- Local variables can only be accessed from within the function.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Functions</h1>

<p>Outside myFunction() carName is undefined.</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
let text = "Outside: " + typeof carName;
document.getElementById("demo1").innerHTML = text;

function myFunction() {
  let carName = "Volvo";
  let text = "Inside: " + typeof carName + " " + carName;
  document.getElementById("demo2").innerHTML = text;
}

myFunction();
</script>

</body>
</html>
```

Parameters vs. Arguments

- In JavaScript, function parameters and arguments are distinct concepts:
- Parameters are the names listed in the function definition.
- Arguments are the values received by the function.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Functions</h1>

<p id="demo"></p>

<script>
// name and age are parameters<br>
function greet(name, age) {
  return `Hello ${name}! You are ${age}
years old.`;
}

document.getElementById("demo").innerH
TML = greet("John", 21);
</script>

</body>
</html>
```

Arrow Functions

Arrow functions were introduced in ES6.(ECMAScript 2015)

Arrow functions allow us to write shorter function syntax:

Before Arrow:

Function to compute the product of a and b

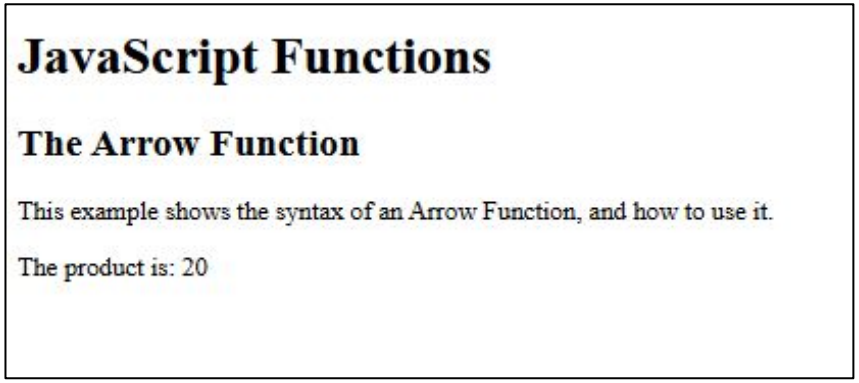
```
let myFunction = function(a, b) {return a * b}
```

With Arrow

```
let myFunction = (a, b) => a * b;
```

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Functions</h1>
<h2>The Arrow Function</h2>

<p>This example shows the syntax of an Arrow Function, and how to use it.</p>

<p id="demo"></p>

<script>
// myFunction computes the product of a and b
myFunction = (a, b) => a * b;

let result = myFunction(4, 5);
document.getElementById("demo").innerHTML = "The product is: " + result
</script>

</body>
</html>
```



References

- Goodman, D., Morrison, M., Novitski, P., Rayl, T. G., JavaScript Bible.
- https://www.w3schools.com/js/default.asp

Thank You