JavaScript

Laiba Imran

What is JavaScript?

JavaScript is a scripting language primarily used to enhance the interactivity of websites.

It allows developers to add dynamic elements to static HTML and CSS pages, including real-time updates, form validations, and animations.

JavaScript is dynamically typed

JavaScript can be used on both client-side (browser) and server-side (Node.js).

What is JavaScript?

JavaScript runs right inside the browser

JavaScript Engine: Each browser has a built-in JavaScript engine that interprets and executes JavaScript code. For example, Google Chrome uses the V8 engine.

What isn’t JavaScript?

It’s not Java

- Java: an object-oriented programming language used for developing mobile apps (Android), web applications (server-side), run on any platform that supports Java Virtual Machine (JVM).

  JS: a client-side scripting language to add interactivity to websites, although it's now used in server-side programming with environments like Node.js. Runs in browser.

Java: Strongly typed and uses explicit declarations of variables and types. Object-oriented, and everything must be part of a class.

JS: Weakly typed, more flexible with types, and no need for explicit declarations. Supports both functional and object-oriented programming styles.

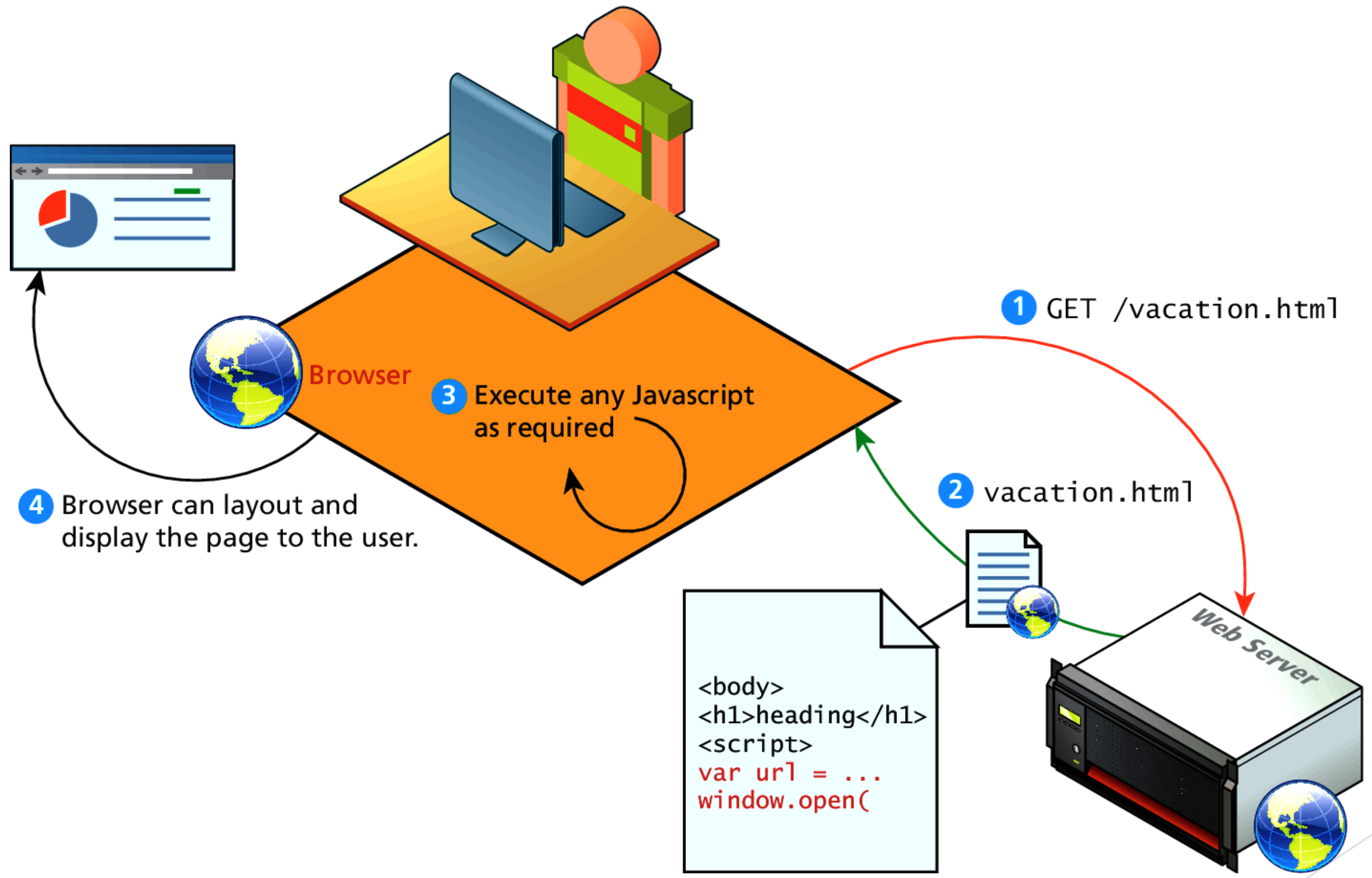
Java: Compiled language. Java code is compiled into bytecode, which is executed by the JVM.

JS: Interpreted language. JavaScript is typically executed by the browser in real-time.

JavaScript is to Java like carpet is to car.

Client-Side Scripting

Let the client compute

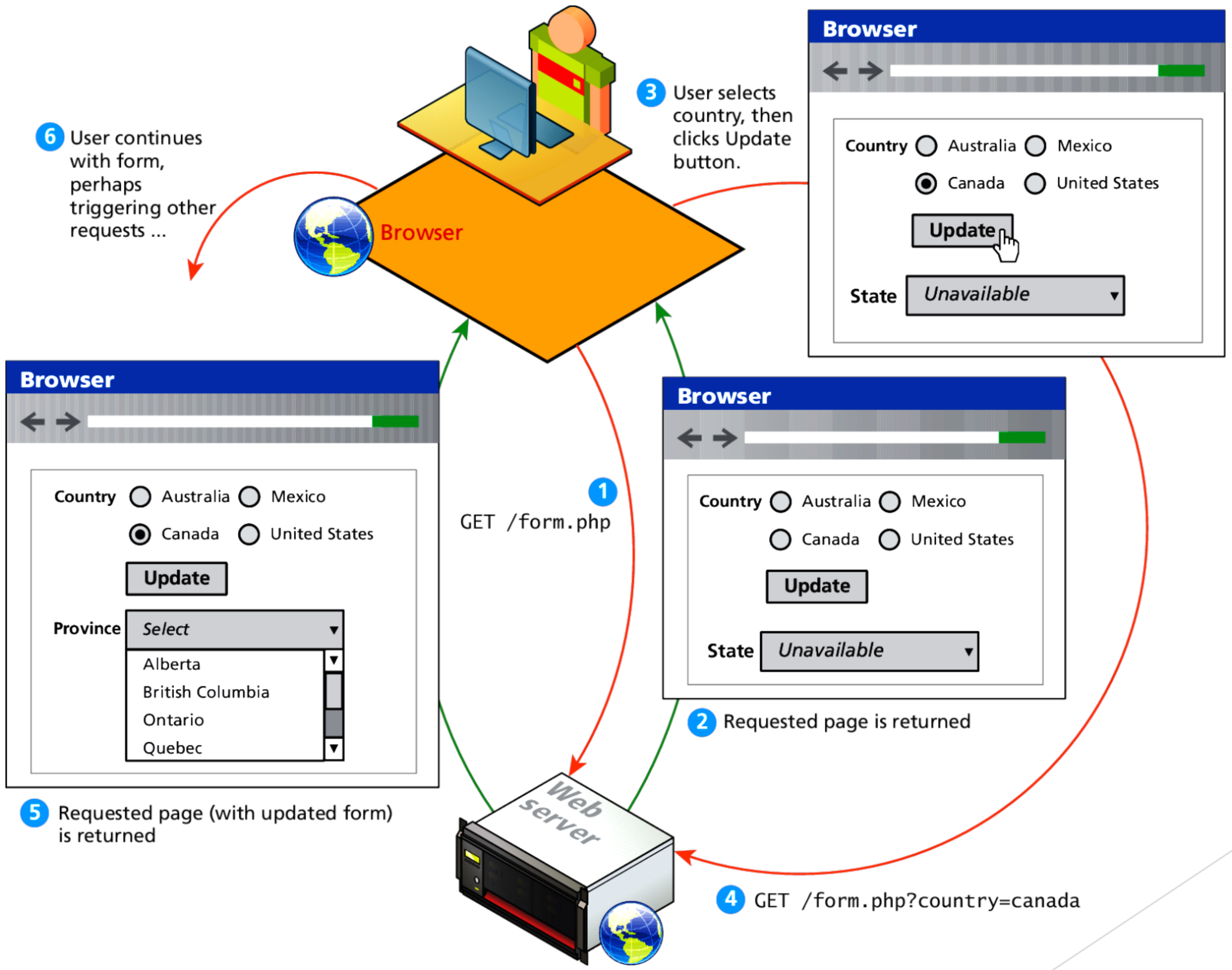


Browser
3 Execute any Javascript as required
4 Browser can layout and display the page to the user.
1 GET /vacation.html
2 vacation.html
Web Server

```
<body>
<h1>heading</h1>
<script>
var url = ...
window.open(
```

HTTP request-response loop

Without JavaScript



Client-Side Scripting

It's good

There are many advantages of client-side scripting:

- Processing can be offloaded from the server to client machines, thereby reducing the load on the server.
- The browser can respond more rapidly to user events than a request to a remote server ever could, which improves the user experience.

Client-Side Scripting

There are challenges

The disadvantages of client-side scripting are mostly related to how programmers use JavaScript in their applications.

- ➢ There is no guarantee that the client has JavaScript enabled
- ➢ What works in one browser, may generate an error in another.

JavaScript Placement

Inline JavaScript: Written directly inside HTML elements (not recommended for best practices).

```
<button onclick="alert('Hello!')">Click me</button>
```

Embedded JavaScript: Written inside <script> tags within an HTML document.

```
<script> alert("Hello, World!"); </script>
```

External JavaScript: Stored in separate .js files and linked to an HTML document.

```
<head>
    <script type="“text/JavaScript”" src="script.js">
    </script>
</head>
```

Variables

- var: Has function-level scope (older way of declaring variables).
- let: Has block-level scope (introduced in ES6) and is preferred over var.
- const: Has block-level scope (introduced in ES6) and is used to declare constants whose values cannot be changed after initialization.

```
var x = 10;   // Can be updated
let y = 20;   // Block-scoped, can be updated
const z = 30; // Block-scoped, cannot be updated
```

Data Types

String: Sequence of characters.

```
let message = "Hello, World!";
```

Number: Represents both integers and floating-point numbers.

```
let age = 25;
```

Boolean: Represents true or false.

```
let isAdult = true;
```

Undefined: A variable that has been declared but not initialized.

```
let name; console.log(name); // Output: undefined
```

Null: Represents an intentional absence of any object value

```
let emptyValue = null;
```

Exercise

Declare variables for their name, age, and semester and display.

Operators and Expressions

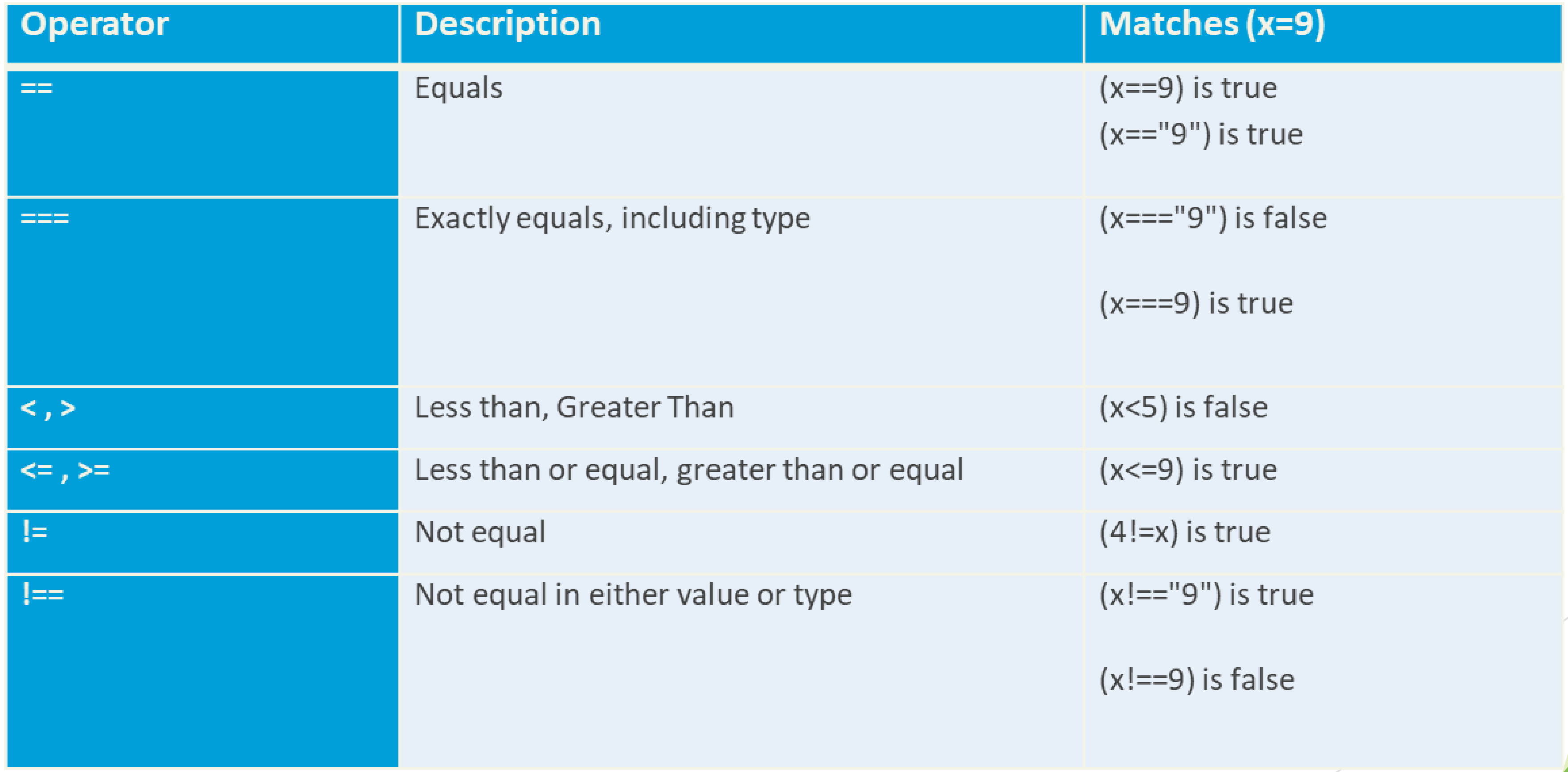
Arithmetic Operators:

Used for basic math operations.

```
let sum = 10 + 5; // 15
let difference = 10 - 5; // 5
let product = 10 * 5; // 50
let quotient = 10 / 5; // 2
let remainder = 10 % 3; // 1
```

Operators and Expressions

Comparison Operators:



Operator	Description	Matches (x=9)
==	Equals	(x==9) is true (x=="9") is true
===	Exactly equals, including type	(x==="9") is false (x===9) is true
< , >	Less than, Greater Than	(x<5) is false
<= , >=	Less than or equal, greater than or equal	(x<=9) is true
!=	Not equal	(4!=x) is true
!==	Not equal in either value or type	(x!=="9") is true (x!==9) is false

Exercise

Create a calculator that performs addition, subtraction, multiplication, and division and display.

Conditionals

if-else statements:

Used to execute code based on a condition.

```
let age = 18;
if (age >= 18)
{
    console.log("You are an adult.");
}
else {
    console.log("You are a minor.");
}
```

Conditionals

Switch Statements:

An alternative to if-else for handling multiple conditions.

```
let color = "red";
switch (color) {
case "red":
    console.log("The color is red.");
break;
case "blue":
    console.log("The color is blue.");
break;
default:
    console.log("Unknown color.");
}
```

Conditionals

Ternary Operator:

A shorthand for if-else statements.

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

Exercise

Write a program that checks if a number is positive, negative, or zero and display.

Loops

For loop:

Used to repeat a block of code a specific number of times.

```
let fruits = ["apple", "banana", "cherry"];
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
```

For/of loop (ES6):

```
let fruits = ["apple", "banana", "cherry"];
  for (const fruit of fruits) {
            console.log(fruit);
}
```

Loops

While loop:

Executes a block of code as long as the condition is true.

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

Loops

Do-while loop:

Executes the code block at least once, and then checks the condition.

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

Exercise

Write a program that calculate the sum of numbers from 1 to 100 and display.

Functions

A function is a block of reusable code that performs a specific task. Functions help organize code, reduce redundancy, and improve readability.

They are defined by using the reserved word function and then the function name and (optional) parameters.

Since JavaScript is dynamically typed, functions do not require a return type, nor do the parameters require type.

```
function greet() {
        console.log("Hello, world!");}
 greet(); // Calling or invoking the function
```

Parameters and Return Values

```
function add(a, b) {
  return a + b;
}
let result = add(5, 10); // 15
console.log(result);
```

Anonymous Functions

```
const add = function(a, b) {
    return a + b;
};
console.log(add(2, 3)); // Output: 5
```

Arrow Functions (ES6)

Arrow functions provide a shorter syntax for writing anonymous functions and are often used in modern JavaScript.
They also handle this keyword differently than traditional functions.

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


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

Arrow Functions (ES6)

One of the most important differences between regular functions and arrow functions is that arrow functions do not have their own this context. Instead, they inherit this from the surrounding (lexical) context.
Example of Arrow Function with this

Regular Function (with this binding):

```
const person = {
        name: "John",
        greet: function() {
                console.log(this.name);  // `this` refers to the `person` object
        }
        };

person.greet();  // Output: John
```

Arrow Functions (ES6)

Arrow Function (no this binding):

```
const person = {
        name: "John",
        greet: () => {
        console.log(this.name);  // `this` does NOT refer to the `person` object }
        };

person.greet();  // Output: undefined
```

Exercise

Write a program that takes two numbers and returns their sum and display using both type of functions.

JavaScript Objects

- Objects can have constructors, properties, and methods associated with them.
- Some objects are included in the JavaScript language; you can also define your own kind of objects.

```
const car = {
          type:"Fiat",
          model:"500",
          color:"white“
  };
```

Constructors

Normally to create a new object we use the new keyword, the class name, and ( ) brackets as follows:

```
// Create an Object
const person = new Object();

// Add Properties
person.firstName = "John";
person.age = 50;
person.eyeColor = "blue";
```

Properties

Each object might have properties that can be accessed, depending on its definition.

When a property exists, it can be accessed using dot notation where a dot between the instance name and the property references that property.

```
const car = {
          type:"Fiat",
          model:"500",
          color:"white“
   };
console.log(car.type, car.model); // Fiat 500
```

Methods

Objects can also have methods, which are functions associated with an instance of an object.
These methods use the same dot notation as for properties, but instead of accessing a variable, we are calling a method.

```
const person = {
  name: "John",
   age: 15,
   greet : function() {
            console.log(`Hello, my name is ${this.name} and I am ${this.age}
years old.`);
  };
}
person.greet();  // Output: Hello, my name is Jack and I am 10 years old.
```

Exercise

Write a program that creates an object representing themselves (name, age, city) and write a method that prints a personalized greeting.

Objects Included in JavaScript

A number of useful objects are included with JavaScript including:

- Array
- Boolean
- Date
- Math
- String
- Dom objects

Arrays

Arrays are one of the most used data structures.
Here is its syntax:

```
var fruits = new Array(); //empty
var fruits = new Array(“apple”, “banana”);

let fruits = ["apple", "banana", "cherry"];
console.log(fruits[0]); // apple
```

Arrays Operations

push(): Adds an item to the end of the array.

```
fruits.push("orange");
console.log(fruits); // ["apple", "banana", "cherry", "orange"]
```

pop(): Removes the last item from the array.

```
fruits.pop();
console.log(fruits); // ["apple", "banana", "cherry"]
```

Arrays Operations

```
cars.length()   // Returns the number of elements
cars.sort()   // Sorts the array
```

Arrays Loops

You can loop through an array using a for loop or forEach method**.**

```
let fruits = ["apple", "banana", "cherry"];
for (let i = 0; i < fruits.length; i++)
{
        console.log(fruits[i]);
}

let fruits = ["apple", "banana", "cherry"];
fruits.forEach( (fruit) => console.log(fruit.toUpperCase()));
```

Exercise

Write a program that creates an array of their favorite cars and write a loop that prints each car in lowercase.