UNIT- II

Unit -II		COs	Hrs.	SEE Marks
Unit Title	**JavaScript**			
Introduction to JavaScript: what is the JavaScript and benefits of the language,		2	2	25%
JavaScript language syntax, Variable declaration, Operators, Control Statements,		2	5	
Error Handling, Understanding arrays, Function Declaration. Built in Functions,		2	5	
Standard Date and Time Functions in java script.		2	1	

1. What is JavaScript? What are the Key Aspects of JavaScript?

JavaScript is a high-level, dynamic programming language that is primarily used to create interactive and responsive web pages and web applications. It is one of the technologies of the web, alongside **HTML** and **CSS**. While HTML provides the structure of a web page and CSS controls the layout and appearance, JavaScript adds **interactivity** and **functionality**, to build the engaging and responsive web pages.

Key Characteristics of JavaScript:

- **Interpreted and lightweight:** No need for compilation; it runs directly in the browser.
- **Object-oriented and event-driven:** Allows for modular, reusable code and reacts to user interactions.
- **Prototype-based:** Supports inheritance through prototypes, rather than traditional classes.
- **Multi-paradigm:** Supports functional, procedural, and object-oriented programming styles.

2. List the advantages and disadvantages of using JavaScript.

Advantages of JavaScript

1. **Client-Side Execution**

   JavaScript runs directly in the browser, reducing server load and improving response time.

2. **Fast Performance**

   Since it's executed on the client side, operations are generally faster.

3. **Rich Interface Features**

   Supports features like drag-and-drop, sliders, and real-time updates, enhancing user interactivity.

4. **Versatility**

   Can be used for both front-end (with HTML/CSS) and back-end (with Node.js) development.

5. **Large Community & Support**

   A huge ecosystem with many libraries, frameworks (like React, Angular,), and community support.

6. **Easy to Learn**

   Simple syntax and widespread use make it beginner-friendly.

Disadvantages of JavaScript

1. **Security Issues**

   Code is visible to users and can be exploited or misused.

2. **Browser Compatibility**

   Some features may work differently or not at all in certain browsers, causing inconsistencies.

3. **Client-Side Dependency**

   If JavaScript is disabled in the user's browser, functionality may break.

4. **Debugging Can Be Tricky**

Though tools exist, debugging JavaScript is sometimes harder compared to compiled languages.

5. **Performance Limits**

   Not suitable for very heavy computation tasks compared to languages like C++ or Java.

3. What is a variable? List and explain the different ways to declare a variable in JavaScript.

A **variable** is a named container used to store data values. In **JavaScript**, variables can hold different types of data such as numbers, strings, objects, etc. The value stored in a variable can be changed during the execution of the program (except when declared with const).

Ways to Declare a Variable in JavaScript

JavaScript provides **three** main ways to declare variables:

1. var

- ✓ Introduced in earlier versions of JavaScript.
- ✓ Function-scoped.
- ✓ Can be **re-declared** and **updated** within the same scope.

Syntax:

```
var x = 10;
```

Example:

```
function example() {
  var a = 5;
  var a = 10; // Allowed
  document.write(a); // 10
}
```

2. let

- ✓ Block-scoped (limited to {} block).
- ✓ Can be updated but not re-declared in the same scope.
- ✓ Preferred over var for most use cases.

Syntax:

let y = 20;

Example:

```
let b = 10;
// let b = 20; // Error: Identifier 'b' has already been declared
b = 30; // Allowed
```

---

3. const

- ✓ Block-scoped.
- ✓ Cannot be re-declared or updated.
- ✓ Must be initialized at the time of declaration.
- ✓ The value itself can't be changed for primitives, but objects/arrays declared with const can be modified internally.

Syntax:

const z = 30;

Example:

```
const PI = 3.14;
// PI = 3.1415; // Error
```

Declaration	Scope	Re-declarable	Updatable	Hoisted
var	Function	Yes	Yes	Yes
let	Block	No	Yes	No
const	Block	No	No	No

4. Explain the following methods of JavaScript with their purpose, syntax, and a neat diagram:

a. alert()

b. prompt()

c. confirm()

a) alert()

Purpose: Displays a popup message to the user with an **OK** button. It is used to show **information or warnings**.

Syntax:

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

b) prompt()

Purpose: Displays a dialog box that asks the user for input. It has a text field, and OK/Cancel buttons.

Syntax:

```
let name = prompt("Enter your name:");
```

Output:

A dialog box with an input field appears:

Note: Returns:

- ✓ The text entered by the user (as a string)
- ✓ null if the user clicks **Cancel**

c) confirm()

Purpose: Displays a confirmation dialog box with OK and Cancel buttons. It is used to confirm user actions.

Syntax:

```
let result = confirm("Are you sure you want to delete?");
```

Output:

A dialog box appears:

Returns:

- ✓ true if the user clicks OK
- ✓ false if the user clicks Cancel

5. Explain the different ways to link JavaScript to an HTML page.

1. Inline JavaScript

JavaScript code is written directly inside an HTML element's attribute using the on event (like onclick, onmouseover, etc.).

Example:

```
<button onclick="alert('Button Clicked!')">Click
Me</button>
```

2. Internal JavaScript

JavaScript code is written within a <script> tag inside the <head> or <body> section of the HTML document.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <script>
    function greet() {
      alert("Hello from internal script!");
    }
  </script>
</head>
<body>
  <button onclick="greet()">Greet</button>
</body>
</html>
```

---

3. External JavaScript

JavaScript code is written in a separate .js file and linked to the HTML page using the <script src="..."> tag.

Example:

script.js

```
function greet() {
  alert("Hello from external file!");
}
```

index.html

```
<!DOCTYPE html>
<html>
<head>
```

```
  <script
type=”text/javascript”src="script.js"></script>
</head>
<body>
  <button onclick="greet()">Greet</button>
</body>
</html>
```

6. Discuss the data types in JavaScript in detail.

In JavaScript, data types define the kind of data a variable can hold. JavaScript is a **dynamically typed** language, which means variables can hold values of any data type and the type can change at runtime.

1. Primitive Data Types

a) Number

- ✓ Represents both integer and floating-point numbers.
- ✓ Includes special values: Infinity, -Infinity, and NaN (Not-a-Number).

```
let a = 10;
let b = 3.14;
let c = NaN;
```

b) String

- ✓ Represents a sequence of characters.
- ✓ Enclosed in single quotes (' '), double quotes (" "), or backticks ( ).

```
let name = "John";
let greeting = `Hello, ${name}`; // Template literal
```

c) Boolean

- ✓ Has only two values: true or false.
- ✓ Used in conditions and logical operations.

```
let isActive = true;
let isLoggedIn = false;
```

d) Undefined

✓ A variable that is declared but not assigned any value.

```
let x;
document.write(x); // undefined
```

e) Null

✓ Represents the intentional absence of any object value.

```
let data = null;
```

f) Symbol

✓ Represents a unique and immutable value, mainly used for object keys.

✓ Example:

```
let sym = Symbol("id");
```

g) BigInt

✓ Used to represent integers larger than 2^53 - 1.

✓ Example:

```
let big = 12345678901234567890123456789n;
```

2. Non-Primitive (Reference) Data Types

✓ These types **store references** to memory locations.

a) Object

✓ Collection of key-value pairs.

Example:

```
let person = { name: "Alice", age: 30 };
```

b) Array

✓ Special type of object used to store ordered collections.

Example:

```
let colors = ["red", "green", "blue"];
```

c) Function

- ✓ A block of code that can be reused.
- ✓ Functions in JavaScript are also objects.

Example:

```
function greet() {
  console.log("Hello!");
}
```

d) Date, RegExp, Map, Set, etc.

- ✓ JavaScript also provides built-in object types like:
- ✓ Date: for date/time
- ✓ RegExp: for regular expressions
- ✓ Map / Set: for collections of unique values or key-value pairs

7. Explain the various types of operators in JavaScript.

1. JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values. Given that **y=5**, the table below explains the arithmetic operators.

Operator	Description	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (Remainder)	a % b
**	Exponentiation	a ** b
++	Increment	a++ or ++a
--	Decrement	a-- or --a

2. JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables. Given that x=10 and y=5, the table below explains the assignment

operators

Operator	Description	Example
=	Assign	x = 10
+=	Add and assign	x += 5 → x = x + 5
-=	Subtract and assign	x -= 5
*=	Multiply and assign	x *= 5
/=	Divide and assign	x /= 5
%=	Modulus and assign	x %= 5

3. Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values

Given that x=5, the table below explains the comparison operators:

Operator	Description	Example
==	Equal to (type conversion)	5 == '5' → true
===	Strict equal (no conversion)	5 === '5' → false
!=	Not equal	5 != 3 → true
!==	Strict not equal	5 !== '5' → true
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

4. Logical Operators

Logical operators are used to determine the logic between variables or values. Given that x=6 and y=3, the table below explains the logical operators

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false

!	not	!(x==y) is true

5. Bitwise Operators

Operator	Description	Example
&	AND	a & b
`	`	OR
^	XOR	a ^ b
~	NOT	~a
<<	Left shift	a << 2
>>	Right shift	a >> 2

6. Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

Var iablename=(condition)?value1:value2

8. Explain the decision statements.

Decision-making statements are used to execute a particular block of code based on certain conditions.

I. **if statement** - use this statement if you want to execute some code only if a specified condition is true.

Syntax:

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

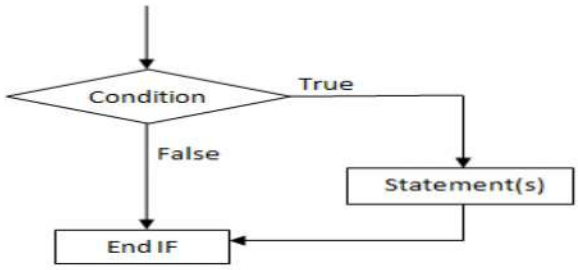


fig: Flowchart for if statement

```
}
```

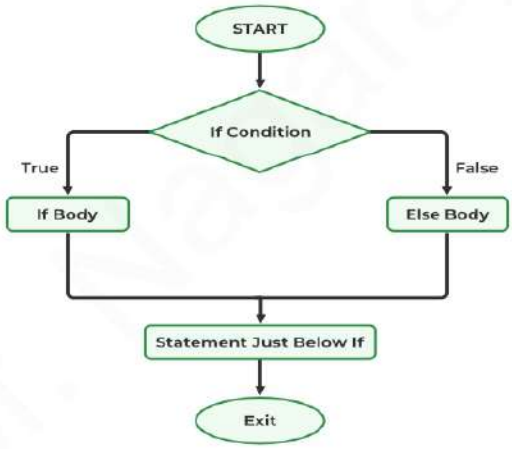
Example:

```
let age = 18;
if (age >= 18) {
  document.write("You are eligible to vote.");
}
```

II. **if...else statement** - use this statement if you want to execute some code if the condition is true and another code if the condition is false

Syntax:

```
if (condition) {
  // code if condition is true
} else {
  // code if condition is false
}
```



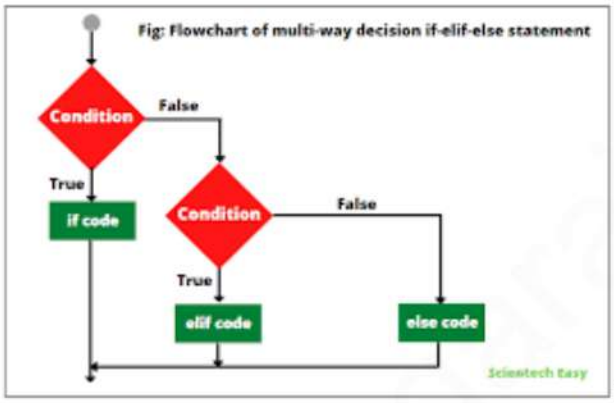
Example:

```
let age = 16;
if (age >= 18) {
  document.write ("You are eligible to vote.");
} else {
  document.write ("You are not eligible to
vote.");
}
```

III. **if...else if ....else statement** - use this statement if you want to select one of many blocks of code to be executed

Syntax:

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



Example:

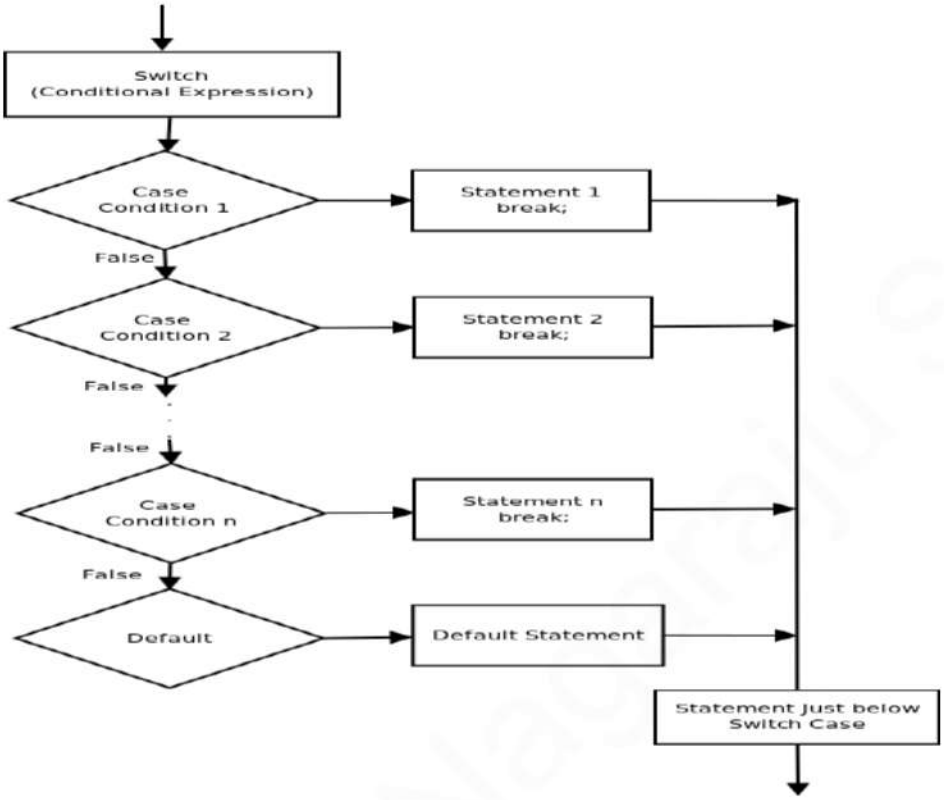
```
let marks = 85;
if (marks >= 90) {
  document.write ("Grade A");
} else if (marks >= 75) {
  document.write ("Grade B");
} else {
  document.write ("Grade C");
  }
```

IV. **switch statement** - use this statement if you want to select one of many blocks of code to be executed

Syntax:

```
switch(expression) {
  case value1:
    // code block
    break;
  case value2:
```

```
    // code block
    break;
  default:
    // default code block
}
```



Example:

```
let day = 3;
switch(day) {
  case 1:
    document.write ("Monday");
    break;
  case 2:
    document.write ("Tuesday");
    break;
  case 3:
    document.write ("Wednesday");
    break;
  default:
    document.write ("Invalid day");
}
```

9. Explain the looping statements.

a) for Loop

The **for loop** in JavaScript is used to repeat a block of code **a specific number of times**. It is one of the most commonly used looping statements in programming.

Syntax:

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

Explanation of the parts:

- ✓ **Initialization** – Runs once before the loop starts. It is typically used to declare and set a loop counter.
- ✓ **Condition** – Evaluated before every loop iteration. The loop continues as long as this condition is true.
- ✓ **Increment/Decrement** – Executed at the end of each loop iteration. It updates the loop counter.

Example:

```
for (let i = 1; i <= 5; i++) {
  document.write(i);
}
```

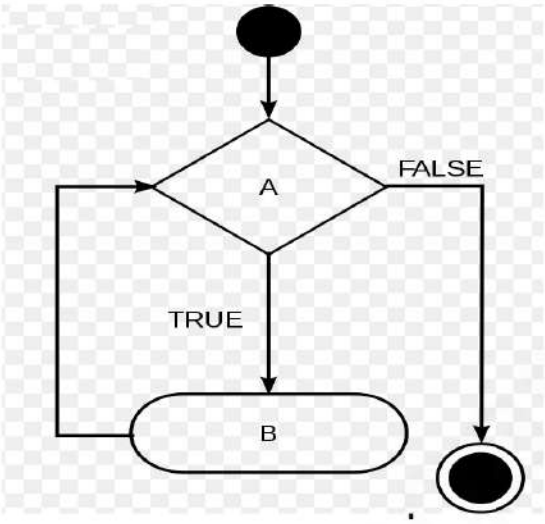
b) while Loop

The while loop in JavaScript is used when you want to repeat a block of code as long as a specified condition is true. It is a pre-test loop, meaning the condition is checked before each iteration.

Syntax:

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

- condition: A Boolean expression. If true, the loop continues; if false, the loop stops.
- The code block will only run **if the condition is true**.



Example:

```
let i = 1;
while (i <= 5) {
  document.write(i);
  i++;
}
```

```
let input = "";
while (input !== "yes") {
  input = prompt("Type 'yes' to continue:");
}
alert("You typed yes!");
```

c) do...while Loop

The do...while loop is a post-test loop, which means:

The loop executes the block at least once, before checking the condition.

After executing once, it keeps repeating as long as the condition is true.

Syntax:

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

- Starts the loop.
- **Code block**: Executes once before checking the condition.
- **while (condition)**: If true, loop continues. If false, loop stops.

Example:

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

d) for...in Loop

The for...in loop is used to **iterate over the enumerable properties (keys) of an object**.

It is **mainly used with objects**, not arrays (though it technically works on arrays too).

```
for (let key in object) {
  // code block using object[key]
}
```

- **key**: The name of each property in the object (as a string).
- **object[key]**: The value of the property.

```
let student = {
  name: "Ravi",
  age: 21,
  course: "BCA"
};

for (let key in student) {
  document.write(key + ": " + student[key]);
}
```

e) for...of Loop

The for...of loop is used to **iterate over iterable objects**, such as:

- Arrays
- Strings
- Maps
- Sets
- DOM collections (like NodeList)

```
for (let value of iterable) {
  // code block using value
}
```

- **value**: Each element from the iterable (e.g., array element, string character, etc.)
- **iterable**: An object that can be iterated (like an array or string)

```
let fruits = ["apple", "banana", "cherry"];

for (let fruit of fruits) {
  document.write(fruit);
}
```

10. Design a JavaScript code to reverse the given number

a. Read the value of N using prompt window.

b. Display the result in alert box

```
let num = parseInt(prompt("Enter a number:"));;

// b. Reverse the number
let reversed = 0;
while (num !== 0) {
  let digit = num % 10;
  reversed = reversed * 10 + digit;
  num = Math.floor(num / 10);
}

// Display the result in alert box
alert("Reversed number is: " + reversed);
```

11. Define an array in JavaScript. Explain the built-in array methods with their purpose and examples.

Definition of Array in JavaScript:

An **array** in JavaScript is a data structure used to store **multiple values in a single variable**. Each value in the array has an **index** (starting from 0) and can be accessed using this index.

Syntax:

```
let arrayName = [value1, value2, value3, ...];
```

Example:

```
let fruits = ["Apple", "Banana", "Mango"];
document.write(fruits[0]);  // Output: Apple
```

1. push()

- **Purpose:** Adds one or more elements to the **end** of an array.
- **Example:**

```
let fruits = ["Apple", "Banana"];
fruits.push("Mango");
document.write(fruits);  // Output: ["Apple", "Banana", "Mango"]
```

2. pop()

- **Purpose:** Removes the **last** element from an array.
- **Example:**

```
let fruits = ["Apple", "Banana", "Mango"];
fruits.pop();
document.write(fruits);  // Output: ["Apple", "Banana"]
```

3. shift()

- **Purpose:** Removes the **first** element from an array.
- **Example:**

```
let fruits = ["Apple", "Banana", "Mango"];
fruits.shift();
document.write(fruits);  // Output: ["Banana", "Mango"]
```

4. unshift()

- **Purpose:** Adds one or more elements to the **beginning** of an array.
- **Example:**

```
let fruits = ["Banana", "Mango"];
fruits.unshift("Apple");
document.write(fruits);  // Output: ["Apple", "Banana", "Mango"]
```

5. concat()

- **Purpose:** Joins two or more arrays and returns a new array.
- **Example:**

```
let a = [1, 2];
let b = [3, 4];
let c = a.concat(b);
document.write(c);  // Output: [1, 2, 3, 4]
```

6. **slice()**

- **Purpose:** Returns a portion of the array without modifying the original array.
- **Example:**

```
let fruits = ["Apple", "Banana", "Mango", "Orange"];
let sliced = fruits.slice(1, 3);
document.write(sliced);  // Output: ["Banana", "Mango"]
```

12. Define a function in JavaScript and explain its syntax with an example.

In JavaScript, a **function** is a block of code designed to perform a particular task. Functions are one of the building blocks of JavaScript and are used to organize code, make it reusable, and improve readability.

Definition of a Function

A **function** in JavaScript is defined using the function keyword, followed by:

- A name (optional for anonymous functions)
- A list of parameters enclosed in parentheses ()
- A block of code enclosed in curly braces {}

Syntax of a Function:

```
function functionName(parameter1, parameter2, ...) {
  // code to be executed
}
```

A function definition consists of the function's header and a compound statement that describes the actions of the function. This compound statement is called the body of the function. A function header consists of the reserved word function, the function's name, and a parenthesized list of parameters if there are any. The parentheses are required even if there are no parameters.

Example:

```
function greet(name) {
  document.write("Hello, " + name + "!");
}

greet("Reva");
```

Benefits of Using Functions

- ✓ Code reusability
- ✓ Better code organization
- ✓ Improves readability and maintainability
- ✓ Reduces repetition

13. Design a JavaScript code to the following.

a. Function: reverser

b. Parameter: A number.

c. Returns: The number with its digits in reverse order.

```
function reverser(number) {
  let reversed = 0;

  while (number > 0) {
    let digit = number % 10;
    reversed = reversed * 10 + digit;
    number = Math.floor(number / 10);
  }

  return reversed;
}

// Read number from user
let number =parseInt(prompt("Enter a number:"));
// Reverse the number
let result = reverser(number);
// Show the result
alert("Reversed number: " + result);
```

14. What is an exception in JavaScript? Explain with an example.

➢ Definition:

An **exception** in JavaScript is a **runtime error** that occurs when the program encounters an unexpected situation or invalid operation. When such an error occurs, JavaScript **"throws"** an exception which can **interrupt** the normal flow of the program.

To handle such errors and prevent the program from crashing, we use the **`try...catch`** block.

➢ Syntax:

```
try {
  // Code that may throw an error
} catch (error) {
  // Code to handle the error
}
```

➢ Example: Handling Division by Zero

```
try {
  let a = 10;
  let b = 0;
  if (b === 0) {
    throw new Error("Division by zero is not allowed");
  }
  let result = a / b;
  document.write("Result:", result);
} catch (e) {
  document.write("Exception Caught:", e.message);
}
```

➢ Explanation:

- ✓ The try block contains code that might cause an error.
- ✓ If an error occurs, control jumps to the catch block.

- ✓ throw is used to manually raise an exception.
- ✓ Error is a built-in JavaScript object used to create error messages.

15. Elaborate on the Date object in JavaScript with examples.

➢ Definition:

The **Date object** in JavaScript is a built-in object that allows you to create, manipulate, and format dates and times. It can represent any moment in time to the millisecond.

➢ Creating a Date Object

We can create a date object using the new Date() constructor as follows.

```
let currentDate = new Date(); // Current date and time
document.write(currentDate);
```

➢ Common Methods of the Date Object

Method	Description	Example	Output
getFullYear()	Gets 4-digit year	date.getFullYear()	2025
getMonth()	Gets month (0-11)	date.getMonth()	6 (July)
getDate()	Gets day of the month	date.getDate()	10
getDay()	Gets day of the week (0-6)	date.getDay()	4 (Thursday)
getHours()	Gets hours (0-23)	date.getHours()	e.g., 13
getMinutes()	Gets minutes	date.getMinutes()	e.g., 45
getSeconds()	Gets seconds	date.getSeconds()	e.g., 22
toDateString()	Returns a readable date string	date.toDateString()	Thu Jul 10 2025

```
// Create a new date object with current date and time
let date = new Date();

// Display the full date and time
document.write("Full Date and Time:", date);

// ----------------------------
// GET Methods
// ----------------------------
document.write("Year:", date.getFullYear());          // Get full year (e.g., 2025)
document.write("Month:", date.getMonth());            // Get month (0-11)
document.write("Date:", date.getDate());              // Get date of the month (1-31)
document.write("Day of Week:", date.getDay());        // Get day (0-6; 0 is Sunday)
document.write("Hours:", date.getHours());            // Get hours (0-23)
document.write("Minutes:", date.getMinutes());        // Get minutes (0-59)
document.write("Seconds:", date.getSeconds());        // Get seconds (0-59)
document.write("Milliseconds:", date.getMilliseconds()); // Get milliseconds (0-999)
document.write("Time in ms since Jan 1, 1970:", date.getTime()); // Get timestamp
document.write("Timezone Offset (minutes):", date.getTimezoneOffset());
```