ECMAScript

Baraneetharan Ramasamy

JavaScript

JavaScript (**JS**) is a lightweight interpreted (or just-in-time compiled) programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles.

The standards for JavaScript are the ECMAScript Language Specification (ECMA-262) and the ECMAScript Internationalization API specification (ECMA-402). As soon as one browser implements a feature, we try to document it. This means that cases where some proposals for new ECMAScript features have already been implemented in browsers, documentation and examples in MDN articles may use some of those new features. Most of the time, this happens between the stages 3 and 4, and is usually before the spec is officially published.

JavaScript vs JQuery

1. **JavaScript**:
    - **Definition**: JavaScript is a versatile programming language used for web development. It enhances website interactivity, complements HTML and CSS,

and powers both client-side (browser) and server-side (e.g., Node.js) applications.

- **Features**:
  - **DOM Manipulation**: JavaScript allows direct manipulation of DOM elements.
  - **Animations**: You can create animations using JavaScript.
  - **Event Handling**: JavaScript handles HTML event interactions.
  - **Ajax Requests**: Making asynchronous requests is possible using JavaScript.
  - **Cross-Browser Support**: Works across different browsers.
  - **Lightweight**: Vanilla JavaScript doesn't rely on external libraries.
- **Example**:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>JavaScript Example</title>
</head>
<body>
    <p id="maddy"></p>
    <script>
        let text = "";
        for (let k = 0; k < 10; k++) {
            if (k === 7) {
                continue;
            }
            text += "The number is " + k + "<br>";
        }
        document.getElementById("maddy").innerHTML = text;
    </script>
</body>
</html>
```

2. **jQuery**:
   - **Definition**: jQuery is a JavaScript library created to simplify common tasks. It provides a concise and feature-rich environment for client-side web development.

- **Features**:
  - **DOM Manipulation**: jQuery simplifies traversing and modifying DOM elements.
  - **Animations**: Built-in features for animations.
  - **Event Handling**: Easy event handling and manipulation.
  - **Ajax Requests**: Simple API for making cross-browser Ajax requests.
  - **CSS Manipulation**: High-level UI widget library.
  - **Cross-Browser Support**: Works well across browsers.
  - **Lightweight**: Only 19KB in size.
- **Example**:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>jQuery Example</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jqu
ery/3.5.1/jquery.min.js"></script>
</head>
<body>
    <h1><mark>Click "Hide_me" to make me disappear!</mark>
</h1>
    <button id="hide">Hide_me</button>
    <button id="show">Show_me</button>
    <script>
        $(document).ready(function () {
            $("#hide").click(function () {
                // Hide logic here
            });
            $("#show").click(function () {
                // Show logic here
            });
        });
    </script>
</body>
</html>
```

- **Note**: While jQuery simplifies development, consider using pure JavaScript for performance-intensive tasks.

In summary, JavaScript is the foundation, and jQuery streamlines common tasks.

Popular JavaScript frameworks as of 2023:

1. **Node.js**:
    - Node.js is the most widely used JavaScript framework. It enables high-performance, non-blocking I/O operations, making it ideal for real-time applications, streaming services, and data-intensive systems. Companies like Netflix, LinkedIn, Uber, and Trello rely on Node.js.
2. **React.js**:
    - React, developed by Facebook, is the second most popular framework. It's used extensively by professional developers. React allows building highly responsive user interfaces using a declarative and component-based approach. Its large community and frequent updates contribute to its popularity.
3. **jQuery**:
    - Although not as dominant as before, jQuery remains the third most popular web framework. It simplifies DOM manipulation and interaction. However, newer frameworks like React and Vue have gained more traction.
4. **Express**:
    - Express is a minimalist web application framework for Node.js. It simplifies server-side development, routing, and middleware handling. Many Node.js applications use Express for building APIs and web servers.
5. **Angular**:
    - Angular, developed by Google, is a comprehensive framework for building dynamic web applications. It provides features like two-way data binding, dependency injection, and component-based architecture. While it has a steeper learning curve, Angular is powerful for large-scale projects.

Is JQuery is dead?

As of 2023, jQuery is not "dead," but its dominance has diminished. Here's the current state:

1. **Usage Decline**:
    - jQuery was immensely popular in the past for simplifying DOM manipulation and cross-browser compatibility.

- However, modern browsers now provide native APIs that handle many tasks jQuery used to address.
- Consequently, developers have shifted toward using newer frameworks like React, Vue, and Angular.

2. **Still Relevant**:
   - jQuery is still relevant for maintaining legacy projects or working with older codebases.
   - Some plugins and libraries rely on jQuery, so it remains in use.
3. **Alternatives**:
   - For new projects, consider alternatives like vanilla JavaScript (leveraging ES6+ features) or more specialized libraries/frameworks.
   - If you need lightweight DOM manipulation, consider using native methods directly.

In summary, while jQuery isn't as central as before, it's not entirely obsolete. Developers now have more choices, and the decision depends on project requirements.

ECMAScript

ECMAScript is a standard for scripting languages, including JavaScript, JScript, and ActionScript. It is best known as a JavaScript standard intended to ensure the interoperability of web pages across different web browsers. It is standardized by Ecma International in the document ECMA-262.

ECMAScript is commonly used for client-side scripting on the World Wide Web, and it is increasingly being used for server-side applications and services using runtime environments - Node.js, deno and bun.

ECMAScript version history

Edition	Date published	Name	Changes from prior edition	Editor
1	June 1997		First edition based on JavaScript 1.1 as implemented in Netscape Navigator 3.0.[1]	Guy L. Steele Jr.

Edition	Date published	Name	Changes from prior edition	Editor
2	June 1998		Editorial changes to keep the specification fully aligned with ISO/IEC 16262:1998.	Mike Cowlishaw
3	December 1999		Based on JavaScript 1.2 as implemented in Netscape Navigator 4.0.[2] Added regular expressions, better string handling, new control statements, try/catch exception handling, tighter definition of errors, formatting for numeric output, and other enhancements	Mike Cowlishaw
4	*Abandoned* (last draft 30 June 2003)	ECMAScript 4 (ES4)	Fourth Edition was abandoned, due to political differences concerning language complexity. Many features proposed for the Fourth Edition have been completely dropped; some were incorporated into the sixth edition.	
5	December 2009		Adds "strict mode", a subset intended to provide more thorough error checking and avoid error-prone constructs. Clarifies many ambiguities in the 3rd edition specification, and accommodates behavior of real-world implementations that differed consistently from that specification. Adds	Pratap Lakshman, Allen Wirfs-Brock

Edition	Date published	Name	Changes from prior edition	Editor
			some new features, such as getters and setters, library support for JSON, and more complete reflection on object properties.[3]	
5.1	June 2011		Changes to keep the specification fully aligned with ISO/IEC 16262:2011.	Pratap Lakshman, Allen Wirfs-Brock
6	June 2015[4]	ECMAScript 2015 (ES2015)	See 6th Edition – ECMAScript 2015	Allen Wirfs-Brock
7	June 2016[5]	ECMAScript 2016 (ES2016)	See 7th Edition – ECMAScript 2016	Brian Terlson
8	June 2017[6]	ECMAScript 2017 (ES2017)	See 8th Edition – ECMAScript 2017	Brian Terlson
9	June 2018[7]	ECMAScript 2018 (ES2018)	See 9th Edition – ECMAScript 2018	Brian Terlson
10	June 2019[8]	ECMAScript 2019 (ES2019)	See 10th Edition – ECMAScript 2019	Brian Terlson, Bradley Farias, Jordan Harband
11	June 2020[9]	ECMAScript 2020 (ES2020)	See 11th Edition – ECMAScript 2020	Jordan Harband, Kevin Smith
12	June 2021[10]	ECMAScript 2021 (ES2021)	See 12th Edition – ECMAScript 2021	Jordan Harband, Shu-yu Guo, Michael Ficarra, Kevin Gibbons
13	June 2022[11]	ECMAScript 2022 (ES2022)	See 13th Edition – ECMAScript 2022	Shu-yu Guo, Michael Ficarra, Kevin Gibbons
14	June 2023	ECMAScript 2023 (ES2023)	See 14th Edition – ECMAScript 2023	Shu-yu Guo, Michael Ficarra, Kevin Gibbons

XMLHttpRequest

All modern browsers support the `XMLHttpRequest` object.

The `XMLHttpRequest` object can be used to exchange data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

XMLHttpRequest Object Methods

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method, url, async, user, psw*)	Specifies the request*method*: the request type GET or POST*url*: the file location*async*: true (asynchronous) or false (synchronous)*user*: optional user name*psw*: optional password
send()	Sends the request to the serverUsed for GET requests
send(*string*)	Sends the request to the server.Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

XMLHttpRequest Object Properties

Property	Description
onload	Defines a function to be called when the request is received (loaded)
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest.0: request not initialized1: server connection established2: request received3: processing request4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data

Property	Description
status	Returns the status-number of a request200: "OK"403: "Forbidden"404: "Not Found"For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Example

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
       // Typical action to be performed when the document is read
       document.getElementById("demo").innerHTML = xhttp.responseT
    }
};
xhttp.open("GET", "filename", true);
xhttp.send();
```

XMLHttpRequest vs AJAX

Certainly! Let's dive into the differences between `XMLHttpRequest` and AJAX:

1. **XMLHttpRequest (XHR):**
    - The `XMLHttpRequest` object is a browser API that allows you to make HTTP requests from JavaScript.
    - It provides a way to exchange data with a web server behind the scenes.
    - You can use it to retrieve data (e.g., JSON, XML, HTML) from a server without reloading the entire web page.
    - Key points:
        - All modern browsers support `XMLHttpRequest`.
        - You create an `XMLHttpRequest` object using: `const xhttp = new XMLHttpRequest();`
        - You define a callback function to handle the response when it's ready: `xhttp.onload = function() { /* Handle the data */ }`
        - To send a request, use the `open()` and `send()` methods:

```
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

- Note: For security reasons, browsers restrict access across domains. Both the web page and the XML file must be on the same server.

2. **AJAX (Asynchronous JavaScript and XML):**
   - AJAX is an approach to making HTTP requests asynchronously without blocking the UI thread.
   - It encompasses various techniques, including the use of `XMLHttpRequest`.
   - AJAX allows you to update parts of a web page dynamically without full page reloads.
   - Key points:
     - AJAX is not a specific object or method; it's a concept.
     - It involves using asynchronous requests to fetch data and update the page.
     - You can use `XMLHttpRequest` directly or libraries like jQuery (which wraps `XMLHttpRequest` for easier use).

In summary, `XMLHttpRequest` is the raw browser object for making HTTP requests, while AJAX refers to the broader approach of using asynchronous requests to enhance web applications.

Datatypes in JavaScript

There are 8 basic data types in JavaScript.

Primitive data types:

number for numbers of any kind: integer or floating-point, integers are limited by ±(253-1).

bigint for integer numbers of arbitrary length.

string for strings. A string may have zero or more characters, there's no separate single-character type.

boolean for true/false.

null for unknown values – a standalone type that has a single value null.

undefined for unassigned values – a standalone type that has a single value undefined.

symbol for unique identifiers.

Non-primitive data type:

object for more complex data structures.

The typeof operator allows us to see which type is stored in a variable.

Usually used as **typeof x**, but **typeof(x)** is also possible.

Returns a string with the name of the type, like "string".

For null returns "object" – this is an error in the language, it's not actually an object.

JavaScript Data Structure

1. **Data Structures in JavaScript**:
   - A data structure is a way to organize, manage, and store data efficiently. It consists of:
     - Data values.
     - Relationships among those values.
     - Functions or operations that can be applied to the data.
   - JavaScript provides both primitive (built-in) and non-primitive (custom) data structures.
2. **Arrays**:
   - An array is a collection of items stored in contiguous memory locations.
   - Each item can be accessed using an index (starting from 0).
   - Example:

```
const arr = ['a', 'b', 'c', 'd'];
console.log(arr[2]); // Output: 'c'
```

3. **Objects (Hash Tables)**:
   - Objects are collections of key-value pairs.
   - Useful for representing structured data.
   - Example:

```
const person = {
    name: 'Alice',
    age: 30,
    occupation: 'Developer'
};
console.log(person.name); // Output: 'Alice'
```

4. **Stacks and Queues**:
    - Stacks follow the Last-In-First-Out (LIFO) principle.
    - Queues follow the First-In-First-Out (FIFO) principle.
    - Implementing them requires custom code.
5. **Linked Lists**:
    - Linked lists consist of nodes, where each node points to the next node.
    - Useful for dynamic data structures.
    - Example:

```
class Node {
    constructor(value) {
        this.value = value;
        this.next = null;
    }
}
const list = new Node('a');
list.next = new Node('b');
```

6. **Trees**:
    - Trees have a hierarchical structure with a root node and child nodes.
    - Binary trees have at most two children per node.
    - Example:

```
class TreeNode {
    constructor(value) {
        this.value = value;
        this.left = null;
        this.right = null;
    }
```

```
}
const root = new TreeNode('root');
root.left = new TreeNode('left');
root.right = new TreeNode('right');
```

7. **Graphs**:
    - Graphs represent relationships between entities.
    - Can be directed or undirected.
    - Example:

```
const graph = {
    A: ['B', 'C'],
    B: ['A', 'D'],
    C: ['A'],
    D: ['B']
};
```

Remember, these data structures serve different purposes, and choosing the right one depends on your specific use case! 😊12345

JavaScript Class

```
class MyClass {
  // class methods
  constructor() { ... }
  method1() { ... }
  method2() { ... }
  method3() { ... }
  ...
}
```

JavaScript Functions

There are several ways to write a function in JavaScript. Here are some of the most common ways:

Function declaration: This is the most common way to define a function in JavaScript. To declare a function, you use the function keyword followed by an

obligatory function name, a list of parameters in brackets, and a pair of curly braces to write the function code.

```
function addTwoNumbers(x, y) {
  return x + y;
}
```

Function expression: This is another way to define a function in JavaScript. With this method, you assign a function to a variable.

```
let addTwoNumbers = function(x, y) {
  return x + y;
};
```

Arrow functions: Arrow functions are a shorthand way of writing functions in JavaScript. They are similar to function expressions but have a more concise syntax.

```
let addTwoNumbers = (x, y) => x + y;
```

Method definition: In JavaScript, you can define functions as methods of objects.

```
let myObject = {
  addTwoNumbers: function(x, y) {
    return x + y;
  }
};
```

Constructor function

```
var Employee = function(employee){
  this.first_name     = employee.first_name;
  this.last_name      = employee.last_name;
  this.email          = employee.email;
  this.phone          = employee.phone;
  this.organization   = employee.organization;
  this.designation    = employee.designation;
  this.salary         = employee.salary;
  this.status         = employee.status ? employee.status : 1;
  this.created_at     = new Date();
```

```
  this.updated_at     = new Date();
};
```

Equivalent Employee class with a constructor

```
class Employee {
  constructor(employee) {
    this.first_name = employee.first_name;
    this.last_name = employee.last_name;
    this.email = employee.email;
    this.phone = employee.phone;
    this.organization = employee.organization;
    this.designation = employee.designation;
    this.salary = employee.salary;
    this.status = employee.status ? employee.status : 1;
    this.created_at = new Date();
    this.updated_at = new Date();
  }
}
```

Create an Object

```
// Define an object with employee details
var employeeDetails = {
  first_name: 'John',
  last_name: 'Doe',
  email: 'john.doe@example.com',
  phone: '123-456-7890',
  organization: 'Example Corp',
  designation: 'Software Engineer',
  salary: 50000,
  status: 1 // Optional, defaults to 1 if not provided
};

// Create an Employee object
var employee = new Employee(employeeDetails);

console.log(employee);
```

What is Nodejs

Node.js is a powerful tool for building fast and scalable web applications. It's an open-source, cross-platform JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. This allows developers to use JavaScript to write command-line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.

Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server- and client-side scripts.

Key Features of Node.js:

- **Asynchronous and Event-Driven**: All APIs of the Node.js library are asynchronous, meaning non-blocking. It essentially means a Node.js based server never waits for an API to return data.
- **Very Fast**: Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- **Single Threaded but Highly Scalable**: Node.js uses a single-threaded model with event looping. This event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests.
- **No Buffering**: Node.js applications never buffer any data. These applications simply output the data in chunks.

Where to Use Node.js?

Node.js can be used for the following kinds of applications:

- Real-time chat applications
- Complex single-page applications
- REST API servers
- Real-time collaboration tools
- Streaming applications
- Microservices architecture

Where Not to Use Node.js?

It's not advisable to use Node.js for CPU-intensive operations; it's built to handle asynchronous events. Using it for heavy computation will annul nearly all of its

advantages.

Remember, Node.js is not a silver bullet that can tackle all kinds of software problems. But for certain domains, such as building fast, scalable network applications, it is an excellent choice that can make your development process easier and more fun.

Node.js version managers

Package Manager

How to create a project?

1. **Install Node.js and npm**:
   - Download and install Node.js from the official website.
   - Verify the installation by running `node -v` and `npm -v` in your terminal.
2. **Set Up Your Project Directory**:
   - Create a new directory for your project: `mkdir my-node-app`.
   - Navigate into your project directory: `cd my-node-app`.
3. **Initialize Your Node.js Project**:
   - Run `npm init` to create a `package.json` file.
   - Follow the prompts to set up your project details.
4. **Create Your Main Application File**:
   - Create a file named `app.js` or `index.js` as your entry point.
5. **Install Dependencies** (if needed):
   - Use `npm install <package_name>` to install any packages your project requires.
6. **Write Your Application Code**:
   - Add your JavaScript code to the main application file you created.
7. **Run Your Node.js Application**:
   - Execute `node app.js` to start your application.

list of common npm commands in bullet points:

- `npm init`: Initialize a new Node.js project and create a `package.json` file.
- `npm install`: Install all the dependencies listed in `package.json`.
- `npm install <package_name>`: Install a specific package.

- `npm install <package_name> -g`: Install a package globally.
- `npm update`: Update all the packages to the latest version.
- `npm update <package_name>`: Update a specific package to the latest version.
- `npm uninstall <package_name>`: Remove a specific package.
- `npm start`: Run the script defined as `start` in `package.json`.
- `npm test`: Run the script defined as `test` in `package.json`.
- `npm run <script>`: Run a script defined in `package.json`.
- `npm run-script <script>`: Alias for `npm run`.
- `npm list`: List installed packages.
- `npm list -g`: List globally installed packages.
- `npm outdate`: Check for outdated packages.
- `npm search <term>`: Search for packages.
- `npm publish`: Publish a package to the registry.
- `npm cache clean --force`: Clear the npm cache.
- `npm audit`: Run a security audit to analyze the dependencies.
- `npm audit fix`: Automatically fix vulnerabilities in dependencies.
- `npm version <update_type>`: Bump the version of your package.
- `npm pack`: Create a tarball from a package.
- `npm link`: Symlink a package folder for local development.
- `npm ci`: Install a project with a clean slate based on `package-lock.json`.
- `npm rebuild`: Rebuild a package.
- `npm dedupe`: Reduce duplication in the node_modules directory.
- `npm doctor`: Check your environments and diagnose common issues.
- `npm config list`: List all npm configuration options.
- `npm config set <key> <value>`: Set a configuration option.
- `npm config get <key>`: Get the value of a configuration option.
- `npm config delete <key>`: Delete a configuration option.
- `npm owner add <user> <package_name>`: Add a user as an owner of a package.
- `npm owner rm <user> <package_name>`: Remove a user from the list of owners of a package.

- `npm access public <package_name>`: Set a package to be publicly accessible.
- `npm access restricted <package_name>`: Set a package to be accessible only to users with permission.

JavaScript Modules

AMD – one of the most ancient module systems, initially implemented by the library require.js.

CommonJS – the module system created for Node.js server.

UMD – one more module system, suggested as a universal one, compatible with AMD and CommonJS.

https://www.freecodecamp.org/news/modules-in-javascript/

CommonJS is a standard for structuring and organizing JavaScript code into reusable modules. Here's an explanation with different examples:

What is CommonJS?

- CommonJS is a module specification used in JavaScript, particularly on the server-side with Node.js.
- It allows you to encapsulate code within modules, which can then be imported and used in other files.

Why Use CommonJS?

- **Modularity**: Breaks down code into manageable pieces.
- **Reusability**: Modules can be reused across different parts of an application.
- **Maintainability**: Easier to maintain and update code in a modular structure.

CommonJS Syntax:

- **Exporting a Module**: You can export objects, functions, or variables from a module using `module.exports`.
- **Importing a Module**: You can import the exported module using the `require()` function.

Examples:

Exporting a Module:

JavaScript

```
// math.js

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

function subtract(a, b) {
  return a - b;
}

module.exports = {
  add,
  subtract
};
```

Importing a Module:

JavaScript

```
// app.js

const math = require('./math.js');

console.log(math.add(2, 3)); // Output: 5console.log(math.subtra
ct(5, 2)); // Output: 3
```

Exporting Multiple Modules:

JavaScript

```
// utils.js

module.exports.add = function(a, b) {
  return a + b;
};

module.exports.subtract = function(a, b) {
  return a - b;
};
```

Importing Specific Functions:

JavaScript

```
// app.js

const { add, subtract } = require('./utils.js');

console.log(add(10, 5)); // Output: 15console.log(subtract(10,
5)); // Output: 5
```

Notes:

- **CommonJS** is mainly used in Node.js environments as browsers do not natively support it.
- With the advent of ES6, **ESModules** have become the standard for JavaScript modules, which use `import` and `export` syntax.

ESModules, or ECMAScript Modules, are the official standard format for packaging JavaScript code for reuse. Here's an explanation with different examples:

What are ESModules?

- **ESModules** provide a way to export and import functions, variables, or classes from one JavaScript file to another.
- They use the `import` and `export` statements to share code between different files, promoting modularity and maintainability.

Why Use ESModules?

- **Code Organization**: Helps in organizing code into small, manageable pieces.
- **Reusability**: Makes it easy to reuse code across different parts of an application or even across different projects.
- **Tree Shaking**: Allows for more efficient bundling by eliminating unused code.

ESModules Syntax:

- **Exporting**: Use `export` to make parts of the module available to other files.
- **Importing**: Use `import` to bring in exports from other modules.

Examples:

Exporting a Single Value:

JavaScript

```
// utils.js

export const PI = 3.14159;
```

Importing a Single Value:

JavaScript

```
// app.js
import { PI } from './utils.js';

console.log(PI); // Output: 3.14159
```

Exporting Multiple Values:

JavaScript

```
// math.js

export function add(a, b) {
  return a + b;
}

export function subtract(a, b) {
  return a - b;
}
```

Importing Multiple Values:

JavaScript

```
// app.js
import { add, subtract } from './math.js';

console.log(add(5, 3)); // Output: 8console.log(subtract(5, 3));
// Output: 2
```

Default Export:

JavaScript

```
// defaultExport.js
export default function() {
  console.log('This is the default export');
}
```

Importing Default Export:

JavaScript

```
// app.js

import myDefaultFunction from './defaultExport.js';

myDefaultFunction(); // Output: This is the default export
```

Renaming Exports and Imports:

JavaScript

```
// math.js

export const squareRoot = Math.sqrt;

// app.jsimport { squareRoot as sqrt } from './math.js';

console.log(sqrt(16)); // Output: 4
```

Dynamic Import:

JavaScript

```
// This is useful for code-splitting and lazy-loading modulesif
(condition) {
  import('./module.js').then((module) => {
    module.doSomething();
  });
}
```

Notes:

- **Browser Support**: Modern browsers support ESModules natively.

- **File Extension**: JavaScript files using ESModules typically have the `.mjs` extension when used on the server-side with Node.js.
- **Static & Dynamic Analysis**: ESModules allow for static analysis of dependencies and can also support dynamic imports.