**Modular JavaScript with ES Modules**

**Overview**

You may have seen or written JavaScript code that is long and difficult to read. This is often the result of developing code in one single file. This makes the code hard to maintain and understand, especially as the a project grows in size and complexity. To address this issue, **JavaScript ES Modules** were introduced to provide a way to organize code into smaller, reusable pieces called modules.

**Prepare**

ES Modules provide ways to help you write cleaner, more organized, and more manageable code by breaking it down into smaller, reusable parts. The following sections addresses the key concepts and reasons how and why to use modules.

**Native Browser Support**

In order to use ES Module features, the type="module" attribute is required in the <script> tag. This attribute tells the browser to treat the script as a module, enabling the use of import and export statements within the script.

For example, you would reference a module script named **app.mjs** like this:

<script type="module" src="scripts/app.mjs"></script>

Note that the script file uses the **.mjs** extension to indicate that it is a module. While not strictly required, using **.mjs** is considered a good practice for clarity and consistency when working with ES Modules.

Modules are loaded asynchronously by default, so they don't block the HTML parser during loading or execution. As a result, there is no need to add the defer attribute to module script references.

**Better Code Organization**

ES Modules allow you to split your code into separate files. This allows you to have better code organization as you keep related functionality together and separate unrelated code. For example, you can have one module for handling DOM manipulation and another module for managing data storage.

Modules can export and import functions, objects, or variables between different files. For example, you might have a module for rendering product cards and another module for managing data storage.

// in storage.mjs - handles data management

export function getData() { /\* implementation \*/ }

export total;

export default class StorageManager { /\* implementation \*/ }

// in output.mjs - handles UI rendering

export default function renderProductCard() { }

// in app.mjs - imports and uses both modules

import { getData, total } from './storage.mjs';

import StorageManager from './storage.mjs';

import renderProductCard from './output.mjs';

The import statement brings functionality from other modules into your current file. Each module can have multiple named exports (using curly braces when importing) and one default export (imported without curly braces). Default exports are useful for a module's primary function or class.

Modules can be imported in any order, and the module system ensures that dependencies are resolved correctly.

Modules are scoped to the module itself, meaning that variables and functions defined in a module are not accessible in the global scope unless explicitly exported. This helps to prevent naming conflicts and keeps the global scope clean.

**Reusability**

Modules can be reused across different parts of your application or even in different projects. This promotes code reuse and reduces duplication, making it easier to maintain and update your codebase.

For example, if you have a module that handles user authentication, you can use it in multiple applications without having to rewrite the code. This is especially useful for libraries or frameworks that provide common functionality that can be shared across different projects.

By using modules, you can create a library of reusable components that can be easily imported and used in other parts of your application. This not only saves time but also helps to keep your codebase clean and organized.

**Activity Instructions**

In this activity you will replace the existing application structure to one that uses modules.

**Inspect**

1. Open [modules.html](https://byui-cse.github.io/wdd231-ww-course/week03/modules/modules.html) page and view/inspect the HTML and JavaScript.
2. Note that there is currently only one deferred, JavaScript file that is named **modules.js**.
3. Review the code. This code will be put into modules.

Note that a <form> element is not used to for the input. This is because a form will cause the page to refresh when the button is clicked. This is not desired behavior in this case. Instead, the button is used to trigger the function enrollStudent() and dropStudent() through event listeners. A form element could be used, but the default behavior of the form would need to be ignored using the event.preventDefault() method.

**Setup and Structure**

1. Create your own **modules.html**, **modules.css**, **modules.*mjs*** files and copy the HTML, CSS, and JavaScript provided in step 1.
2. Create a new JavaScript file named **course.mjs**.
3. Create a new JavaScript file named **sections.mjs**.
4. Create a new JavaScript file named **output.mjs**

**course.mjs**

This file will contain the course object section data and its methods that are used to enroll and drop students from the course.

1. Move the byuiCourse object into the **course.mjs** file from the **modules.js** file.
2. In this **course.mjs** file, export the byuiCourse object as the default. This line can be the last line of the file.

export default byuiCourse;

**sections.mjs**

This file will contain the function that populates the section selection element on the page.

1. Move the setSectionSelection function from the **modules.mjs** file into the **sections.mjs** file.
2. export the setSectionSelection function as a named export.

export function populateSections(sections) { ... }

1. Finally, remove the renderSections(this.sections); line of code in the changeEnrollment method of the byuiCourse object. An run-time error will occur when an update is attempted given this function is no longer available to call within this new module file.

**output.mjs**

This file will contain the functions that are used to render the course title and sections to the page.

1. Move the setTitle and renderSections functions into the **output.mjs** file.
2. export the setTitle and renderSections functions as named exports.

export function setTitle(course) { ... }

export function renderSections(sections) { ... }

**modules.mjs**

The content of this script file has been reduced to event listeners and the function calls.

1. At the top of the file, import the byuiCourse object from the **course.mjs** module.

import byuiCourse from './course.mjs';

1. Next, import the setSectionSelection function from the **sections.mjs** module.

import { setSectionSelection } from './sections.mjs';

Note that the function is encased in squiggly brackets because it is a **named export**. The brackets are not required for a single import, but recommended for clarity. This function is not the **default** export of the module. It could be converted to a default export in the module, but it is not necessary.

1. Next, import the named function exports from the **output.mjs** file.Check Your Understanding
2. Add renderSections(this.sections); to **both** event listeners in order to update the output after the enroll or drop button is clicked.
3. document.querySelector("#enrollStudent").addEventListener("click", function () {
4. const sectionNum = document.querySelector("#sectionNumber").value;
5. byuiCourse.changeEnrollment(sectionNum);
6. renderSections(byuiCourse.sections);
7. });
9. document.querySelector("#dropStudent").addEventListener("click", function () {
10. const sectionNum = document.querySelector("#sectionNumber").value;
11. byuiCourse.changeEnrollment(sectionNum, false);
12. renderSections(byuiCourse.sections);

});

**HTML script**

1. Change the script reference in the **modules.html** file to use the new module structure by removing the defer attribute and adding the type="module" attribute.

<script type="module" src="scripts/modules.js"></script>

**Test and Share**

1. Test your code to ensure that it works as expected. You should be able to select a section and increase or decrease the displayed total enrollment for that section.
2. Share your code and issues, if any, with your peers on Microsoft Teams for review.

**Optional Resources**

* [JavaScript Modules](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules) – MDN
* [Basic Modules](https://github.com/mdn/js-examples/tree/main/module-examples/basic-modules) – MDN Repository

[Back](https://byui-cse.github.io/wdd231-ww-course/week03/index.html)