**INTERMEDIATE JAVASCRIPT MODULES**

**module.exports**

We can get started with modules by defining a module in one file and making the module available for use in another file with Node.js module.exports syntax.

Every JavaScript file run in Node has a local module object with an exports property used to define what should be exported from the file.

Below is an example of how to define a module and how to export it using the statement module.exports.

In **menu.js** we write:

let Menu = {};

Menu.specialty = "Roasted Beet Burger with Mint Sauce";

module.exports = Menu;

Let’s consider what this code means.

1. let Menu = {}; creates the object that represents the module Menu. The statement contains an uppercase variable named Menu which is set equal to an empty object.
2. Menu.specialty is defined as a property of the Menu module. We add data to the Menu object by setting properties on that object and giving the properties a value.
3. "Roasted Beet Burger with Mint Sauce"; is the value stored in the Menu.specialty property.
4. module.exports = Menu; exports the Menu object as a module. module is a variable that represents the module, and exports exposes the module as an object.

The pattern we use to export modules is thus:

1. Create an object to represent the module.
2. Add properties or methods to the module object.
3. Export the module with module.exports.

Let’s create our first module.

**INTERMEDIATE JAVASCRIPT MODULES**

**require()**

To make use of the exported module and the behavior we define within it, we import the module into another file. In Node.js, use the require() function to import modules.

For instance, say we want the module to control the menu’s data and behavior, and we want a separate file to handle placing an order. We would create a separate file **order.js** and import the Menu module from **menu.js** to **order.js** using require(). require() takes a file path argument pointing to the original module file.

In **order.js** we would write:

const Menu = require('./menu.js');

function placeOrder() {

console.log('My order is: ' + Menu.specialty);

}

placeOrder();

We now have the entire behavior of Menu available in **order.js**. Here, we notice:

1. In **order.js** we import the module by creating a const variable called Menu and setting it equal to the value of the require() function. We can set the name of this variable to anything we like, such as menuItems.
2. require() is a JavaScript function that loads a module. It’s argument is the file path of the module: ./menu.js. With require(), the .js extension is optional and will be assumed if it is not included.
3. The placeOrder() function then uses the Menu module. By calling Menu.specialty, we access the property specialty defined in the Menu module.

Taking a closer look, the pattern to import a module is:

1. Import the module with require() and assign it to a local variable.
2. Use the module and its properties within a program.

#### module.exports II

We can also wrap any collection of data and functions in an object, and export the object using module.exports. In **menu.js**, we could write:

module.exports = {

specialty: "Roasted Beet Burger with Mint Sauce",

getSpecialty: function() {

return this.specialty;

}

};

In the above code, notice:

1. module.exports exposes the current module as an object.
2. specialty and getSpecialty are properties on the object.

Then in **order.js**, we write:

const Menu = require('./menu.js');

console.log(Menu.getSpecialty());

Here we can still access the behavior in the Menu module.

#### export default

Node.js supports require()/module.exports, but as of ES6, JavaScript supports a new more readable and flexible syntax for exporting modules. These are usually broken down into one of two techniques, default export and named exports.

We’ll begin with the first syntax, default export. The default export syntax works similarly to the module.exports syntax, allowing us to export one module per file.

Let’s look at an example in **menu.js**.

let Menu = {};

export default Menu;

1. export default uses the JavaScript export statement to export JavaScript objects, functions, and primitive data types.
2. Menu refers to the name of the Menu object, the object that we are setting the properties on within our modules.

When using ES6 syntax, we use export default in place of module.exports. Node.js doesn’t support export default by default, so module.exports is usually used for Node.js development and ES6 syntax is used for front-end development. As with most ES6 features, it is common to transpile code since ES6 is [not supported by all browsers](https://caniuse.com/#feat=es6).

#### import

ES6 module syntax also introduces the import keyword for importing objects. In our **order.js** example, we import an object like this:

import Menu from './menu';

1. The import keyword begins the statement.
2. The keyword Menu here specifies the name of the variable to store the default export in.
3. from specifies where to load the module from.
4. './menu' is the name of the module to load. When dealing with local files, it specifically refers to the name of the file without the extension of the file.

We can then continue using the Menu module in the **order.js** file.

#### Named Exports

ES6 introduced a second common approach to export modules. In addition to export default, named exports allow us to export data through the use of variables.

Let’s see how this works. In **menu.js** we would be sure to give each piece of data a distinct variable name:

let specialty = '';

function isVegetarian() {

};

function isLowSodium() {

};

export { specialty, isVegetarian };

1. Notice that, when we use named exports, we are not setting the properties on an object. Each export is stored in its own variable.
2. specialty is a string object, while isVegetarian and isLowSodium are objects in the form of functions. Recall that in JavaScript, every function is in fact a function object.
3. export { specialty, isVegetarian }; exports objects by their variable names. Notice the keyword export is the prefix.
4. specialty and isVegetarian are exported, while isLowSodium is not exported, since it is not specified in the export syntax.

#### Named Imports

To import objects stored in a variable, we use the import keyword and include the variables in a set of {}.

In the **order.js** file, for example, we would write:

import { specialty, isVegetarian } from './menu';

console.log(specialty);

1. Here specialty and isVegetarian are imported.
2. If we did not want to import either of these variables, we could omit them from the import statement.
3. We can then use these objects as in within our code. For example, we would use specialty instead of Menu.specialty.