**INTERMEDIATE JAVASCRIPT MODULES**

Hello Modules

JavaScript *modules* are reusable pieces of code that can be exported from one program and imported for use in another program.

Modules are particularly useful for a number of reasons. By separating code with similar logic into files called modules, we can:

* find, fix, and debug code more easily;
* reuse and recycle defined logic in different parts of our application;
* keep information private and protected from other modules;
* and, importantly, prevent pollution of the global namespace and potential naming collisions, by cautiously selecting variables and behavior we load into a program.

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. 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 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. Define an object to represent the module.
2. Add data or behavior to the module.
3. Export the module.

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. A common way to do this is with the require() function.

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():

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 Menuavailable in **order.js**. Here, we notice:

1. In **order.js** we import the module by creating a variable with const called Menu and setting it equal to the value of the require() function. We can set this variable to any variable name we like, such as menuItems.
2. require() is a JavaScript function that enables a module to load by passing in the module's filepath as a parameter.
3. './menu.js' is the argument we pass to the require() function.
4. The placeOrder() function then uses the Menu module in its function definition. By calling Menu.specialty, we access the property specialtydefined in the Menu module.
5. We can then invoke the function using placeOrder()

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

1. Import the module
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:

let Menu = {}; 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.