Checkpoint 5

Modules

Using the JavaScript language with Node is quite powerful. However, this combination can become even more useful when you add Node modules into the mix.

Outline

In this checkpoint, you will learn about local modules, which are one of the three different types of modules. At this point in your journey, learning how to make use of modules will allow you to better organize the JavaScript code that you write on your machine.

By the end of this checkpoint, you will be able to do the following:

- Create and connect your own modules

What is a module?

In Node, each file is treated as a separate module. A *module*, sometimes also referred to as a *package*, can contain any JavaScript code. It most commonly exports an object or a function. The exported value then becomes available in other modules.

Put simply, each file in a Node project is its own module. You can then export data from that file to be used in another file.

Do this

Create a new folder

In this checkpoint, you will be asked to make a few different files to practice the concepts that you are learning. Create a new folder with a memorable name, like `learning-node-modules/`.

Local modules

The best way to understand modules is to create and use them. For example, take a look at the following code. Imagine that it is in a file called `plants.js`.

Outline

```
function findPlantById(plants, id) {
  let result = null;
  for (let i = 0; i < plants.length; i++) {
    let plant = plants[i];
    if (plant.id === id) {
      result = plant;
    }
  }
  return result;
}

module.exports = findPlantById;
```

The above function expects an array of objects as the first argument passed to the function, where each object has an `id`. It then accepts an `id`. The function then loops through all of the plants and returns the one that has a matching `id`. If none match, it returns `null`.

At the end of the code block shown above, you may have noticed the following line:

```
module.exports = findPlantById;
```

This line indicates that the file is a module. A function is being exported from this file.

You can use this function in another file. For example, take a look at the following code. Imagine that it is in a file called `main.js`.

```
let find = require("./plants");
let plants = [
  { id: 1, name: "Garden Rocket Arugula" },
  { id: 2, name: "Watercress" },
  { id: 3, name: "Royal Rose Radicchio" },
];

const result = find(plants, 2);
console.log(result);
```

Outline

The new and important part of the code above is the `require()` function. The `require()` function takes a path as an argument. If no file extension is given, it will assume that the file is a JavaScript file. It looks into that file and then finds what is exported, and whatever is exported is returned from the `require()` function.

So, in the above file, the following happens:

1. The `findPlantById()` function is imported from the `plants.js` file. Notice that it is assigned to a new name, `rind`,

although it can be assigned any name.

2. A `plants` variable is declared, which points to an array of objects with IDs.
3. The `find()` function is called with the appropriate arguments.

Do this

Create a module

In the folder that you've created, create two new files: `plants.js` and `main.js`. Copy the above content into the files as appropriate, and then run the `main.js` file.

You will see the following output:

```
{ id: 2, name: "Watercress" }
```

View the default value

In your `plants.js` file, comment out the line that is exporting the function.

```
// module.exports = findPlantById;
```

Then, at the top of your `main.js` file, add the following line:

```
console.log(require("./plants.js"));
```

Outline

What do you see? You should see an empty object. By default, if you require a file that does not have a `module.exports` statement, you will receive an empty object.

Now, undo the changes that you made in this step.

Export an object

After you've undone the changes from the above task, update your `plants.js` file with the following export statement. This export statement should replace the one that you already have.

```
module.exports = { findPlantById: findPlantById };
```

Outline

Now, you are exporting an object instead of a function. How would you update your code in `main.js` so that you can still call the function?

There are a few ways that could work, but one option is to update the code as follows:

```
let plantFunctions = require("./plants");
let plants = [
  { id: 1, name: "Garden Rocket Arugula" },
  { id: 2, name: "Watercress" },
  { id: 3, name: "Royal Rose Radicchio" },
];

console.log(plantFunctions.findPlantById(plants, 2));
```

Checkpoint

This checkpoint will be autograded. Please click the link below to open your assignment in a new tab. Once you complete the assignment, you will see a button allowing you to submit your answers and move on to the next checkpoint.

Your work

03.17.21 **Approved**

Outline

Completed

Next checkpoint

How would you rate this content?

Report a typo or other issue

Go to Overview