**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.

In this lesson, we’ll cover two ways to implement modules in JavaScript: [Node.js’s](https://nodejs.org/en/about/) module.exports and require() syntax, as well as the ES6 import/export syntax.

**Instructions**

Once you feel like you have a high-level understanding of modules, proceed to the next exercise.

**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.

**Instructions**

**1.**

Let’s begin by implementing the pattern above in an example. In **1-airplane.js** create an object named Airplane.

Hint

let Airplane = {};

**2.**

Within the same file, add a property to the Airplane object named myAirplane and set it equal to "StarJet".

Hint

Airplane.myAirplane = "StarJet";

**3.**

Export the module.

Hint

module.exports = Airplane;

let Airplane = {};

Airplane.myAirplane = "StarJet";

module.exports = Airplane;

const Airplane = require('./1-airplane.js');

function displayAirplane() {

  console.log(Airplane.myAirplane);

}

displayAirplane();

**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.

**Instructions**

**1.**

In **2-airplane.js**, set module.exports equal to an empty object.

Hint

module.exports = {

};

**2.**

Within the object, create a key called myAirplane and set it to a value "CloudJet".

Hint

module.exports = {

myAirplane: "CloudJet"

};

**3.**

Again, within module.exports, create another key displayAirplane and set it to an anonymous function. The function should use the this statement to return myAirplane.

Hint

module.exports = {

myAirplane: "CloudJet",

displayAirplane: function() {

return this.myAirplane;

}

};

**4.**

In **2-missionControl.js** use the require() function to import the Airplane module.

Hint

const Airplane = require('./2-airplane.js');

**5.**

In **2-missionControl.js** log the result of calling .displayAirplane() to the console, noting that it is a method of the Airplane object.

Hint

console.log(Airplane.displayAirplane());

module.exports = {

  myAirplane: "CloudJet",

  displayAirplane: function(){

 return this.myAirplane;

  }

};

const Airplane = require('./2-airplane.js');

console.log(Airplane.displayAirplane());

**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).

**Instructions**

**1.**

In **airplane.js**, let’s again create an Airplane module from scratch, this time exporting the module with export default. Create an object to represent the module called Airplane.

Hint

let Airplane = {};

**2.**

Now that we have an object Airplane, we can continue by adding data in the form of properties and values to the Airplane module.

Create an availableAirplanes variable and set it equal to an empty array. Be sure that availableAirplanes is a property of the Airplane object.

Hint

Airplane.availableAirplanes = [];

**3.**

In the availableAirplanes array, add two array elements that are both of type object.

The first object should contain a property name with a value 'AeroJet' and a property fuelCapacity with a value of 800.

The second object should have a property name with a value of SkyJet and a property fuelCapacity with a value of 500.

Hint

Airplane.availableAirplanes = [

{

name: 'AeroJet',

fuelCapacity: 800

},

{name: 'SkyJet',

fuelCapacity: 500

}

];

**4.**

Use export default to export the Airplane module.

Nice work! We added a property that lists the availableAirplanes to the Airplane module.

Hint

export default Airplane;

let Airplane = {};

Airplane.availableAirplanes = [{

  name: 'AeroJet',

  fuelCapacity: 800

 },

 {name: 'SkyJet',

  fuelCapacity: 500

 }

 ];

 export default Airplane;

**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.

**Instructions**

**1.**

In **missionControl.js** we’ll use the module Airplane to display the fuel capacity of both our airplanes.

Use the import keyword to import the Airplane module.

Hint

import Airplane from './airplane';

**2.**

Define a function displayFuelCapacity().

Hint

function displayFuelCapacity() {

}

**3.**

Within the body of the displayFuelCapacity function, use forEach() to iterate over the Airplane.availableAirplanes array.

The forEach() should take an anonymous function as a parameter. We’ll add more in the next step.

Hint

function displayFuelCapacity() {

Airplane.availableAirplanes.forEach(function(){});

}

**4.**

Pass the anonymous function you created in the last step a parameter of element.

Hint

function displayFuelCapacity () {

Airplane.availableAirplanes.forEach(function(element) {

});

}

**5.**

Within the displayFuelCapacity function, use console.log() to output the element’s name and its fuel capacity. The output should look like this:

'Fuel Capacity of + (element name) : + (element fuelCapacity)'

Hint

For this checkpoint, make sure to use single quotes like the code below:

function displayFuelCapacity() {

Airplane.availableAirplanes.forEach(function(element){

console.log('Fuel Capacity of ' + element.name + ': ' + element.fuelCapacity);

});

}

**6.**

Call the displayFuelCapacity function.

Hint

displayFuelCapacity();

import Airplane from './airplane';

function displayFuelCapacity() {

  Airplane.availableAirplanes.forEach(function(element){

    console.log('Fuel Capacity of' + element.name+' : '+ element.fuelCapacity);

  }

  );

}

displayFuelCapacity();

**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.

**Instructions**

**1.**

Remove the statement that sets Airplane to an empty object, and remove the entire export default line.

You will see an error in the console, but we’ll fix this in the next step.

Hint

Remove these lines of code:

let Airplane = {};

export default Airplane;

**2.**

Modify the availableAirplanes array such that it is a variable defined with let and no longer a property on the Airplane object.

Hint

let availableAirplanes = [{

name: 'AeroJet',

fuelCapacity: 800

},

{name: 'SkyJet',

fuelCapacity: 500

}];

**3.**

Let’s add some more data to the availableAirplanes array.

In the first object, AeroJet, add a property availableStaff, and set it equal to an array with the elements 'pilots', 'flightAttendants', 'engineers', 'medicalAssistance', and 'sensorOperators'.

In the second object, SkyJet, add a property availableStaff, and set it equal to an array with the elements 'pilots' and 'flightAttendants'.

Hint

let availableAirplanes = [{

name: 'AeroJet',

fuelCapacity: 800,

availableStaff: ['pilots', 'flightAttendants', 'engineers', 'medicalAssistance', 'sensorOperators'],

},

{name: 'SkyJet',

fuelCapacity: 500,

availableStaff: ['pilots', 'flightAttendants']

}];

**4.**

Define a new variable with let named flightRequirements, and set it equal to an empty object.

Hint

let flightRequirements = {};

**5.**

Within the flightRequirements object, add a property requiredStaff, and set this equal to 4.

Hint

let flightRequirements = {

requiredStaff: 4,

};

**6.**

Define a function with the name meetsStaffRequirements() that takes availableStaff and requiredStaff as parameters.

Hint

function meetsStaffRequirements(availableStaff, requiredStaff) {

}

**7.**

In the body of the meetsStaffRequirements() function, write logic to check if the length of the availableStaff array is greater than or equal to requiredStaff.

The function should contain this logic:

if length of availableStaff is greater than or equal to requiredStaff

return true

else

return false

Hint

function meetsStaffRequirements(availableStaff, requiredStaff) {

if (availableStaff.length >= requiredStaff) {

return true;

} else {

return false;

}

};

**8.**

Using the export keyword, export the variables availableAirplanes, flightRequirements and meetsStaffRequirements.

Hint

export { availableAirplanes, flightRequirements, meetsStaffRequirements};

let availableAirplanes = [{

  name: 'AeroJet',

  fuelCapacity: 800,

  availableStaff : ['pilots', 'flightAttendants', 'engineers', 'medicalAssistance', 'sensorOperators']

 },

 {name: 'SkyJet',

  fuelCapacity: 500,

  availableStaff: ['pilots','flightAttendants']

 }

 ];

let flightRequirements = {

  requiredStaff: 4

};

function meetsStaffRequirements(availableStaff, requiredStaff){

  if (availableStaff.length >= requiredStaff){

  return true;

  }else {

  return false;

  }

}

export { availableAirplanes, flightRequirements, meetsStaffRequirements};

**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.

**Instructions**

**1.**

Let’s remove any reference to Airplane in our code since we are no longer exporting this module.

For example, Airplane.availableAirplanes should be modified to availableAirplanes.

Again, you will see a ReferenceError in the console for now, but we will fix that in our next step.

Hint

Remove the module Airplane from this line:

function displayFuelCapacity() {

availableAirplanes.forEach(function(element) {

...

});

}

displayFuelCapacity();

**2.**

Change the import statement such that it imports the availableAirplanes, flightRequirements, and meetsStaffRequirements variables.

Now, modify any instance of the Airplane.availableAirplanes variable, so that you only use availableAirplanes.

Hint

import {availableAirplanes, flightRequirements, meetsStaffRequirements} from './airplane';

**3.**

Define a function displayStaffStatus().

Hint

function displayStaffStatus() {

}

**4.**

Within the body of the displayStaffStatus() function, use the forEach to iterate over the availableAirplanes array.

Specifically, the forEach() should take a function as a parameter. The function should in turn take element as a parameter.

Hint

function displayStaffStatus() {

availableAirplanes.forEach(function(element){});

}

**5.**

Within the displayStaffStatus() function, use console.log() to output the element’s name. We’ll add more in the next step.

Hint

function displayStaffStatus() {

availableAirplanes.forEach(function(element) {

console.log(element.name);

});

}

**6.**

Continuing within the displayStaffStatus() function, modify the console.log() statement to output

(element name) + ' meets staff requirements: ' + (true/false)

To do this, we can call the meetsStaffRequirements method, passing in two parameters element.availableStaff and flightRequirements.requiredStaff.

Hint

function displayStaffStatus() {

availableAirplanes.forEach(function(element) {

console.log(element.name + ' meets staff requirements: ' + meetsStaffRequirements(element.availableStaff, flightRequirements.requiredStaff) );

});

}

**7.**

Call the displayStaffStatus() function.

Hint

displayStaffStatus();

import {availableAirplanes, flightRequirements, meetsStaffRequirements} from './airplane';

function displayFuelCapacity() {

  availableAirplanes.forEach(function(element){

    console.log('Fuel Capacity of' + element.name+' : '+ element.fuelCapacity);

  }

  );

}

function displayStaffStatus() {

   availableAirplanes.forEach(function(element){

        console.log(element.name + ' meets staff requirements: ' + meetsStaffRequirements(element.availableStaff, flightRequirements.requiredStaff) );

   });

}

displayStaffStatus();

displayFuelCapacity();

**Export Named Exports**

Named exports are also distinct in that they can be exported as soon as they are declared, by placing the keyword export in front of variable declarations.

In **menu.js**

export let specialty = '';

export function isVegetarian() {

};

function isLowSodium() {

};

1. The export keyword allows us to export objects upon declaration, as shown in export let specialty and export function isVegetarian() {}.
2. We no longer need an export statement at the bottom of our file, since this behavior is handled above.

**Instructions**

**1.**

Let’s add some additional data to our **airplane.js** file.

Continue by adding more data to objects within the availableAirplanes variable.

To the first object AeroJet, add a property maxSpeed with a value of 1200 and a property minSpeed with a value of 300.

To the second object SkyJet, add a property maxSpeed with a value of 800 and a property minSpeed with a value of 200.

Hint

let availableAirplanes = [

{name: 'AeroJet',

fuelCapacity: 800,

availableStaff: ['pilots', 'flightAttendants', 'engineers', 'medicalAssistance', 'sensorOperators'],

maxSpeed: 1200,

minSpeed: 300

},

{name: 'SkyJet',

fuelCapacity: 500,

availableStaff: ['pilots', 'flightAttendants'],

maxSpeed: 800,

minSpeed: 200

}

];

**2.**

Within the flightRequirements object, add a property requiredSpeedRange and set this equal to 700.

Hint

let flightRequirements = {

requiredStaff: 4,

requiredSpeedRange: 700

};

**3.**

Continuing with the same file, add a new function meetsSpeedRangeRequirements() that takes three arguments maxSpeed, minSpeed and requiredSpeedRange.

Hint

function meetsSpeedRangeRequirements(maxSpeed, minSpeed, requiredSpeedRange) {

};

**4.**

Within the meetsSpeedRangeRequirements function, create a variable range, and set it to the difference between maxSpeed and minSpeed.

Hint

function meetsSpeedRangeRequirements(maxSpeed, minSpeed, requiredSpeedRange) {

let range = maxSpeed - minSpeed;

};

**5.**

In the body of the meetsSpeedRangeRequirements() function, create logic to check if the range is greater than the requiredSpeedRange.

The function should contain this logic:

if the `range` is greater than the `requiredSpeedRange`

return true

else

return false

Hint

function meetsSpeedRangeRequirements(maxSpeed, minSpeed, requiredSpeedRange) {

let range = maxSpeed - minSpeed;

if (range > requiredSpeedRange) {

return true;

} else {

return false;

}

};

**6.**

Use export to export the variables as soon as they are declared, and remove the export statement at the bottom of the file.

Hint

export let availableAirplanes = [...];

export let flightRequirements = {...};

export function meetsStaffRequirements(){

...

};

export function meetsSpeedRangeRequirements(){

...

};

export let availableAirplanes = [{

  name: 'AeroJet',

  fuelCapacity: 800,

  availableStaff : ['pilots', 'flightAttendants', 'engineers', 'medicalAssistance', 'sensorOperators'],

  maxSpeed: 1200,

  minSpeed: 300

 },

 {name: 'SkyJet',

  fuelCapacity: 500,

  availableStaff: ['pilots','flightAttendants'],

   maxSpeed: 800,

   minSpeed: 200

 }

 ];

export let flightRequirements = {

   requiredStaff: 4,

   requiredSpeedRange: 700

};

export function meetsStaffRequirements(availableStaff, requiredStaff){

  if (availableStaff.length >= requiredStaff){

  return true;

  }else {

  return false;

  }

}

export function meetsSpeedRangeRequirements(maxSpeed, minSpeed, requiredSpeedRange) {

    let range = maxSpeed - minSpeed;

    if (range > requiredSpeedRange) {

    return true;

    } else {

    return false;

    }

};

# Import Named Imports

To import variables that are declared, we simply use the original syntax that describes the variable name. In other words, exporting upon declaration does not have an impact on how we import the variables.

import { specialty, isVegetarian } from 'menu';

**Instructions**

**1.**

Add meetsSpeedRangeRequirements to the import statement at the top of the file.

Hint

import {availableAirplanes, flightRequirements, meetsStaffRequirements, meetsSpeedRangeRequirements} from './airplane';

**2.**

Define a function displaySpeedRangeStatus().

Hint

function displaySpeedRangeStatus() {

}

**3.**

Within the displaySpeedRangeStatus() function, use the forEach method to iterate over each element in the availableAirplanes array.

Again, the forEach() should take a function as a parameter; this function should take element as a parameter.

Hint

function displaySpeedRangeStatus() {

availableAirplanes.forEach(function(element){});

}

**4.**

Within the displaySpeedRangeStatus() function, use console.log() to output the element’s name.

Hint

function displaySpeedRangeStatus() {

availableAirplanes.forEach(function(element) {

console.log(element.name);

});

}

**5.**

Continuing within the displaySpeedRangeStatus() function, modify the console.log() statement to output a statement like this:

(element name) + 'meets speed range requirements: ' + (true/false)

To do this, we can call the meetsSpeedRangeRequirements method, passing in three parameters element.maxSpeed, element.minSpeed and flightRequirements.requiredSpeedRange.

Hint

function displaySpeedRangeStatus() {

availableAirplanes.forEach(function(element) {

console.log(element.name + ' meets speed range requirements: ' + meetsSpeedRangeRequirements(element.maxSpeed, element.minSpeed, flightRequirements.requiredSpeedRange));

});

}

**6.**

Call the displaySpeedRangeStatus() function.

Hint

displaySpeedRangeStatus();

import {availableAirplanes, flightRequirements, meetsStaffRequirements,meetsSpeedRangeRequirements} from './airplane';

function displayFuelCapacity() {

  availableAirplanes.forEach(function(element){

    console.log('Fuel Capacity of' + element.name+' : '+ element.fuelCapacity);

  }

  );

}

function displayStaffStatus() {

   availableAirplanes.forEach(function(element){

        console.log(element.name + ' meets staff requirements: ' + meetsStaffRequirements(element.availableStaff, flightRequirements.requiredStaff) );

   });

}

function displaySpeedRangeStatus() {

    availableAirplanes.forEach(function(element){

        console.log(element.name + ' meets speed range requirements: ' + meetsSpeedRangeRequirements(element.maxSpeed, element.minSpeed, flightRequirements.requiredSpeedRange));

    });

}

displayStaffStatus();

displayFuelCapacity();

displaySpeedRangeStatus();

**Export as**

Named exports also conveniently offer a way to change the name of variables when we export or import them. We can do this with the as keyword.

Let’s see how this works. In our **menu.js** example

let specialty = '';

let isVegetarian = function() {

};

let isLowSodium = function() {

};

export { specialty as chefsSpecial, isVegetarian as isVeg, isLowSodium };

In the above example, take a look at the export statement at the bottom of the of the file.

1. The as keyword allows us to give a variable name an alias as demonstrated in specialty as chefsSpecial and isVegetarian as isVeg.
2. Since we did not give isLowSodium an alias, it will maintain its original name.

**Instructions**

**1.**

Remove the keyword export in front of each variable name, since we will no longer need it.

Hint

As an example

export function meetsStaffRequirements(availableStaff, requiredStaff) {

...

};

would just be written as

function meetsStaffRequirements(availableStaff, requiredStaff) {

...

};

**2.**

Add an export statement to export the availableAirplanes object as aircrafts, flightRequirements as flightReqs, the meetsStaffRequirements method as meetsStaffReqs and meetsSpeedRangeRequirements as meetsSpeedRangeReqs.

Hint

export { availableAirplanes as aircrafts, flightRequirements as flightReqs, meetsStaffRequirements as meetsStaffReqs, meetsSpeedRangeRequirements as meetsSpeedRangeReqs };

let availableAirplanes = [{

  name: 'AeroJet',

  fuelCapacity: 800,

  availableStaff : ['pilots', 'flightAttendants', 'engineers', 'medicalAssistance', 'sensorOperators'],

  maxSpeed: 1200,

  minSpeed: 300

 },

 {name: 'SkyJet',

  fuelCapacity: 500,

  availableStaff: ['pilots','flightAttendants'],

   maxSpeed: 800,

   minSpeed: 200

 }

 ];

let flightRequirements = {

   requiredStaff: 4,

   requiredSpeedRange: 700

};

function meetsStaffRequirements(availableStaff, requiredStaff){

  if (availableStaff.length >= requiredStaff){

  return true;

  }else {

  return false;

  }

}

function meetsSpeedRangeRequirements(maxSpeed, minSpeed, requiredSpeedRange) {

    let range = maxSpeed - minSpeed;

    if (range > requiredSpeedRange) {

    return true;

    } else {

    return false;

    }

};

export { availableAirplanes as aircrafts, flightRequirements as flightReqs, meetsStaffRequirements as meetsStaffReqs, meetsSpeedRangeRequirements as meetsSpeedRangeReqs };

**Import as**

To import named export aliases with the as keyword, we add the aliased variable in our import statement.

import { chefsSpecial, isVeg } from './menu';

In **orders.js**

1. We import chefsSpecial and isVeg from the Menu object.
2. Here, note that we have an option to alias an object that was not previously aliased when exported. For example, the isLowSodium object that we exported could be aliased with the as keyword when imported: import {isLowSodium as saltFree} from 'Menu';

Another way of using aliases is to import the entire module as an alias:

import \* as Carte from './menu';

Carte.chefsSpecial;

Carte.isVeg();

Carte.isLowSodium();

1. This allows us to import an entire module from **menu.js** as an alias Carte.
2. In this example, whatever name we exported would be available to us as properties of that module. For example, if we exported the aliases chefsSpecial and isVeg, these would be available to us. If we did not give an alias to isLowSodium, we would call it as defined on the Carte module.

**Instructions**

**1.**

Within the body of **missionControl.js**, change each variable to its alias, with the exception of the variable in the import statement.

In the body, change:

* availableAirplanes to aircrafts
* flightRequirements to flightReqs
* meetsStaffRequirements to meetsStaffReqs
* meetsSpeedRangeRequirements to meetsSpeedRangeReqs

You will see an error in the console, but we’ll fix this in the next exercise.

Hint

A function with correct aliases should look like:

function displaySpeedRangeStatus() {

aircrafts.forEach(function(element) {

console.log(element.name' + ' meets speed range requirements: ' + meetsSpeedRangeReqs(element.maxSpeed, element.minSpeed, flightReqs.requiredSpeedRange));

});

}

displaySpeedRangeStatus();

**2.**

Now modify the import statement to import aircrafts, flightReqs, meetsStaffReqs, meetsSpeedRangeReqs.

Hint

import { aircrafts, flightReqs, meetsStaffReqs, meetsSpeedRangeReqs} from './airplane';

import {aircrafts, flightReqs, meetsStaffReqs,meetsSpeedRangeReqs} from './airplane';

function displayFuelCapacity() {

  aircrafts.forEach(function(element){

    console.log('Fuel Capacity of' + element.name+' : '+ element.fuelCapacity);

  }

  );

}

function displayStaffStatus() {

   aircrafts.forEach(function(element){

        console.log(element.name + ' meets staff requirements: ' + meetsStaffReqs(element.availableStaff, flightReqs.requiredStaff) );

   });

}

function displaySpeedRangeStatus() {

    aircrafts.forEach(function(element){

        console.log(element.name + ' meets speed range requirements: ' + meetsSpeedRangeReqs(element.maxSpeed, element.minSpeed, flightReqs.requiredSpeedRange));

    });

}

displayStaffStatus();

displayFuelCapacity();

displaySpeedRangeStatus();

# Combining Export Statements

We can also use named exports and default exports together. In **menu.js**:

let specialty = '';

function isVegetarian() {

};

function isLowSodium() {

};

function isGlutenFree() {

};

export { specialty as chefsSpecial, isVegetarian as isVeg };

export default isGlutenFree;

Here we use the keyword export to export the named exports at the bottom of the file. Meanwhile, we export the isGlutenFree variable using the export default syntax.

This would also work if we exported most of the variables as declared and exported others with the export default syntax.

export let Menu = {};

export let specialty = '';

export let isVegetarian = function() {

};

export let isLowSodium = function() {

};

let isGlutenFree = function() {

};

export default isGlutenFree;

Here we use the export keyword to export the variables upon declaration, and again export the isGlutenFree variable using the export default syntax

While it’s better to avoid combining two methods of exporting, it is useful on occasion. For example, if you suspect developers may only be interested in importing a specific function and won’t need to import the entire default export.

**Instructions**

**1.**

In **airplanes.js**, use the export keyword to export availableAirplanes, flightRequirements, and meetsStaffRequirements as soon as they are declared.

**2.**

Use export default to export meetsSpeedRangeRequirements

export let availableAirplanes = [

{name: 'AeroJet',

 fuelCapacity: 800,

 availableStaff: ['pilots', 'flightAttendants', 'engineers', 'medicalAssistance', 'sensorOperators'],

 maxSpeed: 1200,

 minSpeed: 300

},

{name: 'SkyJet',

 fuelCapacity: 500,

 availableStaff: ['pilots', 'flightAttendants'],

 maxSpeed: 800,

 minSpeed: 200

}

];

export let flightRequirements = {

  requiredStaff: 4,

  requiredSpeedRange: 700

};

export function meetsStaffRequirements(availableStaff, requiredStaff) {

  if (availableStaff.length >= requiredStaff) {

    return true;

  } else {

    return false;

  }

};

export function meetsSpeedRangeRequirements(maxSpeed, minSpeed, requiredSpeedRange) {

  let range = maxSpeed - minSpeed;

  if (range > requiredSpeedRange) {

    return true;

    } else {

    return false;

  }

};

export default meetsSpeedRangeRequirements;

**Combining Import Statements**

We can import the collection of objects and functions with the same data.

We can use an import keyword to import both types of variables as such:

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

import GlutenFree from './menu';

**Instructions**

**1.**

Remove the import statement at the top of **missionControl.js**.

Once you have removed import, change each variable to its original, unaliased name within the rest of the file.

* aircrafts to availableAirplanes
* flightReqs to flightRequirements
* meetsStaffReqs to meetsStaffRequirements
* meetsSpeedRangeReqs to meetsSpeedRangeRequirements

If you see errors in the console, not to worry. We’ll resolve this in our last step!

**2.**

At the top of the file, we’ll now import all variables from the module.

* Use import to import availableAirplanes, flightRequirements, and meetsStaffRequirements between a set of {}
* Use import to import meetsSpeedRangeRequirements

Hint

import { availableAirplanes, flightRequirements, meetsStaffRequirements} from './airplane';

import meetsSpeedRangeRequirements from './airplane';

import { availableAirplanes, flightRequirements, meetsStaffRequirements} from './airplane';

import meetsSpeedRangeRequirements from './airplane';

function displayFuelCapacity() {

  availableAirplanes.forEach(function(element) {

    console.log('Fuel Capacity of ' + element.name + ': ' + element['fuelCapacity']);

  });

}

displayFuelCapacity();

function displayStaffStatus() {

  availableAirplanes.forEach(function(element) {

   console.log(element.name + ' meets staff requirements: ' + meetsStaffRequirements(element.availableStaff, flightRequirements.requiredStaff) );

  });

}

displayStaffStatus();

function displaySpeedRangeStatus() {

  availableAirplanes.forEach(function(element) {

   console.log(element.name + ' meets speed range requirements:' + meetsSpeedRangeRequirements(element.maxSpeed, element.minSpeed, flightRequirements.requiredSpeedRange));

  });

}

displaySpeedRangeStatus();

# Review

We just learned how to use JavaScript modules. Let’s review what we learned:

Modules in Node.js are reusable pieces of code that can be exported from one program and imported for use in another program.

* module.exports exports the module for use in another program.
* require() imports the module for use in the current program.

ES6 introduced a more flexible, easier syntax to export modules:

* default exports use export default to export JavaScript objects, functions, and primitive data types.
* named exports use the export keyword to export data in variables.
* named exports can be aliased with the as keyword.
* import is a keyword that imports any object, function, or data type.