Tip 47: Isolate Functionality with Import and Export

In this tip, you'll learn how to share code between files.

We'll cover the following

- Sharing code between files
- Importing and exporting code
  - Example: Exporting a function
  - Example: Exporting selective functions
  - Example: Importing functions
  - Default export
  - Importing the default function

Sharing code between files

In the bad old days of JavaScript, you kept all code in a single file. Even worse, developers would put all their JavaScript code in the DOM under a single `<script>` tag.

Things got better slowly. First, someone created code to minify and concatenate files so at least you had only one small *import* statement. Then projects such as *Require.js* and *CommonJS* gave developers a way to share code between files using *modules*. With the module system, JavaScript developers were finally able to easily *reuse* code in a project.

Modules have been simplified and are now simple *import* and *export* statements. And with this simple interface, not only can you share code between files in a project, but you can also use community code with nearly identical syntax. You'll see more about community code in the next tip. For now, let's look at how to import and export code.

Importing and exporting code

This code won't work out-of-the box. It's still a good idea to combine and minify your code to a single file. Eventually, browsers will be able to *dynamically* import

code, but for now, you still need to create *single* files, often called **bundles** or **packages**. You'll see how in Tip 50, Use Build Tools to Combine Components.

You've actually been using *exported* code throughout the course. You wouldn't know unless you looked at the courses's source code because the examples hid the code *export*. *Importing* and *exporting* is just that simple. You export any existing code with a *single* statement.

Example: Exporting a function

Here's some code from Tip 36, Prevent Context Confusion with Arrow Functions.

```
const validator = {
    message: 'is invalid.',
    setInvalidMessage: field => `${field} ${this.message}`,
};
```

If you want to share the code, you just need to add a simple `export` statement.

```
const validator = {
    message: 'is invalid.',
    setInvalidMessage: field => `${field} ${this.message}`,
};
export { validator };
```

How does it work? At the most basic level, all you need to do is *export* an object containing the data you want to share. That means you can export *functions, variables, and classes*. And you don't need to export anything. If you choose to export some functions and not others, you've essentially created *public* and *private* functions.

In the preceding example, you exported a single function. In other situations, you may have a function you don't want to share essentially making it *private*. In that situation, export all the functions you're willing to share.

Example: Exporting selective functions

Here's how it would look if you wanted to share two functions while hiding one:

```
function getPower(decimalPlaces) {
    return 10 ** decimalPlaces;
}
function capitalize(word) {
    return word[0].toUpperCase() + word.slice(1);
}
function roundToDecimalPlace(number, decimalPlaces = 2) {
    const round = getPower(decimalPlaces);
    return Math.round(number * round) / round;
}
export { capitalize, roundToDecimalPlace };
```

Now that you've exported the functions, you'll probably want to use them. To use a function in another file, use the `import` keyword and the functions you'd like to import to *curly* braces. After you declare what you're importing, give the *path* relative to the file you're in.

You can also import *library* code and you'll see how in the next tip. For now, you're only importing code from other files you own.

Example: Importing functions

Try importing some utility functions into a new file. It would look like this:

index.js

util.js

All code files are copied to end of the page...

You don't have to import everything. If you want only a single item, that's fine.

index.js

util.js

All code files are copied to end of the page...

And you don't have to limit yourself to functions. You can also export variables and classes.

```
const PI = 3.14;
const E = 2.71828;
export { E, PI };
```

This probably looks familiar.

> Exporting and importing use nearly the same syntax as *destructuring*.

In fact, if you want you keep all your imports as properties on an object, you simply import everything to a variable name.

The syntax is a little different from destructuring. Declare that you're importing all functions using the *asterisks* and then give the variable name. You can now call the functions as if they were on an object.

index.js

util.js

All code files are copied to end of the page...

As with destructuring, you can also rename functions or data you import. The syntax is slightly different. Instead of a *colon*, like you'd use in destructuring, you use the keyword `as` to assign the data to a new variable.

Exports are already simple, but there are a few shortcuts that make things even easier.

Instead of declaring an object and adding each piece of data at the end, you can add the `export` keyword before each function. This makes your code even easier because you don't need an object at the bottom of the file.

```
export function getPower(decimalPlaces) {
    return 10 ** decimalPlaces;
}

export function capitalize(word) {
    return word[0].toUpperCase() + word.slice(1);
}

export function roundToDecimalPlace(number, decimalPlaces = 2) {
    const round = getPower(decimalPlaces);
    return Math.round(number * round) / round;
}
```

Exporting functions one at a time doesn't change how you import. You can use any of the techniques mentioned.

Default export

As you start to separate out your code, you'll often have files that contain a single

As you start to separate out your code, you'll often have files that contain a single entry point. Or you may have a function that's more important. In those situations, you can declare a *default export*. This will make the import process a little shorter.

Consider a file that converts an `address` object to a *string*. The main goal of the utility is to convert an object. There's a clear *default* export. But you may still want to share some helper functions.

Add the keyword `default` after the `export` keyword on `normalize()` to make it the main `export`. Add `export` to any remaining functions.

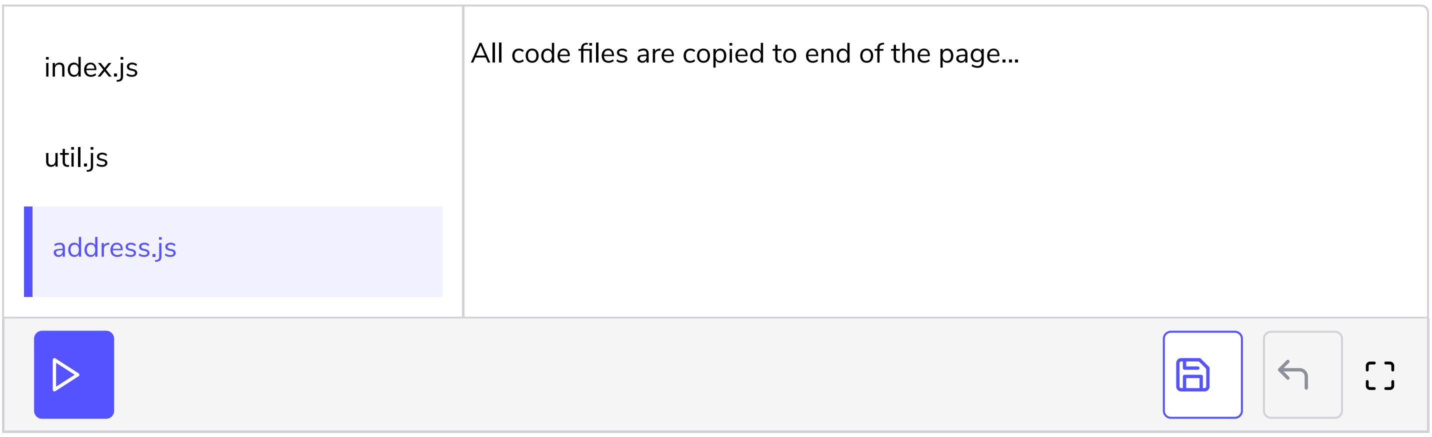
```
import { capitalize } from './util';

export function parseRegion(address) {
    const region = address.state || address.providence || '';
    return region.toUpperCase();
}

export function parseStreet({ street }) {
    return street.split(' ')
        .map(part => capitalize(part))
        .join(' ');
}

export default function normalize(address) {
    const street = parseStreet(address);
    const city = address.city;
    const region = parseRegion(address);
    return `${street} ${city}, ${region}`;
}
```

Importing the default function

Now when you want to import `normalize()`, you use the same syntax but *without* the curly braces. If you don't use curly braces, you'll get the *default* export and nothing else. You don't need to use the exact function name—*you can import the default to any variable name you want*—but it's a good idea to use the *same* name as the default to keep things readable.

index.js	All code files are copied to end of the page...
util.js	
address.js	

If you want to import the default function along with some other functions, you can mix and match `import` statements. Separate the `default` and the *curly* brace import using a *comma*.



All code files are copied to end of the page...

Default imports are particularly useful on classes because there should be only one class per file, so there's no reason to export other code.

```
import { capitalize } from './util';

export default class Address {
    constructor(address) {
        this.address = address;
    }
    normalize() {
        const street = this.parseStreet(this.address);
        const city = this.address.city;
        const region = this.parseRegion(this.address);
        return `${street} ${city}, ${region}`;
    }
    parseStreet({ street }) {
        return street.split(' ')
            .map(part => capitalize(part))
            .join(' ');
    }
    parseRegion(address) {
        const region = address.state || address.providence || '';
        return region.toUpperCase();
    }
}
```

As you can see, imports and exports are so intuitive there's not really much to say. But there is one slight problem—because you can split code easily, your projects will start to grow. Don't worry, though. As your project grows, you can split code into different files. This will let you organize code more efficiently and logically. In Tip 49, Build Applications with Component Architecture, you'll learn one way to organize code. But before you get to that point, you'll almost certainly want to use code outside your own codebase. Fortunately, that's easier than ever.

In the next tip, you'll learn how to use community code with `npm`.

Code Files Content !!!

⟱ ⟱

```
--------------------------------------------------------------------------
|  index.js [1]
--------------------------------------------------------------------------



import { capitalize, roundToDecimalPlace } from './util';

function giveTotal(name, total) {
    return `${capitalize(name)}, your total is: ${roundToDecimalPlace(total)}`;
}
console.log(giveTotal('sara', 10.3333333));

export { giveTotal };



--------------------------------------------------------------------------
|  util.js [1]
--------------------------------------------------------------------------



function getPower(decimalPlaces) {
    return 10 ** decimalPlaces;
}
function capitalize(word) {
    return word[0].toUpperCase() + word.slice(1);
}
function roundToDecimalPlace(number, decimalPlaces = 2) {
    const round = getPower(decimalPlaces);
    return Math.round(number * round) / round;
}
export { capitalize, roundToDecimalPlace };






***********************************************************************************



--------------------------------------------------------------------------
|  index.js [2]
--------------------------------------------------------------------------



import { capitalize } from './util';

function greet(name) {
    return `Hello, ${capitalize(name)}!`;
}
```

```
console.log(greet('ashley'));

export { greet };


-------------------------------------------------------------------------
|  util.js [2]
-------------------------------------------------------------------------


function getPower(decimalPlaces) {
    return 10 ** decimalPlaces;
}
function capitalize(word) {
    return word[0].toUpperCase() + word.slice(1);
}
function roundToDecimalPlace(number, decimalPlaces = 2) {
    const round = getPower(decimalPlaces);
    return Math.round(number * round) / round;
}
export { capitalize, roundToDecimalPlace };





**********************************************************************************



-------------------------------------------------------------------------
|  index.js [3]
-------------------------------------------------------------------------


import * as utils from './util';

function greet(name) {
    return `Hello, ${utils.capitalize(name)}!`;
}
console.log(greet('ashley'));

export { greet };


-------------------------------------------------------------------------
|  util.js [3]
-------------------------------------------------------------------------


function getPower(decimalPlaces) {
    return 10 ** decimalPlaces;
}
function capitalize(word) {
    return word[0].toUpperCase() + word.slice(1);
}
function roundToDecimalPlace(number, decimalPlaces = 2) {
    const round = getPower(decimalPlaces);
    return Math.round(number * round) / round;
}
export { capitalize, roundToDecimalPlace };
```

```
**********************************************************************************



-------------------------------------------------------------------------
|  index.js [4]
-------------------------------------------------------------------------


import normalize from './address';

function getAddress(user) {
    return normalize(user.address);
}

export default getAddress;


-------------------------------------------------------------------------
|  util.js [4]
-------------------------------------------------------------------------


function getPower(decimalPlaces) {
    return 10 ** decimalPlaces;
}
function capitalize(word) {
    return word[0].toUpperCase() + word.slice(1);
}
function roundToDecimalPlace(number, decimalPlaces = 2) {
    const round = getPower(decimalPlaces);
    return Math.round(number * round) / round;
}
export { capitalize, roundToDecimalPlace };


-------------------------------------------------------------------------
|  address.js [4]
-------------------------------------------------------------------------


import { capitalize } from './util';

export function parseRegion(address) {
    const region = address.state || address.providence || '';
    return region.toUpperCase();
}

export function parseStreet({ street }) {
    return street.split(' ')
        .map(part => capitalize(part))
        .join(' ');
}

export default function normalize(address) {
    const street = parseStreet(address);
    const city = address.city;
    const region = parseRegion(address);
    return `${street} ${city}, ${region}`;
}
```

```
*********************************************************************************



-------------------------------------------------------------------------
|  index.js [5]
-------------------------------------------------------------------------



import normalize, { parseRegion } from './address';

function getAddress(user) {
    return normalize(user.address);
}

export function getAddressByRegion(users) {
    return users.reduce((regions, user) => {
        const { address } = user;
        const region = parseRegion(address);
        const addresses = regions[region] || [];
        regions[region] = [...addresses, normalize(address)];
        return regions;
    }, {});
}

const bars = [
    {
        name: 'Saint Vitus',
        address: {
            street: '1120 manhattan ave',
            city: 'Brooklyn',
            state: 'NY',
        },
    },
];
console.log(getAddressByRegion(bars));


-------------------------------------------------------------------------
|  util.js [5]
-------------------------------------------------------------------------



function getPower(decimalPlaces) {
    return 10 ** decimalPlaces;
}
function capitalize(word) {
    return word[0].toUpperCase() + word.slice(1);
}
function roundToDecimalPlace(number, decimalPlaces = 2) {
    const round = getPower(decimalPlaces);
    return Math.round(number * round) / round;
}
export { capitalize, roundToDecimalPlace };


-------------------------------------------------------------------------
|  address.js [5]
-------------------------------------------------------------------------
```

```
import { capitalize } from './util';

export function parseRegion(address) {
    const region = address.state || address.providence || '';
    return region.toUpperCase();
}

export function parseStreet({ street }) {
    return street.split(' ')
        .map(part => capitalize(part))
        .join(' ');
}

export default function normalize(address) {
    const street = parseStreet(address);
    const city = address.city;
    const region = parseRegion(address);
    return `${street} ${city}, ${region}`;
}




*********************************************************************************
```