Make a Node.js Package Using Class by Reference

by

Christopher Andrew Topalian

Christopher Topalian

Dedicated to God the Father

College of Scripting Music & Science

CollegeOfScripting.weebly.com

How to Make a Node.js Package

Let's make a package of functions for us to use in all of our projects, which we will install system wide on our computer. Later, we can even choose to upload our package to the internet and share it with the world.

By making a package of functions we can update our package anytime that we want and all of our projects will utilize the same updated package of functions!

This makes programming much easier, because we only have to make the package one time, instead of over and over again.

This allows us to be much more productive, because the package can easily be imported into any folder using npm link nameOfPackage

Let's make a package named cos

On the next pages we walk through creating the package that we name cos

cos stands for College of Scripting :-)

Class by Reference

Namespace

Class by Reference is a way to create a namespace for our functions to avoid name conflicts with other packages.

The static methods of our custom class named OurMath allow us to organize our functions in this single class without creating instances. The static methods can be accessed directly on the class itself without needing to create an object instance.

No Instances and No State

By using static methods there's no need to create an object instance of OurMath. In this Class by Reference style of programming that we are using, the class doesn't maintain any state.

Using Static methods is a way to group related functions together in a way that's easy to access, but without needing to instantiate an object to use them. It creates a namespace that allows us to work with our package of functions without function name conflicts with other packages.

How to Create a Node.js Package

* We make a folder, named project1

* In project1 folder we make a folder named cos

Naming Our Package

When naming packages, we never use:

* uppercase letters
* underscores
* commas
* special chars (!@#$%^&*())

Instead, we use

* lowercase only
* hyphens or dots

So, we can choose to name it,

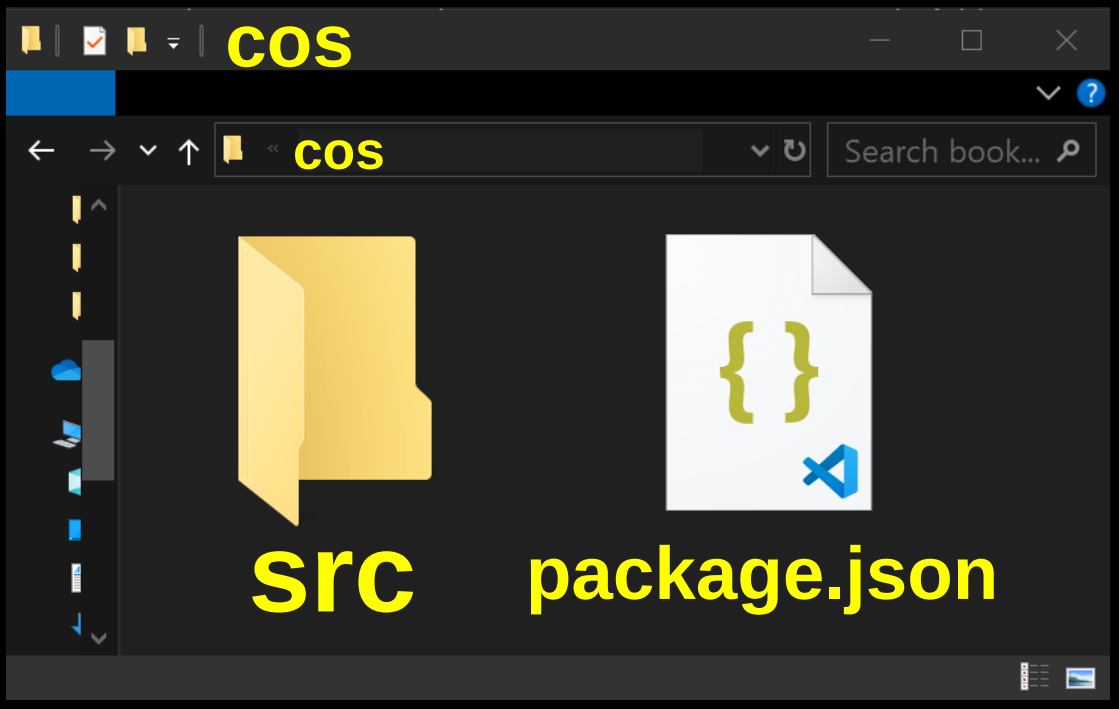
our-package or ourpackage

but NOT ourPackage or our_package

We can choose to use dots, such as:

our.package or ourpackage.001

In the cos folder, we make a folder named src



In the cos folder, we also make a file named package.json

We have multiple ways to make the package.json file.

But, instead of using the npm method of making a package.json file using npm init let us create the file manually, without using the npm way, as shown on the next page.

package.json has meta data that is used to Install and Distribute Our Package

We create a package.json file, which defines our package and its metadata. This file is essential for distributing and installing our package across projects.

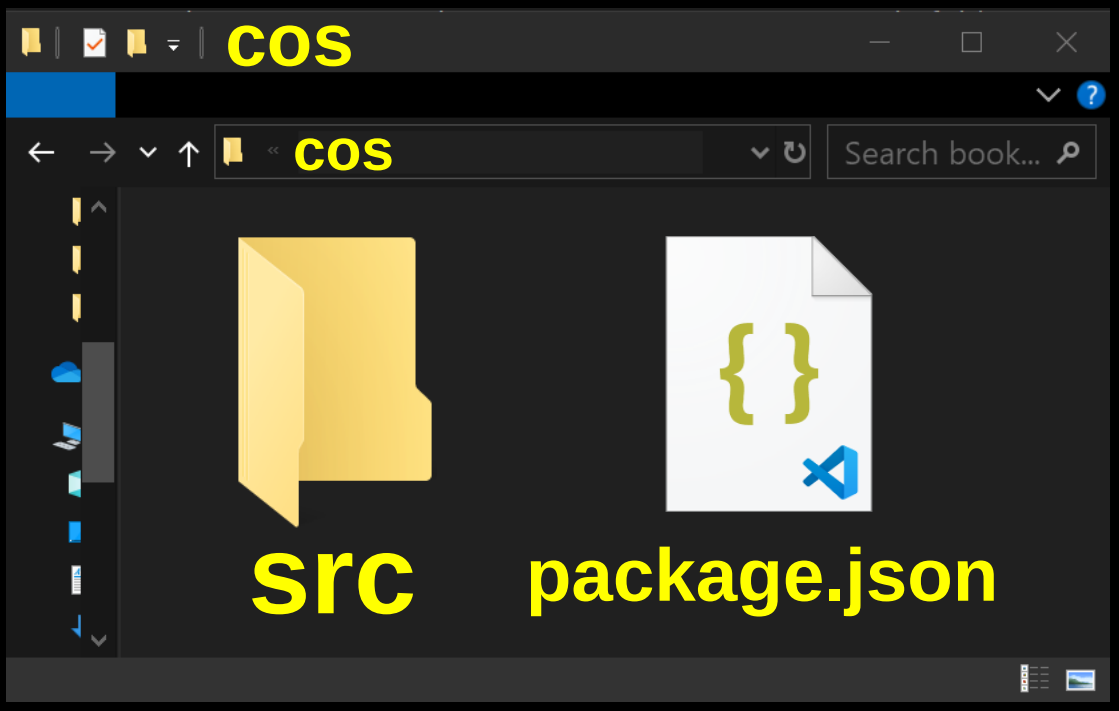
First, we make a new text file in VSCode Editor and type the script that we see on the next page and save it as package.json

We make sure to save the package.json file inside of the cos folder.

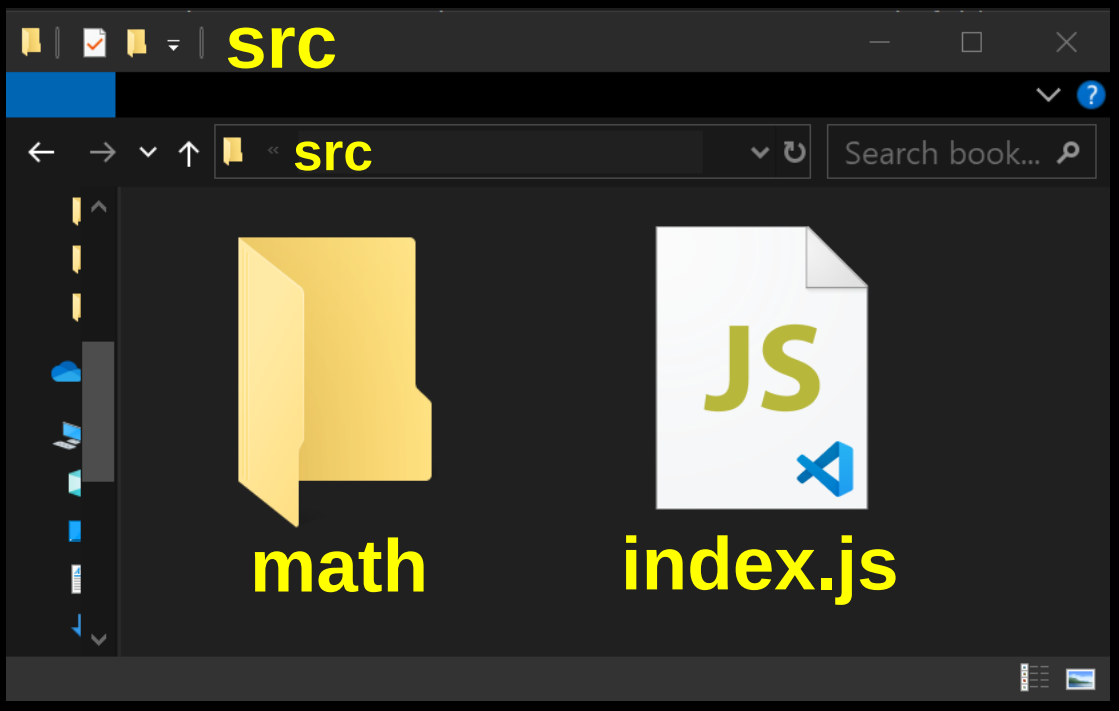
NEXT PAGE SHOWS package.json

```
{
    "name": "cos",
    "version": "1.0.0",
    "description": "Our Package",
    "main": "src/index.js",
    "scripts": {
        "test": "echo \"Error: no test
specified\" && exit 1"
    },
    "author": "Christopher Andrew
Topalian",
    "license": "",
    "files": [
        "src"
    ]
}
```

We have a src folder and a package.json file inside of our cos folder as shown here:



Inside of our src folder, we make a folder named math and we make an index.js file



On the next page, we show the index.js file code.

We Type the Code as we see it on the next page, and save it as index.js

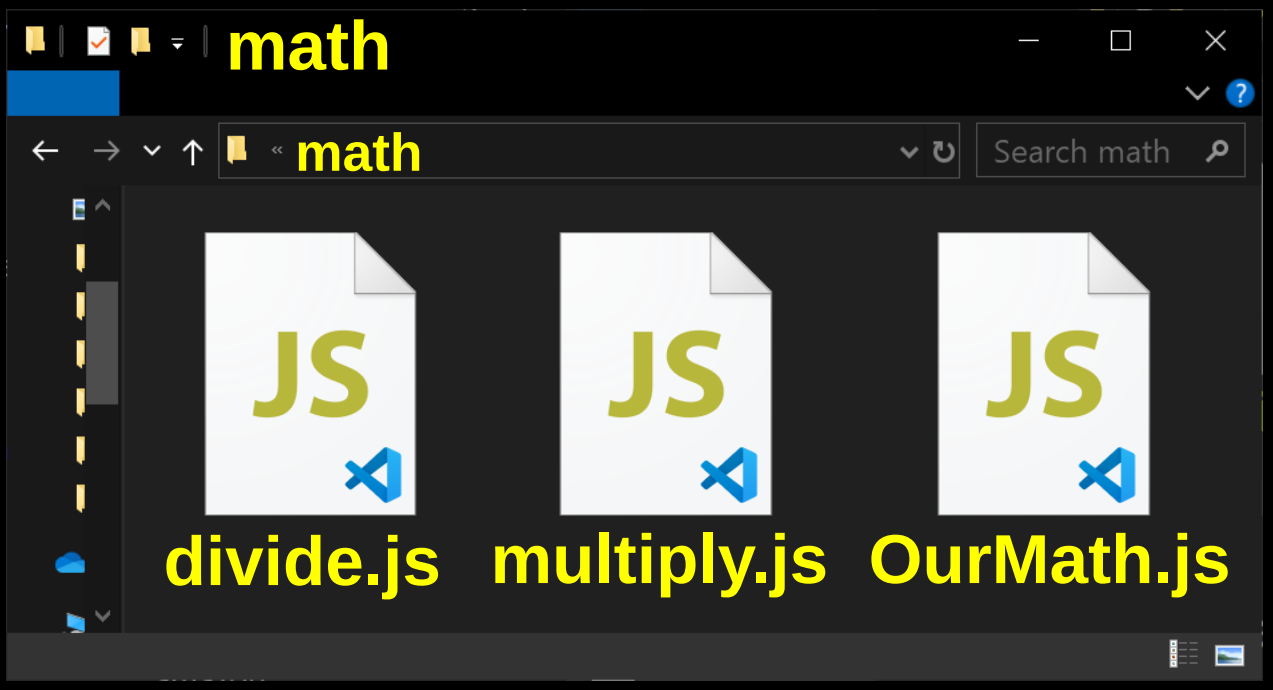
We make sure that this index.js file is inside of the src folder.

This index.js file will reference the functions that we place in the math folder.

SEE NEXT PAGE for: index.js

```
// index.js

const OurMath = require('./math/OurMath');

module.exports = OurMath;

//----//

// Dedicated to God the Father
// All Rights Reserved Christopher Andrew Topalian
Copyright 2000-2024
// https://github.com/ChristopherTopalian
// https://github.com/ChristopherAndrewTopalian
// https://sites.google.com/view/CollegeOfScripting
```

Inside of our math folder, we make three .js files named divide.js, multiply.js and OurMath.js



Thus, the two function files we are adding to our package are: divide.js and multiply.js

The class file we are adding references our divide and multiply functions.

On the next page, we see the function script that we save as multiply.js

NEXT PAGE SHOWS: multiply.js

```
// multiply.js

function multiply(a, b)
{
    return a * b;
}

module.exports = multiply;

//----//

// Dedicated to God the Father
// All Rights Reserved Christopher Andrew Topalian
Copyright 2000-2024
// https://github.com/ChristopherTopalian
// https://github.com/ChristopherAndrewTopalian
// https://sites.google.com/view/CollegeOfScripting
```

On the next page, we see the function script that we save as divide.js

NEXT PAGE SHOWS: divide.js

```
// divide.js

function divide(a, b)
{
    return a / b;
}

module.exports = divide;


//----//

// Dedicated to God the Father
// All Rights Reserved Christopher Andrew Topalian Copyright 2000-2024
// https://github.com/ChristopherTopalian
// https://github.com/ChristopherAndrewTopalian
// https://sites.google.com/view/CollegeOfScripting
```

On the next page, we see the class script that we save as OurMath.js

NEXT PAGE SHOWS: OurMath.js

```
// OurMath.js

const divide = require('./divide');
const multiply = require('./multiply');

class OurMath
{
    static divide(a, b)
    {
        return divide(a, b);
    }

    static multiply(a, b)
    {
        return multiply(a, b);
    }
}

module.exports = OurMath;

//----//
```

// Dedicated to God the Father

// https://github.com/ChristopherTopalian
// https://github.com/ChristopherAndrewTopalian
// https://sites.google.com/view/CollegeOfScripting

How to Install Our Package Computer Wide

We Type cmd into the cos folder address bar and press Enter

This opens our cos folder in the command prompt

We type in the Command prompt:

npm link

press Enter

We have Installed Our Package System Wide!

Now, that we have used

npm link

which made our package available system wide, we can now easily link to our package from any of our Node.js project folders by using

npm link cos

This makes creating and updating our applications very easy :-)

The next page shows how to use our custom package.

To use our package in another folder, we first make a new folder anywhere else on our computer and name our folder project2

In the address bar of project2 folder,
we type
cmd
press Enter

This opens the command prompt.

We type in the command prompt
npm link cos
press Enter

This will install the node_modules link folder of our package named cos into our project2 folder.

This way we can use the functions from our package in our project2 folder project.

Any time that we change a function in our cos package or add a function, those changes will now be useable in our project2 folder too.

This saves a lot of work, since now we only have to change the package of functions one time, but it allows us to use the package in as many projects as we want with the updated changes, just by typing in the address bar of any folder:
npm link cos
and pressing Enter

We make a new script in VSCode and save it as, usesOurPackage.js

We save usesOurPackage.js in a folder that we make any where on our computer. We named this new folder project2 to make things easy.

Thus, inside of project2 folder, we have the usesOurPackage.js file.

The code to the usesOurPackage.js file is on the next page.

We make sure to first link to our package by using npm link cos in project2 folder to be able to use the functions from our package named cos.

NEXT PAGE SHOWS usesOurPackage.js

```
// usesOurPackage.js

const OurMath = require('cos');

console.log(OurMath.divide(10, 5));
console.log(OurMath.multiply(4, 4));

//-//

process.stdin.resume();
process.stdin.once('data', function()
{
    process.exit();
});

//----//

// Dedicated to God the Father
// All Rights Reserved Christopher Andrew Topalian
Copyright 2000-2024
// https://github.com/ChristopherTopalian
// https://github.com/ChristopherAndrewTopalian
// https://sites.google.com/view/CollegeOfScripting
```

Updating Our Package is Easy

We type cmd into the cos folder address bar and press Enter

This opens our cos folder in the command prompt

We type in the Command prompt:

npm link

press Enter

We can show what package links are present on our computer system using:

```
npm list -g --link
```

We can unlink a package system wide using:

```
npm unlink -g cos
```

To unlink the package locally from a project folder, we can use:

```
npm unlink cos
```

Christopher Topalian

How to Paste Code from a PDF that has Junk Characters.

How to Paste Code from a PDF that has Junk Characters.

When we paste from a pdf into VSCode, it might look like this:

```
function combineJSFiles(directory, scriptFilename)
{
    let outputFilePath = path.join(directory, 'main.js');

    let fileContents = [];
```

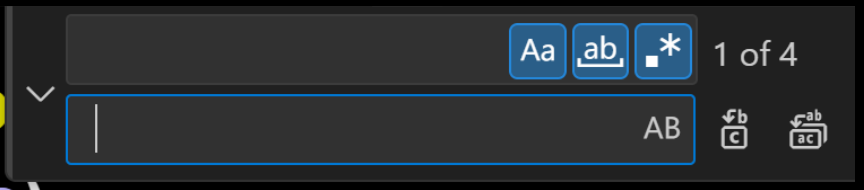
We can't leave those junk characters in the code, so we remove them with find/replace.

We Find 1 of the spaces.
We Replace All with the 1 space that we typed.
This gets rid of the junk characters in the code.

We highlight 1 space with our mouse arrow:

```
function combineJSFiles(directory, scriptFilename)
{
    let outputFilePath = path.join(directory, 'main.js');

    let fileContents = [];
```

We press Control + H to open the Find/Replace feature and Replace All with our own Space




```
function comb
scriptFilename)
{
    let outputFilePath = path.join(directory, 'main.js');

    let fileContents = [];
```

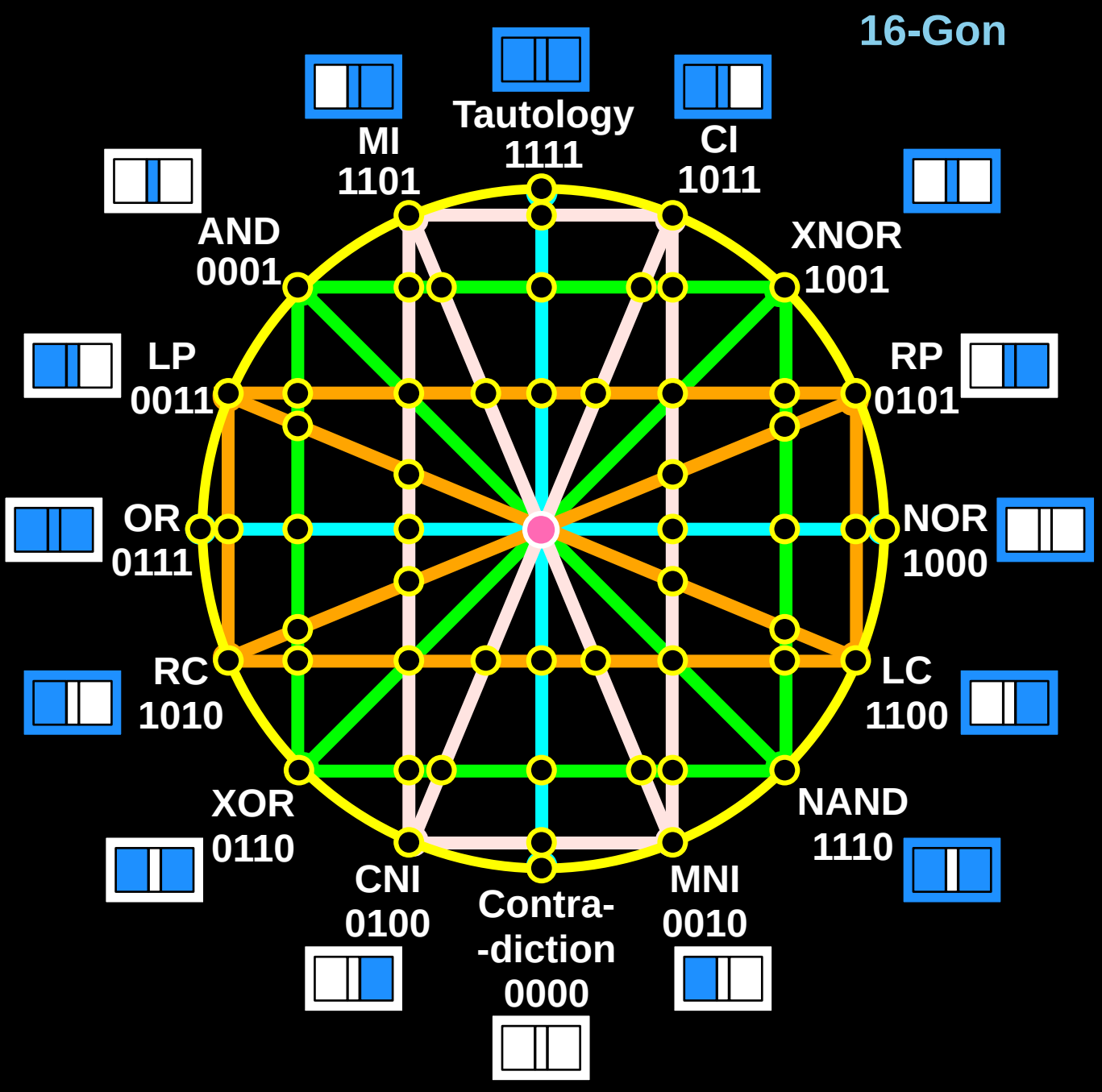
Here we see that the Find/Replace All has replaced the junk characters with our working spaces instead:

```
function combineJSFiles(directory, scriptFilename)
{
    let outputFilePath = path.join(directory, 'main.js');

    let fileContents = [];
```

Now that the code has no junk characters, it can run.

Christopher Topalian

True Artificial Intelligence System



For More Tutorials:

CollegeOfScripting.weebly.com

CollegeOfScripting.wordpress.com

GitHub.com/ChristopherTopalian

GitHub.com/ChristopherAndrewTopalian

Youtube.com/ScriptingCollege

Twitter.com/CollegeOfScript

Rumble.com/user/CollegeOfScripting

Sites.google.com/view/CollegeOfScripting

Dedicated to God the Father

**This book is created by the
College of Scripting Music & Science.**

Always remember, that each time you write a script with a pencil and paper, it becomes imprinted so deeply in memory that the material and methods are learned extremely well. When you Type the scripts, the same is true.

The more you type and write out the scripts by keyboard or pencil and paper, the more you will learn programming!

**Write & Type EVERY example that you find.
Keep all of your scripts organized.
Every script that you create increases your programming abilities.**

**SEEING CODE, is one thing,
but WRITING CODE is another.
Write it, Type it, Speak it, See it, Dream it.**

www.CollegeOfScripting.weebly.com