
Go Over the Edge at
FILES FALLS

LEVEL 3

FILES FALLS

OBJECTIVE: TRAIN SYSTEM STATUS

We want to print a list of running and stopped trains for passengers

WHICH TRAINS ARE RUNNING?

We need a printout for each train since each train number is unique

```
> "Train #" + 1 + " is running."
```

→"Train #1 is running."

```
> "Train #" + 2 + " is running."
```

→"Train #2 is running."

```
> "Train #" + 3 + " is running."
```

→"Train #3 is running."

And on and on and on . . . wow, this sucks with significance.

RUNNING JAVASCRIPT IN AN HTML FILE

Embedding code that signals which JavaScript file to use

The src signals which file our runnable JS code is in. In this case, let's call our file trains.js.

index.html

```
<html>
<header>
...
</header>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
 <script src="trains.js"></script>
</body>
</html>
```

These dots just mean that a bunch of html code exists here to handle other parts of the website.

The <script> tag says "Hey, we want some JavaScript code executed!"

Remember to turn off your 'script' tag!

SO NOW, WHAT IS THAT TRAINS.JS FILE?

Building a file of JavaScript code that can be used repeatedly by our site

You can use your favorite simple text-based editor to make any .js file.

trains.js

```
Our JavaScript code
for printing the
running trains will
go in here!
```

Inside the file, we write the JavaScript code that we want to be executed when the index.html file reaches our script tag.

WHERE DO THESE FILES GO?

Adding our files to an appropriate location

root/

index.html

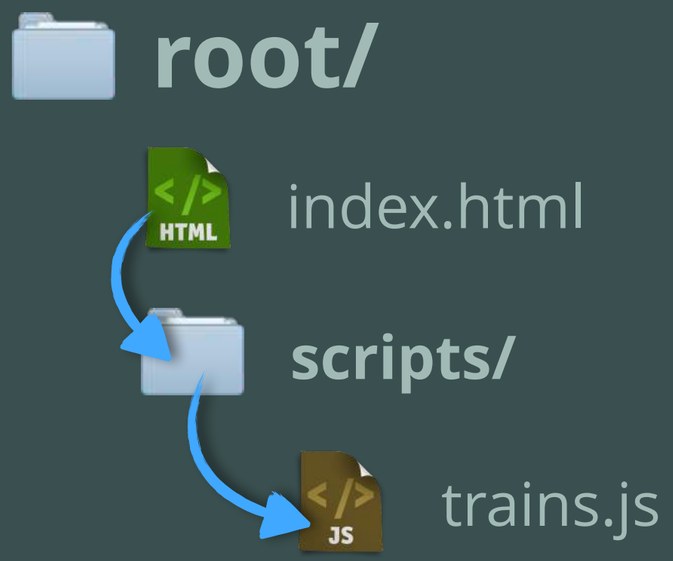
trains.js

So that our `src = "trains.js"` code works correctly, we'd need to place our trains.js file in the same directory as the index.html file.

```
<html>
<header>
...
</header>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
  <script src="trains.js"></script>
</body>
</html>
```


STAYING ORGANIZED AS YOU CODE

Many websites will keep all scripts in descriptive locations

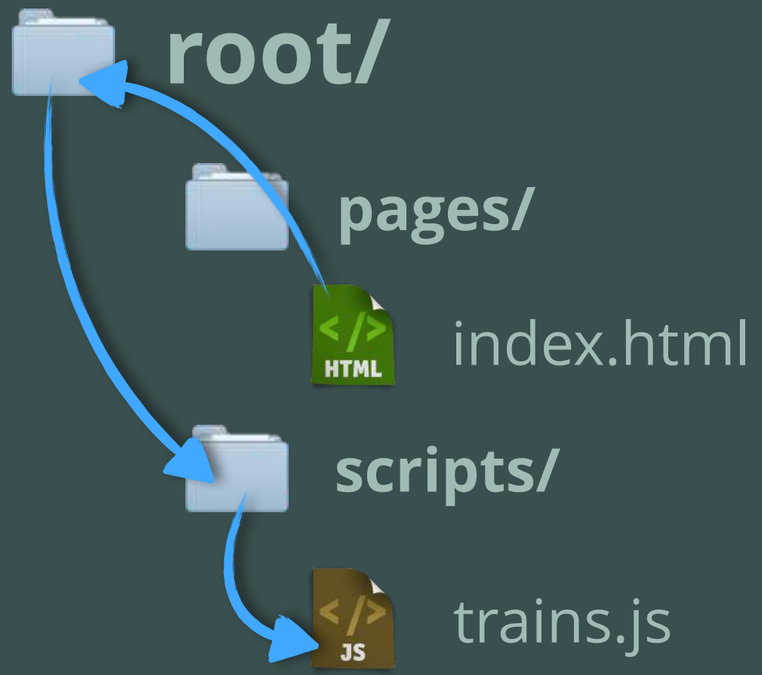



```
<html>
<header>
...
</header>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
  <script src="scripts/trains.js"></script>
</body>
</html>
```

This syntax says to `index.html`, first go down to the `scripts` folder, and there you'll find the train.js file.

LINK SYNTAX FOR DISTANT FILES

Staying organized means we'd have to be more detailed on our src links.




```
<html>
<header>
...
</header>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
  <script src="../scripts/trains.js"></script>
</body>
</html>
```

The double-dot syntax indicates to "go up a folder." This first takes the path back to the root folder, and then takes it down to the scripts folder, where our train.js is found.

TRYING SOME CODE IN A FILE

Let’s try executing a few console-style expressions.

trains.js

```
"Train #" + 1 + " is running."
"Train #" + 2 + " is running."
"Train #" + 3 + " is running."
```

Something's up! The compiler doesn't understand what we've placed in our file in the same way that the console does.

→ ERROR

BUILDING STATEMENTS IN FILES

We need a way to differentiate between executable statements

So here's our Console Entry:

```
> let trainsOperational = 8
```

But guess what? If we enter multiple console statements to execute in one file...

```
let trainsOperational = 8
trainsOperational = trainsOperational + 4
"There are " + trainsOperational + " running trains."
```

...it is interpreted by the compiler pretty much like this utter nonsense:

```
let trainsOperational = 8trainsOperational = trainsOperational +
4"There are " trainsOperational + " running trains."
```


ENTER SEMICOLONS!

We like semicolons; they do not suck.

Console Entry

```
> let trainsOperational = 8
```

File Entry

```
let trainsOperational = 8;
```

Add a semicolon in files to tell the compiler where statements end.

Multiple executable statements are all separated by semicolons

wootForSemicolons.js

```
let trainsOperational = 8;
trainsOperational = trainsOperational + 4;
"There are " + trainsOperational + " running trains.";
```

No error, but now our output is blank? No printed string? What's up?

→ (blank) ❌

PRINTING FROM A FILE TO THE CONSOLE

The console.log() method outputs results of code operations in files

```
let totalTrains = 12;
let trainsOperational = 8;
```

Place expression inside the enclosing parentheses of the console.log() method

```
console.log("There are " + trainsOperational + " running trains.");
```

```
There are 8 running trains.
```

Notice now that we get the actual String contents, with no quote notation.

```
console.log(trainsOperational == totalTrains);
```

The console.log() returns the results of any expression we want printed.

```
false
```

USING CONSOLE.LOG( ) IN TRAINS.JS

Now we can print out every running train.

trains.js

```
console.log("Train #" + 1 + " is running.");
console.log("Train #" + 2 + " is running.");
console.log("Train #" + 3 + " is running.");
console.log("Train #" + 4 + " is running.");
console.log("Train #" + 5 + " is running.");
console.log("Train #" + 6 + " is running.");
console.log("Train #" + 7 + " is running.");
console.log("Train #" + 8 + " is running.");
```

```
Train #1 is running.
Train #2 is running.
Train #3 is running.
Train #4 is running.
Train #5 is running.
Train #6 is running.
Train #7 is running.
Train #8 is running.
```

Combine semicolons to separate executable statements, and the console.log() method to get results printed to the console, and we have a winner!...sort of.