

Intro to JavaScript

Ryan Collins

A short history

JavaScript first appeared in 1995, it's main purpose was to handle some input validation that was run server side with Perl. A round trip was needed to validate the data and Netscape Navigator wanted to change that with the introduction of JavaScript.

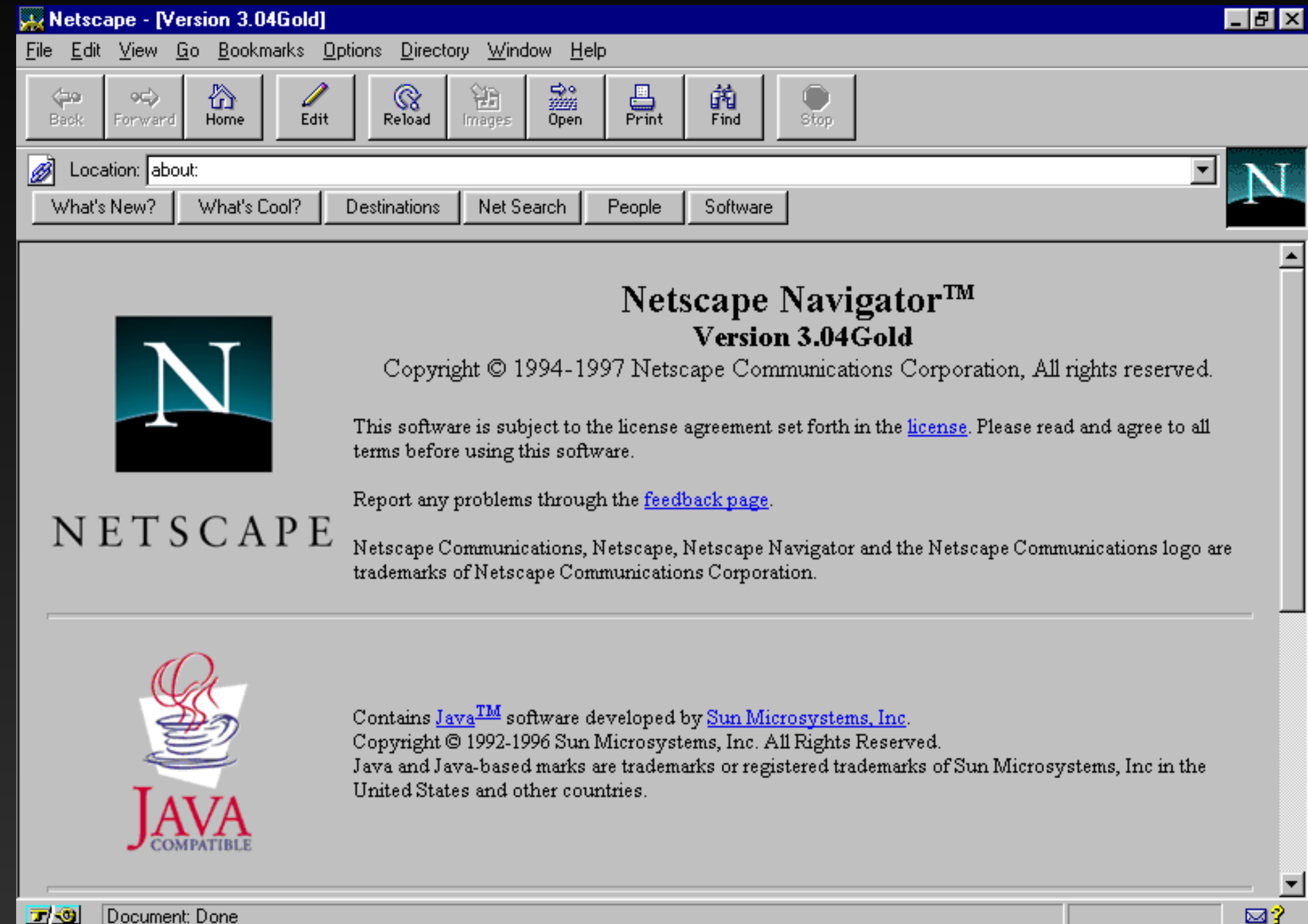
In 1995 a Netscape developer named Brendan Eich began developing a scripting language named Moca(later named LiveScript).

Netscape entered an alliance with Sun Microsystems and changed the name to JavaScript. Because at the time, there was a buzz in the press around Java.

In 1996 Microsoft introduced Internet Explorer 3 with a JavaScript implementation named JScript.

In 1997 after the Microsoft excitement, JavaScript 1.1 was submitted to the ECMA, to standardize the syntax and semantics of a general purpose language.

The following year ISO/IEC adopted ECMAScript as a standard.



Naming Conventions

Always use the same conventions for all your code

- Variable and function names use camelCase
- Global variables written in UPPERCASE
- Constants (const `PI = 3.14;`) in UPPERCASE
- HTML5 attributes can start with data (data-quantity, data-price)
- CSS uses hyphens in property names (font-size)
- Underscores (date_of_birth)

Do not start names with a \$ sign, this will conflict with JavaScript library names

Adding JavaScript

To a webpage

In-Line, is placing JavaScript code directly in the HTML document between two `<script></script>` tags.

The first tag `<script>` tag declares that the following script will be executed.

```
<html>
```

```
  <script type="text/javascript">
```

```
    alert("Hi there!");
```

```
  </script>
```

```
</html>
```

Copy and save this sample code as a .html file and open in your browser, you will get the following

This page says

Hi there!

OK

[Check sample 1](#)

Internal JavaScript

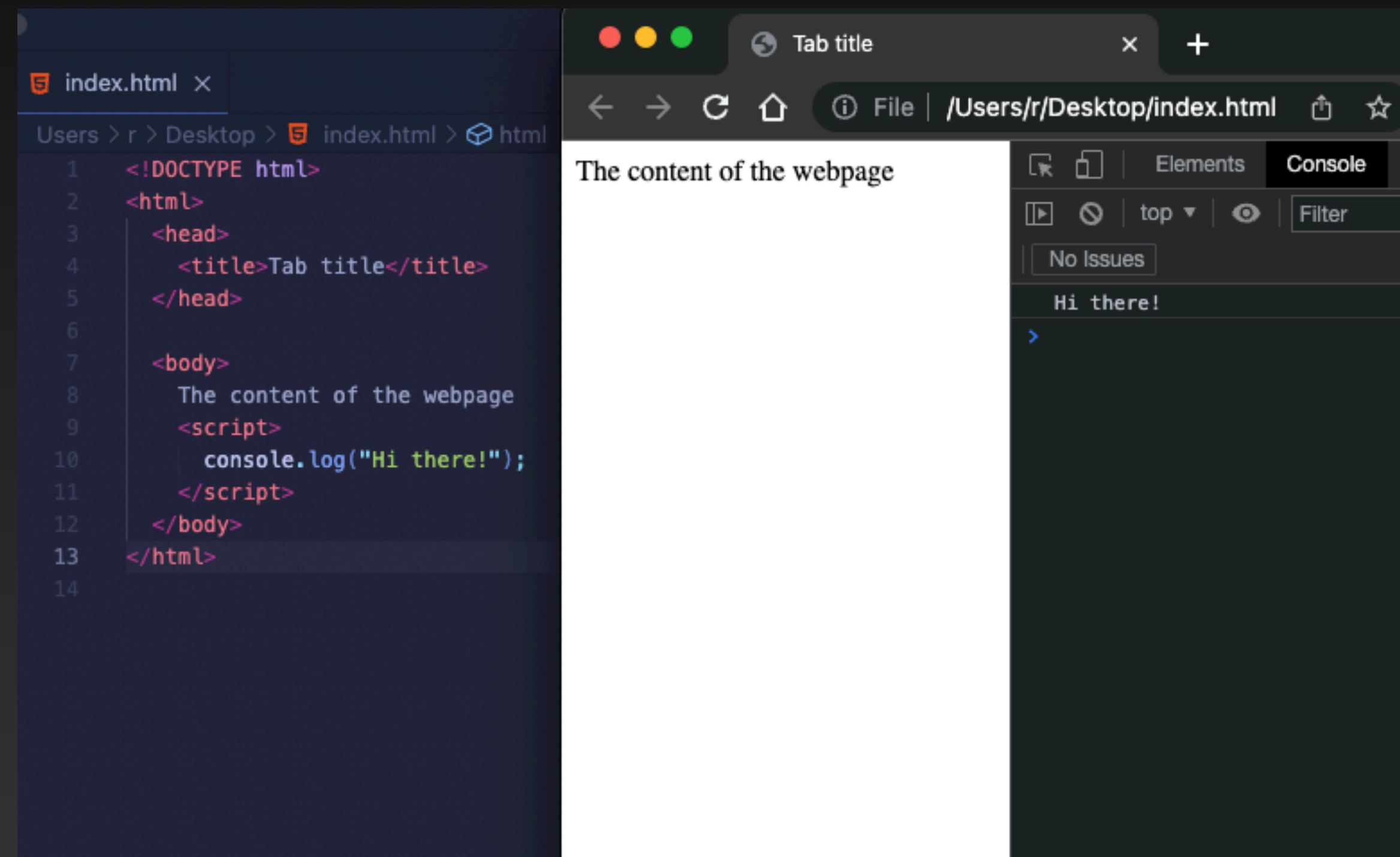
The previous method used in-line javascript, that is not really the best practice.

Another way is to add additional elements inside <html> - <head> and <body>

In the head element we write metadata, and later connect external files to our HTML file.

At the bottom of the document, we can place a script tag,

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Tab title</title>
    <script>
      console.log("Hi there!");
    </script>
  </head>
  <body>
    Content of the webpage
  </body>
</html>
```



External file

Linking to an external file is considered best practice, as it organizes your code better and help avoiding really long HTML pages due to the extra JavaScript code.

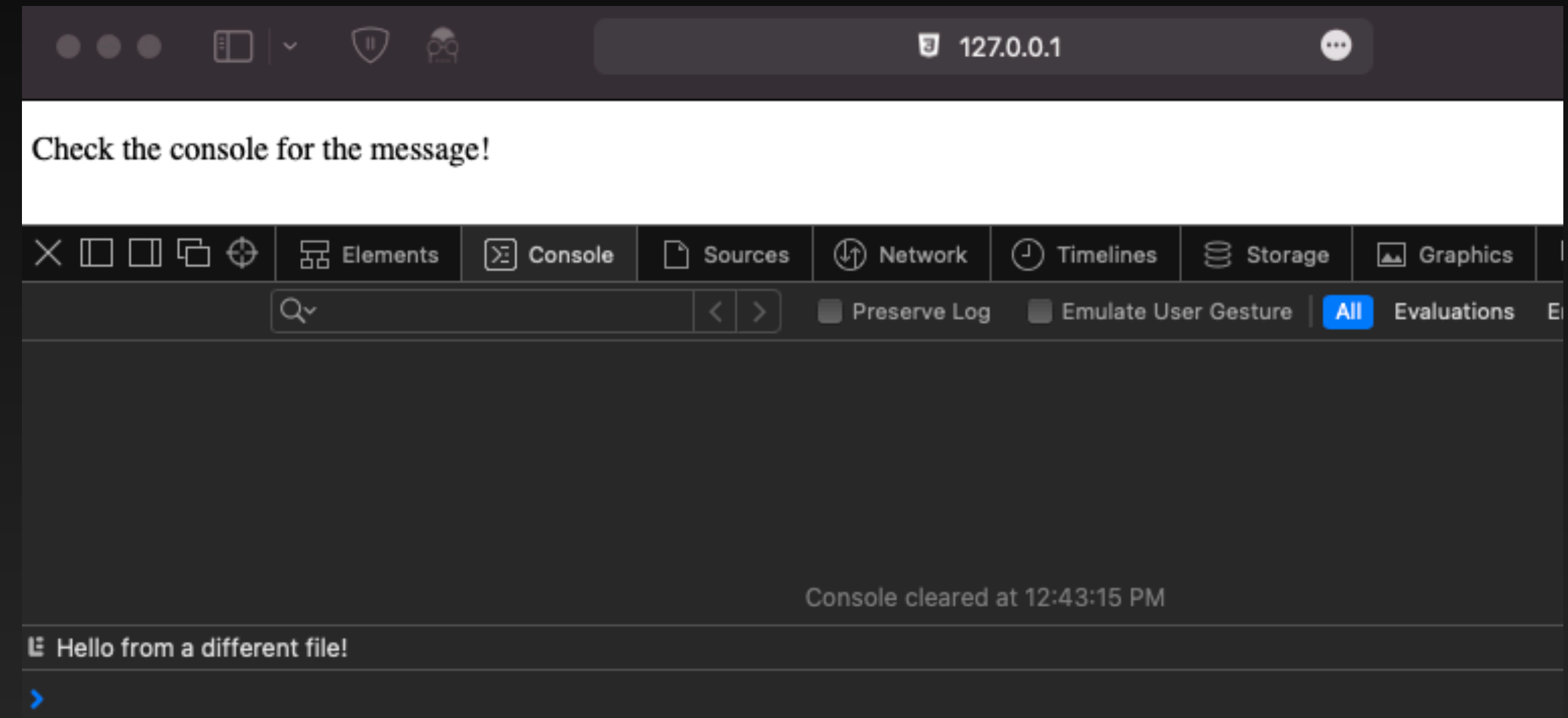
First you create an external JavaScript file containing the postfix .js

Adding the following link to the external file.

```
<html>
```

```
  <script type="text/javascript"  
    src="scripts/hello.js"></script>
```

```
</html>
```



Check the external-js-file source code

Formatting code

So how do you actually write JavaScript code?

There are a few important concepts to keep in mind, like code format.

We have to use the right indentation, semicolons and comments, for our code

If we don't format our code well, we will have really long lines and it will be hard to read. For example, comments;

```
// I'm a single comment
```

```
/*I'm a multi
```

```
line comment*/
```

Without new lines:

```
let status = "new"; let scared = true; if (status === "new") { console.log("Welcome to JavaScript!"); } if (scared) { console.log("Don't worry you will be fine!"); } else { console.log("You're brave! You are going to do great!"); } } else { console.log("Welcome back, I knew you'd like it!"); } }
```

With new lines but without indentation:

```
let status = "new";
let scared = true;
if (status === "new") {
  console.log("Welcome to JavaScript!");
  if (scared) {
    console.log("Don't worry you will be fine!");
  } else {
    console.log("You're brave! You are going to do great!");
  }
} else {
  console.log("Welcome back, I knew you'd like it!");
}
```

With new lines and indentation:

```
let status = "new";
let scared = true;
if (status === "new") {
  console.log("Welcome to JavaScript!");
  if (scared) {
    console.log("Don't worry you will be fine!");
  } else {
    console.log("You're brave! You are going to do great!");
  }
} else {
  console.log("Welcome back, I knew you'd like it!");
}
```

Let's start some theory

Variables

Variables are containers that store values. You start by declaring a variable with the `let` keyword, followed by the name you give to the variable:

```
let myVariable;
```

- A semicolon at the end of a line indicates where a statement ends. It is only required when you need to separate statements on a single line.
- You can name a variable nearly anything, but there are some restrictions.
- JavaScript is case sensitive. This means *myVariable* is not the same as *myvariable*.

After declaring a variable, you can give it a value:

```
myVariable = 'Bob';
```

Also, you can do both these operations on the same line:

```
let myVariable = 'Bob';
```

You retrieve the variable by calling the variable name:

```
myVariable;
```

Variable datatypes

After assigning a value to a variable, you can change it later in the code:

```
let myVariable = 'Bob';
```

```
myVariable = 'Steve';
```

Variables may hold values,
That have different data types:

Variable	Explanation	Example
String	This is a sequence of text known as a string. To signify that the value is a string, enclose it in single quote marks.	<pre>let myVariable = 'Bob';</pre>
Number	This is a number. Numbers don't have quotes around them.	<pre>let myVariable = 10;</pre>
Boolean	This is a True/False value. The words <code>true</code> and <code>false</code> are special keywords that don't need quote marks.	<pre>let myVariable = true;</pre>
Array	This is a structure that allows you to store multiple values in a single reference.	<pre>let myVariable = [1, 'Bob', 'Steve', 10]; Refer to each member of the array like this: myVariable[0], myVariable[1], etc.</pre>
Object	This can be anything. Everything in JavaScript is an object and can be stored in a variable. Keep this in mind as you learn.	<pre>let myVariable = document.querySelector('h1'); All of the above examples too.</pre>

let, var, and const

A variable consists of three parts, a variable-defining keyword (let, var, or const), a name and a value. What's the difference?

```
let nr1 = 12;
```

```
var nr2 = 8;
```

```
const PI = 3.14159;
```

- let declaration declares a block-scoped local variable, optionally initializing it to a value
- var statement declares a function-scoped or globally-scoped variable, optionally initializing it to a value. var global scope, you can use the variables defined in your entire script.
- *A block of code always starts with { and ends with }, which is how we recognize it.*
- const on the other hand is used for variables that only get the value assigned once. For example the variable of PI, which will not change.

Comments

Comments are snippets of text that can be added along with code. The browser ignores text marked as comments. You can write comments in JavaScript just as you can in CSS:

```
/*
```

Everything in between is a comment.

```
*/
```

If your comment contains no line breaks, it's an option to put it behind two slashes like this:

```
// This is a comment
```


Strings

A string is used to store a text value. A string is a sequence of characters and there are a few ways to write them.

- Double quotes : "High there"
- Single quotes : 'Hi there'
- Backticks : `backticks` special template strings to use a variable directly

Escape characters, say we want to have double, single quotes and backslashes in our string? The solution is to use an escape character, a backslash \

```
let str = "Hello, what's your name? Is it \"Mike\"?";
```

```
console.log(str);
```

Logs the following to the console

Hello, what's your name? Is it "Mike"?

Concatenation

The `concat()` method concatenates the string arguments to the calling string and creates a new string.

Syntax;

`concat(str1)`

`concat(str1, str2)`

`concat(str1, str2, ... , strN)`

If the arguments are not the type of the type string, they are converted to string values before concatenating.

JavaScript Demo: String.concat()

```
1 const str1 = 'Ryan';
2 const str2 = 'Collins';
3
4 console.log(str1.concat(' ', str2));
5 // expected output: "Hello World"
6
7 console.log(str2.concat(', ', str1));
8 // expected output: "World, Hello"
9
```

Run ›

Reset

```
> "Ryan Collins"
> "Collins, Ryan"
```

Expressions & operators

Javascript has the following types of operators;

- Assignment operators
- Comparison operators
- Arithmetic operators
- Bitwise operators
- Logical operators
- String operators
- Conditional (ternary) operator
- Comma operator
- Unary operators
- Relational operators

Operator logic

JavaScript has both binary and unary operators, and one special ternary operator, the conditional operator. A binary operator requires two operands, one before the operator and one after the operator:

operand1 operator operand2

For example, 3+4 or x*y.

A unary operator requires a single operand, either before or after the operator:

operator operand

or

operand operator

For example, x++ or ++x.

Define:
op·er·and
/'äpə,rand/

noun MATHEMATICS

noun: operand; plural noun: operands

1. the quantity on which an operation is to be done.

Let's look at a couple examples

Assignment operators

An assignment operator assigns a value to its left operand based on the value of its right operand.

The simple assignment operator is equal (=), which assigns the value of its right operand to its left operand.

That is, `x = f()` is an assignment expression that assigns the value of `f()` to `x`.

There are also compound assignment operators that are shorthand for the operations listed in the following table:

Assignment	<code>x = f()</code>	<code>x = f()</code>
Addition assignment	<code>x += f()</code>	<code>x = x + f()</code>
Subtraction assignment	<code>x -= f()</code>	<code>x = x - f()</code>
Multiplication assignment	<code>x *= f()</code>	<code>x = x * f()</code>
Division assignment	<code>x /= f()</code>	<code>x = x / f()</code>
Remainder assignment	<code>x %= f()</code>	<code>x = x % f()</code>
Exponentiation assignment	<code>x **= f()</code>	<code>x = x ** f()</code>
Left shift assignment	<code>x <= f()</code>	<code>x = x << f()</code>
Right shift assignment	<code>x >= f()</code>	<code>x = x >> f()</code>
Unsigned right shift assignment	<code>x >>= f()</code>	<code>x = x >>> f()</code>
Bitwise AND assignment	<code>x &= f()</code>	<code>x = x & f()</code>
Bitwise XOR assignment	<code>x ^= f()</code>	<code>x = x ^ f()</code>
Bitwise OR assignment	<code>x = f()</code>	<code>x = x f()</code>
Logical AND assignment	<code>x &&= f()</code>	<code>x && (x = f())</code>
Logical OR assignment	<code>x = f()</code>	<code>x (x = f())</code>
Logical nullish assignment	<code>x ??= f()</code>	<code>x ?? (x = f())</code>

Arithmetic operators

An arithmetic operator takes numerical values (either literals or variables) as their operands and returns a single numerical value. The standard arithmetic operators are addition (+), subtraction (-), multiplication (*), and division (/). These operators work as they do in most other programming languages when used with floating point numbers (in particular, note that division by zero produces Infinity)

```
1 / 2; // 0.5  
1 / 2 == 1.0 / 2.0; // this is true
```



Conditional (ternary) operator

The conditional operator is the only JavaScript operator that takes three operands. The operator can have one of two values based on a condition. The syntax is:

`condition ? val1 : val2`

If condition is true, the operator has the value of val1. Otherwise it has the value of val2. You can use the conditional operator anywhere you would use a standard operator.

For example,

```
const status = age >= 18 ? 'adult' : 'minor';
```

This statement assigns the value "adult" to the variable status if age is eighteen or more. Otherwise, it assigns the value "minor" to status.

This operator is frequently used as an alternative to an if...else statement

Logical operators

Logical operators are typically used with Boolean (logical) values; when they are, they return a Boolean value. However, the `&&` and `||` operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they may return a non-Boolean value.

Operator	Usage	Description
Logical AND (<code>&&</code>)	<code>expr1</code> <code>&&</code> <code>expr2</code>	Returns <code>expr1</code> if it can be converted to <code>false</code> ; otherwise, returns <code>expr2</code> . Thus, when used with Boolean values, <code>&&</code> returns <code>true</code> if both operands are true; otherwise, returns <code>false</code> .
Logical OR (<code> </code>)	<code>expr1</code> <code> </code> <code>expr2</code>	Returns <code>expr1</code> if it can be converted to <code>true</code> ; otherwise, returns <code>expr2</code> . Thus, when used with Boolean values, <code> </code> returns <code>true</code> if either operand is true; if both are false, returns <code>false</code> .
Logical NOT (<code>!</code>)	<code>!expr</code>	Returns <code>false</code> if its single operand that can be converted to <code>true</code> ; otherwise, returns <code>true</code> .

Conditionals

Conditionals are code structures used to test if an expression returns true or not. A very common form of conditionals is the if...else statement. For example:

```
let iceCream = 'chocolate';  
if(iceCream === 'chocolate') {  
  alert('Yay, I love chocolate ice cream!');  
} else {  
  alert('Awww, but chocolate is my favourite...');  
}
```

Functions

Functions are a way of packaging functionality that you wish to reuse. It's possible to define a body of code as a function that executes when you call the function name in your code. This is a good alternative to repeatedly writing the same code. You have already seen some uses of functions. For example:

```
let myVariable = document.querySelector('h1');  
alert('hello!');
```

These functions, `document.querySelector` and `alert`, are built into the browser. If you see something which looks like a variable name, but it's followed by parentheses— `()` —it is likely a function.

Creating a function

Let's write a function that asks for your name and then greets you

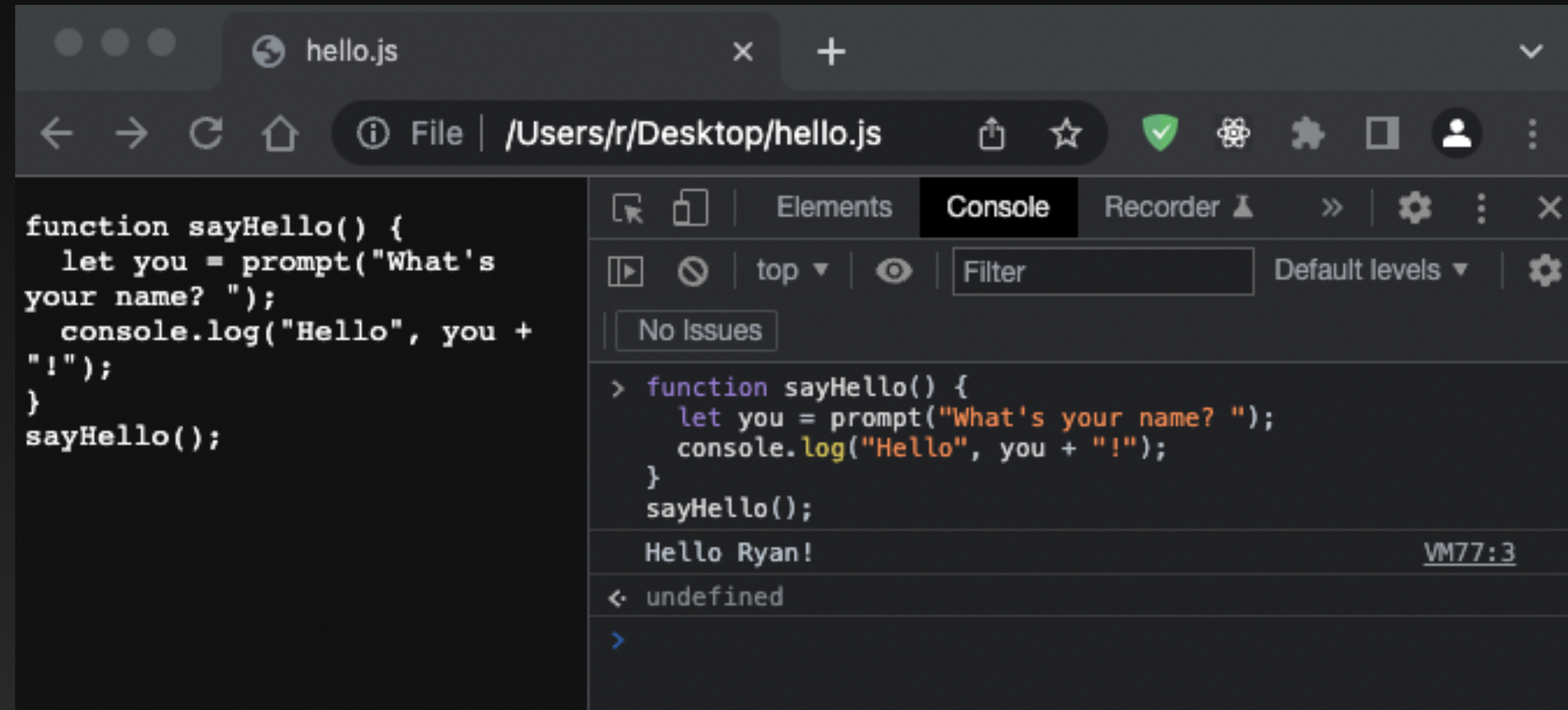
```
function sayHello() {
```

```
  let you = prompt("What's your  
  name? ");
```

```
  console.log("Hello", you + "!");
```

```
}
```

```
sayHello();
```



The screenshot shows a web browser window with a single tab titled 'hello.js'. The address bar shows the file path '/Users/r/Desktop/hello.js'. The browser's developer tools are open, with the 'Console' tab selected. The console shows the following output:

```
> function sayHello() {  
  let you = prompt("What's your name? ");  
  console.log("Hello", you + "!");  
}  
sayHello();  
  
Hello Ryan! VM77:3  
< undefined  
>
```

Check source code named calling-a-function

Events

Real interactivity on a website requires event handlers. These are code structures that listen for activity in the browser, and run code in response. The most obvious example is handling the click event, which is fired by the browser when you click on something with your mouse. To demonstrate this, enter the following into your console, then click on the current webpage:

```
document.querySelector('html').addEventListener('click', function() {  
    alert('Ouch! Stop poking me!');  
});
```

shorter version

```
let myHTML = document.querySelector('html');  
myHTML.addEventListener('click', function() {  
    alert('Ouch! Stop poking me!');  
});
```

Example JavaScript feature website

[Open source code titled image-changer now!](#)

Image Changer

In this section, you will learn how to use JavaScript and DOM API features to alternate the display of one of two images. This change will happen as you click the displayed image.

Open the sample source code titled “image-changer”, what’s happening?

1. The code retrieves the value of the “src” attribute
2. The code uses a conditional to check if the “src” value is equal to the path of the original image, if it is, the code changes the “src” path to the second image, forcing the image to load inside the element
3. If it isn’t, (if it’s already changed) the src value swaps back to the original image path, to the original state

Loops

Loops offer a quick and easy way to do something repeatedly.

You can think of a loop as a computerized version of the game where you tell someone to take X steps in one direction, then Y steps in another. For example, the idea "Go five steps to the east" could be expressed this way as a loop:

```
for (let step = 0; step < 5; step++) {  
    // Runs 5 times, with values of step 0 through 4.  
    console.log('Walking east one step');  
}
```


for statement

A for loop repeats until a specified condition evaluates to false. The JavaScript for loop is similar to the Java and C for loop.

- A for statement looks as follows:

```
for ([initialExpression]; [conditionExpression]; [incrementExpression])  
    statement
```

for statement example

Here, the for statement declares the variable `i` and initializes it to 0. It checks that `i` is less than the number of options in the `<select>` element, performs the succeeding if statement, and increments `i` by 1 after each pass through the loop.

```
function howMany(selectObject) {  
  let numberSelected = 0;  
  for (let i = 0; i < selectObject.options.length; i++) {  
    if (selectObject.options[i].selected) {  
      numberSelected++;  
    }  
  }  
  return numberSelected;  
}  
  
const btn = document.getElementById('btn');  
btn.addEventListener('click', () => {  
  const musicTypes = document.selectForm.musicTypes;  
  console.log(`You have selected ${howMany(musicTypes)} option(s).`);  
});
```

[Check for-loop sample code](#)

Assignment

Create a basic JavaScript project