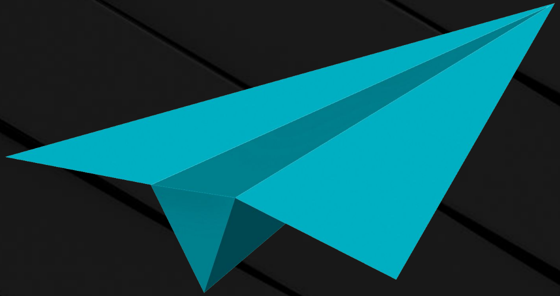
Introduction to ES6

Skills Bootcamp in Front-End Web Development

Lesson 11.1



Office Hours

30 Minutes

WELCOME

Be sure to install Node.js it using the resources found on the
Node.js installation guide on The Full-Stack Blog

Learning Objectives

By the end of class, you will be able to:

- Run very simple JavaScript files from the command line using Node.js.
- Explain arrow functions and how they impact the `this` context.
- Use template strings and use `const` and `let` in place of `var`.
- Use functional loops like `map()` and `filter()`.

What is Node.js?

NodeJS...

- Is an open source, cross-platform JavaScript runtime environment designed to be run outside of the browser.

- Is a general utility that can be used for a variety of other purposes, including asset compilation, scripting, monitoring, and as the basis for web servers.

Instructor Demonstration

Mini-Project

What are we learning?

We are learning more about Node.js, third-party modules, and Node’s native `fs` module.

```
var fs = require('fs');
```

How does this project build off or extend previously learned material?

We are continuing to expand our knowledge of using JavaScript to build programs, but this time we are working outside the browser.

Questions?

Instructor Demonstration

Node.js

Questions?

Activity: Node.js

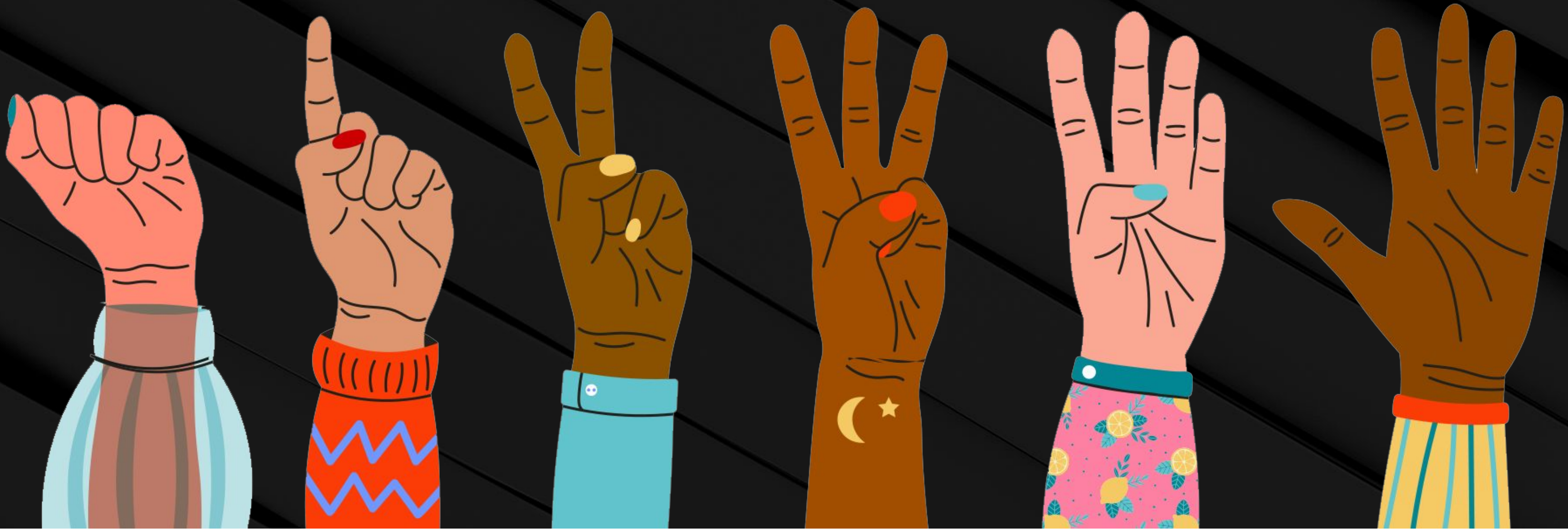
Suggested Time:

10 Minutes

Time’s Up! Let’s Review.

FIST TO FIVE:

How comfortable do you feel with the Node.js thus far?



Review: Node.js

01

What happens if we were to log `window` to the console?

02

What kinds of things do we think are possible in the browser, but not possible in Node.js?

03

What can we do if we don’t completely understand this?

Review: Node.js

01

What happens if we were to log `window` to the console?

We get an error—`window` is not defined in Node.js.

02

What kinds of things do we think are possible in the browser, but not possible in Node.js?

We can't use prompts, confirms, or alerts because of the `window` object.

03

What can we do if we don't completely understand this?

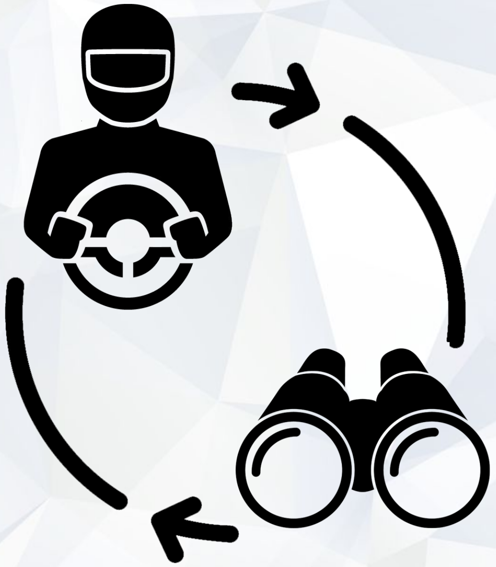
We can refer to supplemental material, read the **Node.js documentation**, and stick around for office hours to ask for help.

Questions?

Instructor Demonstration

Arrow Functions

Questions?



Pair Programming Activity:

Arrow Function Practice

Suggested Time:

15 Minutes

Time’s Up! Let’s Review.

Review: Arrow Function Practice

The following funnyCase() function is able to use arrow syntax, because there is no this context that needs to be preserved:

```
var funnyCase = string => {
  var newString = "";
  for (var i = 0; i < string.length; i++) {
    if (i % 2 === 0) newString += string[i].toLowerCase();
    else newString += string[i].toUpperCase();
  }
  return newString;
};
```

Review: Arrow Function Practice

When using arrow functions, we can use an implied return to reduce the code even further, as shown in the following example:

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

var doubled = map(numbers, element => element * 2);
```

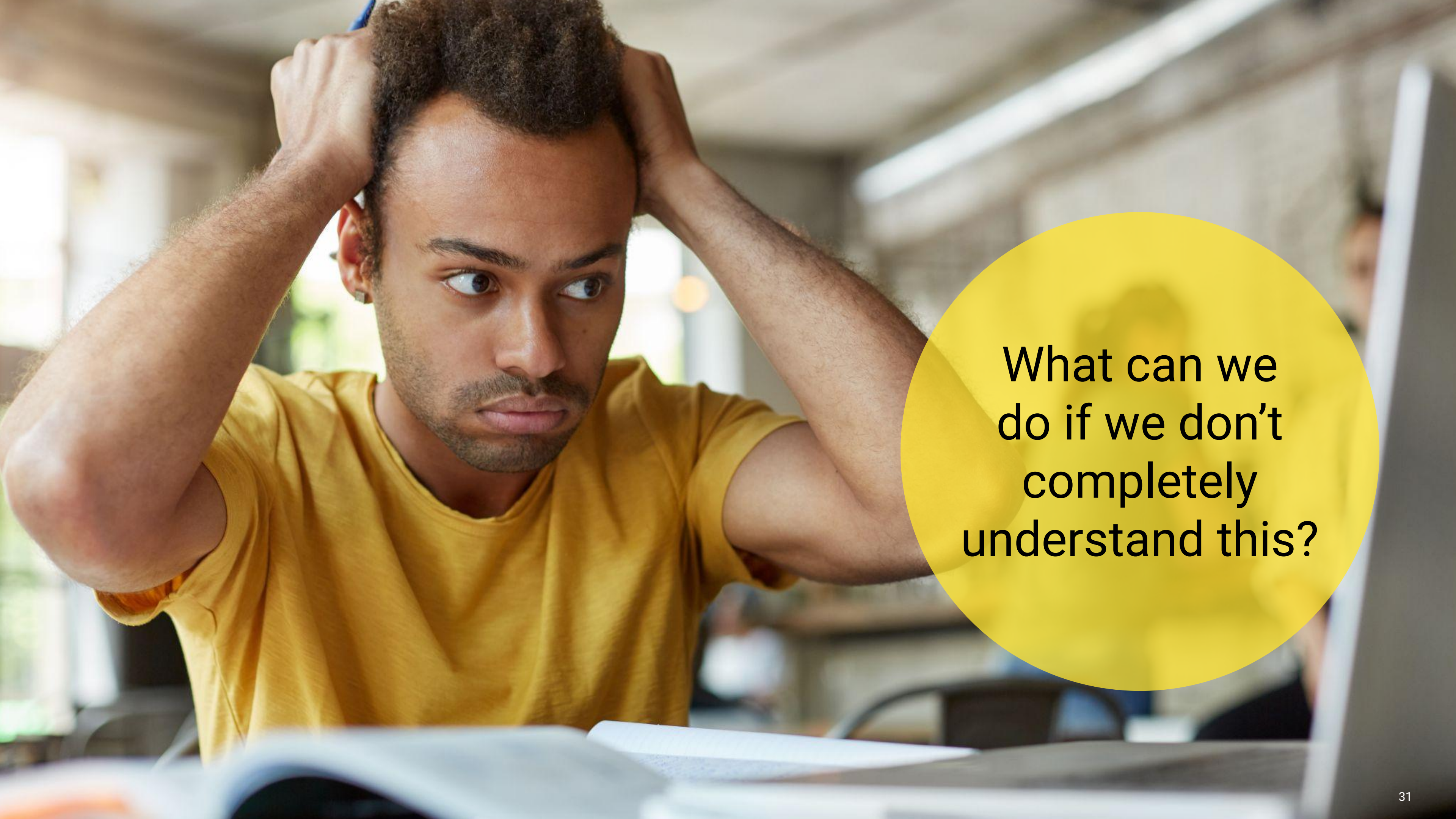
Review: Arrow Function Practice

In the following example, we had to convert the arrow functions back to regular functions to preserve the context of this in the object:

```
var netflixQueue = {
  queue: [
    "Mr. Nobody",
    "The Matrix",
    "Eternal Sunshine of the Spotless Mind",
    "Fight Club"
  ],
  watchMovie: function() {
    this.queue.pop();
  },
};
```

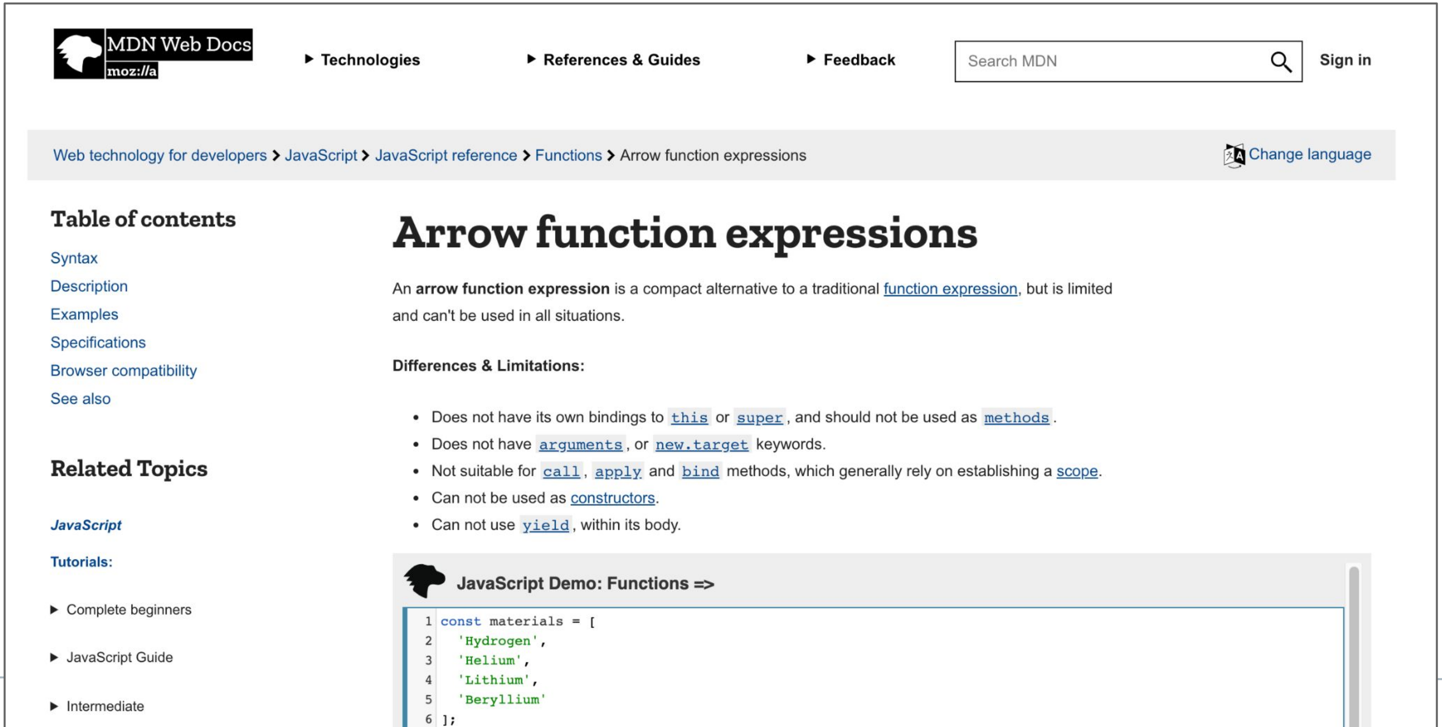
Why would you use arrow functions?

The syntax is easier to write and makes for cleaner-looking code.



What can we do if we don’t completely understand this?

We can refer to supplemental material, read the MDN Web Docs on arrow functions, and stick around for office hours to ask for help.



Web technology for developers > JavaScript > JavaScript reference > Functions > Arrow function expressions

Table of contents

- Syntax
- Description
- Examples
- Specifications
- Browser compatibility
- See also

Related Topics

JavaScript

Tutorials:

- Complete beginners
- JavaScript Guide
- Intermediate

Arrow function expressions

An **arrow function expression** is a compact alternative to a traditional function expression, but is limited and can't be used in all situations.

Differences & Limitations:

- Does not have its own bindings to `this` or `super`, and should not be used as `methods`.
- Does not have `arguments`, or `new.target` keywords.
- Not suitable for `call`, `apply` and `bind` methods, which generally rely on establishing a scope.
- Can not be used as constructors.
- Can not use `yield`, within its body.

JavaScript Demo: Functions =>

```
const materials = [
  'Hydrogen',
  'Helium',
  'Lithium',
  'Beryllium'
];
```

Instructor Demonstration

let and const

Questions?

Pair Programming Activity:

Convert to ES6 Syntax

Suggested Time:

15 Minutes

Time’s Up! Let’s Review.

Review: Convert to ES6 Syntax

A good way to think about these variable names is to ask yourself “does this need to be changed in future?” If the answer is no, you should use `const`.

```
const $root = document.querySelector("#root");
```

Review: Convert to ES6 Syntax

Ask yourself if you need to take advantage of the `this` context inside your function. If not, convert it to an arrow function.

```
const makeGuess = () => {
  const $score = document.querySelector("#root p");
  $score.textContent = "Score: " + score + " | " + "Target: " + targetScore;

  if (score > targetScore) {
    alert("You lost this round!");
    playRound();
  } else if (score === targetScore) {
    alert("You won this round!");
    playRound();
  }
};
```

Review: Convert to ES6 Syntax

This kind of function is called a **constructor** function. Arrow functions can't be used in constructor functions.

```
const Crystal = function(color) {
  this.element = document.createElement("div");
  this.element.className = "crystal " + color;
  this.value = 0;

  this.element.addEventListener(
    "click",
    () => {
      score += this.value;
      makeGuess();
    },
    false
  );
};
```

What is a good use for `let`?

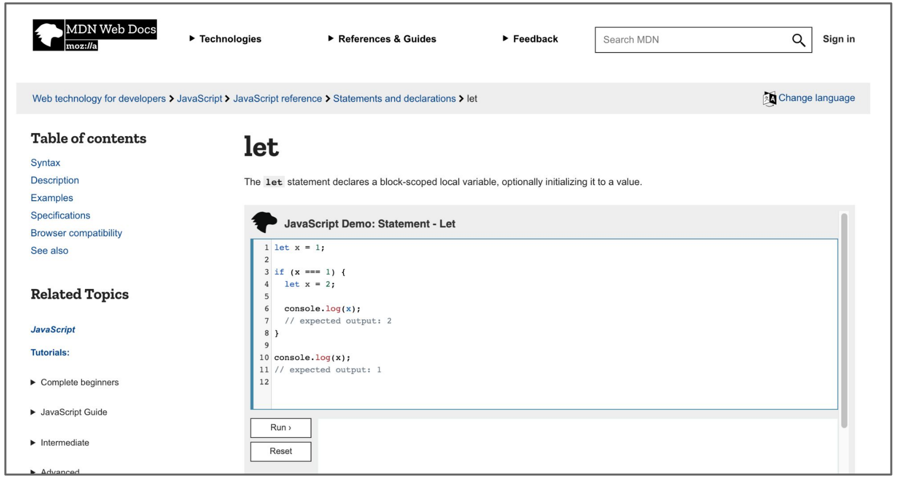
When we need to reassign a value. An example of this would be a counter variable like `i`.

What can we do if we don’t completely understand this?

We can refer to supplemental material and stick around for office hours to ask for help.

Read the MDN Web Docs on `let`

Read the MDN Web Docs on `const`



Web technology for developers > JavaScript > JavaScript reference > Statements and declarations > let

Table of contents
- Syntax
- Description
- Examples
- Specifications
- Browser compatibility
- See also

Related Topics

JavaScript

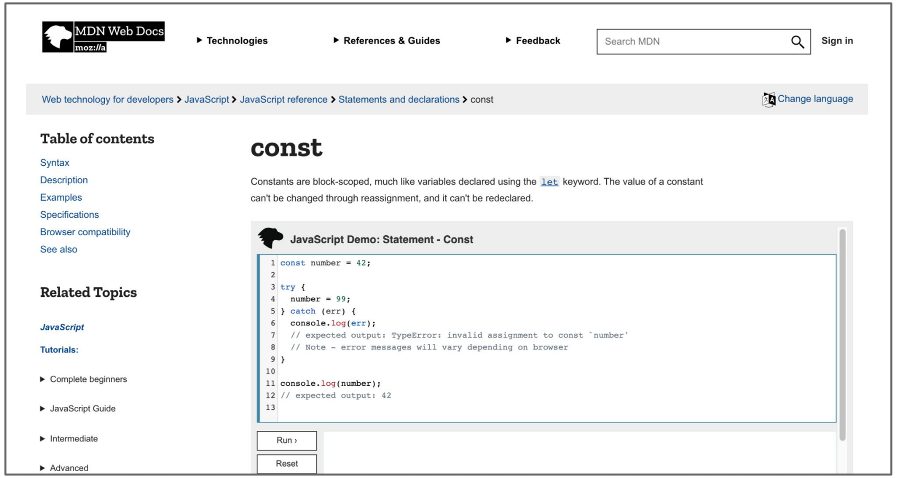
Tutorials:
- Complete beginners
- JavaScript Guide
- Intermediate
- Advanced

let

The `let` statement declares a block-scoped local variable, optionally initializing it to a value.

JavaScript Demo: Statement - Let

```
let x = 1;

if (x === 1) {
  let x = 2;

  console.log(x);
  // expected output: 2
}

console.log(x);
// expected output: 1

```



Web technology for developers > JavaScript > JavaScript reference > Statements and declarations > const

Table of contents
- Syntax
- Description
- Examples
- Specifications
- Browser compatibility
- See also

Related Topics

JavaScript

Tutorials:
- Complete beginners
- JavaScript Guide
- Intermediate
- Advanced

const

Constants are block-scoped, much like variables declared using the `let` keyword. The value of a constant can't be changed through reassignment, and it can't be redeclared.

JavaScript Demo: Statement - Const

```
const number = 42;

try {
  number = 99;
} catch (err) {
  console.log(err);
  // expected output: TypeError: invalid assignment to const `number'
  // Note - error messages will vary depending on browser
}

console.log(number);
// expected output: 42

```

Questions?

Break

Instructor Demonstration

Functional Loops

What is the difference between `filter()` and `forEach()`?

Functional Loops

`filter()`

returns a brand-new array

`forEach()`

mutates the existing array

How is `map()` different from `filter()`?

Functional Loops

`map()` will return a brand-new array like `filter()` does; however, the length of the array that `map()` returns will be the exact same as the input array.

This isn't always the case for the `filter()` method.

Questions?

Instructor Demonstration

Template Literals

Template Literals

Using string interpolation, or template strings, we have a new way of concatenating variables to the rest of strings.

This is a new feature included in ES6.

Template strings are much more readable and easier to manage.

Consider the following example:

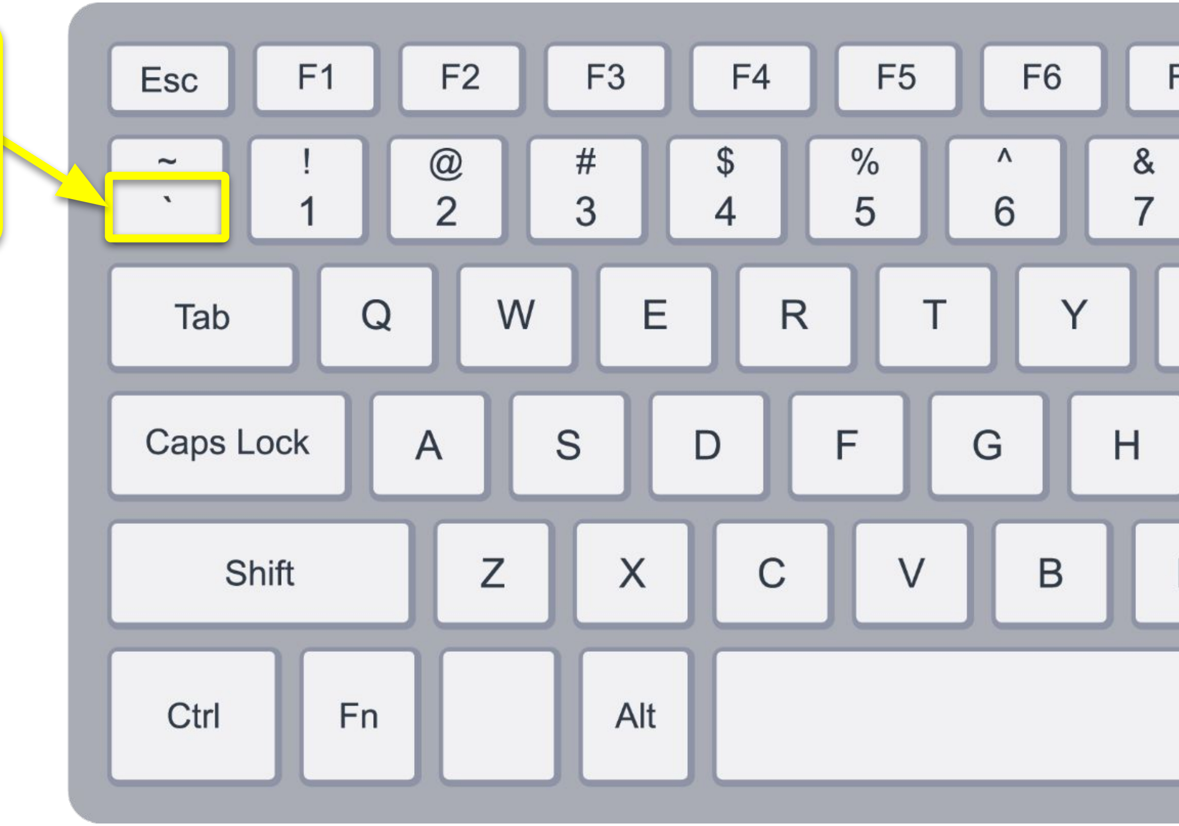
```
const arya = {
  first: "Arya",
  last: "Stark",
  origin: "Winterfell",
  allegiance: "House Stark"
};

const greeting = `My name is ${arya.first}!
I am loyal to ${arya.allegiance}.`;
```

What are the main differences that you notice in syntax between regular string concatenation and template literals?

Template Literals

Immediately we notice that template strings use backticks instead of quotes.

Additionally, instead of using plus signs, we can now reference variables explicitly using the `${}` syntax.



Questions?

Activity: Template Literals

Suggested Time:

10 Minutes

Time’s Up! Let’s Review.

Review: Template Literals

- Template strings are much easier to read than traditional string concatenation.
- Dealing with spacing is a lot easier using template literals.
- Don’t forget to use backticks instead of quotes. This is a very easy mistake to make.

Review: Template Literals

In the following example, we create a template string that will eventually be injected into the DOM:

```
const music = {
  title: "The Less I Know The Better",
  artist: "Tame Impala",
  album: "Currents"
};

// write code between the <div> tags to output your objects data
const songSnippet = `
  <div class="song">
    <h2>${music.title}</h2>
    <p class="artist">${music.artist}</p>
    <p class="album">${music.album}</p>
  </div>
`;
const element = document.getElementById("music");
element.innerHTML = songSnippet;
```

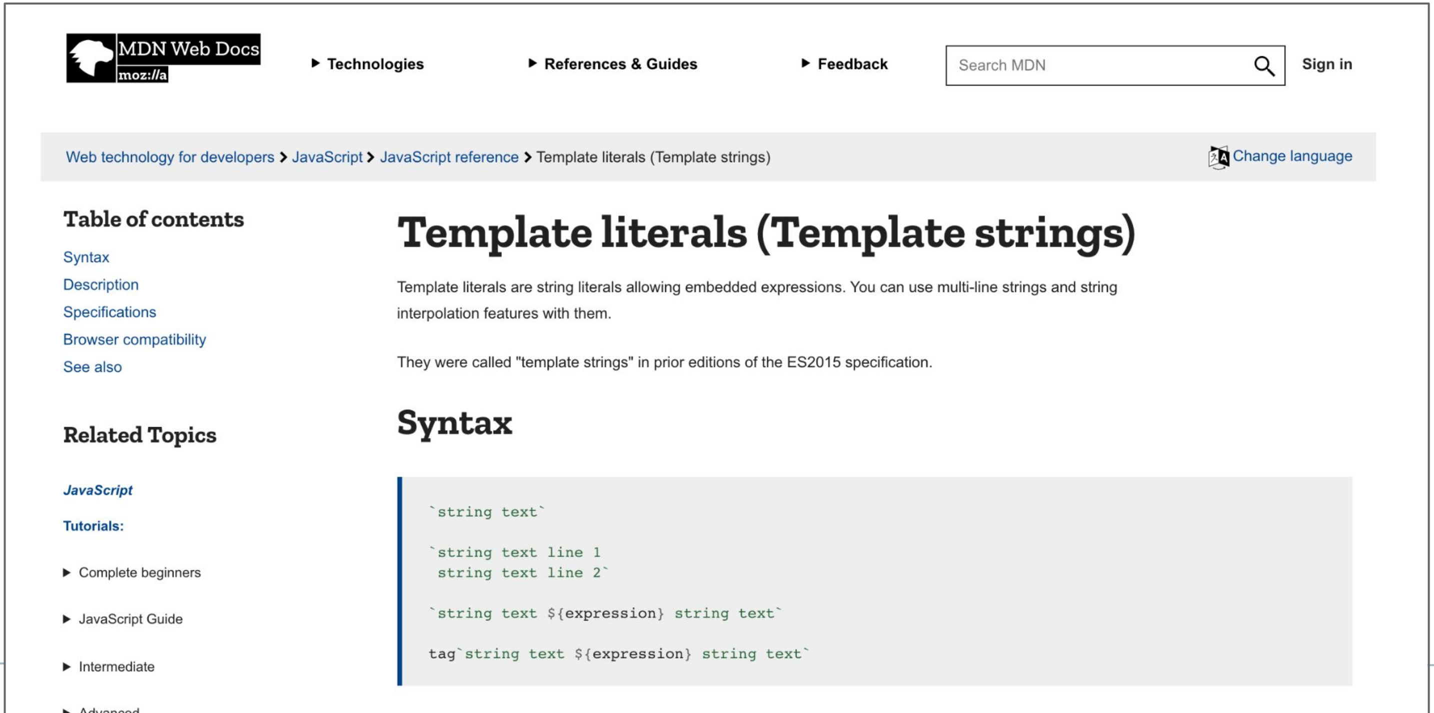
We use the `${}` syntax to reference the music object and the variables within it in the template string. That template string eventually gets added to the DOM as pure HTML.

What are the benefits of using template strings?

They are easier to read and easier to manage. They also allow us to maintain indentation and formatting of the content when inside the backticks.

What can we do if we don't completely understand this?

We can refer to supplemental material, read the MDN Web Docs on template literals, and stick around for office hours to ask for help.



Web technology for developers > JavaScript > JavaScript reference > Template literals (Template strings)

Table of contents

- Syntax
- Description
- Specifications
- Browser compatibility
- See also

Related Topics

JavaScript

Tutorials:

- ▸ Complete beginners
- ▸ JavaScript Guide
- ▸ Intermediate
- ▸ Advanced

Template literals (Template strings)

Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them.

They were called "template strings" in prior editions of the ES2015 specification.

Syntax

```
`string text`

`string text line 1
 string text line 2`

`string text ${expression} string text`

tag`string text ${expression} string text`
```

Questions?

The End