Slideshow 8

Javascript basics II

DOM manipulation, functions, objects

INFO 6150
Fernando Augusto López Plascencia

In this lesson:

- DOM manipulation: Creating new content
- Refreshers on objects and functions
- Debugging

Embedding JS in a page

Two ways:

- Insert directly inside a <script> tag:

```
        <script>
            console.log('This script is directly embedded into the page.')
        </script>
```

- Link an external script through `<script src=”path/to/file.js”>`:

```
<html>
    <head>
        <script src="script.js"></script>
    </head>
```

Created by HideMaru
from Noun Project

A challenge!

Create a page, then link a script to it.

The script must:

- Ask the user for their name (use `window.prompt`)
- Greet the user using that name

Icon by HideMaru for Noun Project

Variables or constants?

Use a variable if the value will change later in the program.

Otherwise, use a constant.

```
const userName = window.prompt('Please enter your name.');
window.alert(`Salutations, ${userName}!`)
```

It is good practice to use constants whenever you can.

Greet the user… in the web page

Your next challenge:

- Modify your code so that the message is shown in the web page instead. You need to create HTML content for the message.

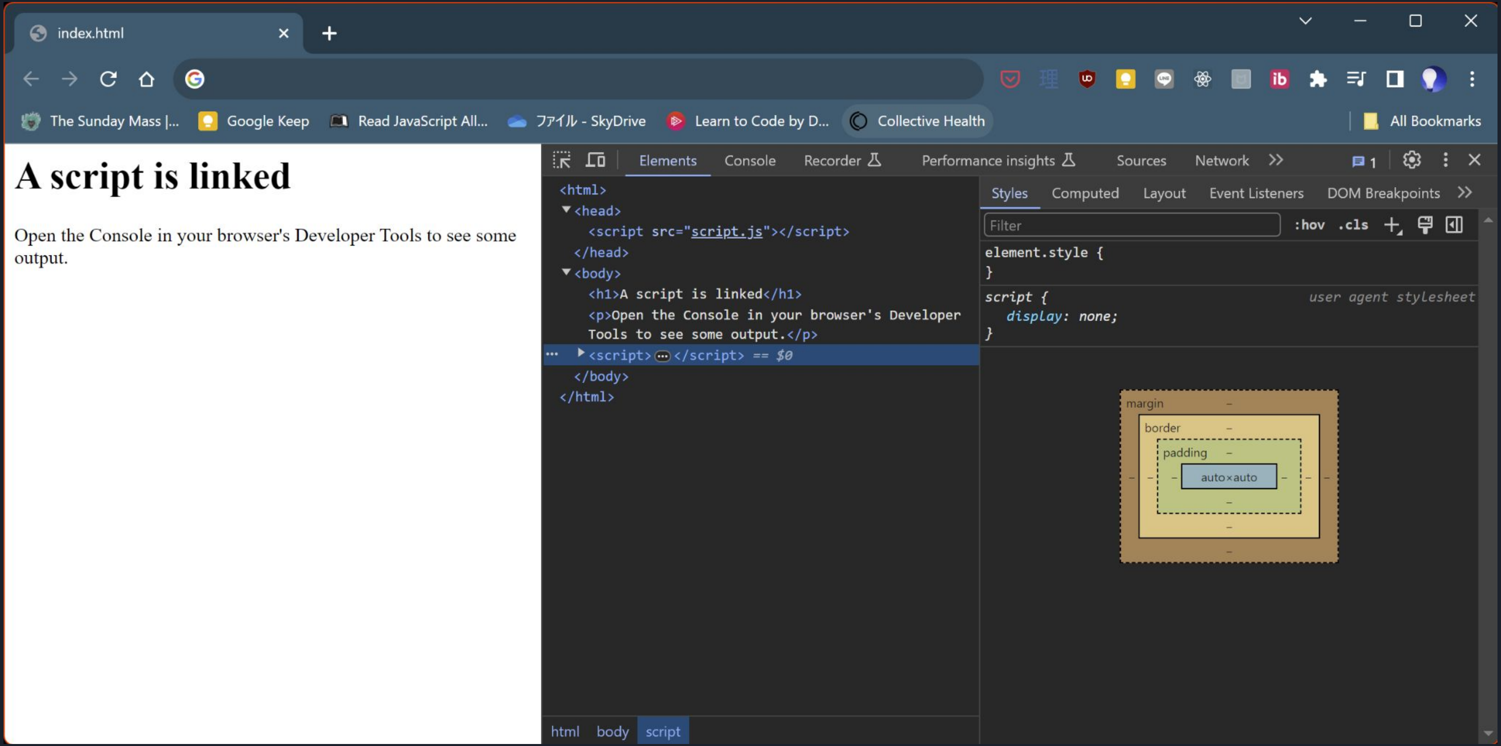
How to do this?

DOM and DOM manipulation

The DOM allows Javascript to interact with HTML elements.

There is a DOM node per HTML element.

DOM and DOM manipulation



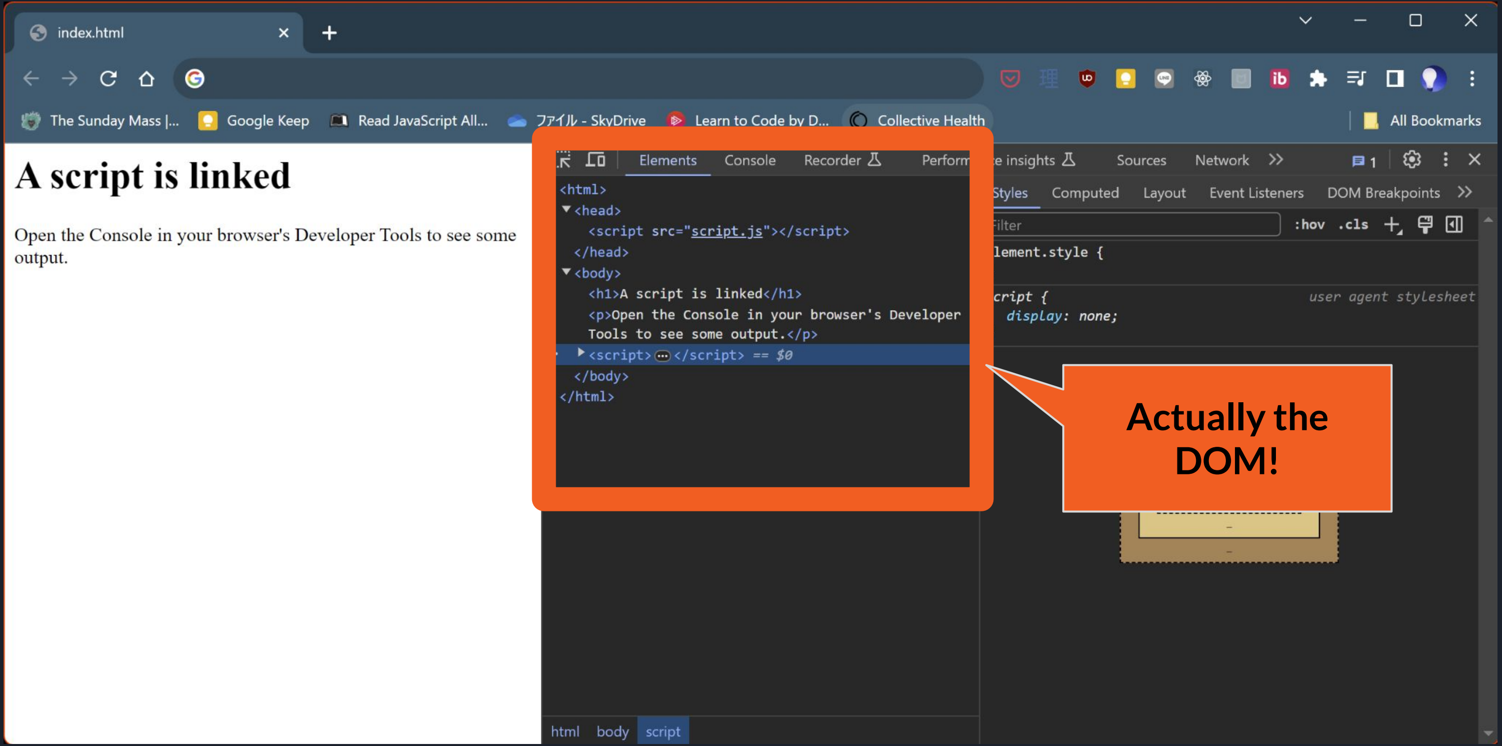
A script is linked

Open the Console in your browser's Developer Tools to see some output.

```
<html>
▼<head>
    <script src="script.js"></script>
  </head>
▼<body>
    <h1>A script is linked</h1>
    <p>Open the Console in your browser's Developer
    Tools to see some output.</p>
  ▶<script>…</script> == $0
  </body>
</html>
```

```
element.style {
}
script {                                    user agent stylesheet
   display: none;
}
```

DOM and DOM manipulation



A script is linked

Open the Console in your browser's Developer Tools to see some output.

```
<html>
▼<head>
    <script src="script.js"></script>
  </head>
▼<body>
    <h1>A script is linked</h1>
    <p>Open the Console in your browser's Developer
    Tools to see some output.</p>
  ▶<script>⋯</script> == $0
  </body>
</html>
```

```
lement.style {
}
cript {                         user agent stylesheet
  display: none;
```

Actually the DOM!

DOM and DOM manipulation

The DOM allows Javascript to interact with HTML elements.

There is a DOM node per HTML element.

Each DOM node is an object: it has attributes (things it has) and methods (functions it can call).

DOM and DOM manipulation

The DOM allows Javascript to interact with HTML elements.

There is a DOM node per HTML element.

Each DOM node is an object: it has attributes (things it has) and methods (functions it can call).

Wait, what?

Before we continue: a refresher on objects

An object is a data structure that symbolizes something complex.

An object contains:

- Attributes - basically, other values.
    - They describe the object
- Methods - basically, functions that use the object data.
    - They describe what the object can do

Before we continue: a refresher on objects

```
// An object.
// Separate object components using commas
const person = {
    name: "Fernando",
    age: 44, // age is just a number
    sayName: function() {
        console.log(this.name); // will print "Fernando"
        // "this" refers to the current object
    },
}
```

Before we continue: a refresher on objects

In Javascript, most things are objects, even if they have no (apparent) class.

We can explain this one later - just roll with it for now

DOM and DOM manipulation

1. Create a new node (an object) with our content
2. Insert it in the DOM tree (the page)

Creating a new node

```
const username = "Fernando";

// create a new tag node
const newParagraph = document.createElement('p');

// create a new text node; this represents the text that goes inside the tag
const newContent = document.createTextNode(`Hi, ${username}!`);

// with this, the text is attached to the new "p"
newParagraph.appendChild(newContent);
```

The global objects: `document` and `window`

They always exist if running JS in a web page.

- `document` represents the whole HTML page.
  - The DOM tree is a child of `document`
- window represents the browser window (even if there is no page loaded).

Both contain useful functions that are always available.

Insert the new node in the web page

The new node exists in memory, but we still need to insert it into the DOM tree.

It’s good practice to have a specific “parent” in your HTML:

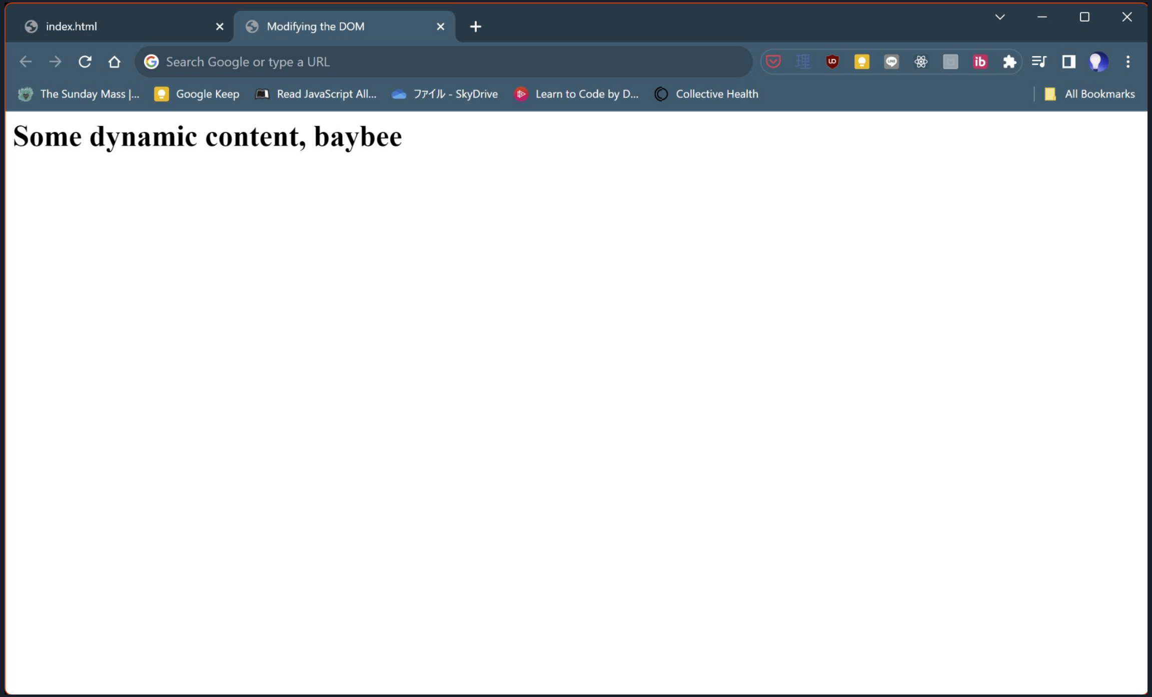
```html
<html>
    <head>
        <title>Modifying the DOM</title>
        <script src="script.js"></script>
    </head>
    <body>
        <h1>Some dynamic content, baybee</h1>
        <section id="dynamic">
            <!-- content here will be generated -->
        </section>
    </body>
</html>
```

Insert the new node in the web page

Now, back in our script:

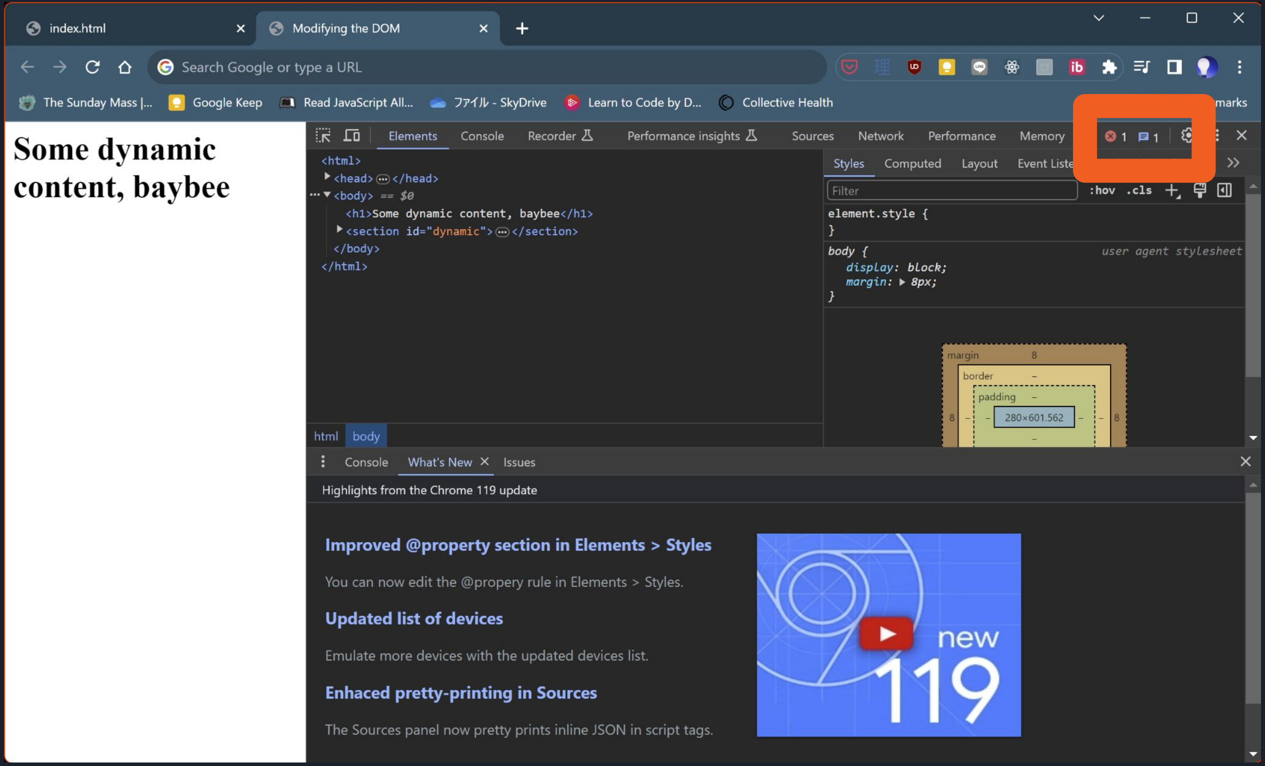
```
// insert the node into the DOM tree:
// 1. get a reference to the node where we'll insert
const parent = document.getElementById('dynamic');

// 2. make our node a child of this node
parent.appendChild(newParagraph);
```

The result: WHY IS IT NOT WORKIIIING



Some dynamic content, baybee

Let's check on the Developer Console!



Some dynamic content, baybee

```
<html>
 ▶<head>⋯</head>
… ▼<body> == $0
    <h1>Some dynamic content, baybee</h1>
   ▶<section id="dynamic">⋯</section>
  </body>
</html>
```

```
element.style {
}
body {                              user agent stylesheet
   display: block;
   margin: ▶ 8px;
}
```

Highlights from the Chrome 119 update

Improved @property section in Elements > Styles

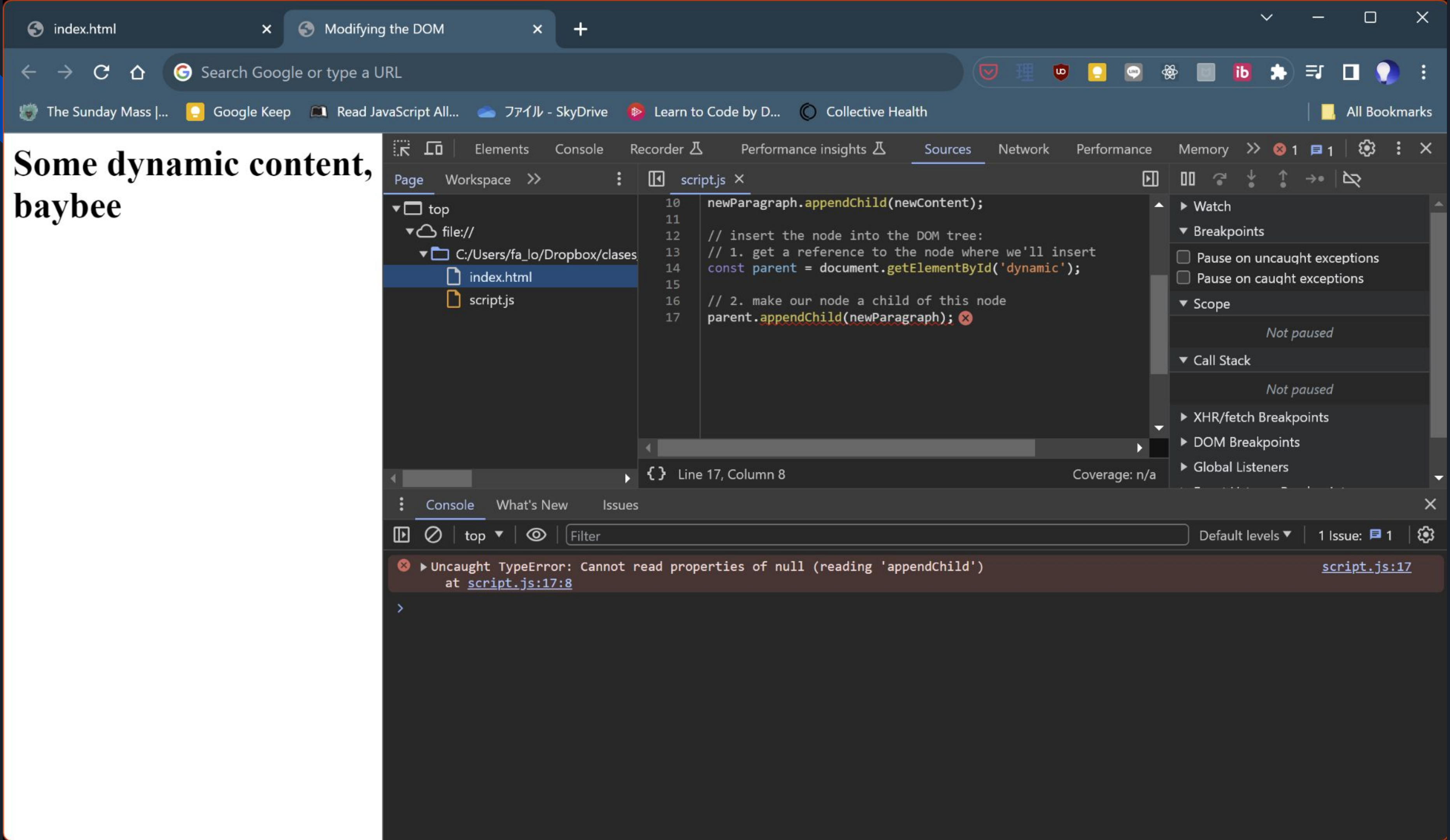
You can now edit the @propery rule in Elements > Styles.

Updated list of devices

Emulate more devices with the updated devices list.

Enhaced pretty-printing in Sources

The Sources panel now pretty prints inline JSON in script tags.



Some dynamic content, baybee

```
newParagraph.appendChild(newContent);

// insert the node into the DOM tree:
// 1. get a reference to the node where we'll insert
const parent = document.getElementById('dynamic');

// 2. make our node a child of this node
parent.appendChild(newParagraph);
```

Line 17, Column 8

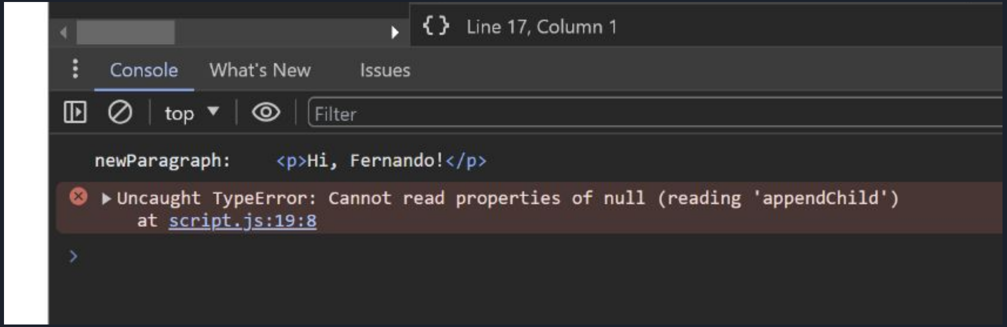
```
Uncaught TypeError: Cannot read properties of null (reading 'appendChild')
    at script.js:17:8
```

script.js:17

What do you mean “null”? What’s the value?
Debugging!

Technique 1: outputting values

```
// create a new tag node
const newParagraph =
document.createElement('p');

console.log('newParagraph: ', newParagraph);
```

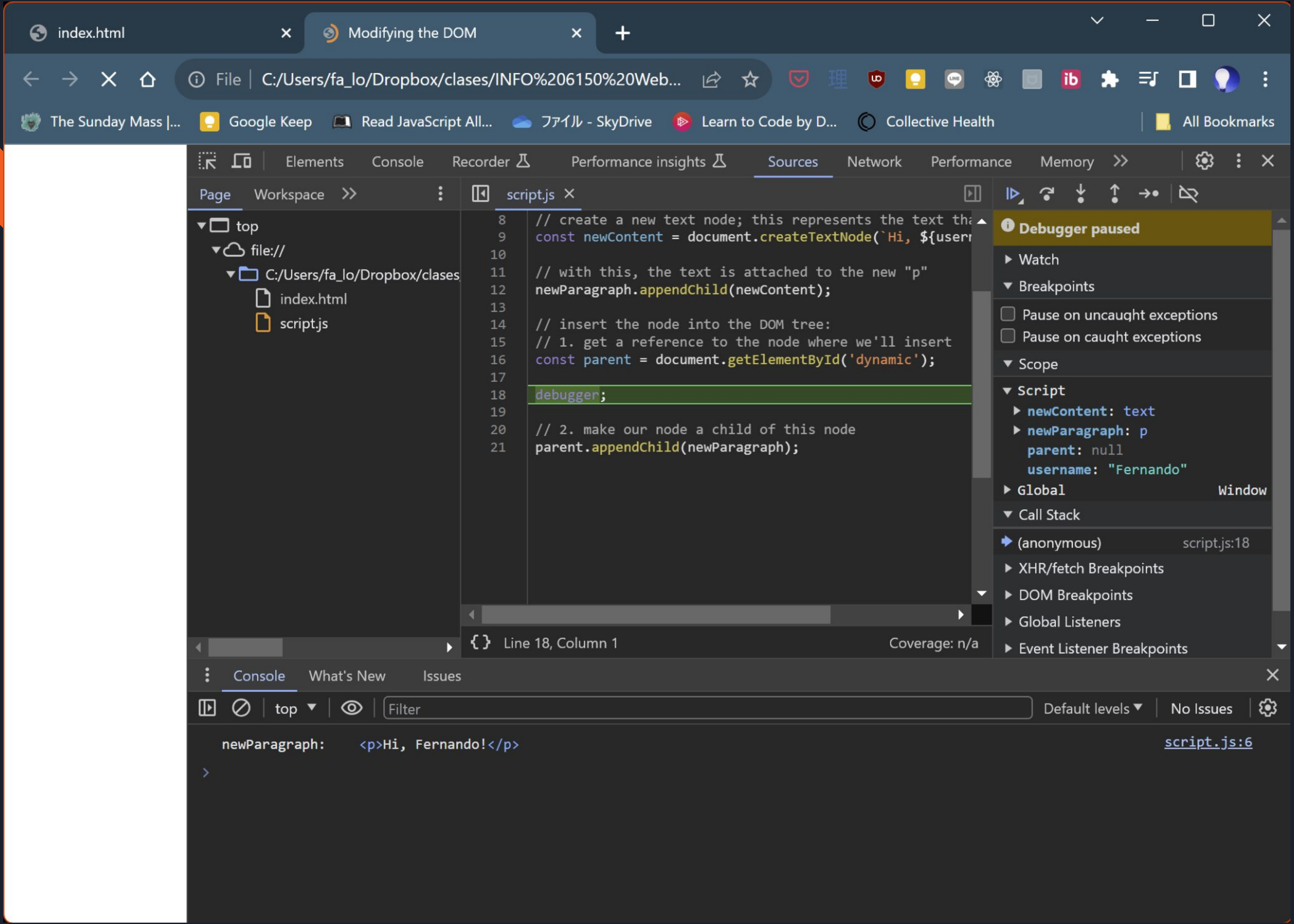


```
newParagraph:    <p>Hi, Fernando!</p>
▶Uncaught TypeError: Cannot read properties of null (reading 'appendChild')
    at script.js:19:8
```

What do you mean “null”? What’s the value?
Debugging!

Technique 2: breakpoints

```
// insert the node into the DOM tree:
// 1. get a reference to the node where we'll insert
const parent = document.getElementById('dynamic');

debugger;

// 2. make our node a child of this node
parent.appendChild(newParagraph);
```



```
// create a new text node; this represents the text tha
const newContent = document.createTextNode(`Hi, ${usern

// with this, the text is attached to the new "p"
newParagraph.appendChild(newContent);

// insert the node into the DOM tree:
// 1. get a reference to the node where we'll insert
const parent = document.getElementById('dynamic');

debugger;

// 2. make our node a child of this node
parent.appendChild(newParagraph);
```

Line 18, Column 1   Coverage: n/a

Debugger paused

Scope

Script

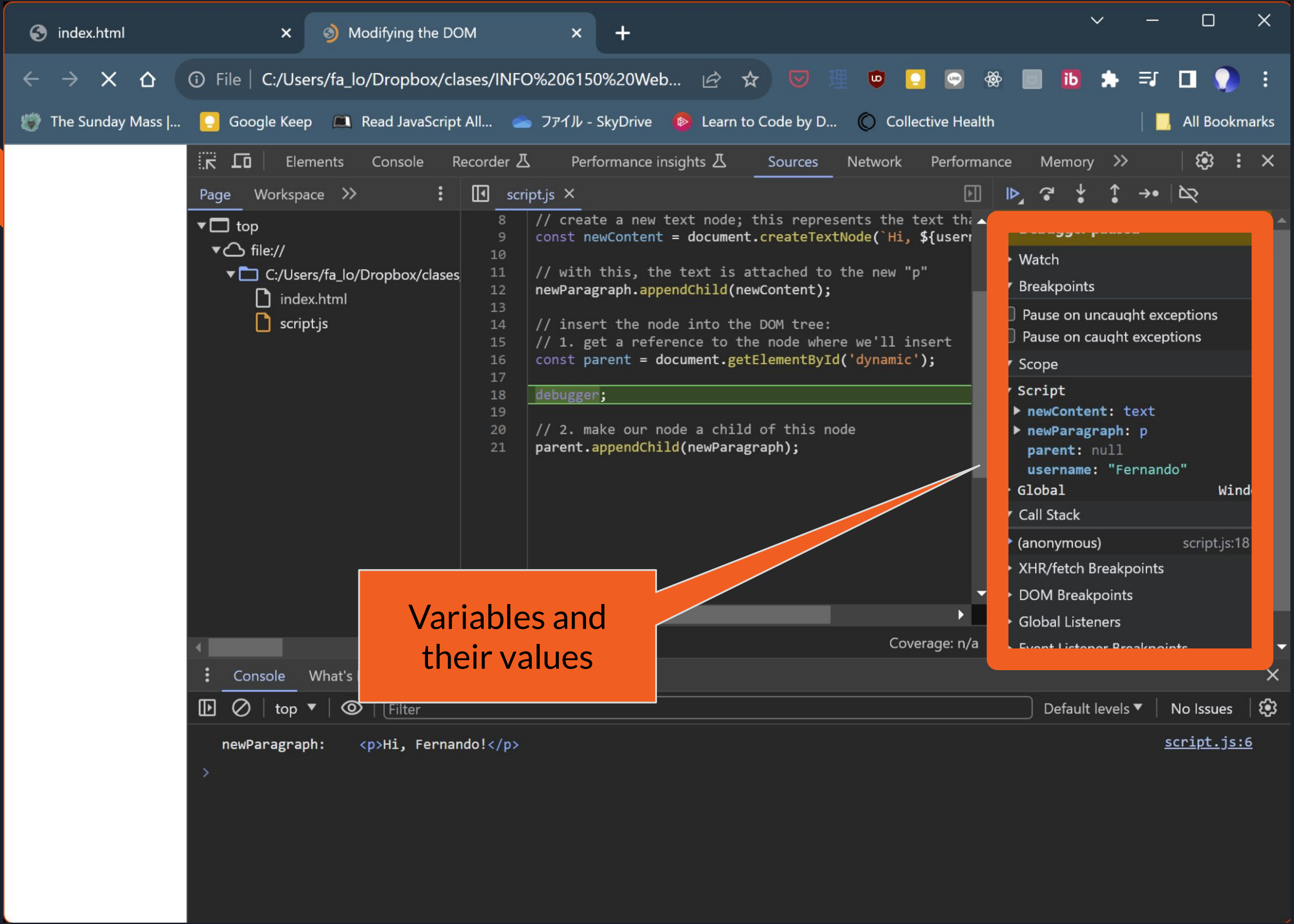
```
newContent: text
newParagraph: p
parent: null
username: "Fernando"
```

Global   Window

Call Stack

(anonymous)   script.js:18

```
newParagraph:   <p>Hi, Fernando!</p>          script.js:6
```

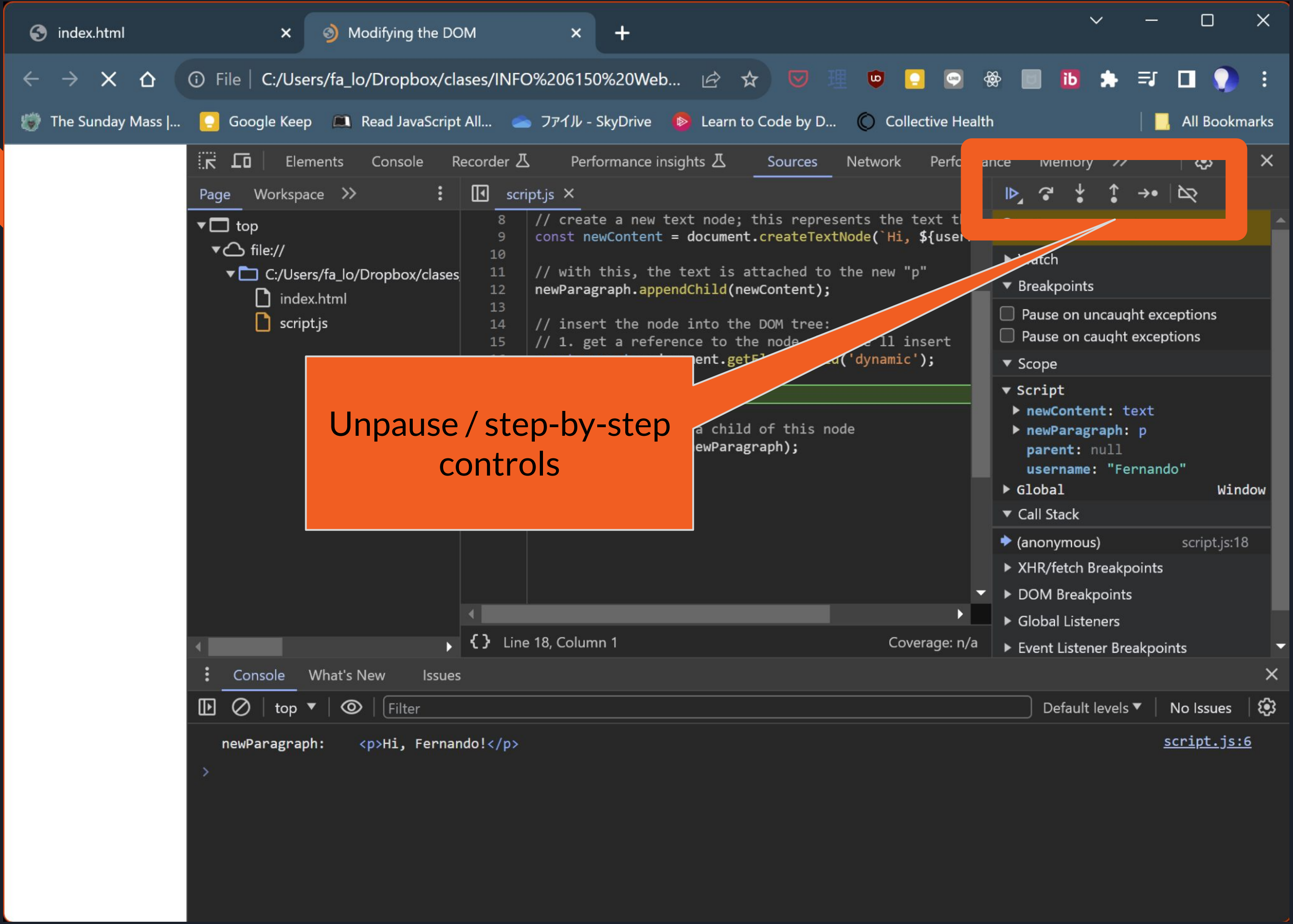


```
// create a new text node; this represents the text tha
const newContent = document.createTextNode(`Hi, ${usern

// with this, the text is attached to the new "p"
newParagraph.appendChild(newContent);

// insert the node into the DOM tree:
// 1. get a reference to the node where we'll insert
const parent = document.getElementById('dynamic');

debugger;

// 2. make our node a child of this node
parent.appendChild(newParagraph);
```

```
Script
▶ newContent: text
▶ newParagraph: p
  parent: null
  username: "Fernando"
Global                    Wind
Call Stack
(anonymous)        script.js:18
```

```
newParagraph:   <p>Hi, Fernando!</p>                script.js:6
```

Variables and their values



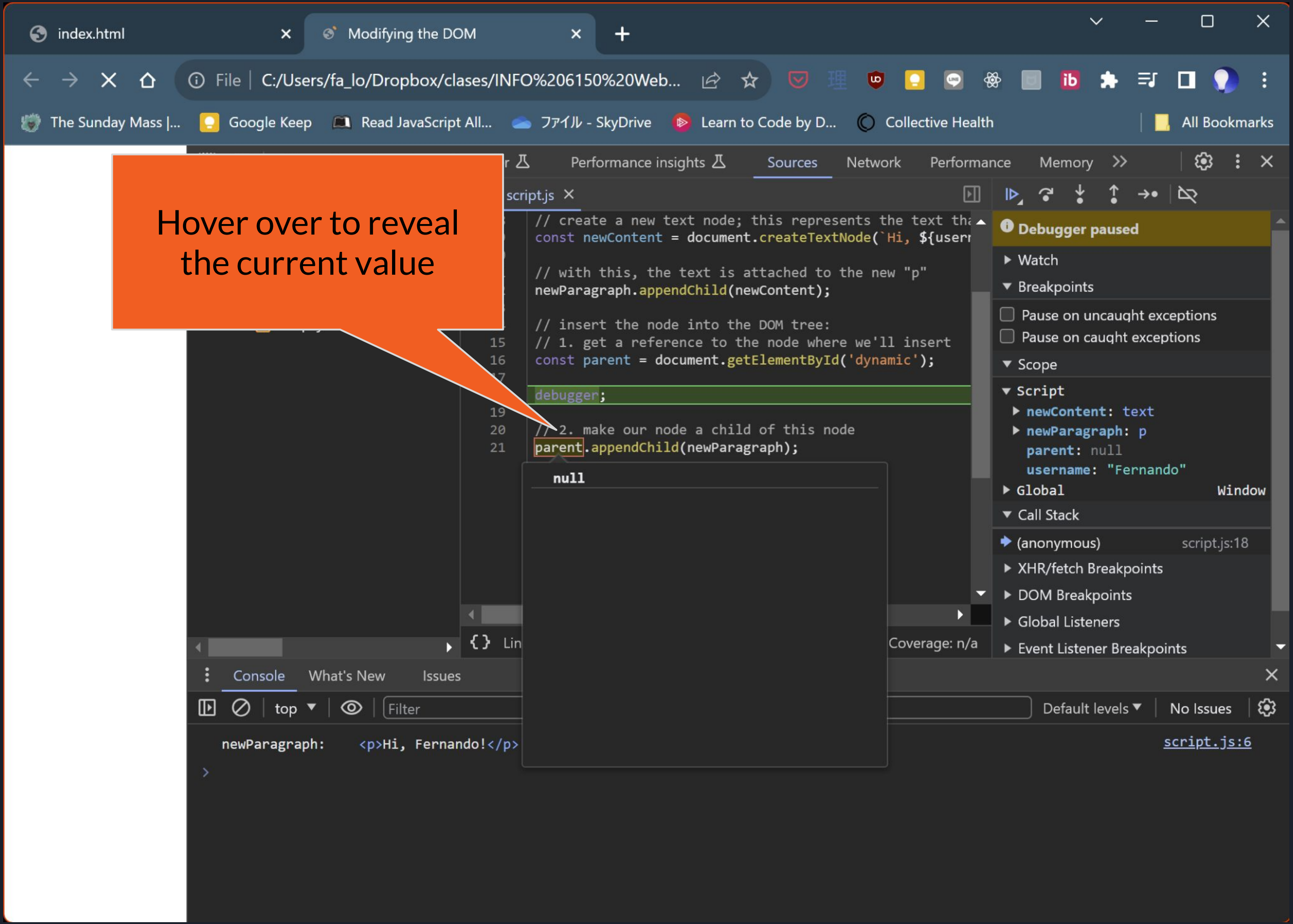
```
// create a new text node; this represents the text t
const newContent = document.createTextNode(`Hi, ${user

// with this, the text is attached to the new "p"
newParagraph.appendChild(newContent);

// insert the node into the DOM tree:
// 1. get a reference to the node              ll insert
                             ent.getE          d('dynamic');

                                a child of this node
                                ewParagraph);
```

```
newContent: text
newParagraph: p
parent: null
username: "Fernando"
Global                                   Window
```

```
newParagraph:   <p>Hi, Fernando!</p>                     script.js:6
```

Unpause / step-by-step controls

Hover over to reveal the current value



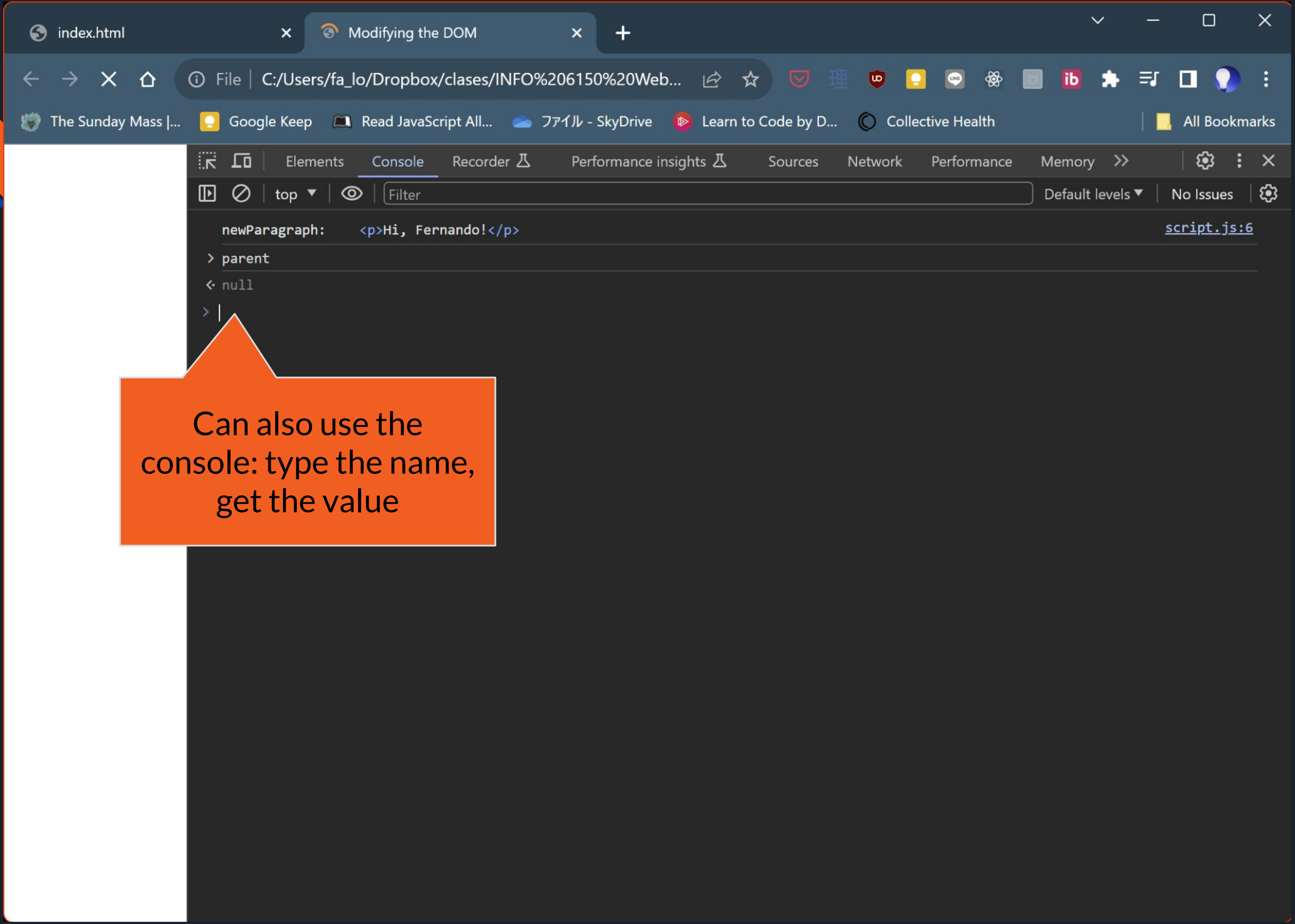
```
// create a new text node; this represents the text tha
const newContent = document.createTextNode(`Hi, ${userr

// with this, the text is attached to the new "p"
newParagraph.appendChild(newContent);

// insert the node into the DOM tree:
// 1. get a reference to the node where we'll insert
const parent = document.getElementById('dynamic');

debugger;

// 2. make our node a child of this node
parent.appendChild(newParagraph);
```

```
null
```

Debugger paused

- Watch
- Breakpoints
  - Pause on uncaught exceptions
  - Pause on caught exceptions
- Scope
  - Script
    - newContent: text
    - newParagraph: p
    - parent: null
    - username: "Fernando"
  - Global — Window
- Call Stack
  - (anonymous) — script.js:18
- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners
- Event Listener Breakpoints

```
newParagraph:    <p>Hi, Fernando!</p>                script.js:6
>
```



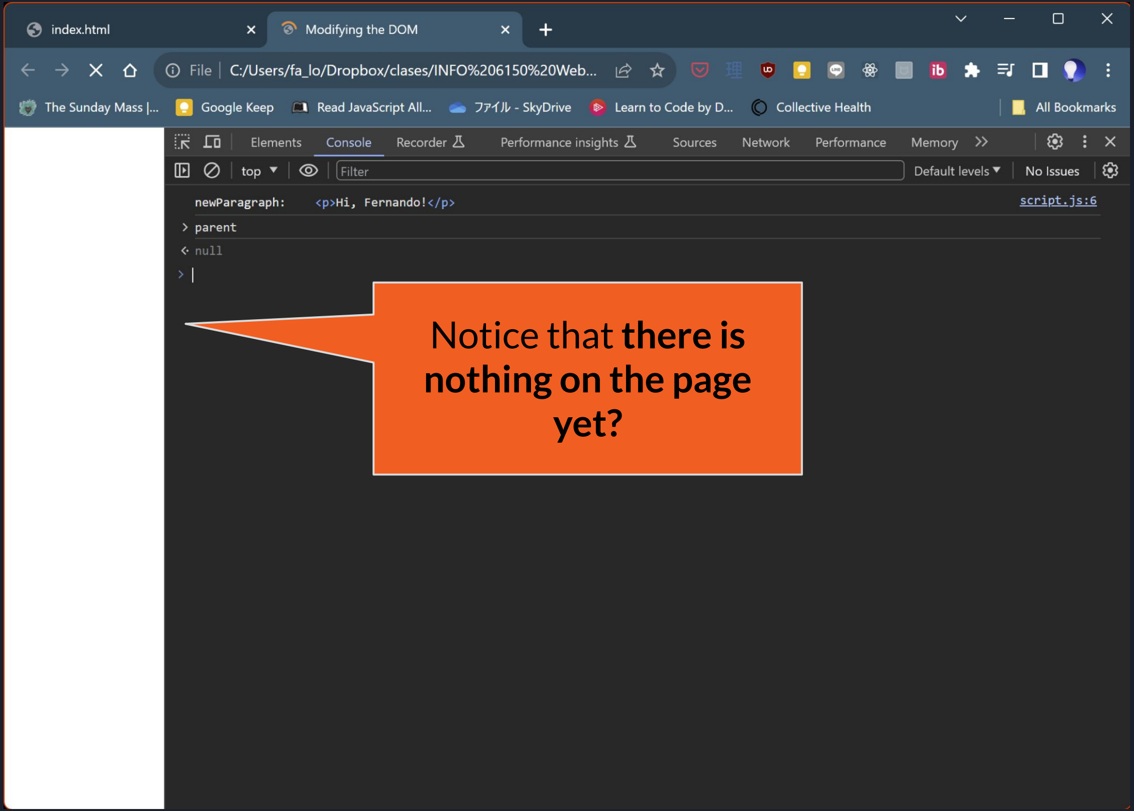
```
newParagraph:   <p>Hi, Fernando!</p>                         script.js:6
> parent
<· null
>
```

Can also use the console: type the name, get the value

So… what happened?

`parent` is `null` because there is no HTML node with an id of “dynamic”... yet

So… what happened?



```
newParagraph:    <p>Hi, Fernando!</p>
> parent
<· null
>
```

Notice that **there is nothing on the page** yet?

Execution order and blocking

- When inserted directly, the browser executes Javascript first, then renders the page.
  - This means we can do some operations efficiently, but cannot use the page contents
- To avoid this, we will instruct the browser to only run the code after the page has loaded.
  - We need to put our code in a function

Execution order and blocking

- When inserted directly, the browser executes Javascript first, then renders the page.
  - This means we can do some operations efficiently, but cannot use the page contents
- To avoid this, we will instruct the browser to only run the code after the page has loaded.
  - We need to put our code in a function

Hey, can you explain that first?

Before we continue: a refresher on functions

A function is a piece of a program that can be invoked on its own.

A function has inputs called parameters and an output (a return value).

Before we continue: a refresher on functions

```
function sum(a, b) {
    return a + b;
}

const result = 1 + 2;
console.log(result); // 3
```

Before we continue: a refresher on functions

We put code in functions to avoid repeating it (DRY: Don’t Repeat Yourself).

A function is (or should be) idempotent:

- If called with the same input…
- …it will always have the same effect (return the same output)

Is `sum(a, b)` idempotent?

Running content on page load

First let’s put our code in a function:

```
function init() {
    const username = "Fernando";

    // create a new tag node
    const newParagraph = document.createElement('p');

    // create a new text node; this represents the text that goes inside the tag
    const newContent = document.createTextNode(`Hi, ${username}!`);

    // with this, the text is attached to the new "p"
    newParagraph.appendChild(newContent);

    // insert the node into the DOM tree:
    // 1. get a reference to the node where we'll insert
    const parent = document.getElementById('dynamic');

    // 2. make our node a child of this node
    parent.appendChild(newParagraph);
}
```

Our code goes into a new function

Running content on page load

Now, let's instruct the browser to run the function when the page is loaded. We will see two methods (there are more). This is the first (old) method:

```html
<html>
    <head>
        <title>Modifying the DOM</title>
        <script src="script.js"></script>
    </head>
    <body onload="init()">
        <h1>Some dynamic content, baybee - for realz this time</h1>
        <section id="dynamic">
            <!-- content here will be generated -->
        </section>
    </body>
</html>
```

`onload` tells the browser to run a function when the content finishes loading

Running content on page load

This is a more modern method, using event loaders. (We will talk more about events later in the course.)

In the .js file, below the init function, write:

```
window.addEventListener("DOMContentLoaded", init);
```

We are telling the browser to run "init" when the DOM content is loaded. (Some images might not have finished loading)

Or (less preferred):

```
window.addEventListener("load", init);
```

Run "init" after all images, etc. finished loading.

Running content on page load

This is a more modern method, using event loaders. (We will talk more about events later in the course.)

In the .js file, below the init function, write:

```
window.addEventListener("DOMContentLoaded", init);
```

We are telling the browser to run “init” when the DOM content is loaded. (Some images might not have finished loading)

Or (less preferred):

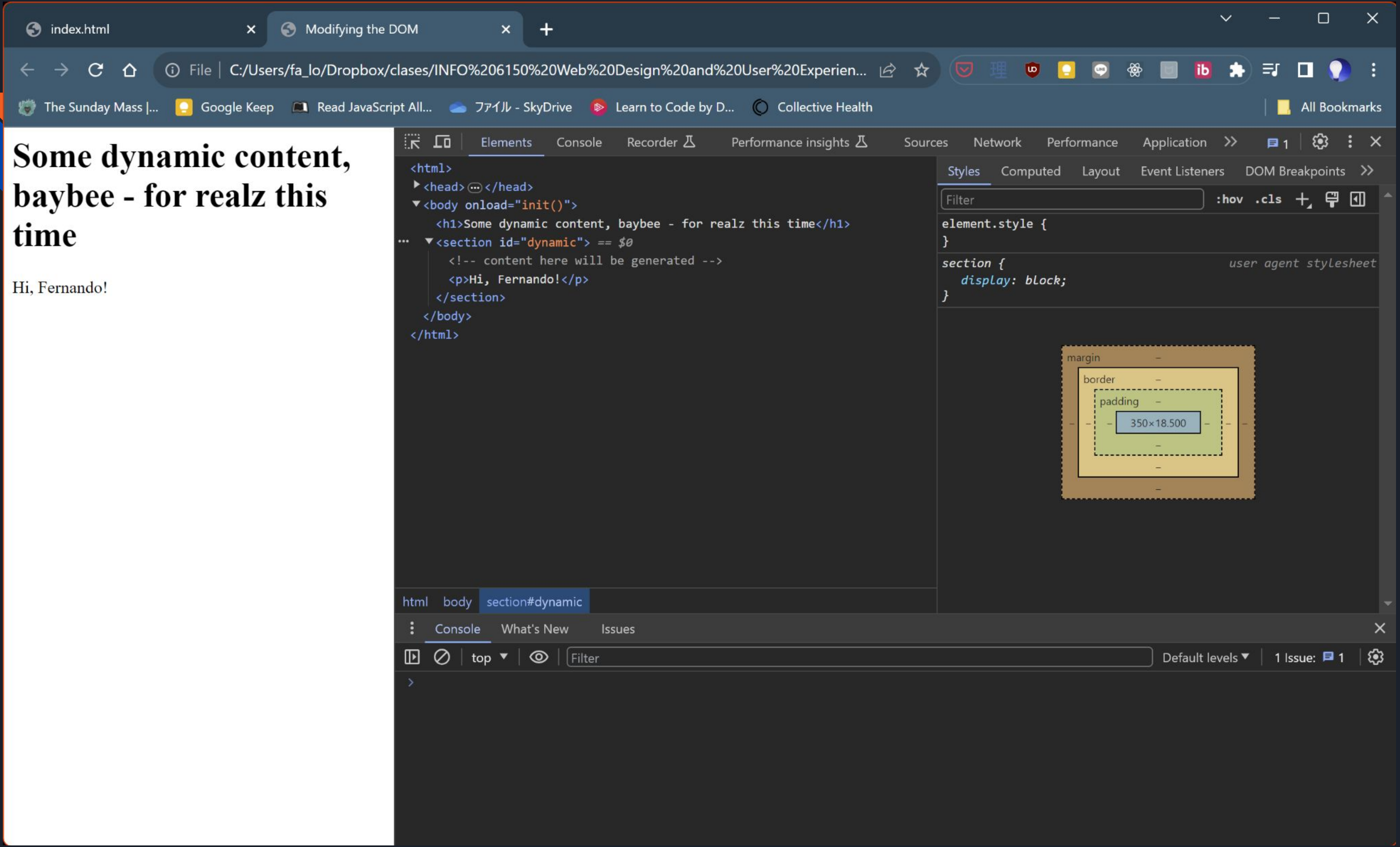
```
window.addEventListener("load", init);
```

Load “init” after images loaded (may take some time)

Running content on page load

It's also possible to write the function inside the addEventListener:

```
window.addEventListener("DOMContentLoaded", function init() {
    const username = "Fernando";
   ...
```



Some dynamic content, baybee - for realz this time

Hi, Fernando!

```
<html>
  <head>…</head>
  <body onload="init()">
    <h1>Some dynamic content, baybee - for realz this time</h1>
    <section id="dynamic"> == $0
      <!-- content here will be generated -->
      <p>Hi, Fernando!</p>
    </section>
  </body>
</html>
```

```
element.style {
}
section {                                  user agent stylesheet
  display: block;
}
```

Created by HideMaru
from Noun Project

Greet the user… in the web page

Now that you know how:

- Modify your code so that the message is shown in the web page instead. You need to create HTML content for the message.
  - Remember to make your script run on page load, not before