



Slideshow 8

# **Javascript basics**



**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>
```



# A challenge!

Created by HideMaru  
from Noun Project

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



# 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

The screenshot shows a web browser window with a single tab titled 'index.html'. The address bar shows a Google search icon. The browser's bookmark bar contains several items: 'The Sunday Mass [...]', 'Google Keep', 'Read JavaScript All...', 'ファイル - SkyDrive', 'Learn to Code by D...', and 'Collective Health'. The page content displays the heading 'A script is linked' and a paragraph: 'Open the Console in your browser's Developer Tools to see some output.'

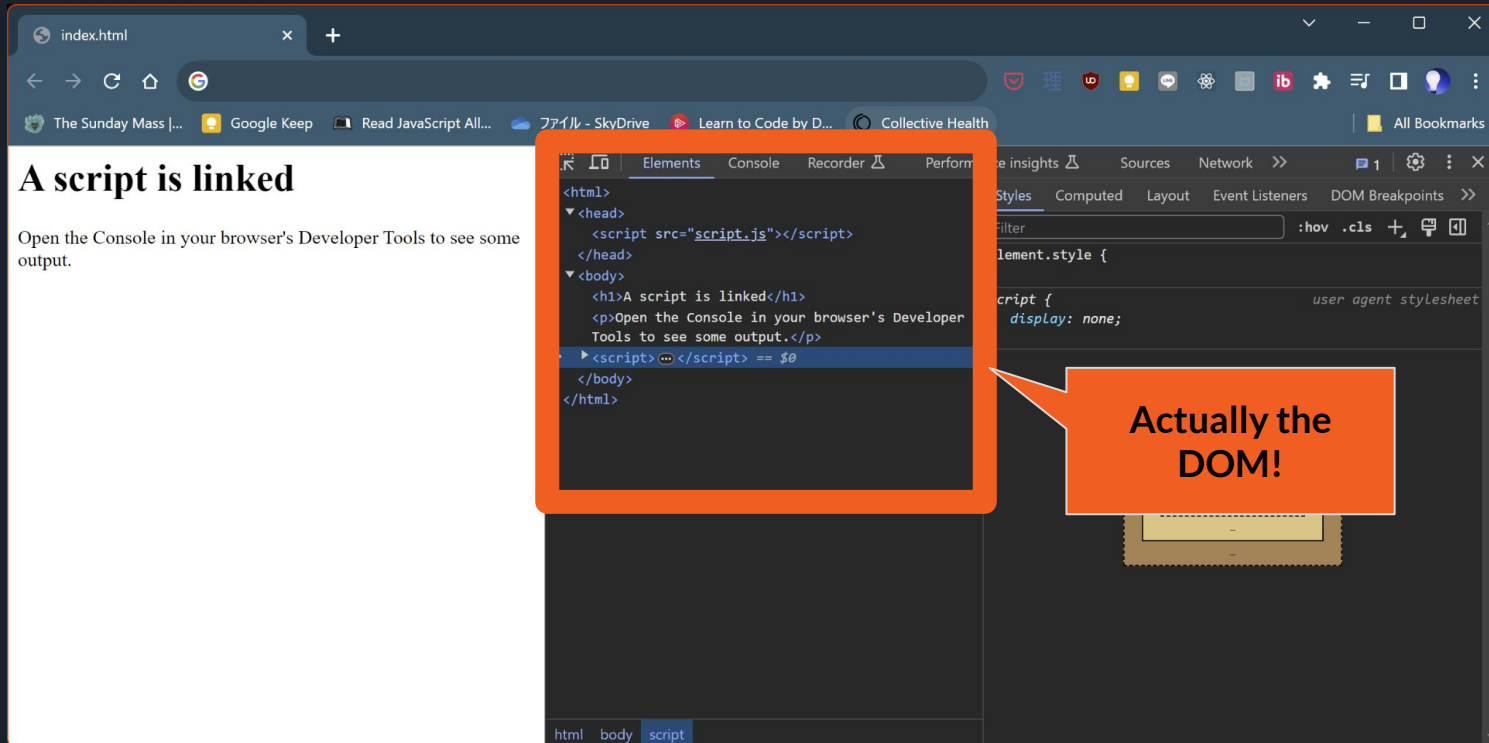
The Chrome DevTools interface is open, showing the 'Elements' panel on the left, the 'Console' panel in the middle, and the 'Styles' panel on the right. The 'Elements' panel shows the following HTML structure:

```
<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>
  </body>
</html>
```

The 'Console' panel shows a single log entry: `<script> </script> == $0`. The 'Styles' panel shows the 'element.style' and 'script' styles. The 'element.style' is empty, and the 'script' style has `display: none;` from the 'user agent stylesheet'. A visual representation of the box model is shown in the bottom right of the Styles panel, with a central blue box labeled 'auto x auto' surrounded by green 'padding' and yellow 'border' areas, all within a larger yellow 'margin' area.



# DOM and DOM manipulation



The screenshot shows a web browser window with a single tab titled "index.html". The address bar shows a Google search engine. The page content includes a heading "A script is linked" and a paragraph "Open the Console in your browser's Developer Tools to see some output." The browser's Developer Tools are open, showing the "Elements" panel on the left and the "Console" panel on the right. The "Elements" panel shows the HTML structure of the page, with the following code:

```
<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>
  </body>
</html>
```

The "Console" panel shows the following log entries:

```
> <script> </script> == $0
```

An orange box highlights the "Elements" panel, and a speech bubble points to it with the text "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>
  <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>
```

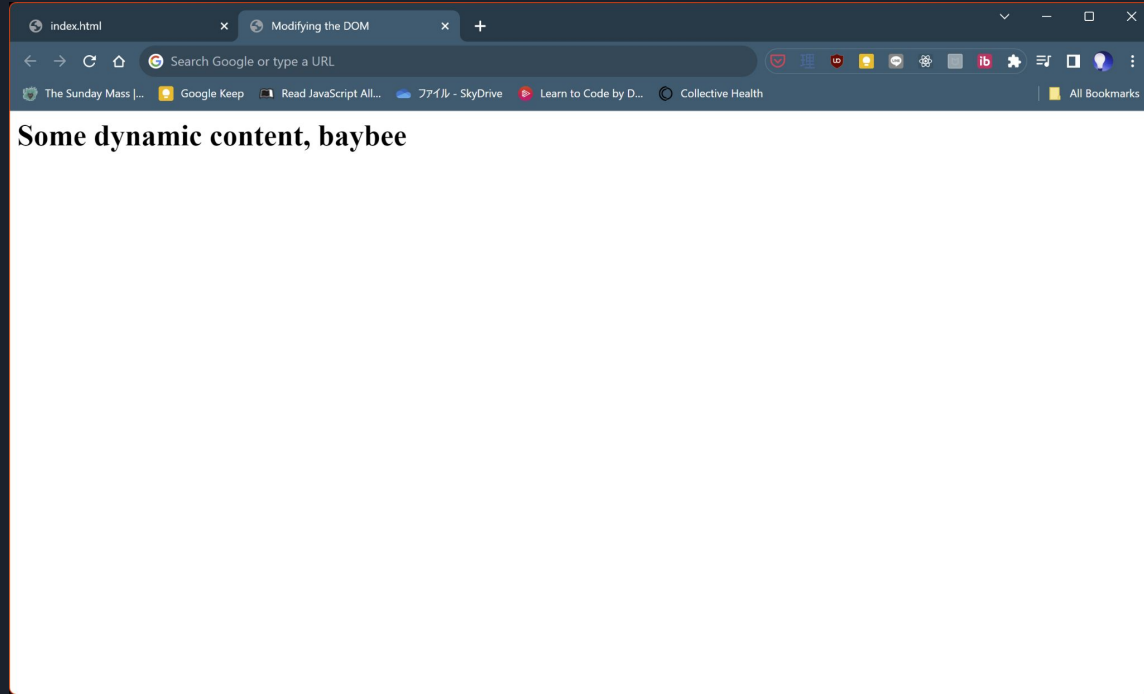


# Insert the new node in the web page

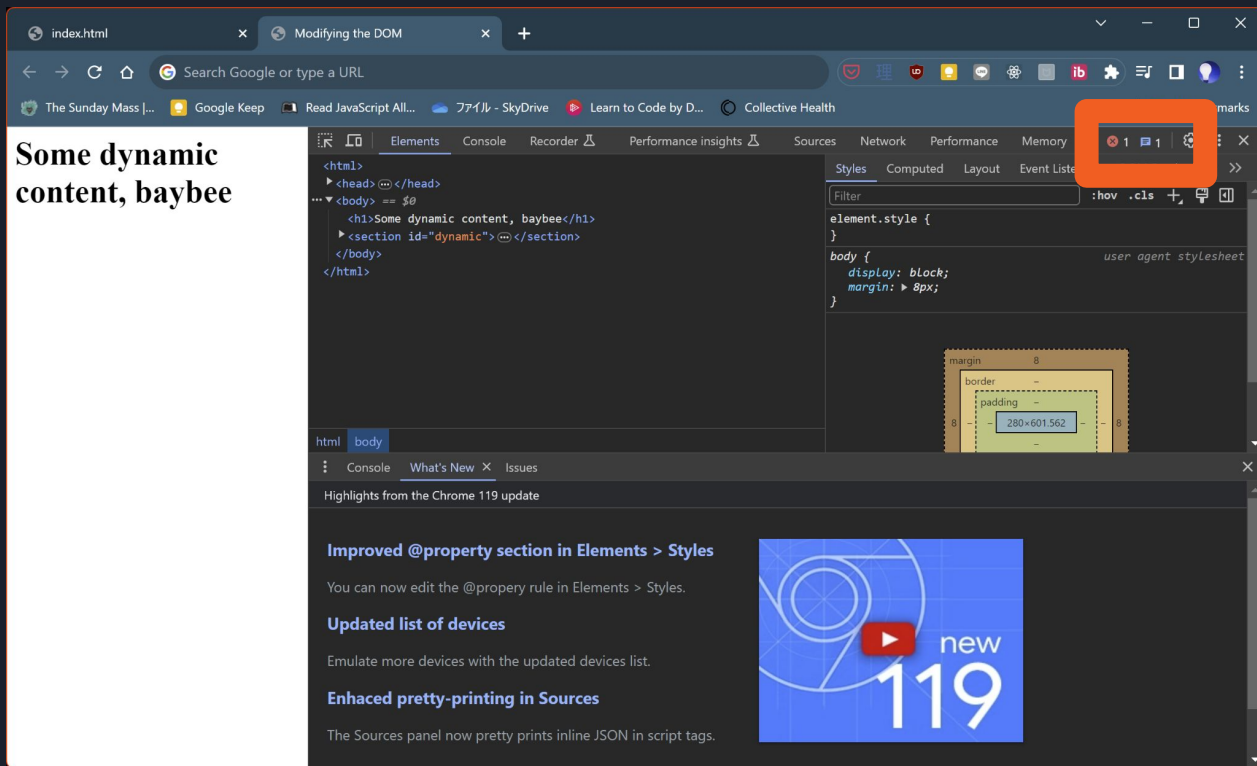
Now, back in our script:

```
// 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 WORKIIING



# Let's check on the Developer Console!




# Some dynamic content, baybee

The screenshot shows a web browser with two tabs: 'index.html' and 'Modifying the DOM'. The address bar contains 'Search Google or type a URL'. The browser's bookmark bar includes 'The Sunday Mass [...]', 'Google Keep', 'Read JavaScript All...', 'ファイル - SkyDrive', 'Learn to Code by D...', and 'Collective Health'. The page content displays the text 'Some dynamic content, baybee'.

The browser's developer tools are open, showing the 'Sources' panel. The file explorer on the left indicates the current file is 'script.js' located at 'C:/Users/fa\_lo/Dropbox/clases/'. The code editor shows the following JavaScript code:

```
10 newParagraph.appendChild(newContent);
11
12 // insert the node into the DOM tree:
13 // 1. get a reference to the node where we'll insert
14 const parent = document.getElementById('dynamic');
15
16 // 2. make our node a child of this node
17 parent.appendChild(newParagraph);
```

The error console at the bottom shows an 'Uncaught TypeError: Cannot read properties of null (reading 'appendChild')' at 'script.js:17:8'. The 'script.js' file is highlighted in the file explorer, and the error message is displayed in the console.



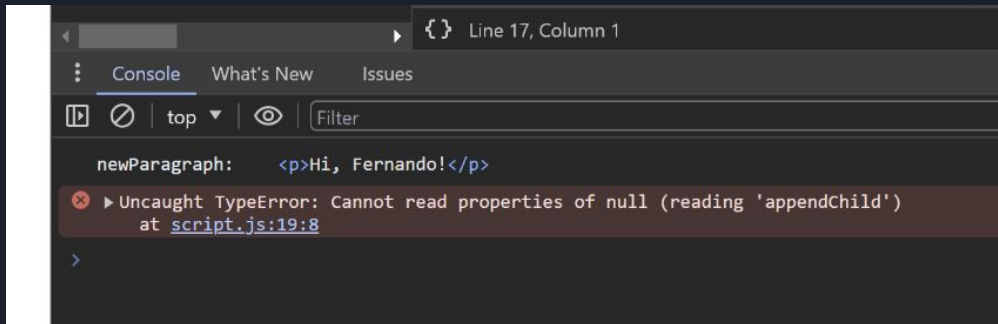
# What do you mean “null”? What’s the value?


## Debugging!

Technique 1: outputting values

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

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





# 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);
```





The screenshot shows a web browser with the address bar displaying `C:/Users/fa_lo/Dropbox/classes/INFO%206150%20Web...`. The browser's developer tools are open, with the 'Sources' panel selected. The file `script.js` is loaded, and the debugger is paused at line 18, which contains the `debugger;` statement. The code in `script.js` is as follows:

```
8 // create a new text node; this represents the text that
9 const newContent = document.createTextNode('Hi, ${username}!');
10
11 // with this, the text is attached to the new "p"
12 newParagraph.appendChild(newContent);
13
14 // insert the node into the DOM tree:
15 // 1. get a reference to the node where we'll insert
16 const parent = document.getElementById('dynamic');
17
18 debugger;
19
20 // 2. make our node a child of this node
21 parent.appendChild(newParagraph);
```

The 'Script' section of the debugger shows the following variables and their values:

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

A red callout box with the text "Variables and their values" points to the 'Script' section of the debugger.

The screenshot shows a web browser with the address bar displaying "index.html" and "Modifying the DOM". The browser's address bar shows the file path: "C:/Users/fa\_lo/Dropbox/classes/INFO%206150%20Web...". The browser's toolbar includes various icons for navigation and extensions. The developer tools are open, showing the "Sources" panel. The "script.js" file is selected, and the code is visible. The code includes comments and a function that creates a new text node and appends it to a paragraph. The "Unpause / step-by-step controls" buttons are highlighted in an orange box. A callout box points to these buttons with the text "Unpause / step-by-step controls".

index.html x Modifying the DOM x +

File | C:/Users/fa\_lo/Dropbox/classes/INFO%206150%20Web... ☆

The Sunday Mass |... Google Keep Read JavaScript All... ファイル - SkyDrive Learn to Code by D... Collective Health All Bookmarks

Elements Console Recorder Performance insights Sources Network Performance Memory

Page Workspace >> script.js x

top

file://

C:/Users/fa\_lo/Dropbox/classes

index.html

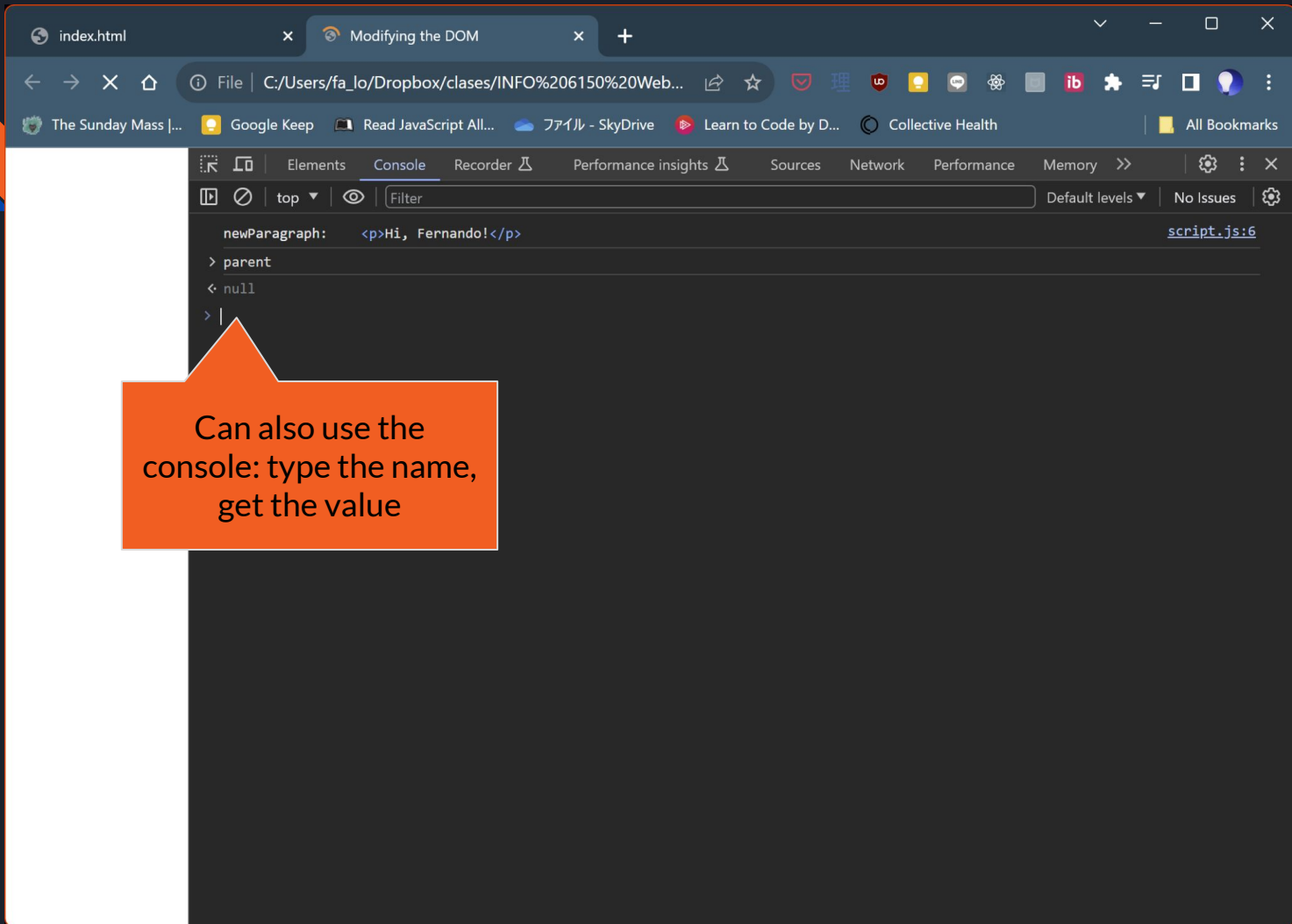
script.js

```
8 // create a new text node; this represents the text to be inserted
9 const newContent = document.createTextNode('Hi, ${username}!');
10
11 // with this, the text is attached to the new "p"
12 newParagraph.appendChild(newContent);
13
14 // insert the node into the DOM tree:
15 // 1. get a reference to the node we want to insert into
16 const parent = document.getElementById('dynamic');
```

Unpause / step-by-step controls

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



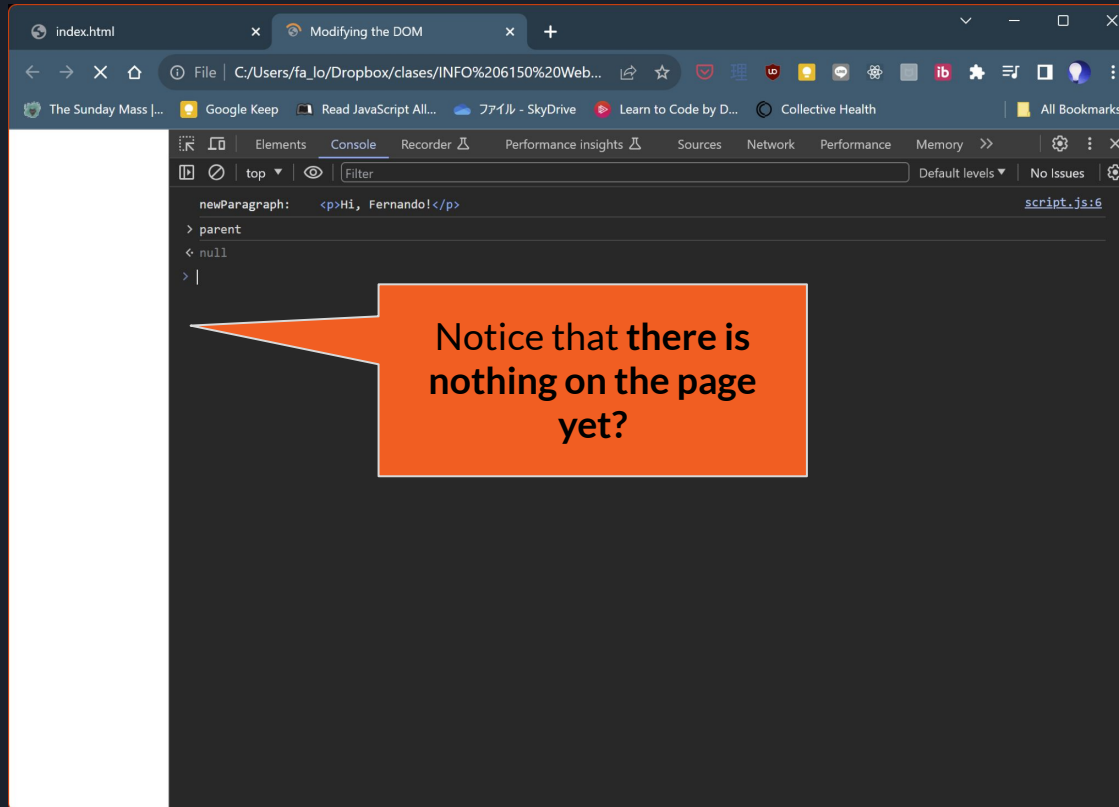




# So... what happened?

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

# So... what happened?





# 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>
  <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!

The screenshot shows a web browser window with two tabs: 'index.html' and 'Modifying the DOM'. The address bar shows the file path: 'C:/Users/fa\_lo/Dropbox/classes/INFO%206150%20Web%20Design%20and%20User%20Experien...'. The browser's toolbar includes various icons for navigation and extensions. The page content displays the text 'Some dynamic content, baybee - for realz this time' and 'Hi, Fernando!'. The Chrome DevTools interface is open, showing the 'Elements' panel with the following HTML structure:

```
<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>
```

The 'Styles' panel shows the 'element.style' and 'section' styles, with 'display: block;' applied to the 'section' element. A visual representation of the box model is shown, with dimensions 350x18.500. The 'Console' panel is also visible, showing the 'html' and 'body' objects, and the 'section#dynamic' element selected.



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