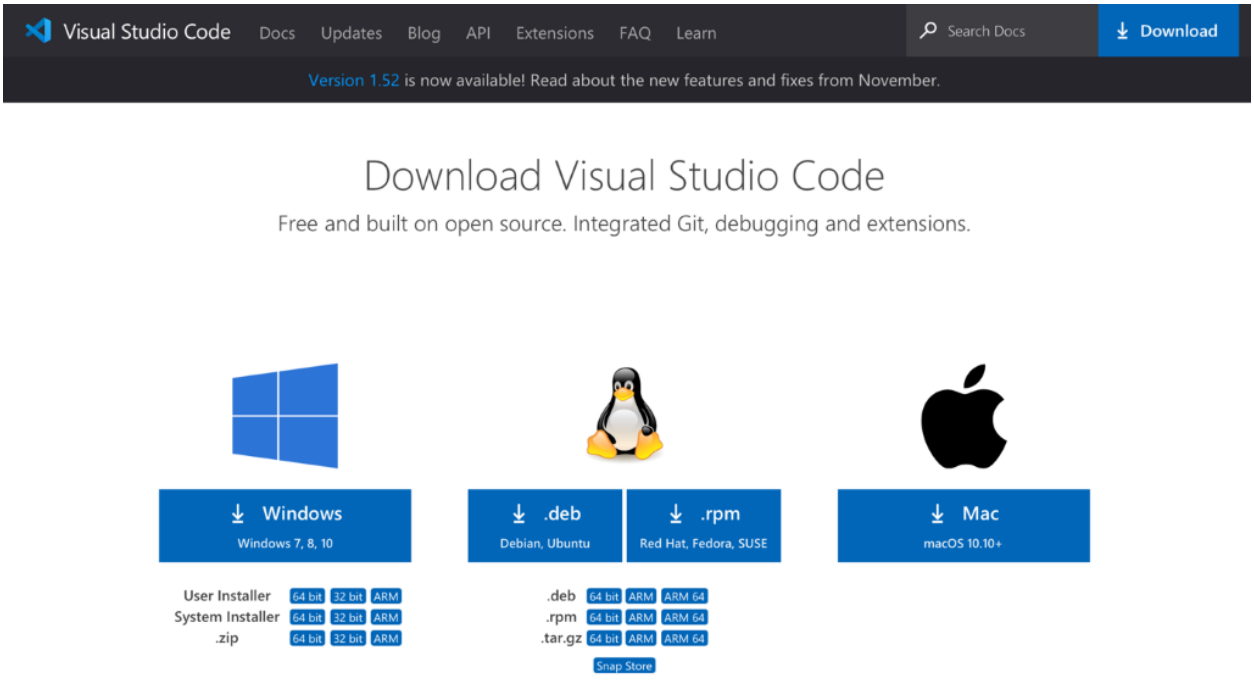
EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: INSTALANDO VISUAL STUDIO CODE EN NUESTRO COMPUTADOR.
- EXERCISE 2: NUESTRO PRIMER "HOLA MUNDO" CON JAVASCRIPT.
- EXERCISE 3: DECLARACIÓN DE VARIABLES Y CAMBIOS DE TIPOS DE DATOS EN JAVASCRIPT.

EXERCISE 1: INSTALANDO VISUAL STUDIO CODE EN NUESTRO COMPUTADOR

Nota: a JavaScript se le hará referencia con la abreviación "js" en este documento.

Haremos un ejemplo para familiarizarnos con la declaración de variables, y lograr imprimir nuestro primer "Hola Mundo" en JavaScript. Para ello, utilizaremos la herramienta Visual Studio Code, que podrás descargar desde su sitio oficial.

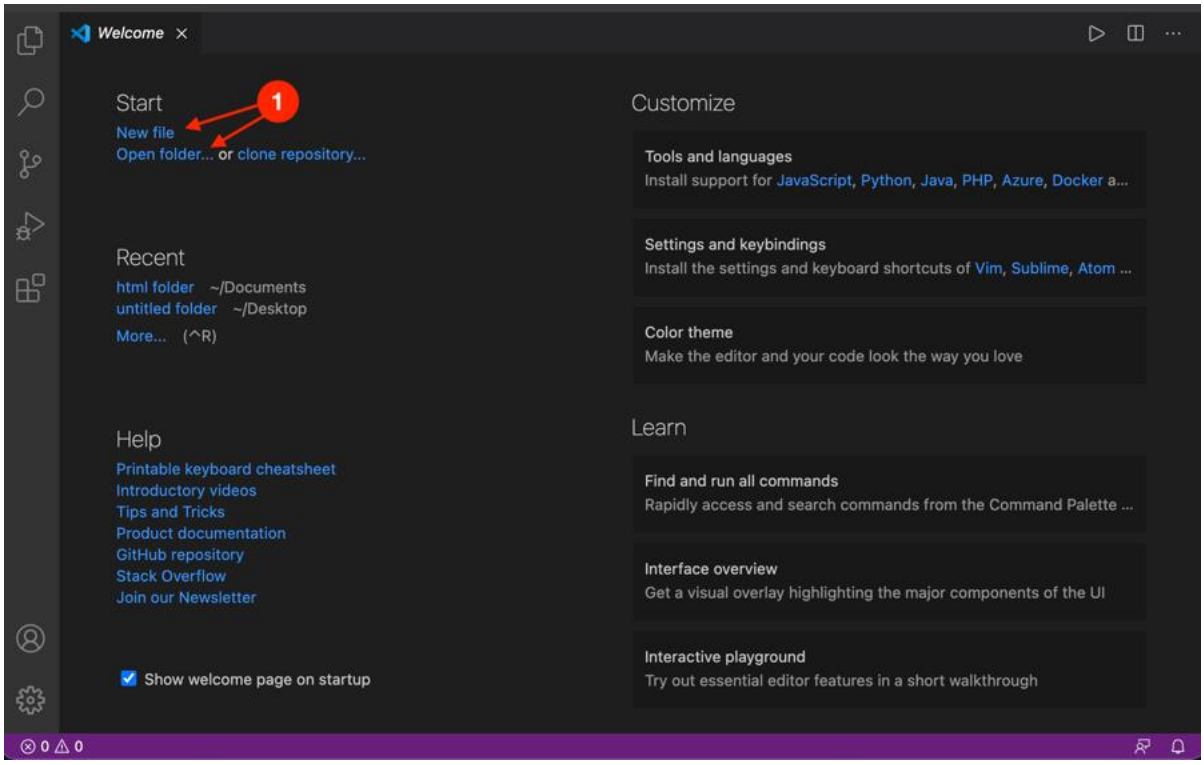


Después de descargar la versión que se adapta al sistema operativo de tu computadora, aparecerá un instalador con instrucciones, que te guiarán a través del proceso de configuración; una vez listas, habrás

instalado con éxito Visual Studio Code. Ahora, nuestro entorno de desarrollo estará completamente configurado, y listo para programar en JavaScript. ¡Empecemos!

EXERCISE 2: NUESTRO PRIMER "HOLA MUNDO" CON JAVASCRIPT

Al iniciar VS code, debemos ir a **Start,** y seleccionar **"New File"** o **"Open Folder"**, para crear un nuevo archivo, o abrir una carpeta que contenga nuestros archivos de código respectivamente. A continuación, podrás encontrar esas opciones en la enumeración 1.



Crearemos un nuevo archivo HTML, llamado index.html, el cual incluirá un H1 para visualizar el contenido de la página. Hasta el momento, nuestro código es el siguiente:

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prueba de JavaScript</title>
</head>

<body>
    <h1>Hola! Esta es una prueba de JavaScript</h1>
</body>
</html>
```

¿Dónde se coloca el código JavaScript?: esto depende de si lo incluiremos incrustado en el archivo HTML, o si lo incluiremos como un archivo externo.

Para incrustarlo, basta con escribir el código JS entre etiquetas: `<script></script>`. Éste se puede colocar tanto en el head, como en el body de un HTML (con esto nos referimos entre las etiquetas `<head></head>`, o entre `<body></body>`). Es una buena práctica colocar las etiquetas `<script>` para JavaScript, justo antes de la etiqueta `</body>` de cierre, en vez de colocarlo en la sección `<head>` de un HTML; se considera así porque si tiene mucho JavaScript, puede ralentizar visiblemente la carga de la página, pues el navegador cargará todo el JavaScript antes de cargar el HTML.

```
<body>
    <h1>Hola! Esta es una prueba de JavaScript</h1>

    <script>
    // Aqui podemos programar con JavaScript.
    </script>
</body>
</html>
```

Para incluir JS como un archivo externo, se debe tener un archivo con la extensión `".js"`, como por ejemplo: `"script.js"`. Como a cualquier otro archivo externo, a éste se le debe hacer referencia en el head de un HTML, de la siguiente manera:

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

Para incluir nuestro código JS como un archivo externo, debemos ir a donde se encuentra index.html, y en el espacio enmarcado con azul hacemos **CLIC** derecho > **New File** > y la llamaremos: `"test.js"`.

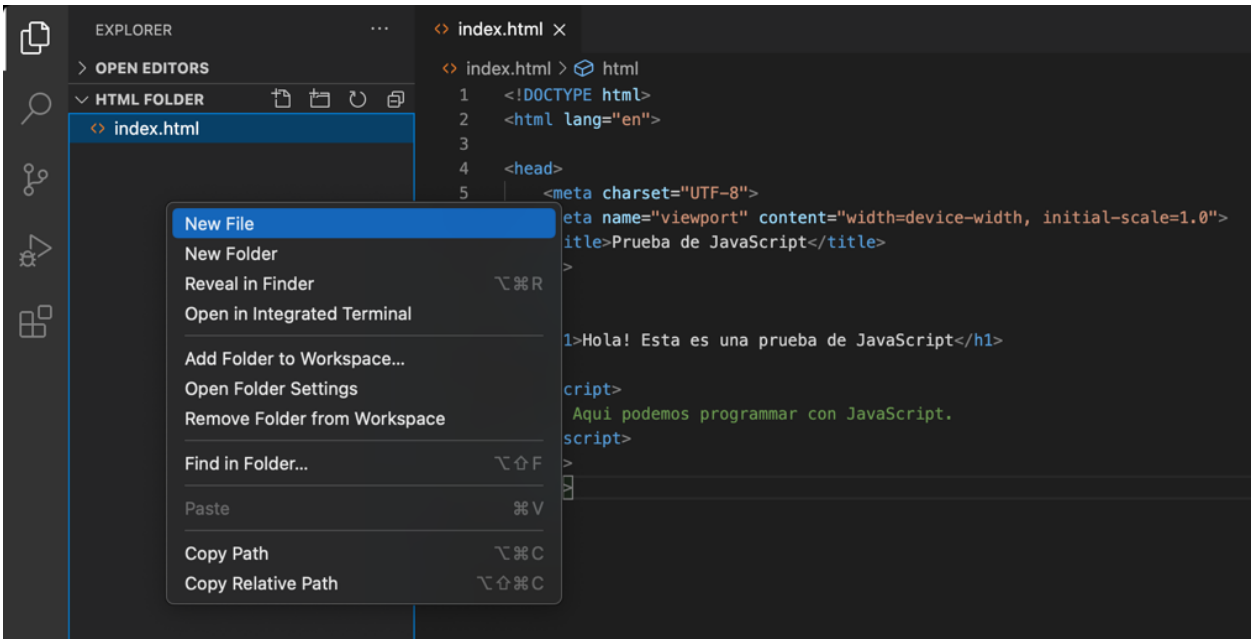


Imagen 2: Creación de un nuevo archivo, el cual llamaremos test.js.

Como podemos ver a continuación, hacemos referencia a este archivo de JS, dentro de una etiqueta script en el **head** del HTML.

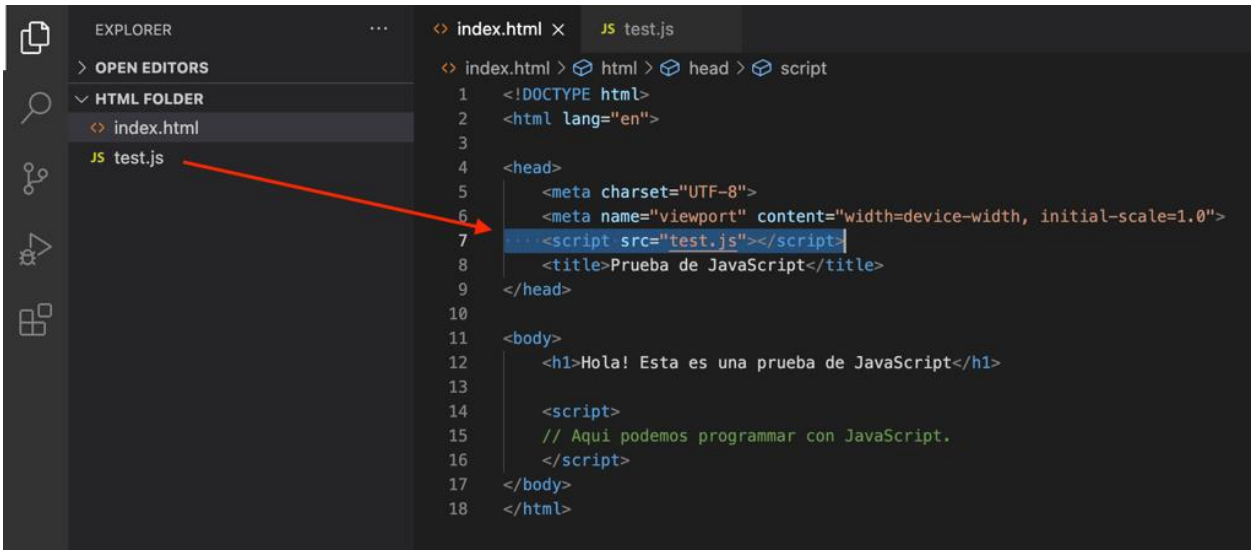
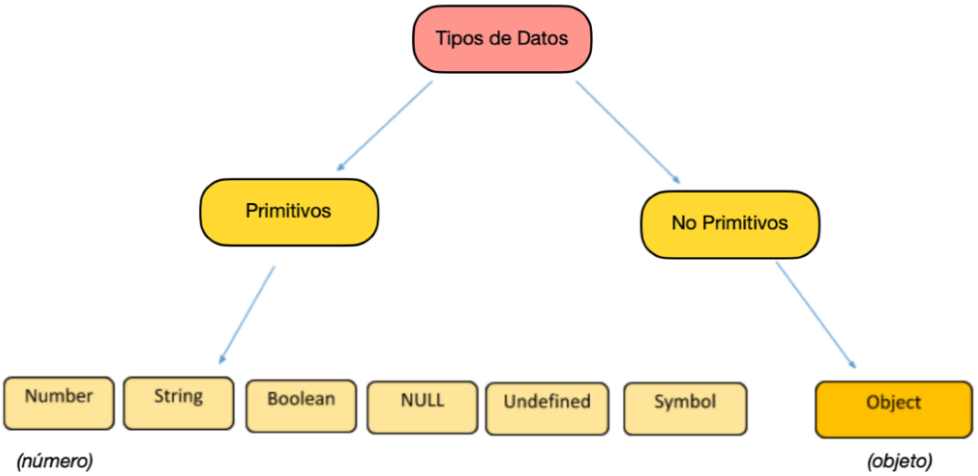


Imagen 3: Incluimos el archivo test.js en nuestro HTML, por medio de referenciarlo en el head.

Ahora ya tenemos demostrado *donde* ubicar nuestro código JavaScript, pero nos falta ver *como* se utiliza. Empezaremos revisando cómo se declaran las variables. Además, aprenderemos a emplear la consola de un navegador, para visualizar el *output* o *resultado* de nuestro código. En primer lugar, debemos familiarizarnos con los distintos tipos de datos que se pueden utilizar en el lenguaje de JavaScript, éstos constan de: datos primitivos y no primitivos; como muestra la siguiente imagen:



En JavaScript, los comentarios se escriben anteponiendo 2 barras inclinadas `“//”`. Podemos expresar literales numéricos, con y sin decimales, así como `Strings` con comillas dobles(`“ ”`), y comillas simples (`‘ ’`).

```
// Esto es un comentario.

// Estos son literales, valores fijos:

// números se pueden escribir así
10.00
10

// Strings se pueden escribir así:
"hola"
'hola'
```

En JavaScript, estas distintas tipografías no marcan una diferencia real al momento de utilizar los valores en código.

```
        // Variables se escriben así:

        var m; // variable no instanciada.

        var m = 10

        // No es necesario poner un ';' al final de cada línea.

        var m = 10; // No difiere en nada.
```

Otra particularidad de este lenguaje es que no requiere de la inserción de un punto y coma (**";"**), y el hecho de incluirla o no, es totalmente opcional y no afecta el valor. Además, si nos fijamos en las líneas de código anteriores, las variables pueden tomar cualquier valor de nombre. En este caso, utilizamos el nombre **"m"** para la variable.

```
        m = 'hola'

        m = 12.324

        m = "los valores en JS son considerados dinámicos"

        var m = 'porque se les puede cambiar el valor y tipo de dato
sencillamente'

        var enSerio = true

        enserio = 0
```

Cómo muestran los valores de **m** en estas líneas, JavaScript tiene variables que son consideradas dinámicas, ya que pueden cambiar de tipo y de valor de manera sencilla, sin tener que establecer previamente el tipo de dato que son. Es importante destacar que los nombres de las variables distinguen mayúsculas y minúsculas, por lo que **"enSerio"**, es totalmente distinta a **"enserio"**. ¿Cómo lo podemos comprobar?: mediante una consola, esta es una de las muchas herramientas del browser, disponibles para ayudarnos a desarrollar código JavaScript.

Todos los navegadores tienen una consola, la cual nos permite inspeccionar el código fuente de una página. Por lo general, se logra llegar a ella haciendo **CLIC** derecho en cualquier página web, y luego seleccionar la opción de: "inspeccionar"

En Google Chrome, la opción se encuentra al hacer **CLIC** derecho.

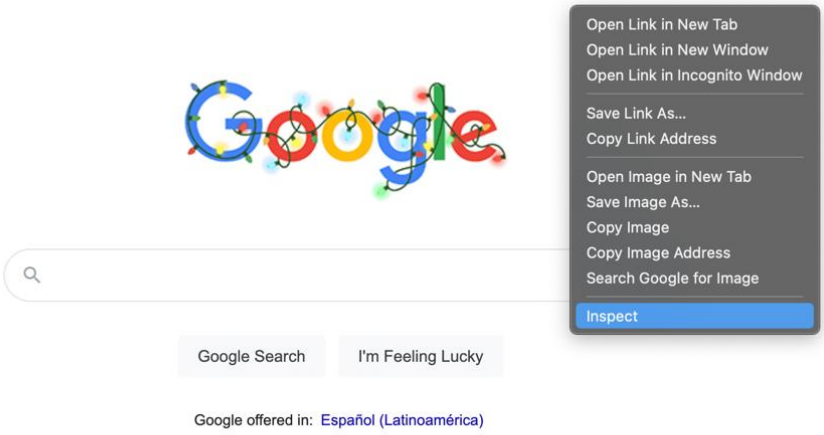


Imagen 4: Podemos llegar a la consola inspeccionando cualquier página web.

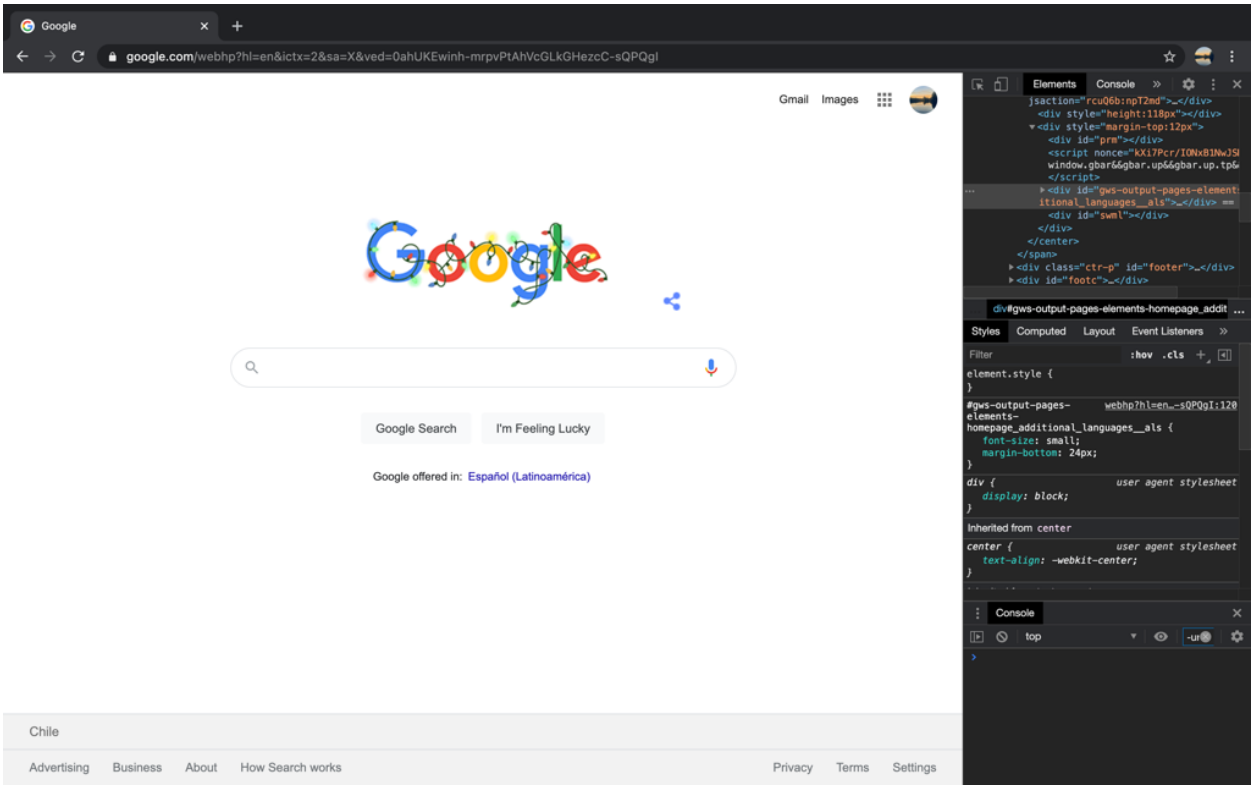


Imagen 5: Lista de la consola en Google Chrome.

La consola se puede asemejar a una máquina de rayos X, dado que nos permite ver todo lo que hay detrás de una página web, desde el estilo que tiene, hasta la funcionalidad que puede cumplir. E incluso, nos puede mostrar las falencias en nuestro código.

En nuestro caso, para poder visualizar algunos elementos en JavaScript, debemos primero ir a la ruta donde se encuentra el archivo HTML en nuestro navegador. Al realizar esto, veremos lo siguiente:

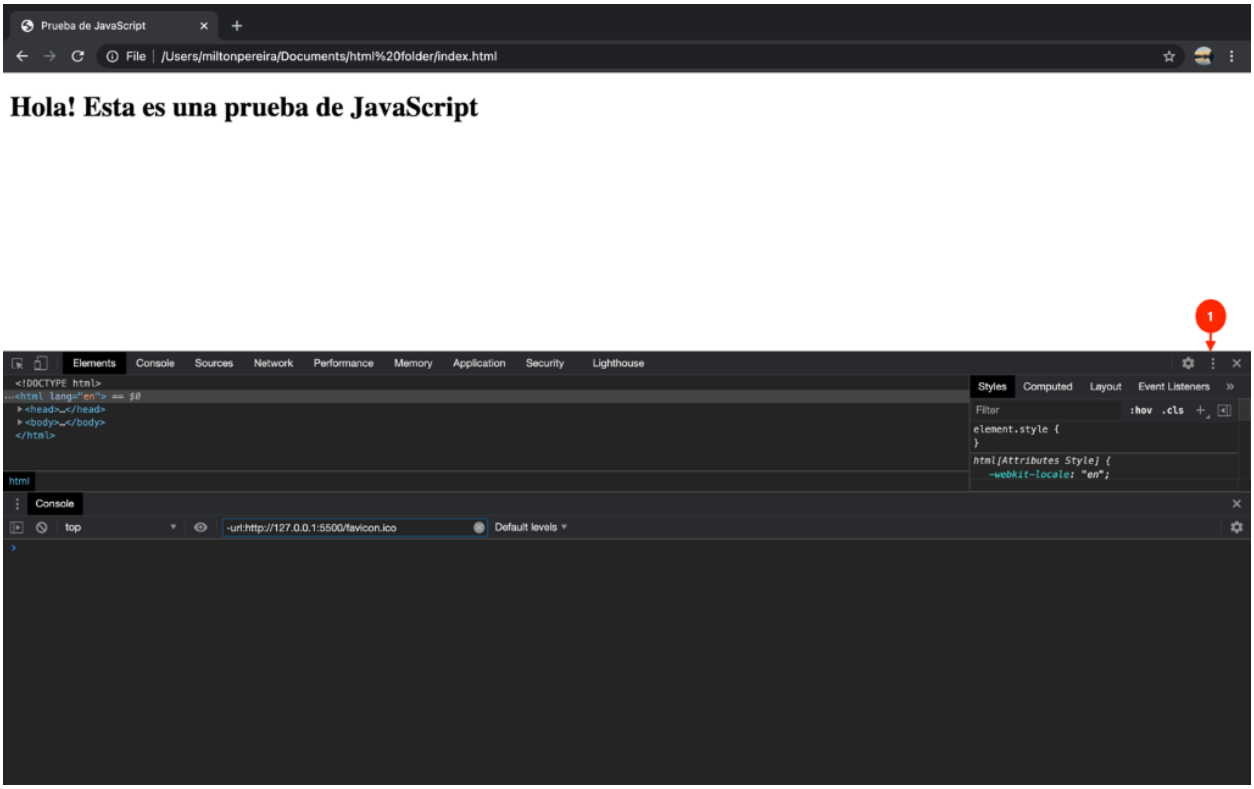


Imagen 6: Nuestro archivo HTML en el navegador.

Cómo puedes ver, nuestro archivo HTML no contiene mucho contenido, y nuestra consola ahora está orientada horizontalmente en la parte inferior de la página, dado que la cambiamos en la opción que se encuentra enumerada con el 1.

El código de nuestra página HTML es el siguiente:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prueba de JavaScript</title>
</head>
<body>
    <h1>Hola! Esta es una prueba de JavaScript</h1>
    <script>
        // Esto es un comentario.
```

```
12
13          // Estos son literales, valores fijos:
14
15          // números se pueden escribir así
16          10.00
17          10
18
19          // Strings se pueden escribir así:
20          "hola"
21          'hola'
22
23          // Variables se escriben así:
24
25          var m; // variable no instanciada.
26
27          var m = 10
28
29          // No es necesario poner un ';' al final de cada línea.
30
31          var m = 10; // No difiere en nada.
32
33          m = 'hola'
34
35          m = 12.324
36
37          m = "los valores en JS son considerados dinámicos"
38
39          var m = 'porque se les puede cambiar el valor y tipo de dato
40  sencillamente'
41
42          var enSerio = true
43
44          enserio = 0
45
46      </script>
47  </body>
48  </html>
49
```

Podemos lograr imprimir un valor de JavaScript en la consola, utilizando el método: `console.log()`. En este caso, intentaremos imprimir nuestro primer “Hola Mundo” en JavaScript. Para ello, vamos a declarar lo siguiente:

```
var primer = "Este es mi primer "
var hola = 'Hola Mundo '
var js = "en JavaScript!"
console.log(primer + hola + js);
```

El resultado de estas líneas de código por consola, es el siguiente:

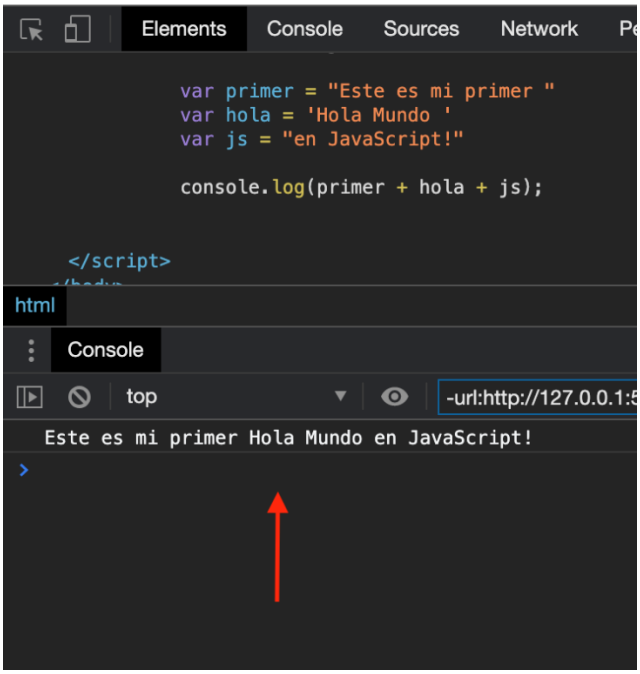


Imagen 7: Nuestro primer Hola Mundo en JavaScript.

De esta manera, hemos logrado mostrar por consola nuestro primer Hola Mundo en el lenguaje de JavaScript.

EXERCISE 3: DECLARACIÓN DE VARIABLES Y CAMBIOS DE TIPOS DE DATOS EN JAVASCRIPT

Después de haber logrado mostrar por consola nuestro primer “Hola Mundo” en JavaScript, nos resta revisar cómo podemos seguir declarando nuestras variables y a la vez, efectuar los cambios de tipos de datos en este lenguaje. En el ejemplo anterior, teníamos declarado el siguiente código en nuestro HTML:

```
<script>
        // Strings se pueden escribir así:
        "hola"
        'hola'
        // Variables se escriben así:
        var m; // variable no instanciada.
        var m = 10
        // No es necesario poner un ';' al final de cada línea.
        var m = 10; // No difiere en nada.
        m = 'hola'
        m = 12.324
        m = "los valores en JS son considerados dinámicos"
        var m = 'porque se les puede cambiar el valor y tipo de dato
sencillamente'
        var enSerio = true
        enserio = 0
</script>
```

Ahora, enfocaremos nuestra atención en las últimas dos variables, que son: `enSerio` y `enserio`. Si quisiéramos imprimir por consola su valor, tendríamos que hacer lo siguiente:

```

var enSerio = true
enserio = 0

console.log("valor de enSerio: "+ enSerio); // este es un boolean
console.log("valor de enserio: "+ enserio); // este es un valor numérico

```

Si guardamos nuestros cambios, y refrescamos la página en el navegador, el resultado por consola será:

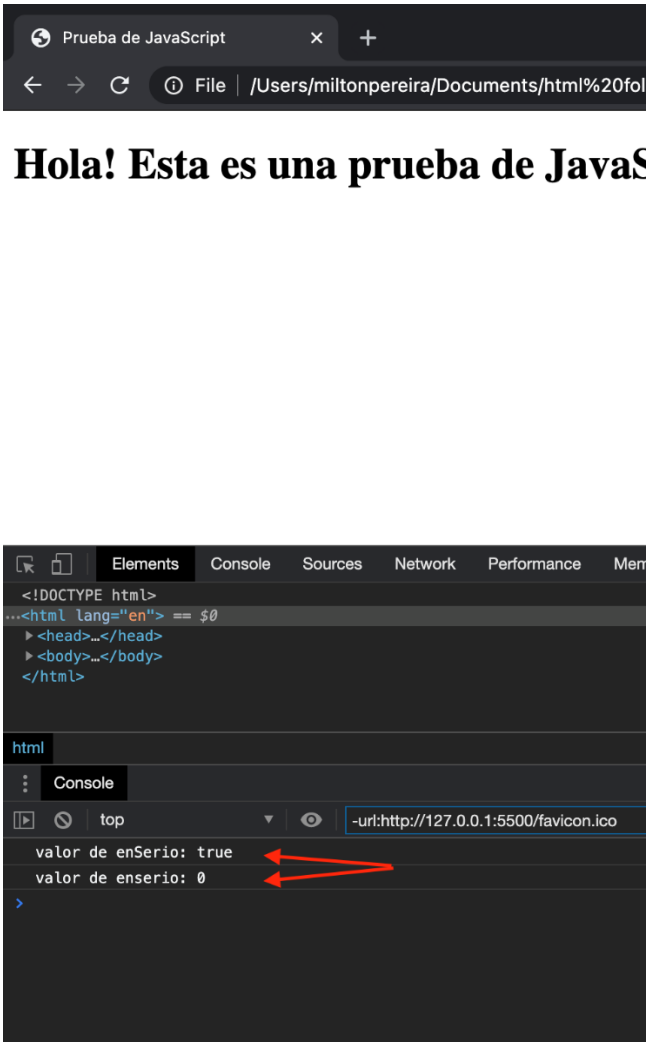


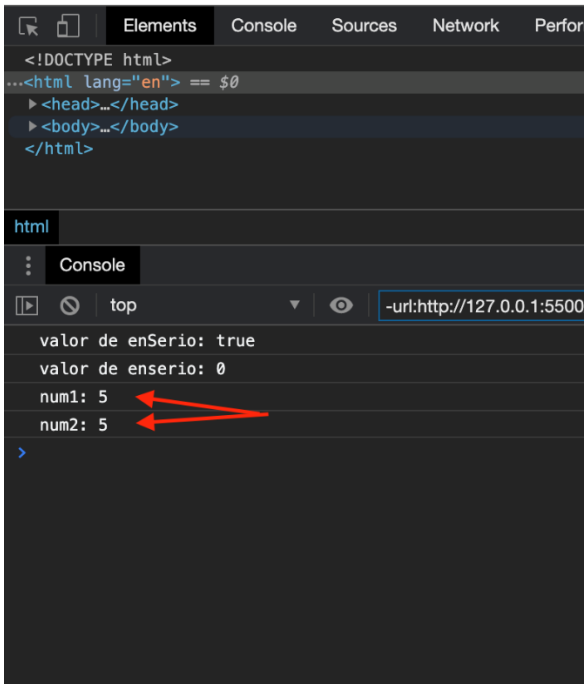
Imagen 1: El resultado de usar el método console.log().

El método `console.log()` es muy útil, pero se puede prestar para ambigüedades. Como por ejemplo, el siguiente caso:

```
var num1 = 5
var num2 = '5'

console.log("num1: "+ num1) // number
console.log("num2: "+ num2) // String
```

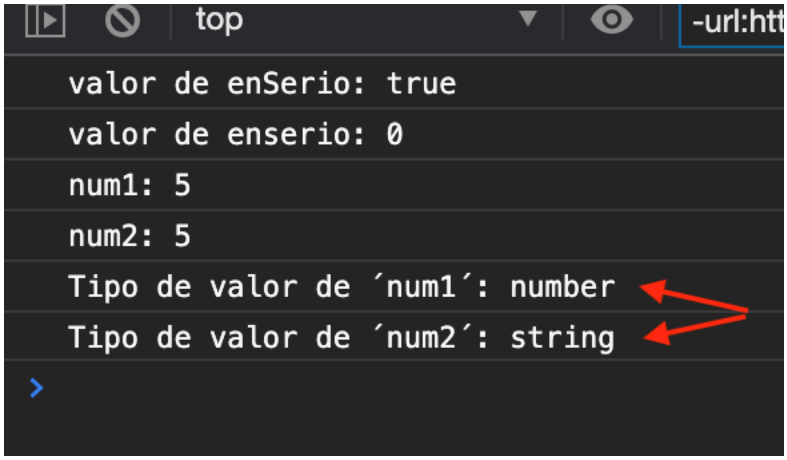
Por consola tenemos el siguiente resultado:



Cómo se visualiza, esto puede causar un poco de confusión, pues el valor por consola pareciera ser idéntico, aunque no son del mismo tipo de datos. ¿Existe otro método que nos ayude a determinar el tipo de dato?: si, el método `typeof()` recibe por parámetro el valor del cual deseamos determinar el tipo de dato, y retorna un String con éste ya identificado.

```
1 console.log("Tipo de valor de ´num1´: "+ typeof(num1)) // number
2 console.log("Tipo de valor de ´num2´: "+ typeof(num2)) // String
```

Estas dos líneas de código resultan en el siguiente *output (salida)*:



Ahora sí, ya tenemos identificados los tipos de datos correspondientes, pero ¿cómo podemos cambiarlos?: valiéndonos de las siguientes funciones:

`Number()` – cambia cualquier tipo de dato por parámetro, a un número.
`String()` – cambia cualquier tipo de dato por parámetro, a un `String`.
`Boolean()` – cambia cualquier tipo de dato por parámetro, a un `boolean`.

Por ejemplo, supongamos que tenemos una variable con un valor numérico, que deseamos transformarla a un `String` o a un `boolean`:

```
var x = 1

// Cambiamos el tipo de dato de number a String
console.log("x to String: " + String(x) + ", type: " + typeof (String(x)));

// Cambiamos el tipo de dato de number a boolean (0 = false, 1 = true)
console.log("x to Boolean: " + Boolean(x) + ", type: " + typeof
(Boolean(x)));
```

Al efectuar esto, el resultado por la consola es el siguiente:

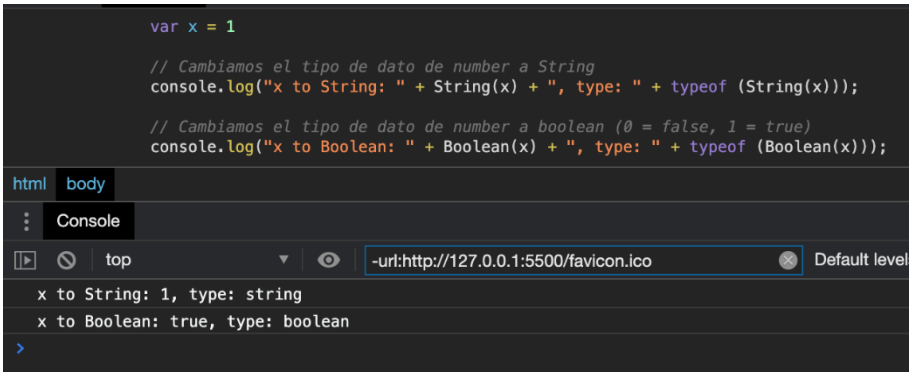


Imagen 2: Resultado de conversión de tipo de datos.

JavaScript también nos permite convertir valores de tipo `String`, a números:

```
// String a numero
console.log("1: " + Number("7.3453"));
console.log("2: " + Number(" "));
console.log("3: " + Number(""));
console.log("4: " + Number("102 32"));
```

Por consola podemos apreciar los resultados:

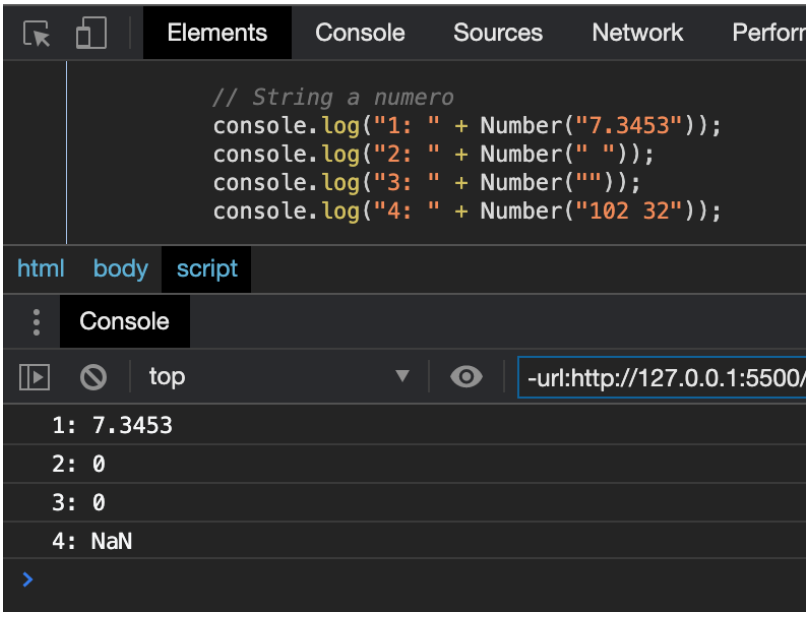


Imagen 3: Convirtiendo un String a un número.

También tenemos la oportunidad de utilizar símbolos que representan operaciones aritméticas, y al usarlas en conjunto con Strings, podemos encontrar comportamiento muy distintivo, como muestra el siguiente código e imagen:

```
var doce = '12'

console.log(doce + 3)
console.log(doce / 3)
console.log(doce * 3)
console.log(doce - 3)
```

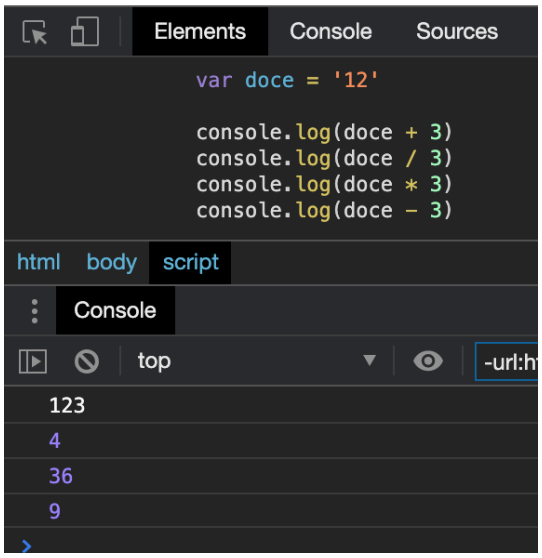


Imagen 4: Operaciones aritméticas con un String.

Se puede observar que, al momento de sumarle un número o un `String`, el resultado es `"123"`, siendo esta una adición del `3` al `String`. Las demás operaciones terminan en un resultado numérico, tomando el `12` como si fuese otro valor numérico con el cual poder operar.

Hasta el momento, el método `console.log()` nos ha ayudado a visualizar nuestro código JavaScript. Este es uno de los muchos métodos considerados "salidas" (*outputs*). `console.log()`, y los demás que hemos implementado, son denominados: "objetos". Esto se debe a que JS se puede escribir con distintos estilos de programación, como: imperativo, declarativo o funcional; pero primordialmente, JavaScript es un idioma

orientado a objetos, que consta de “usar objetos para modelar cosas del mundo real que queremos representar dentro de nuestros programas, y/o proporcionar una forma sencilla de acceder a la funcionalidad que, de otro modo, sería difícil o imposible de utilizar.”

(Puedes encontrar más información sobre la programación orientada a objetos en JavaScript aquí)

Aparte del `console.log()`, los demás métodos que nos permiten ver las salidas de nuestro código son los siguientes:

```
1  // Para sobre-escribir el valor de un elemento HTML
2  document.getElementById("primero").innerHTML = 'Esto es una demostración
3  de cómo usar JavaScript';
4  document.getElementById("segundo").innerHTML = "La suma de 8 y 7 es:";
5
6  // Para escribir en el documento HTML
7  document.write(8 +7);
8
9  // Para escribir en una caja de advertencia (Alert box)
10 alert('Hola Mundo desde JavaScript')
```

En las primeras 4 líneas de código, podrás notar que empieza con la palabra `document`, que hace referencia al archivo HTML. La continuación de ésta, es una serie de métodos que podemos utilizar. `“getElementById()”` obtiene un elemento por el id que le asignamos, y `“write()”` sobreescribe un valor en el HTML. `“alert()”` genera una caja de advertencia o un pop-up en la página, donde podemos poblar esa cajita con valores, ya sean numéricos o de tipo `String`.

Al guardar los cambios, y recargar nuestra página en el navegador, lo primero que observamos es el `alert`:

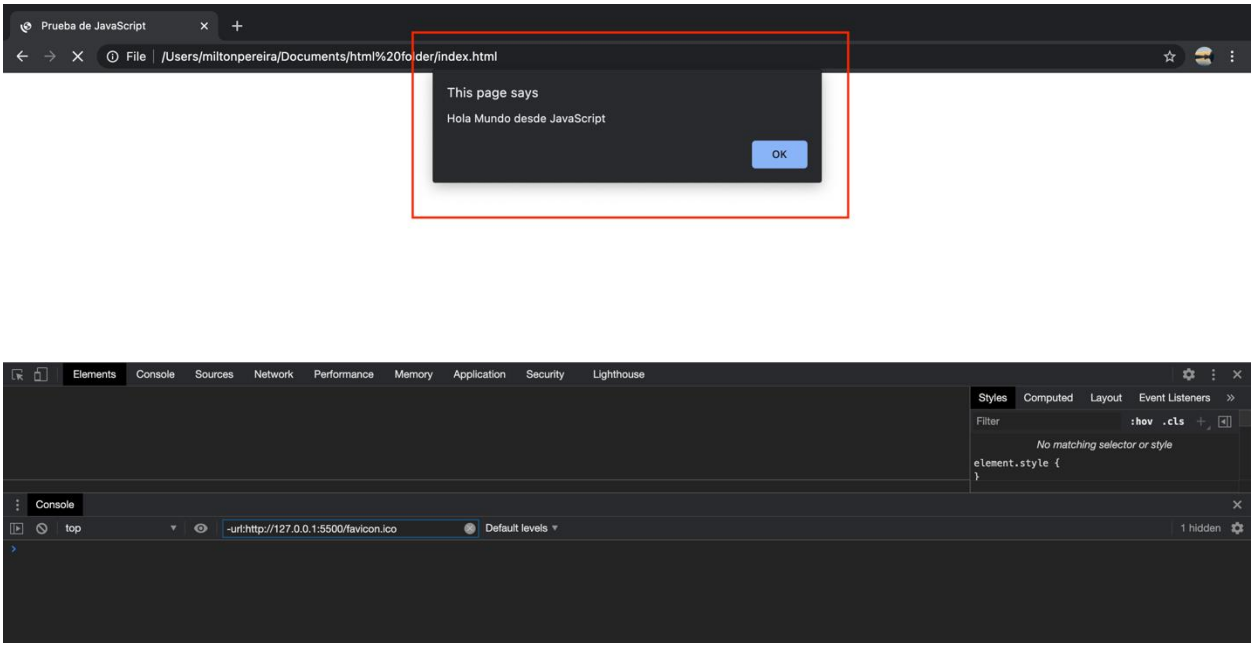


Imagen 5: Un mensaje de tipo alert.

Posterior al mensaje `alert`, vemos el siguiente resultado de nuestras primeras 4 líneas de código:

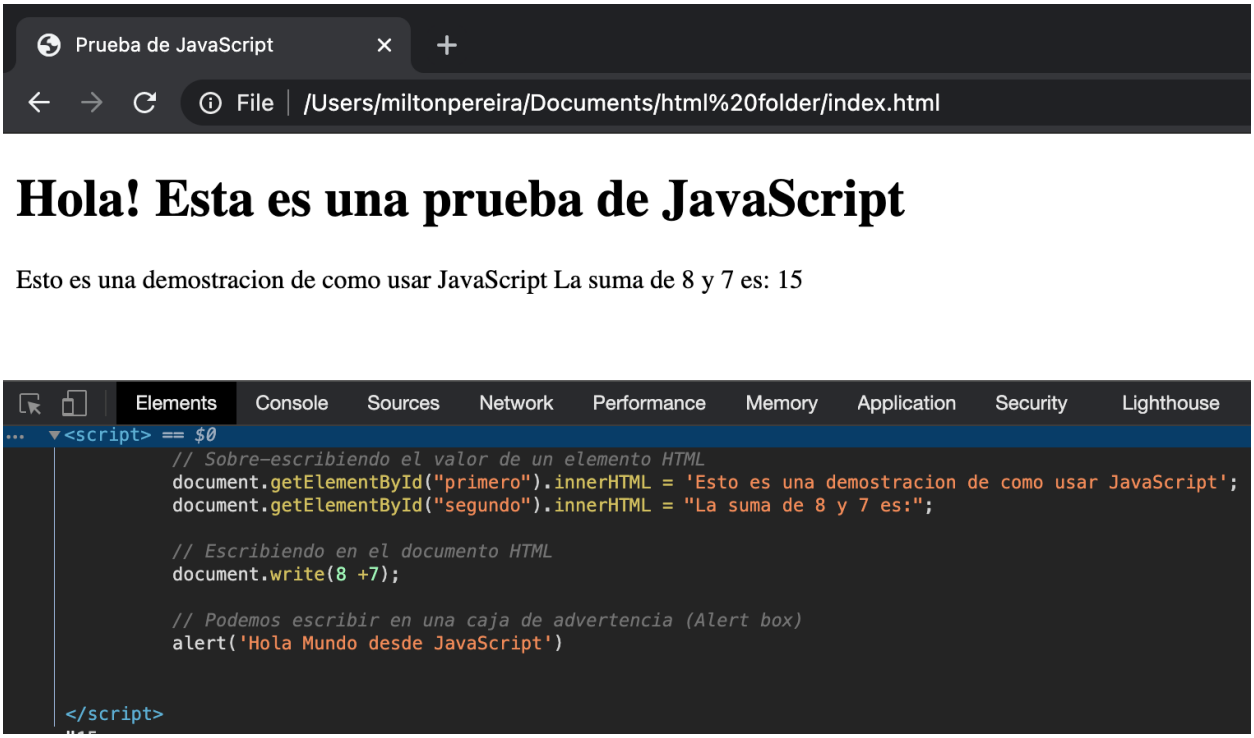


Imagen 6: El resultado de las otras líneas de código que habíamos escrito para manipular el archivo html.

"document.write()" también nos permite generar elementos HTML completos de manera dinámica. Por ejemplo, si quisiéramos escribir el código que teníamos dentro de nuestras etiquetas Body, podemos generarlo dinámicamente con las siguientes líneas de código JavaScript, obteniendo el mismo resultado visual:

```
<!DOCTYPE html>
<html lang="en">

<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Prueba de JavaScript</title>
</head>

<body>
        <script>
        document.write("    <h1>Hola! Esta es una prueba de
JavaScript<\/h1>");
        document.write("    <span id=\"primero\"><\/span>");
```

```
16          document.write("    <span id=\"segundo\"><\/span>");
17          document.write("    <span id=\"tercero\"><\/span>");
18          </script>
19  </body>
20  </html>
```

Si observas, nuestro HTML está completamente vacío, salvo el código JavaScript; pero el resultado es exactamente igual a si hubiésemos escrito manualmente los elementos HTML.

De esta manera, queda demostrada la funcionalidad que puede tener JavaScript al momento de manipular un archivo HTML, y cómo podemos usar las variables que este lenguaje nos permite.