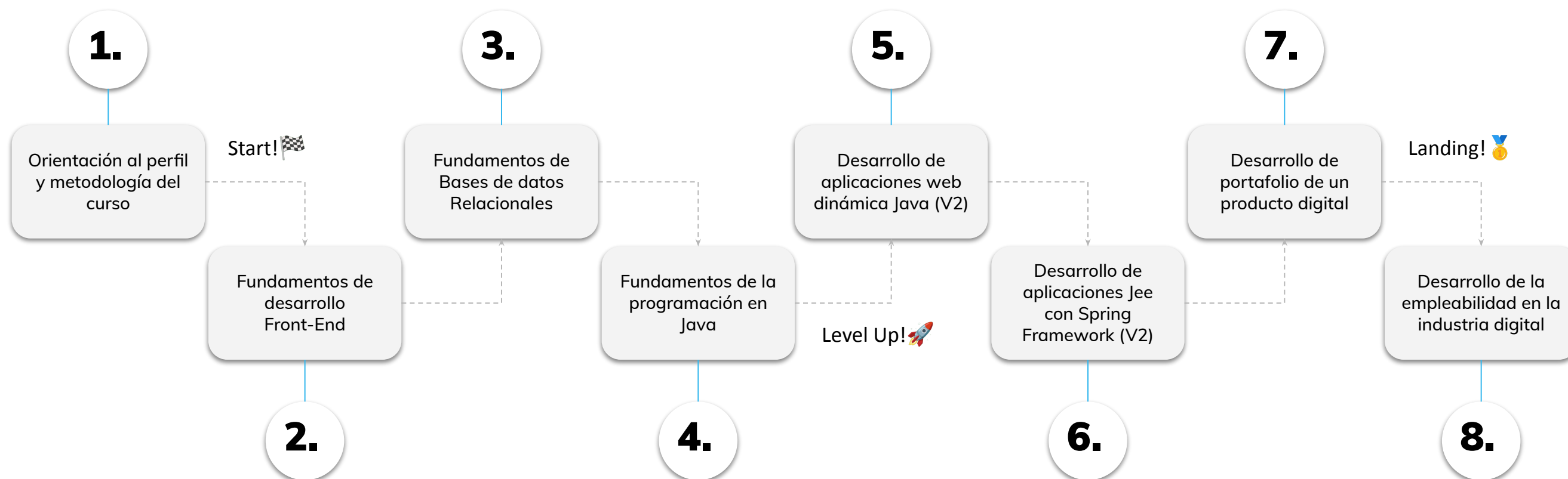


➤ Bases del lenguaje Javascript

Plan formativo: Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



REPASO CLASE ANTERIOR

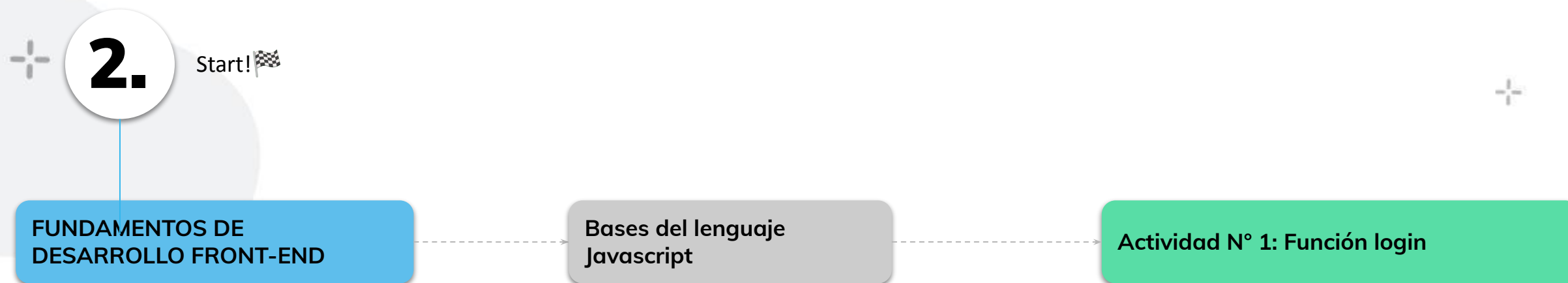
En la clase anterior trabajamos :

- ✓ Incorporar el componente form de bootstrap de manera más rápida para crear formularios
- ✓ Aprender a modificar y agregar menú a partir del componente navbar



LEARNING PATHWAY

¿Sobre qué temas trabajaremos?



En esta lección, aprenderás qué son las funciones en JavaScript y cómo se utilizan para agrupar y reutilizar bloques de código. Comprenderás cómo declarar y llamar a funciones en JavaScript, incluidos los parámetros y el valor de retorno y explorarás ejemplos prácticos de funciones y cómo pueden simplificar tu código.



OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



Explorar ejemplos prácticos de funciones y cómo pueden simplificar tu código.



Utilizar la consola de desarrolladores de tu navegador para ejecutar código JavaScript de forma interactiva.



Aprender a usar instrucciones de depuración como `console.log()` para mostrar información útil durante la ejecución del programa.






Rompehielo

Contexto:

El reto es escribir conjuntamente una historia de 150 palabras.

En la ventana del chat o archivo, cada participante en su turno podrá escribir únicamente 3 palabras, e inmediatamente la siguiente persona tendrá la oportunidad de escribir 3 nuevas palabras para continuar de manera lógica la historia. 

Al final del ejercicio, leer la historia resultante

¡Reta al equipo a hacerlo más rápido cada vez!



➤ Funciones



Funciones

Cuando se desarrolla una aplicación o sitio web, es muy habitual utilizar una y otra vez las mismas instrucciones.

En programación, una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta, que luego se puede reutilizar a lo largo de diferentes instancias del código.

Las funciones en JavaScript son bloques de código reutilizables que se pueden invocar para realizar una tarea específica. Proporcionan una forma de encapsular un conjunto de instrucciones y ejecutarlas en cualquier momento que se necesiten. Las funciones pueden aceptar argumentos (parámetros) y devolver un resultado.





Funciones



Declaración

La declaración de una función en JavaScript se realiza utilizando la palabra clave `function`, seguida del nombre de la función y un par de paréntesis `()`. Dentro de los paréntesis, puedes especificar los parámetros que la función puede aceptar, separados por comas. Luego, se abre un bloque de código con llaves `{}` donde se define el cuerpo de la función, es decir, las instrucciones que se ejecutarán cuando la función sea invocada.



```
function saludar() {  
  //Bloque de código  
  console.log("¡Hola estudiantes!");  
}
```



Funciones



Declaración

Una vez que declaramos la función, podemos usarla en cualquier otra parte del código todas las veces que queramos.

Para ejecutar una función sólo hay que escribir su nombre y finalizar la sentencia con (). A esto se lo conoce como llamado a la función. Donde escribamos el llamado, se interpretarán las instrucciones definidas en esa función.

```
saludar()
```





Funciones



Parámetros

Una función simple, puede no necesitar ningún dato para funcionar. Pero cuando empezamos a codificar funciones más complejas, nos encontramos con la necesidad de recibir cierta información. Cuando enviamos a la función uno o más valores para ser empleados en sus operaciones, estamos hablando de los parámetros de la función.

Los parámetros se envían a la función mediante variables y se colocan entre los paréntesis posteriores al nombre de la función.

```
//ejemplo de sintaxis
function conParametros(parametro1, parametro2) {
    console.log(parametro1 + " " + parametro2);
}

//sintaxis de una función con parametros
function saludar(nombre, edad) {
    console.log("¡Hola, " + nombre + "! Tenes " +
edad + " años.");
}
```



Funciones



Parámetros

Así, podemos armar funciones dinámicas que, siguiendo la lógica que queramos, pueden generar distintos resultados al recibir diferentes valores.

Cuando invocas una función que tiene parámetros, debes proporcionar los valores correspondientes a esos parámetros. El valor que toman estos parámetros se definen en el llamado. Cuando llamamos a la función, los valores que pasamos a la función entre paréntesis se asignan posicionalmente a los parámetros correspondientes, generando posibles resultados diferentes.

```
saludar("Juan", 30);  
// Muestra "¡Hola, Juan! Tienes 30 años."
```





Funciones



Ejemplo función Sumar y Mostrar

Se definen dos funciones, una para realizar la suma y otra para mostrar el resultado.

La función sumar toma dos números como argumentos y actualiza la variable resultado, mientras que la función mostrar muestra un mensaje en la consola.



```
//Declaración de variable para guardar el resultado
de la suma
let resultado = 0;
//Función que suma dos números y asigna a resultado
function sumar(primerNumero, segundoNumero) {
    resultado = primerNumero + segundoNumero
}
//Función que muestra resultado por consola
function mostrar(mensaje) {
    console.log(mensaje)
}
//Llamamos primero a sumar y luego a mostrar
sumar(6, 3);
mostrar(resultado);
```



Funciones



Resultado de una función

Las funciones pueden generar un valor de retorno usando la palabra `return`, obteniendo el valor cuando la función es llamada.

La función puede comportarse como una operación que genera valores (como en las operaciones matemáticas y lógicas previas).

En el espacio donde se llama a la función se genera un nuevo valor: este valor es el definido por el **return** de la misma.

```
function sumar(primerNumero, segundoNumero) {  
    return primerNumero + segundoNumero;  
}  
  
let resultado = sumar(5, 8);  
  
let resultado = sumar(5, 8);  
console.log(resultado) // => 13
```

› Scope

Scope

El scope o ámbito de una variable es la zona del programa en la cual se define, el contexto al que pertenece la misma dentro de un algoritmo, restringiendo su uso y alcance.

JavaScript define dos ámbitos para las variables:

- **Global**
- **Local.**





Scope



Ámbito global:

Las variables declaradas fuera de cualquier función tienen un ámbito global. Esto significa que están disponibles en todo el programa y pueden ser accedidas desde cualquier parte. Estas variables se conocen como variables globales y se definen utilizando las palabras clave var, let o const fuera de cualquier función.



```
let nombre = "Juan"; // Variable global

function saludar() {
  console.log("Hola, " + nombre);
  // Accediendo a la variable global dentro de la
  función
}

saludar(); // Imprime "Hola, Juan"
console.log(nombre); // Imprime "Juan"
```



Scope



Ámbito local:

Las variables declaradas dentro de una función tienen un ámbito local, lo que significa que solo están disponibles dentro de esa función. Estas variables se conocen como variables locales y se definen utilizando las palabras clave var, let o const dentro de una función.



```
function saludar() {  
  var mensaje = "Hola, bienvenido";  
  // Variable local  
  
  console.log(mensaje);  
}  
  
saludar(); // Imprime "Hola, bienvenido"  
console.log(mensaje);  
// Error: mensaje is not defined
```

✖ ▶ Uncaught ReferenceError: mensaje is not defined



Scope



Es importante tener en cuenta el alcance de las variables para evitar colisiones de nombres y asegurar un correcto funcionamiento del programa. Es una buena práctica limitar el alcance de las variables en la medida de lo posible utilizando ámbitos locales y evitando la declaración de variables globales innecesarias.



```
let nombre = "John Doe" // variable global

function saludar() {
  let nombre = "Juan Lopez" // variable local
  console.log(nombre)
}

//Accede a nombre global
console.log(nombre) // → "John Doe"

//Accede a nombre local
saludar() // → "Juan Lopez"
```



Scope



Entender que cada scope local es un espacio cerrado nos permite crear bloques de trabajo bien diferenciados e independientes, sin preocuparnos por repetir nombres de variables, sabiendo que se entienden como diferentes según donde las llamemos.



```
function sumar(num1, num2) {  
  let resultado = num1 + num2  
  return resultado  
}  
  
function restar(num1, num2) {  
  let resultado = num1 - num2  
  return resultado  
}
```

➤ Funciones anónimas



Funciones anónimas

Las funciones anónimas en JavaScript son funciones que no tienen un nombre definido. Se definen directamente como expresiones de función sin asignarles un nombre específico. Estas funciones se utilizan comúnmente en situaciones donde solo se requiere una función temporal o como argumentos para otras funciones.

Una función anónima es una función que se define sin nombre y se utiliza para ser pasada como parámetro o asignada a una variable. En el caso de asignarla a una variable, pueden llamar usando el identificador de la variable declarada.





Funciones anónimas

Las funciones anónimas son útiles cuando se necesitan funciones temporales o cuando se desea utilizar una función como argumento en otra función, como en el caso de funciones de devolución de llamada (callback functions) o funciones de orden superior (higher-order functions).

Es importante tener en cuenta que, aunque las funciones anónimas no tienen un nombre específico, se asignan a variables u otros elementos para poder ser utilizadas posteriormente.



```
//Generalmente, las funciones anónimas se asignan  
a variables declaradas como constantes  
const suma = function (a, b) { return a + b }  
const resta = function (a, b) { return a - b }  
console.log( suma(15,20) )  
console.log( resta(15,5) )
```



Funciones anónimas

Funciones Flechas =>

Las funciones flecha (arrow functions) son una forma más concisa y expresiva de definir funciones en JavaScript. Se introdujeron en ECMAScript 6 y proporcionan una sintaxis simplificada para crear funciones. Identificamos a las funciones flechas como funciones anónimas de sintaxis simplificada. No usan la palabra function pero usa => (flecha) entre los parámetros y el bloque.



```
const miFuncion = () => {  
  // Código de la función  
};
```





Funciones anónimas

Funciones Flechas =>

Se declara una variable miFuncion y se le asigna una función flecha como valor. El cuerpo de la función se encuentra dentro de los corchetes {}, donde se pueden escribir las operaciones que realizará la función.

Si la función flecha tiene un solo parámetro, se pueden omitir los paréntesis alrededor del parámetro.



```
const suma = (a, b) => { return a + b }  
//Si es una función de una sola línea con retorno  
podemos evitar escribir el cuerpo.  
const resta = (a, b) => a - b ;  
console.log( suma(15,20) )  
console.log( resta(20,5) )
```



Funciones anónimas



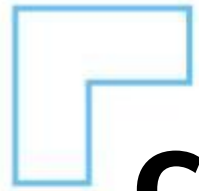
Callback

Los callbacks son funciones que se pasan como argumentos a otras funciones. Son una forma común de lograr la ejecución asíncrona y la programación basada en eventos en JavaScript.

La idea principal detrás de los callbacks es permitir que una función sea llamada dentro de otra función después de que se haya completado una tarea o evento específico. Esto permite una programación más flexible y dinámica, ya que puedes definir el comportamiento que se debe ejecutar después de ciertas operaciones.

```
function operacionAsincrona(callback) {  
    // Realizar una operación asíncrona  
    // Una vez que la operación se haya completado,  
    llamar al callback  
    callback();  
}  
  
function miCallback() {  
    console.log("La operación asíncrona se ha  
completado.");  
}  
  
operacionAsincrona(miCallback);
```

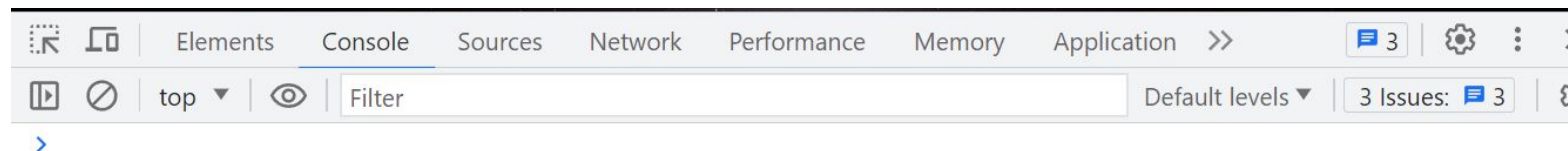
› Cómo ejecutar código Javascript en la consola



Cómo ejecutar código Javascript en la consola

Para ejecutar código JavaScript en la consola del navegador y realizar la depuración del código, puedes seguir estos pasos:

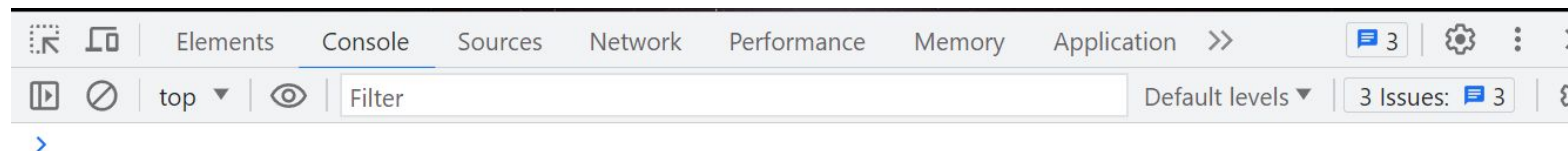
1. Abre cualquier página web en tu navegador.
2. Haz clic derecho en la página y selecciona "Inspeccionar" o presiona la tecla F12 para abrir las herramientas de desarrollo del navegador.
3. En la pestaña "Consola" de las herramientas de desarrollo, verás un prompt donde puedes ingresar y ejecutar código JavaScript. Acá puedes escribir cualquier instrucción o función en JavaScript y presionar "Enter" para ejecutarla.





Cómo ejecutar código Javascript en la consola

4. Observa los resultados o mensajes de error que se muestran en la consola después de ejecutar el código. puedes utilizar la consola para imprimir valores, realizar cálculos, interactuar con el DOM de la página y depurar problemas en tu código.
5. La consola del navegador es una herramienta muy útil para probar y depurar código JavaScript. Te permite ejecutar código en tiempo real y ver los resultados de tus instrucciones. Además, puedes utilizar métodos específicos de la consola, como `console.log()`, para imprimir valores y mensajes en la consola y facilitar el seguimiento de tu código durante la depuración.



› Depurando el código Javascript con la consola

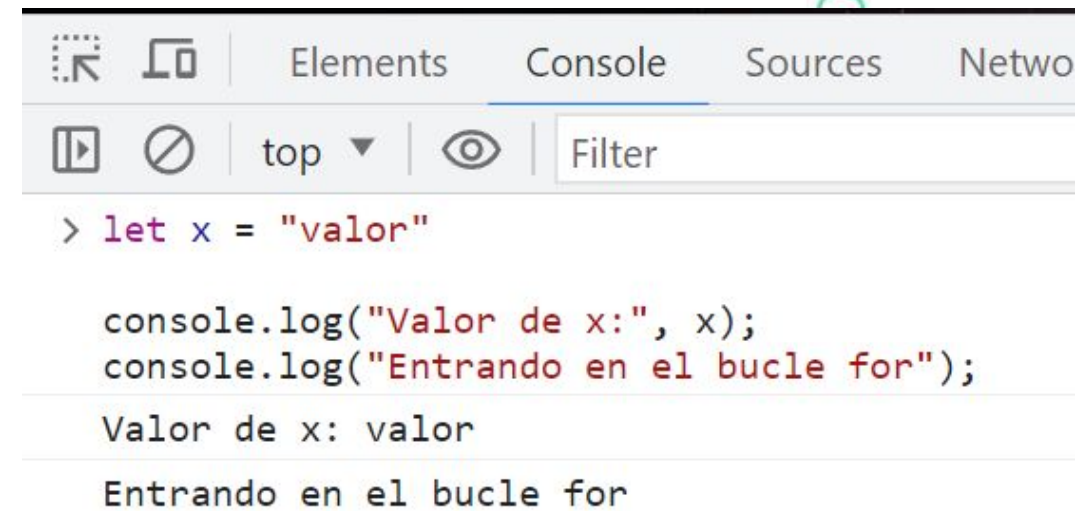


Depurando el código Javascript con la consola

La consola del navegador también es una herramienta muy útil para depurar el código JavaScript y encontrar errores o problemas en tu programa. puedes utilizar varias técnicas para depurar y examinar el código en la consola:

1. Mostrar mensajes de depuración:

Podes utilizar `console.log()` para imprimir mensajes de depuración en la consola. puedes agregar mensajes en diferentes partes de tu código para verificar el valor de variables, comprobar si ciertas condiciones se cumplen o simplemente seguir el flujo de ejecución del programa.



```
> let x = "valor"

console.log("Valor de x:", x);
console.log("Entrando en el bucle for");

Valor de x: valor

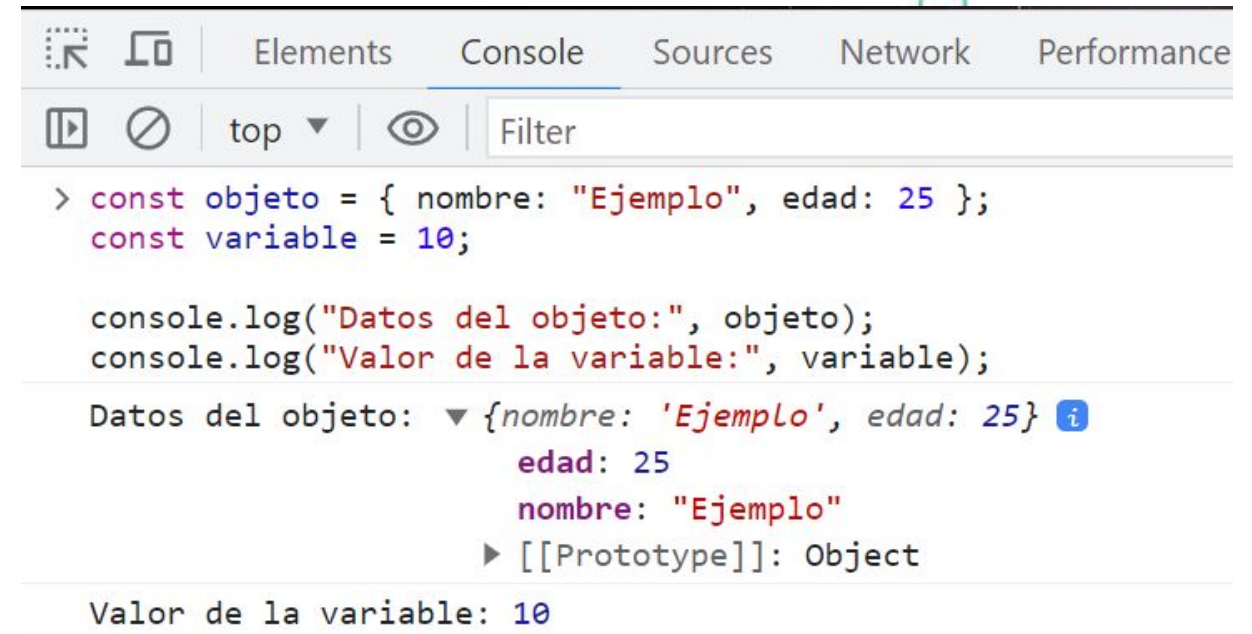
Entrando en el bucle for
```



Depurando el código Javascript con la consola

2. Inspeccionar objetos y variables:

Podes utilizar `console.log()` para imprimir el contenido de objetos y variables complejas. Esto te permite examinar su estructura y asegurarte de que contengan los datos esperados.



```
> const objeto = { nombre: "Ejemplo", edad: 25 };  
const variable = 10;  
  
console.log("Datos del objeto:", objeto);  
console.log("Valor de la variable:", variable);  
  
Datos del objeto: ▼ {nombre: 'Ejemplo', edad: 25} ⓘ  
    edad: 25  
    nombre: "Ejemplo"  
    ► [[Prototype]]: Object  
  
Valor de la variable: 10
```





Depurando el código Javascript con la consola



3. Establecer puntos de interrupción:

Podes utilizar puntos de interrupción en el código para detener la ejecución en un punto específico y examinar el estado de las variables en ese momento. En las herramientas de desarrollo del navegador, puedes hacer clic en el número de línea a la izquierda del código para establecer un punto de interrupción. Cuando se alcance el punto de interrupción, la ejecución se detendrá y podrás examinar el contexto y las variables en ese punto.



LIVE CODING

Ejemplo en vivo

Comandos de Depuración en JavaScript

Eres un instructor de programación y estás realizando una sesión en vivo para enseñar a los estudiantes cómo utilizar comandos de depuración en JavaScript para mejorar su habilidad para encontrar y solucionar problemas en el código.

Realizar un live coding para demostrar cómo utilizar comandos de depuración en JavaScript, incluyendo debugger, console.trace(), y console.error(), para detectar y solucionar problemas en el código.

 **Tiempo: 10 Minutos**



LIVE CODING

Ejemplo en vivo

Preparación

Crea un nuevo archivo JavaScript (puedes nombrarlo como "depuracion.js").

Define una función que contenga un error deliberado.

```
function dividir(a, b) {  
  return a / b;  
}  
let resultado = dividir(10, 0);
```

Usando console.trace()

Agrega una declaración console.trace() para imprimir la pila de llamadas en la consola.

Explícales cómo console.trace() muestra la secuencia de funciones que se han llamado hasta el momento y cómo puede ayudar a rastrear cómo se llegó a un punto específico en el código.

LIVE CODING

Ejemplo en vivo

Usando debugger

Agrega la instrucción debugger en un punto específico del código donde quieras detener la ejecución y comenzar a depurar.

Explícales cómo debugger pausa la ejecución y permite inspeccionar variables y el flujo de ejecución en ese punto.

Usando console.trace()

Agrega una declaración `console.trace()` para imprimir la pila de llamadas en la consola.

Explícales cómo `console.trace()` muestra la secuencia de funciones que se han llamado hasta el momento y cómo puede ayudar a rastrear cómo se llegó a un punto específico en el código.

Evaluación Integradora ✨



Trabajo Integrador del Módulo 💪

Iremos completándolo progresivamente clase a clase

[CLICK AQUÍ](#)

para ver la consigna completa



Ejercicio N° 1

Función Login



Función Login

Breve descripción

Contexto: 🙌

Avanzamos con nuestra función login del encuentro anterior.

Consigna: ✍️

Crear una función de inicio de sesión en JavaScript que solicita al usuario un nombre de usuario y una contraseña, verifica si son correctos y devuelve un mensaje de éxito o error.

Tiempo🕒: 10 minutos





Función Login

Breve descripción

Paso a paso: 

Crear la Función de Inicio de Sesión

- Define una función llamada `iniciarSesion` que tome dos parámetros: `usuario` y `contrasena`.
- Dentro de la función, compara el `usuario` y la `contrasena` con valores predeterminados para verificar si son correctos. Puedes usar valores de ejemplo como `"usuario123"` y `"claveSecreta"` para este propósito.
- Si el `usuario` y la `contrasena` coinciden con los valores predeterminados, la función debe devolver un mensaje de éxito como `"Inicio de sesión exitoso"`.
- Si no coinciden, la función debe devolver un mensaje de error como `"Credenciales incorrectas"`.





Función Login

Breve descripción

Paso a paso: 

Solicitar Credenciales al Usuario

- Fuera de la función iniciarSesion, utiliza prompt para solicitar al usuario que ingrese su nombre de usuario y contraseña.
- Almacena los valores ingresados por el usuario en variables.

Llamar a la Función de Inicio de Sesión

- Llama a la función iniciarSesion con los valores ingresados por el usuario como argumentos.
- Almacena el resultado de la función en una variable llamada resultado.

Mostrar el Resultado al Usuario

- Utiliza console.log() para mostrar el valor almacenado en la variable resultado. Esto mostrará un mensaje de éxito o error al usuario.



Función Login

```
function iniciarSesion(usuario, contrasena) {  
    // Valores predeterminados de usuario y contraseña (pueden ser reemplazados)  
    const usuarioValido = "usuario123";  
    const contrasenaValida = "claveSecreta";  
  
    // Verificar si las credenciales coinciden con los valores predeterminados  
    if (usuario === usuarioValido && contrasena === contrasenaValida) {  
        return "Inicio de sesión exitoso";  
    } else {  
        return "Credenciales incorrectas";  
    }  
}  
  
const usuarioIngresado = prompt("Ingresa tu nombre de usuario:");  
const contrasenaIngresada = prompt("Ingresa tu contraseña:");  
const resultado = iniciarSesion(usuarioIngresado, contrasenaIngresada);  
console.log(resultado);
```

○

¿Alguna consulta?

+



RESUMEN

¿Qué logramos en esta clase?

- ✓ Comprendes el concepto de funciones en JavaScript y cómo se utilizan para agrupar y reutilizar bloques de código.
- + ✓ Has aprendido cómo declarar funciones, incluyendo la especificación de parámetros y el uso de return para devolver valores. +
- ✓ Utilizar la función flecha y la función anónima y aprender la diferencia entre un scope local y un scope global.
- ✓ Estár familiarizado con el uso de comandos de depuración, como debugger, console.trace(), y console.error(), para detectar y solucionar problemas en el código JavaScript.
- ✓ Comprender cómo debugger pausa la ejecución en un punto específico, console.trace() muestra la pila de llamadas y console.error() permite imprimir mensajes de error en la consola.



#WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 🙌 🙌 🙌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
 - a. Material 1: Lección 5: Bases del lenguaje Javascript: 25-38
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

Momento: ✚

Time-out!

🕒 5 min.

