

Conectar los puntos

*"Cuando la acción se vuelve poco rentable, recopila información.
Cuando la información se vuelve poco rentable, duerme."*

Ursula K. Le Guin – La Mano Izquierda de la Oscuridad

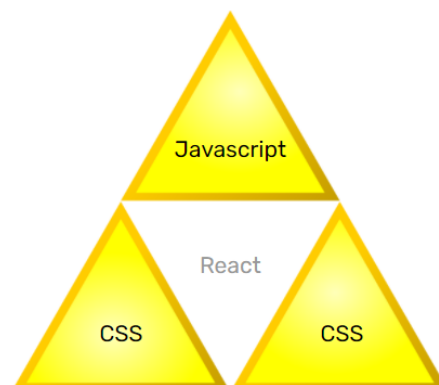
Misión: posible.

El destino final de este programa es enseñarte a construir el frontend de aplicaciones web dinámicas. Durante el primer sprint adquiriste habilidades que te permitieron construir páginas web estáticas, y ahora, en este segundo sprint, vamos por mas. En este sprint aprenderás a comunicarte con el cerebro de las computadoras, a través de un lenguaje de programación, y podrás acelerar todo lo que hiciste en el primer sprint, a través de React.

Para alcanzar esta misión necesitarás dominar 3 tecnologías:

- **HTML:** para crear la estructura de una página web
- **CSS:** para dar estilos
- **Javascript:** para agregar lógica y comportamiento

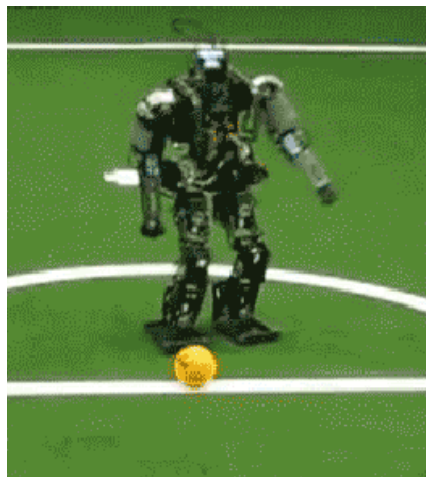
Durante la primera mitad del segundo sprint aprenderás a utilizar Javascript, y en la segunda, verás una introducción a React, teniendo la oportunidad de mezclar estas tecnologías (HTML, CSS y Javascript) para poder llevar tus páginas estáticas, a súper aplicaciones web.



Para cumplir la misión de desarrollar aplicaciones web dinámicas, aparte de crear la estructura y los estilos con HTML y CSS, deberás aprender a programar un sitio web con Javascript. Para empezar este recorrido te queremos contar sobre **tres cosas**: la programación, los lenguajes de programación y Javascript.

¿Qué es programar?

En el desarrollo de software, **programar es el acto de escribir instrucciones que puedan ser entendidas por una computadora**, las cuales deberán estar escritas en un lenguaje de programación. Programar nos da la habilidad de poder comunicarnos con el cerebro de una computadora para que haga tareas por nosotros. Contrario a la creencia popular, las computadoras son muy rápidas pero muy, muy tontas. Programar es instruir a la computadora lo que debe y no debe hacer ante ciertas situaciones.



Cuando decimos “**escribir instrucciones para una computadora**” podemos estar refiriéndonos a varias cosas: un robot, un microondas o un reloj. En el contexto que nos interesa, nosotros estaremos escribiendo instrucciones para el navegador, quien es el responsable de procesar el código de nuestras aplicaciones web.

¿Qué es un lenguaje de programación?

Un lenguaje natural, como el español o inglés, es un conjunto de palabras y símbolos que le permiten a las personas comunicarse, transmitir ideas, información o datos. De la misma forma, un lenguaje de programación es un conjunto de palabras y símbolos que le permite a los programadores comunicarse con una computadora. Durante el primer sprint pudiste dar



instrucciones al navegador, a través de HTML y CSS, sobre cómo dibujar interfaces gráficas.

Con Javascript, podrás extender los poderes de estas instrucciones, agregando comportamiento y lógica a los componentes gráficos creados con HTML y CSS. Este lenguaje te permitirá agregar reacciones y acciones a tus sitios web, así como manipular los elementos existentes, crearlos o borrarlos.

Los lenguajes de programación son creados por personas como tu y yo, bueno, por **grupos** de personas, ¿puedes creer que actualmente existen alrededor de [700 lenguajes de programación](#) registrados? No le tengas miedo a ese número. ya que de esos 700, quizás 50 aún estén en uso, y de esos 50, aproximadamente 10 son los más demandados, dependiendo del área.

Un lenguaje de programación solventa distintos tipos de problemas, y estos se van popularizando durante el tiempo por comunidades de developers que demuestran que el uso de un lenguaje de programación les facilita ciertas tareas en sus desarrollo.

Por ejemplo, Python, es un lenguaje de programación muy popular dentro de la comunidad de la ciencia de datos, mientras que COBOL, es un lenguaje muy usado dentro de las aplicaciones que utilizan los bancos. En el caso del front-end, [Javascript es la opción más popular en estos días](#). En la primera parte de este sprint deberás concentrarte en conocer Javascript como lenguaje mientras exploras los principios de la programación.

¿Por qué Javascript?

Javascript fue lanzado en 1995, proporcionando una forma de crear páginas web en el navegador Netscape. Este lenguaje ha sido adoptado por los navegadores más populares a la fecha ya que facilita el desarrollo de aplicaciones web.

¡Atención! Javascript no tiene nada que ver con el lenguaje de programación Java. La similitud entre los nombres fue una decisión de marketing, ya que cuando Javascript estaba siendo introducido en el mundo, el lenguaje Java estaba en boca de muchas personas y empresas, y se pensó que nombrar los lenguajes de forma similar era una buena forma de aprovechar la ola de popularidad de Java.

Si te pones a pensar, es algo como la Pepsi-Cola y la Coca-Cola. Se parecen, se ven igual pero claramente son dos cosas distintas. Otra razón sobre el por que hacemos énfasis en Javascript es que también es un lenguaje de fácil adopción



para principiantes y muy flexible a la hora de codear. Poco a poco irás descubriendo estas razones.

Habiendo dilucidado estas tres preguntas, es hora de meter los pies en el océano que es Javascript. Empecemos con tres conceptos cruciales para iniciarnos en esto: [variables](#), [valores](#) y [operadores](#).

Variables / Valores / Operadores

Cuando pones en funcionamiento un programa, el navegador ejecuta las líneas de código que hayas escrito en tu editor (de arriba hacia abajo) y las ejecuta. Las instrucciones que va pasando, las olvida ¿Qué pasa si quieres volver a usarlas? ¿Cómo puedes hacer para que la computadora recuerde esa información?



En programación, para guardar información usamos [variables](#). El concepto de variable es el primero de muchos que aprenderás sobre lenguajes de programación, específicamente de Javascript.

Una variable es **un espacio reservado de memoria para almacenar un valor** que corresponde a un tipo de [valor](#). Toda aplicación necesita variables para poder recordar la información.

Las aplicaciones web y las personas tienen muchas cosas en común. Tu nombre, tu edad, tu color de ojos, tu número de identificación gubernamental son variables que te han sido asignadas en diferentes momentos desde tu creación. Personas usuarias registradas, listas de hoteles disponibles, cantidad de seguidores, likes por publicación y cantidad de reproducciones son algunos ejemplos de variables de una aplicación web.

Existirán variables que podrán ser cambiadas a lo largo del tiempo en una aplicación web. Una persona puede cambiar su nombre siguiendo un procedimiento, y la edad de alguien innegablemente irá cambiando mientras pasan los años. Esas variables son mutables, es decir, que pueden cambiar,



mientras que tu identificación gubernamental o tu color de piel, son cosas que difícilmente cambien, y en algunos casos, no podrán cambiar nunca, es decir que son inmutables. ¿Supiste de alguien que pudo cambiar su número de dni alguna vez?

Javascript ofrece tres tipos de variables para estos casos:

- `var`
- `let`
- `const`

Estas tres palabras son reservadas del lenguaje, lo cual significa que solo podrás usarlas para construir variables. Y hablando de construir variables, he aquí la sintaxis para hacer eso:

tipo de variable	nombre de la variable	valor
<code>let</code>	<code>miNombre</code>	<code>= "Javier";</code>
<code>const</code>	<code>DNI</code>	<code>= 123456789;</code>
<code>var</code>	<code>apellido</code>	<code>= "Rivera";</code>

Observa las convenciones que hemos usado para definir los nombres de las variables: todas en minúscula, utilizando [camel case](#) para variables que tengan más de una palabra, y mayúsculas para las `const`. Además, para expresar texto utilizamos comillas "dobles", 'simples' o `inclinadas` (¡todas son válidas!), mientras que los números no las necesitan.

Los nombres que les asignamos a las variables son una elección muy importante, ya que aportan a la buena legibilidad y organización del código. Es importante que estos nombres sean lo más descriptivos posible: si son demasiado cortos o, al contrario, demasiado largos, pueden dificultar la legibilidad.

🚨 ¡Ten cuidado! Hay algunas reglas que seguir para asignar los nombres de las variables:

1. **Caracteres.** Los únicos caracteres que pueden tener son letras, números, el signo pesos (\$) y el guión bajo (_). Evita palabras reservadas y espacios.



De lo contrario, ¡no va a funcionar!

2. **Case-sensitive.** Las variables son susceptibles de cambiar su sentido si alguna de sus letras está en mayúscula o minúscula. Por ejemplo, la variable “miNombre” no será la misma que “minombre”.
3. **Más de una palabra.** Usamos los nombres para describir la información que queremos guardar. Por lo cual, necesitamos encontrar una forma precisa y evidente de escribirlas para que su legibilidad sea buena.

🔥 Además de las reglas mencionadas, ¡los developers intentamos replicar otras **buenas prácticas**: separamos las palabras con guiones bajos (_); utilizamos la de técnica de [lowerCamelCase](#), que consiste en iniciar siempre en minúscula y luego comenzar cada palabra con una mayúscula.

Aquí van algunos ejemplos:

<https://gist.github.com/havebeeair/0e339877b2850150f1dc2625bb00da0b>

Es importante que, más allá de la forma en la que prefieras nombrar variables, al elegir una seas consistente en el programa ¡y utilices siempre la misma!

Para mostrarte un poco la sintaxis de definición de variables, hemos preparado un video, en el que también te contamos cómo preparar tu entorno de desarrollo con Javascript, en el editor de texto VS Code.

[Preparando el entorno y tipos de variable](#)

¡Mini challenge! 🔥

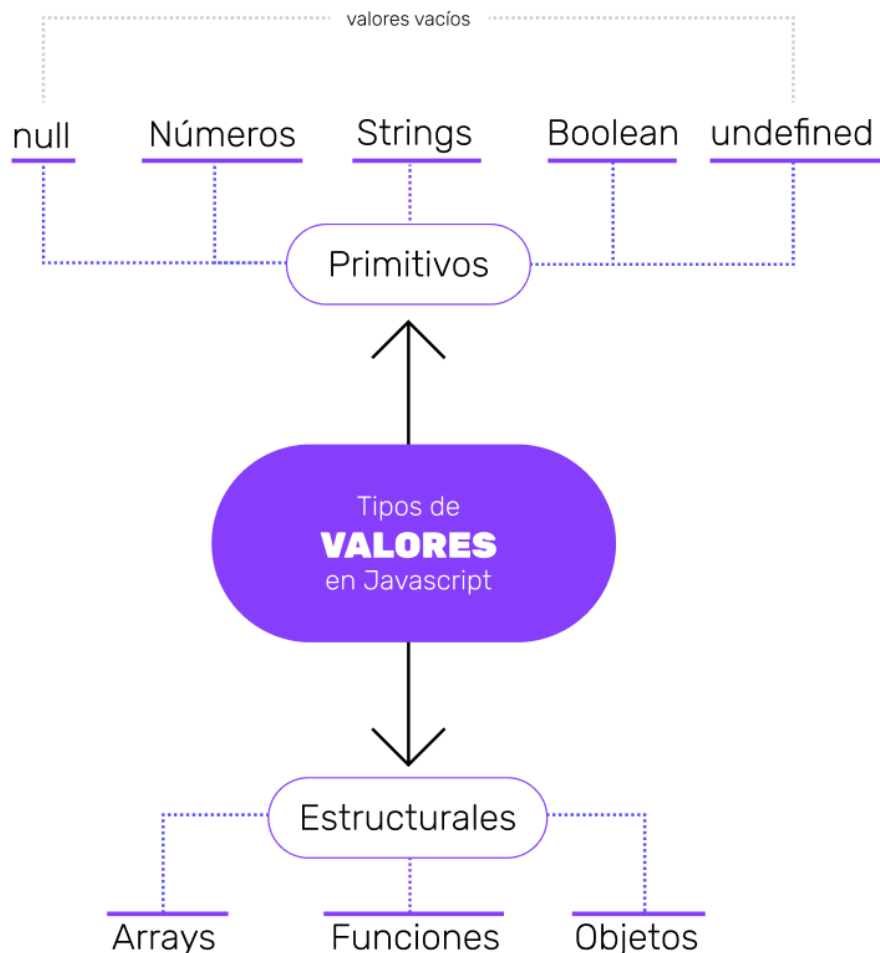
Prepara tu entorno de trabajo con VS Code, creando un proyecto web, y asocia un archivo Javascript, tal como te mostramos en el video. Dentro de este archivo, crea variables que definan cosas de ti, teniendo en cuenta la diferencia entre los tipos [let](#) y [const](#).

Variables / Valores / Operadores

El segundo concepto que te queremos contar en esta introducción son los valores. En los ejemplos anteriores te enseñamos a crear variables usando los tipos `let` y `const`, las cuales iban a almacenar algo, específicamente un valor.

El valor es lo que una variable guarda, es decir, su significado. Los valores se dividen por “tipos”, y cada uno de estos provee distintos tipos de datos, para distintos casos de uso. Por ejemplo, si quieres guardar la edad en una variable, deberás utilizar un dato numérico, mientras que un nombre, lo deberás guardar en un dato “string”, o cadena de caracteres.

Existen 8 tipos de valores en Javascript, que iremos estudiando poco a poco a lo largo de este segundo sprint. Estos 8 valores se subdividen en dos categorías: primitivos y estructurales.



Los valores primitivos son los que nos permiten guardar un solo tipo de valor. Por ejemplo, un número, una cadena de caracteres, o un valor lógico (booleano). Los valores estructurales nos permiten guardar varios tipos de valores. La función es

un tipo de valor estructural, muy especial, que estudiaremos más adelante. Por ahora te podemos adelantar que la función cumple un rol protagónico en Javascript y React, por lo que le dedicaremos unas cuantas meetingsss.

Por ahora, vamos a enfocarnos en estos tres valores primitivos:

- **Números.** Son la unidad conceptual con la que representamos cantidades. Este tipo de datos permite hacer operaciones matemáticas con los valores numéricos, tanto decimales como enteros.
- **Strings.** Representan las cadenas de texto. Puede ser una letra "L", una palabra "Perro", una frase o incluso un párrafo entero. No son solo letras o números, sino también caracteres especiales (como "@" o espacios en una frase). Siempre están delimitados por comillas 'simples', "dobles", o `inclinadas simples`. ¡TODO ENTRE COMILLAS ES UN STRING!
- **Booleanos.** Un booleano es el tipo de dato lógico que representa valores de lógica binaria, representados en sólo dos valores posibles: verdadero (TRUE) y falso (FALSE). Estos, nos van a permitir hacernos preguntas para saber la condición de veracidad o falsedad sobre algo. Por eso, decimos que los booleanos permiten determinar condiciones y tomar decisiones en base a si esas condiciones se cumplen o no. **¡Son muy útiles para construir programas!**

Javascript es un lenguaje de tipado débil, lo cual significa que no tenemos que declarar el tipo de variable (numero, string o booleano), contrario a lenguajes de programación como de tipado fuerte, como Java.

Variables / Valores / Operadores

¡Solamente con las variables no alcanza! Los operadores nos permiten realizar cálculos matemáticos, evaluar relaciones, conectar o manipular los datos. Dependiendo del tipo de dato que se vaya a manipular Javascript nos proveerá diferentes operadores. Por ejemplo:

- Puedo sumar $4 + 5$ y como resultado obtener `9`. En este caso nuestro operador será el signo +

Pero este mismo operador funcionará diferente si el dato que estoy manipulando es un string. Por ejemplo:

- $"4" + 5$ será "45". En este, al tener un valor tipo número y otro tipo string el operador `+` concatena los valores y devuelve un resultado tipo string.

Hay tres tipos de operadores: los aritméticos, los de comparación y los lógicos.



1. Los operadores aritméticos

Toman valores numéricos (ya sean literales o variables) como sus operandos y retornan un valor numérico único. **Siempre toman dos valores y producen uno nuevo como resultado.**

Los operadores aritméticos estándar son adición o suma (+), sustracción o resta (-), multiplicación (*), y división (/)



[Números y operadores aritméticos](#)

2. Los operadores de comparación.

Cotejan dos valores y producen un resultado booleano. El resultado puede ser **false** o **true**.

[Operadores de comparación](#)

Estrictamente igual ===

Evalúa que ambos datos sean iguales tanto en su valor como en su tipo. Se escribe utilizando tres veces el signo de igualdad. Por ejemplo:

"4" === 4 el operador estrictamente evaluará esta expresión como **false** (falsa) ya que el carácter "4" es distinto al número 4.

Estrictamente distinto !==

Evalúa que ambos valores y tipo de dato sean diferentes. Por ejemplo:

"1" !== 1 el operador evaluará esta expresión como verdadera, o true, ya que estos dos valores son estrictamente distintos.

"1" !== "1" el operador devuelve **false** ya que ambos valores son estrictamente iguales.

Mayor que >

Evalúa que el primer valor sea mayor al segundo. Por ejemplo:

7 > 3 el operador devuelve **true**, ya que 7 es mayor que 3

7 > 7 el operador mayor que devuelve **false**, por que 7 no es mayor que 7

3 > 7 el operador mayor que devuelve **false**

Mayor o igual que >=

Evalúa que el primer valor sea mayor o igual al segundo. Por ejemplo:

7 >= 3 el operador mayor que devuelve **true**

7 >= 7 el operador mayor que devuelve **true**

3 >= 7 el operador mayor que devuelve **false**

Menor que <

Evalúa que el primer valor sea menor al segundo. Por ejemplo:

4 < 5 el operador mayor que devuelve **true**

4 < 4 el operador mayor que devuelve **false**

4 < 2 el operador mayor que devuelve **false**

Menor o igual que `<=`

Evalúa que el primer valor sea menor o igual al segundo. Por ejemplo:

`4 <= 5` el operador mayor que devuelve **true**

`4 <= 4` el operador mayor que devuelve **true**

`3 <= 2` el operador mayor que devuelve **false**

3. Operadores lógicos.

Estos son operadores que nos permiten trabajar con expresiones booleanas. Una expresión booleana es aquella que arroja un valor booleano, como los ejemplos anteriores. Dale un vistazo al siguiente video para entender cómo trabajan estos operadores.

[Operadores lógicos](#)

En resumen, hay tres operadores lógicos:

- **AND:** evalúa que ambas expresiones booleanas sean ciertas
- **OR :** evalúa que al menos una expresión booleana
- **NOT:** invierte la expresión booleana

El siguiente video pone en práctica todos los conceptos descritos previamente. ¡Asegúrate de verlo! Para este video, hemos utilizado un sandbox de tipo “vainilla Javascript” con una configuración similar al entorno con VS Code. El código fuente podrás encontrarlo en la descripción del video.

[Operaciones en Javascript](#)



Antes de despedirnos

Programar puede ser una actividad tediosa y frustrante, y habrá momentos durante el sprint en el que podrás sentirte abrumadx ante la cantidad de información nueva que recibirás. ¡Apela a tu tolerancia y anímate a desafiarte para ir más allá de los límites! Recuerda siempre preguntar a tu squad leades y compañerxs de clase. **Aprender es una actividad que se hace mejor en grupo.**



Ante la frustración, no saques conclusiones sobre tí o tu talento, porque es perfectamente normal que esto suceda. Comunícate con tu equipo durante estos momentos, y toma una pausa, un café, o una siesta. ¡Programar no es solo estar sentadx frente a una computadora! También es una actividad social, es mirar cosas que antes no estaban. En algunos casos, programar puede ser el arte de conectar los puntos que tu mismx has dibujado sobre la pared.

Familiarizarse con la sintaxis de código de cualquier lenguaje de programación no ocurre de la noche a la mañana, lleva días, semanas y hasta meses, y para eso, necesitaremos escribir mucho código, e inclusive, leerlo. No se espera en lo absoluto que te memorices la sintaxis de Javascript. Para ser un buen developer no se necesita memorizar nada. Un buen developer sabe cómo buscar lo que necesita.


¡Prepárate para la próxima meeting!


Challenge



#1. Prepara un entorno de trabajo para tus prácticas como te mostramos en los videos de la toolbox, bien sea en Code Sandbox o en VS Code.



 #2. ¡Defínete en variables! Crea un archivo Javascript que contenga datos sobre ti, utilizando los 3 tipos primitivos que te presentamos en esta toolbox: números, strings y booleanos. Adjuntalo a tu proyecto web, el cual preparaste en el punto 1 del challenge. No es necesario mostrar nada por consola.

 #3. Accede a [este gist](#) y copia su contenido en tu entorno de trabajo y sigue las instrucciones. Puedes crear un archivo nuevo o copiar el contenido del gist en el archivo del challenge #2. Si creas un archivo nuevo, deberás agregar una nueva etiqueta `<script>` en el html, debajo del archivo que contiene los datos sobre ti.