Fundamentos JavaScript

DEV.F:: DESARROLLAMOS(PERSONAS);

¿Qué es JavaScript?

- **No es Java 🤣.**
- Lenguaje de programación.
- Scripting o **interpretado**.
- Multiparadigma (estructurada y POO).
- Basado en **prototipos**.
- **Débilmente tipado.**
- Se basa en el estándar **ECMAScript**.
- Para el front end funciona en el navegador gracias a la Web API.
- Para el back end funciona con Node.

Documentación Oficial

- Eloquent JS.
- Mdn.
- W3Schools.
- Devdocs.io.
- Curso de principiantes JS.
- Cómo documentar en JS.

DEV.F

Comparativa con otros lenguajes

Java

Server-side development,
compiled language,
static typed language, class-based,
multi-threaded

Javascript

Client-side scripts,
interpreted scripted language,
dynamic typed language,
prototype-based, single-threaded

Funcionamiento

DEV.F::
DESARROLLAMOS(PERSONAS);

¿Cómo ejecutar JS?

- Dentro del HTML.
- Con un archivo JS importado mediante la tag script.

```
<script>
    console.log('hola desde el HTML')
</script>
<script src="js/index.js"></script>
```

Variables

DEV.F::
DESARROLLAMOS(PERSONAS);

Variables

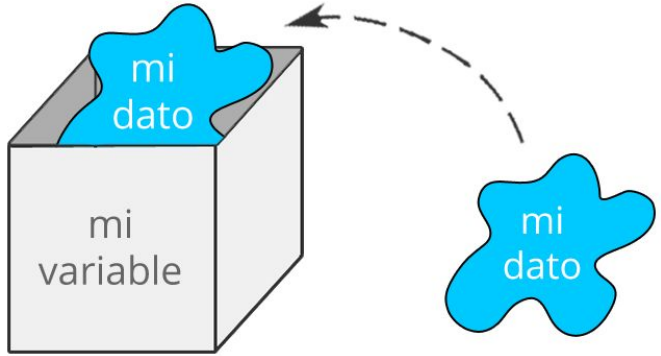
- Una variable es un nombre o **identificador único** que **puede tomar distintos valores**.
- Es un **espacio** que le designamos a un **valor** para **almacenarlo**.

```
// tipoDato nombreVariable = valor;
int applePrice = 2;
```

```
// alcance nombreVariable = valor;
var name = 'jose';
let app = 'montoya';
const apm = 'guzman';
```

¿Para qué sirve una variable?

1. Guardar datos y estados.
2. Representar valores dentro de una expresión matemática.
3. Asignar valores de una variable a otra.
4. Mostrar valores por pantalla.



DEV.F

Nombres de variables

- Las variables deben comenzar con una letra minúscula.
- Los nombres deben estar formados por letras (a .. z, A .. Z) y dígitos (0 .. 9) y guión bajo _.
- Evitar el uso de caracteres especiales (<*^¨=?) o internacionales ($£àêß).
- Se deben evitar nombres de una sola letra. **Debemos usar nombres descriptivos que ayuden a entender el código.**
- Nunca debemos usar nombres que se puedan confundir con **palabras reservadas** del lenguaje como break, const, else, new, entre otras.

Estilos de nombrado

- **camelCase:** Comúnmente usado al escribir código.
- **PascalCase:** Nombrar archivos de tipo clase.
- **kebab-case:** Comúnmente usado al nombrar carpetas y archivos.
- **snake_case.**
- **dot.case:** Comúnmente usado al nombrar carpetas y archivos.

Tipado

DEV.F:: DESARROLLAMOS(PERSONAS);

Tipado

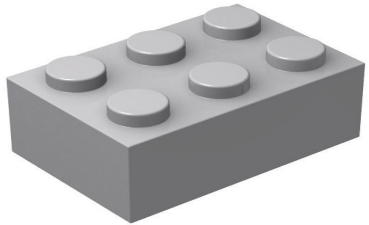
- El **tipado débil**: Los lenguajes de programación permiten que las variables pueden ser cambiadas de tipo.
- El **tipado fuerte:** Los lenguajes de programación permiten que las variables solo sean de un solo tipo de dato y no pueden cambiarlo a menos que se haga un parseo.

Tipos de datos

DEV.F:
DESARROLLAMOS(PERSONAS);

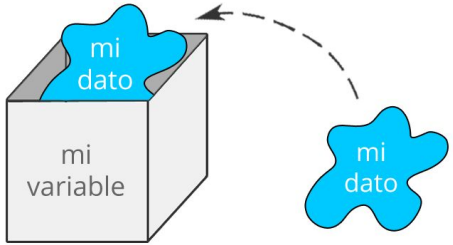
Tipos de datos

- Variables **primitivas** / tipos de datos **simples**.
  - Number (int o double).
  - String.
  - Char.
  - Boolean.



- Variables **objeto** o tipo de dato **complejo**.
  - Objetos.
  - Arrays.

Simple vs Complejo




```
// alcance nombreVariable = valor;
var name = 'jose';
let app = 'montoya';
const apm = 'guzman';
```


```
var arr = [1, 2, 3, 4, 5];
const verduras = ['tomate', 'calabaza', 'espinaca'];
var alumno = new Alumno();
```

Arreglos y objetos

- Un **objeto** es la **agrupación de varias variables de un mismo tipo dentro de una entidad**.
- Un **arreglo es como un vagón de tren** que **guarda muchos valores** uno por posición.

```
// objects
var video = {
    name: "peaches", // video.name
    author: "jack black", // video.author
}

// arrays
var comments = [
    "el video es buenardo", // comments[0]
    "a mi no me gusto", // comments[1]
    "todos somos browser", // comments[2]
];
```

Otros tipos

- **undefined:** Cuando una variable no está ni siquiera declarada.
- **null:** Es un tipo objeto, valor vacío por defecto genérico de JS.
- **NaN:** Es un tipo de dato numérico que no es un número válido.

Comprobar el tipo de dato de una variable

- función typeof.

Inputs y outputs

DEV.F::
DESARROLLAMOS(PERSONAS);

Entrada y salida

ENTRADA

- prompt
- confirm

SALIDA

- console.log
- alert
- document.write

Conceptos Clave

DEV.F::
DESARROLLAMOS(PERSONAS);

Conceptos Clave

- **Definición:** Es la acción de definir el alcance, tipo de dato y nombre de una variable.
- **Asignación:** Es el proceso de darle valor a una variable.
- **Reasignación:** Consiste en cambiarle el valor a una variable.
- **Variable indefinida:** Es cuando tratamos de acceder a una variable que nunca fue definida.
- **Sintaxis:** Las reglas de escritura de ese lenguaje de programación.
- **Palabras reservadas:** Son palabras clave de un lenguaje de programación y solo pueden ser usadas para el fin que el lenguaje definió.
- **Comentarios:** Es texto dentro del código que no se ejecuta, sirve para explicar mejor el código.

Conceptos Clave

- **Parsear:** Convertir un tipo de dato en otro, tipo de dato.
- **Concatenar:** Unir variables y texto.
- **Invocar una función:** Es mandar llamar a una función.
- **Ámbito de vida:** Es el espacio donde una variable o declaración funciona.
- **Sentencias:** Son un conjunto de líneas de código.

Operadores

DEV.F::
DESARROLLAMOS(PERSONAS);

Operadores

- Operadores de asignación.
  - Asignación
- Operadores aritméticos.
  - Suma, resta, multiplicación, división, potencia.
  - Residuo.
  - Incremento y decremento.
- Operadores de comparación.
  - Igual, diferente, Mayor que, Menor que, Exactamente igual, exactamente diferente.

Operadores

- Operadores lógicos.
  - And (&&), or (||) y not (!).
  - Tablas de verdad y procedencia de operadores.
  - Docs.
  - Operador doble negación: !!

Truthy y falsy

En JavaScript cuando creamos cualquier tipo de variable se le asigna una herencia de Boolean, si la **variable existe y está definida con** un **valor diferente** al **de la** siguiente **lista**, es **truthy**. Si **no existe y/o no está definido y/o es un valor de la lista** es **falsy**.

Value	Type
0	Number
NaN (not a number)	Number
'' (empty string)	String
false	Boolean
null	Object
undefined	Undefined

Sentencias de control

DEV.F::
DESARROLLAMOS(PERSONAS);

Sentencias de control

Las estructuras de control permiten **determinar el orden de ejecución de las** sentencias o **instrucciones de un programa**.

En programación existen tres tipos de estructuras de control:

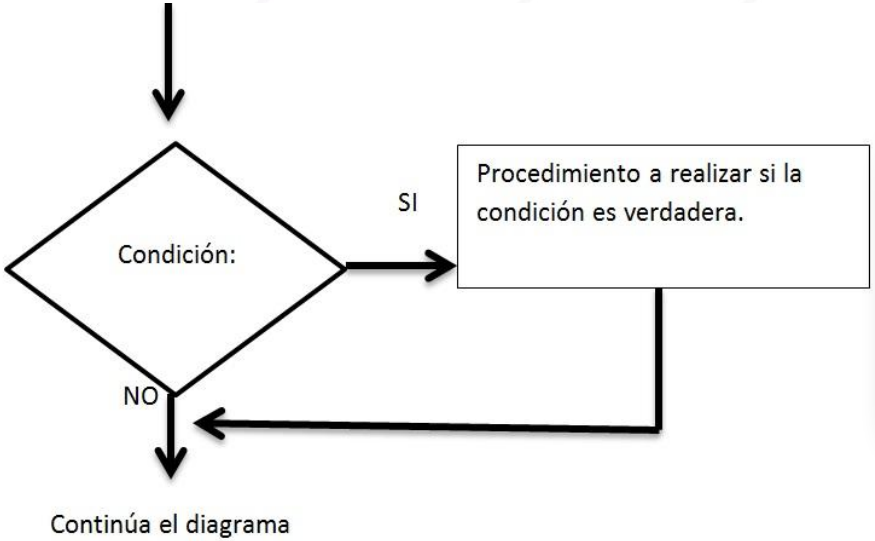
- La secuencia.
- Las de decisión.
- Las repetitivas.

DEV.F

Condicionales

Son reglas de código que escribimos para decidir que se ejecute cierto código sólo si se satisfacen esas reglas y si no se cumplen suceda algo distinto.

Permiten controlar el flujo de ejecución.




```
if(edad > 18) {
    console.log('ere mayor de edad');
} else {
    console.log('ere mayor de edad');
}
```

If, if else y else

Son reglas de código que escribimos para decidir que se ejecute cierto código sólo si se satisfacen esas reglas y si no se cumplen suceda algo distinto.

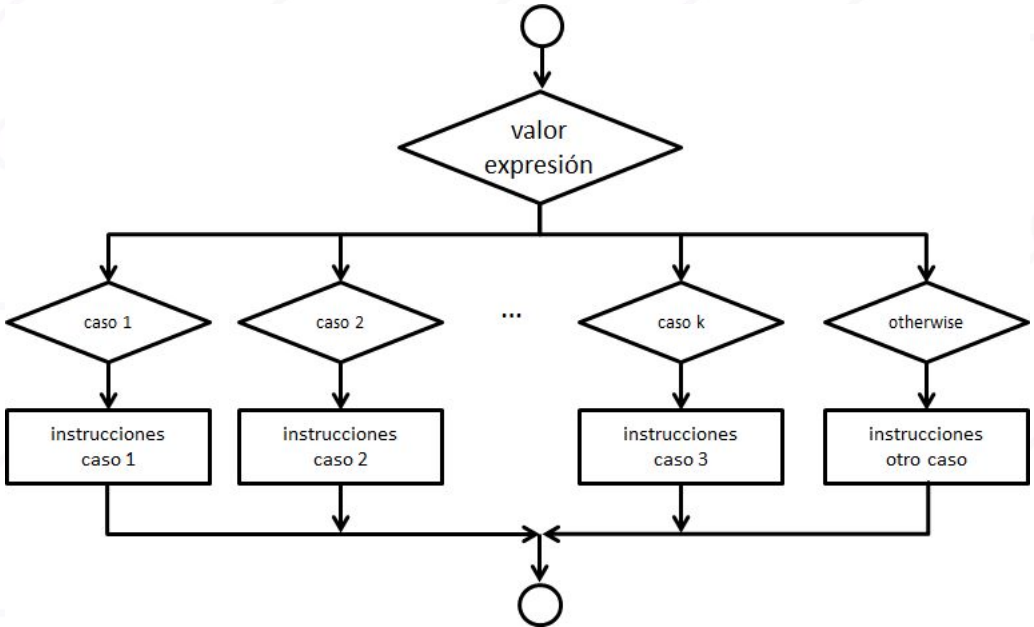
Permiten controlar el flujo de ejecución.

```
if (video == 'CHAMPIONS') {
    console.log('ver video champions');
} else if (video == 'MIRANDA') {
    console.log('ver video miranda');
} else {
   console.log('video no existente ... ');
}
```

Switch

Son reglas de código que escribimos para decidir que se ejecute cierto código sólo si se satisfacen esas reglas y si no se cumplen suceda algo distinto.

Permiten controlar el flujo de ejecución.



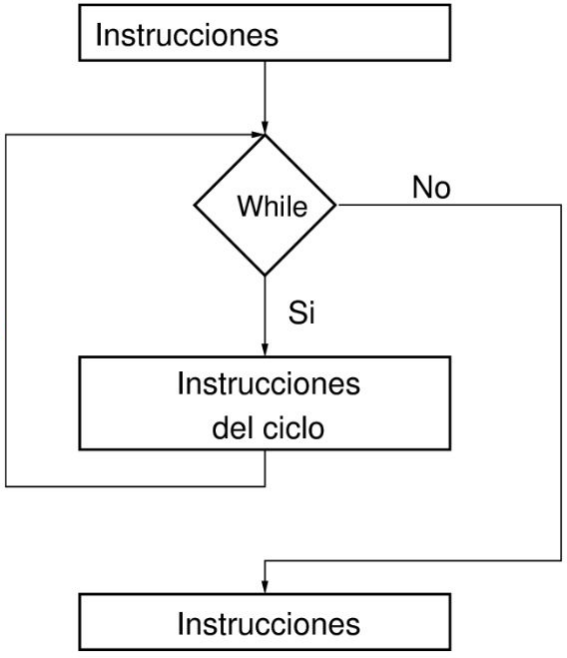
Ciclos

DEV.F::
DESARROLLAMOS(PERSONAS);

Ciclos

Son sentencias que **se ejecutarán de forma repetitiva mientras una cierta condición se siga cumpliendo**. Se componen de:

- Un valor de inicio.
- Una razón de cambio (incremento o decremento).
- Una condición de fin.



while

- **Mientras la condición sea verdadera**, el código dentro del bloque se estará repitiendo.

```
// inicio
let edad = 0;
while(edad <= 10) { // condicion
    console.log('este año tengo', edad, 'años');
    edad++; // valor de cambio
}
```

```
// inicio
let aguinaldo = 100;
while(aguinaldo > 0) { // condicion
    console.log('me quedan', aguinaldo, 'pesitos');
    aguinaldo = aguinaldo - 10; // valor de cambio
}
```

do while

- Se ejecuta **el bloque de código** que está dentro de las llaves **UNA VEZ**, **después se valida la condición while**, si se cumple se seguirá repitiendo el ciclo hasta que se deje de cumplir.

```
let start = 1; // inicio
do {
    // game is fire!!
    start = prompt('Quieres continuar? Escribe 0 para salir'); // valor de cambio
} while (start !== 0); // condicion
```

for

- Es un ciclo que conjunta en una sola línea el valor de inicio, condición y valor de cambio. Por lo que **las líneas de código dentro del for se ejecutan desde el INICIO hasta que ya no se cumpla la condición**.

```
// inicio ; condicion ; valor de cambio
for(var i=0; i<=10; i++) {
    console.log('Numero: ', i);
}
```

```
// inicio ; condicion ; valor de cambio
for(var i=0; i>=10; i--) {
    console.log('Numero: ', i);
}
```

Funciones (métodos)

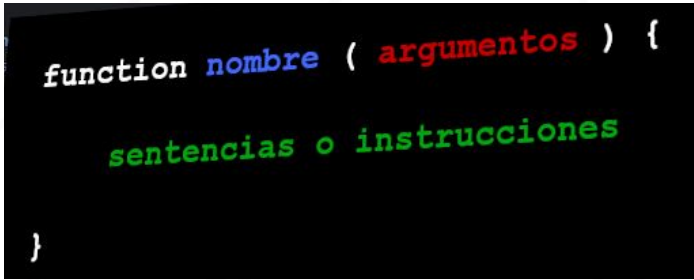
DEV.F:
DESARROLLAMOS(PERSONAS);

Funciones

Un **conjunto de instrucciones** que **realiza una tarea** o calcula un valor, pero para que un procedimiento califique como función, puede o no tener alguna entrada y puede o no devolver una salida donde hay alguna relación obvia entre la entrada y la salida.

A las **entradas** de una función se le llaman **parámetros** y a las **salidas** se les conoce como **valores de retorno**.

Dicho de otra manera, una función engloba un conjunto de instrucciones.



Clasificación de las funciones

En programación:

- Funciones que retornan.
- Funciones que no retornan.

Funciones en JS

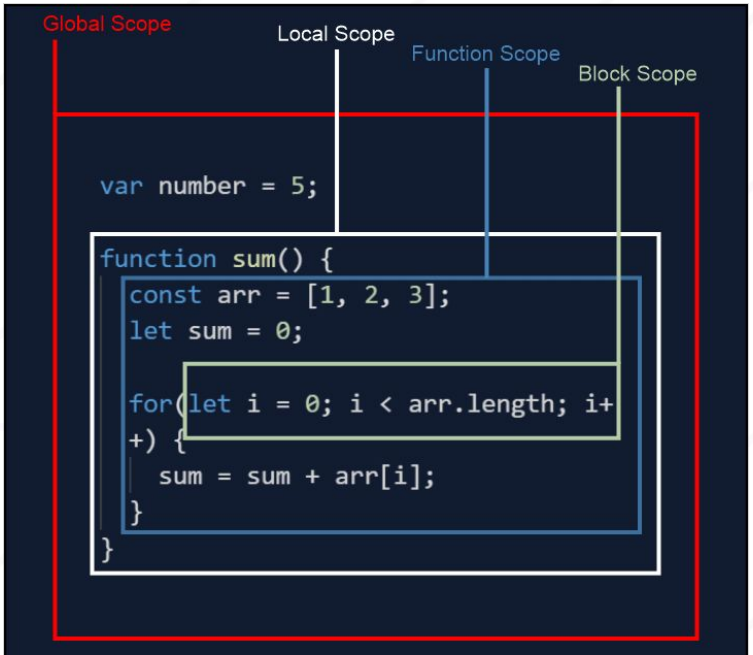
- Funciones convencionales (declaraciones).
- Funciones anónimas.
- IIFE.
- Funciones flecha.

Ámbito de vida de una variable Scope

DEV.F:
DESARROLLAMOS(PERSONAS);

Scope

El ámbito (Scope) se refiere a **los lugares dentro de nuestro código en donde las variables están disponibles para su uso**. Cuando una variable tiene un ámbito global, significa que está disponible en cualquier lugar de tu programa.



Hoisting

DEV.F:: DESARROLLAMOS(PERSONAS);

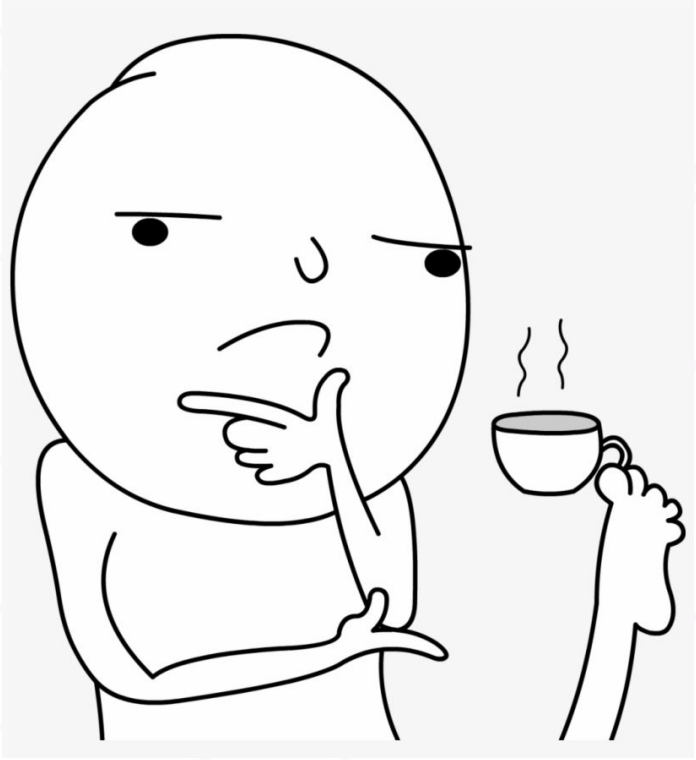
Hoisting

El concepto de Hoisting fue pensado como una manera general de referirse a cómo funcionan los **contextos de ejecución** en JavaScript, es un concepto de ES2015 (es6).

Características

- Consiste en elevar la declaración de variables y funciones al inicio.
- No funciona para la inicialización sólo para la declaración.
- Docs.
- Video explicativo.

Qué sigue?



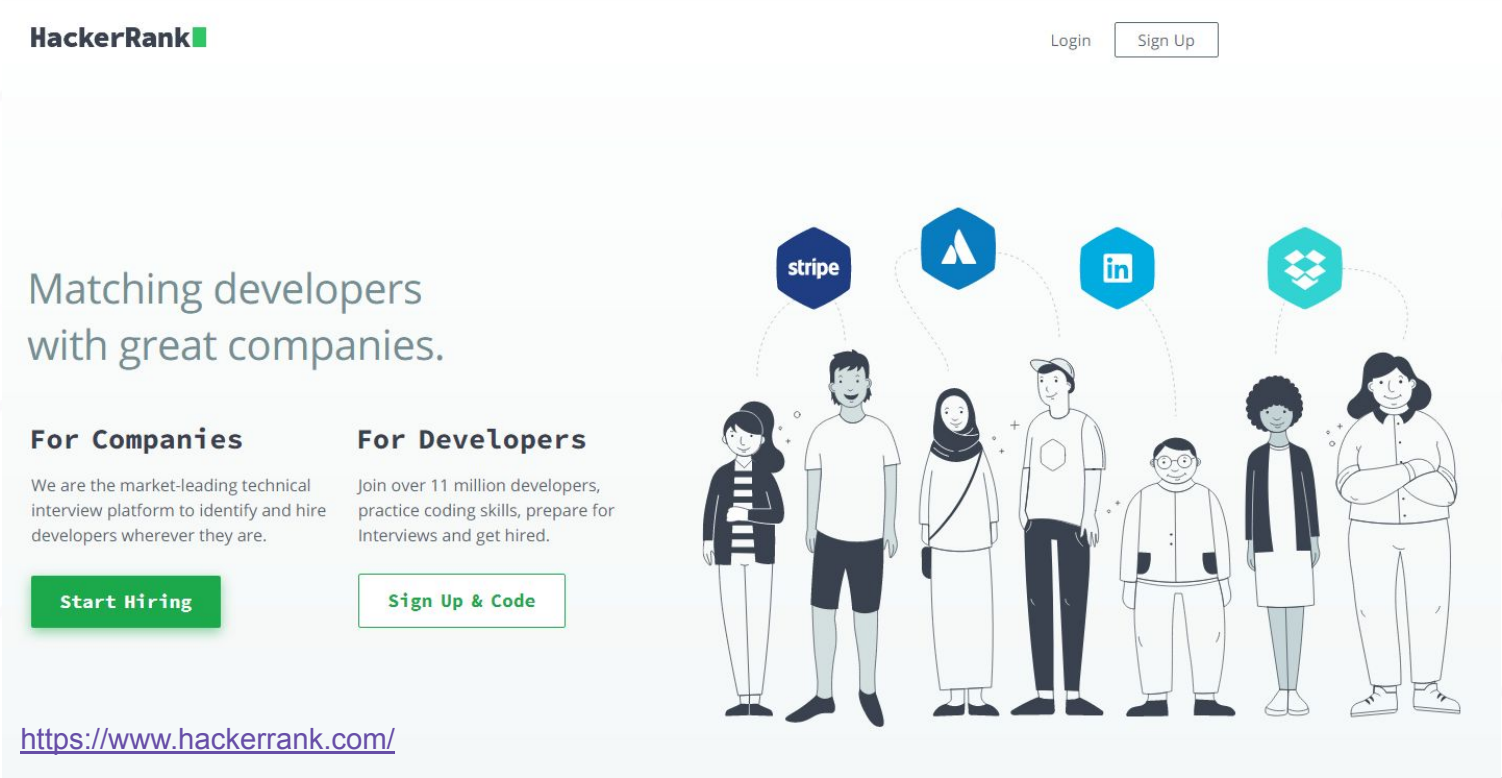
¿Cómo mejorar mi lógica?

DEV.F:
DESARROLLAMOS(PERSONAS);

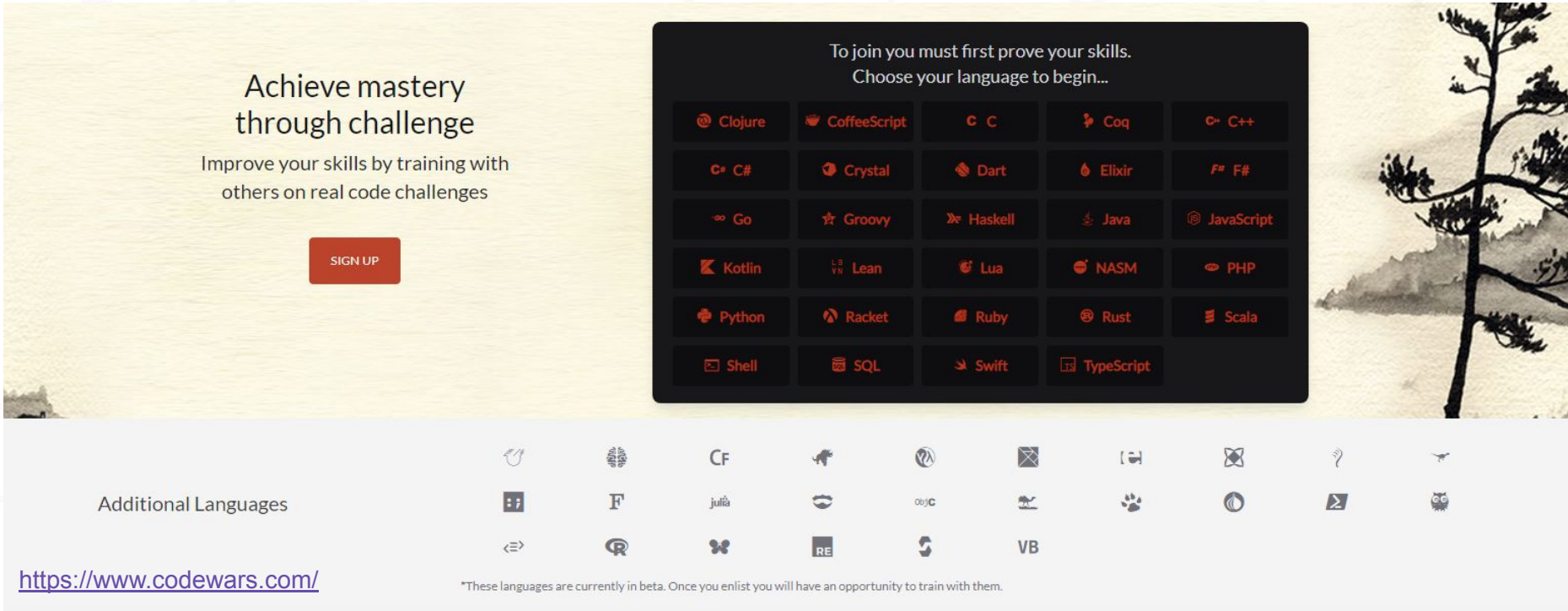
¿Cómo puedo mejorar mi lógica de programador?

1. **Intenta entender el código de otros programadores.** Cuando trabajes en equipo o aprendas por tu cuenta. Al ver el código de otros programadores, descubres formas distintas de solucionar un mismo problema, formas que quizás a ti nunca se te hubiera ocurrido.
2. **Resuelve algoritmos.** Un algoritmo es la forma que tenemos para solucionar un problema sin la necesidad de programarlo. Soluciona tus ejercicios de programación en un papel, antes de intentar llevarlo a código.
3. **Aprende pseudocódigo.** El pseudocódigo es una forma de estructurar tu algoritmo en un lenguaje intermedio. No llega a ser un lenguaje de programación, pero tampoco es el lenguaje común con el que hablamos todos los días.
4. **Programa mucho.** No intentes desarrollar de frente programas grandes; sino, empieza desarrollando cientos de ejercicios pequeños.

Plataformas para estimular la lógica



https://www.hackerrank.com/



https://www.codewars.com/

Estudia un poco más de JS

- JavaScript curso en SoloLearn.
- JavaScript en tutorialesYa.
- Curso de principiantes JS.
- Explicación en video de las sentencias de control y repetitivas:
  - if
  - for
  - while
  - switch