Curso Frontend Developer

- validator.w3.org → Validación de sintáxis html.
- Emoji - Keyboard.

Pseudo-clases

```
p:first-child {
  ... }
```

↳ Propiedad solo para primer elemento (hijo)

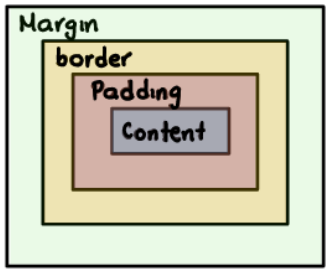
Pseudo-elementos

```
p::first-letter {
  ... }
```

↳ Propiedad solo para primer elemento (hijo)

Modelo de caja

- Padding afecta modelo de caja (Cambia width y height de div).
- Propiedad "box-sizing:border-box"
  → Respeta width/height.
- Margin



- Head ≠ Header.

Arquitecturas CSS

1 Predecible
2 Reutilizable
3 Mantenible
4 Escalable.

→

- Establecer reglas.
- Explicar estructura base.
- Establecer estándares de codificación.
- Evitar largas hojas de estilo.
- Documentación.

OOCSS → CSS orientado a objetos. (El de toda la vida).
BEM → Block element modify (WTF).
SMACSS → Arquitectura escalable y modular.
ITCSS → Dividir hoja de estilo. [Triángulo invertido]
Atomic Design

→ StoryBooks. npm install storybook

- flexboxfroggy.com -> Maquetación para flexbox.
- labs.jensimmons.com.
- las páginas pueden estar en css-grid y con flex-box.
- cssgridgarden.com -> Maquetación para grid

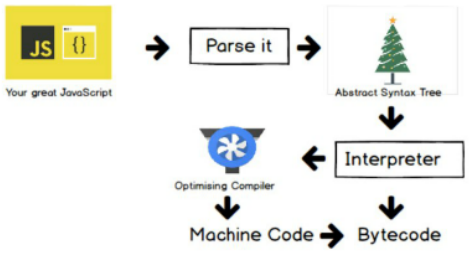
Accesibilidad -> ANDI

tabindex = "0"

aria-label = "algo"

JAVASCRIPT

- Lenguaje compilado
- Orientado a objetos
- Debilmente tipado
- Dinámico



- Node.Js → Backend.

Dos componentes principales.

- Data que guardamos en memoria
- Tareas (funciones) que haremos con esta data.

Valores

`10` → Valor tipo número

`"sergio G"` → Valor tipo string.

`true, false` → valor booleano

`null, undefined` → V. placeholder.

`[1, 2, 3]` → Array (valor tipo objeto)

`{nombre: "sergio G"}` → Valor tipo objeto.

- Con `typeof` puedo saber el tipo del valor.

Variables

`var nombre = "sergio"` → Guarda espacio en memoria.

- Cuando una variable se declara sin inicializar, su valor es `undefined`.

Funciones

- Conjunto de sentencias que hacen cosas mágicas

➢ Funciones declarativas.

```
function miFunction(){
    return 3; }
```

➢ Función de expresión (anónimas)

```
Var miFunción = function(a,b){
    return a+b; }
```

- Las dos se llamarían `miFuncion();`

Importante

" A las funciones declarativas se les aplica hoisting, y a la expresión de función, no. Ya que el hoisting se aplica en las palabras reservadas var y function".

Funciones Declarativas → Se puede llamar la función antes de declararla.

Funciones de Expresión → **No** se puede llamar la función antes de declararla.

Hoisting is Javascript's default behavior of moving declaration to the top.

Scope

Alcance que tienen las variables.
Scope global y scope local.

Scope global.
Scope local.

```
var nombre = "Diego";

function fun(){
  var apellido = "~"
  return nombre + apellido }
```

fun(); → Diego ~
console.log(apellido) → ✗

- No puedes acceder desde el scope global a variables que habiten en el scope local.

Hoisting

- Cuando las variables y las funciones se procesan antes de la ejecución del código.

- Solo ocurre en las versiones de ECNAScript 5 hacia abajo.

Coerción

- Cambiar un tipo de valor a otro.

```
Var a = 20;  ──→ tipo número.
Var b = String(a) ──→ Cambio a tipo String.
```

Truthy - False Boolean()

Valores falsos → 0, null, NaN, undefined, false, "".
Valores True → 1, "a", [], {}, function(){}.

Operadores

== → No compara el tipo de valor, sino el valor per se
↳ 3=="3" True
=== → Extrictamente igual.
+= → para sumar mas numeros var a = b+=2

Condicionales

```
condition ? true : false;
```
→ if, else en una sola linea.

```
var num = 1
var result = num === 1 ? "Si soy uno" : "No, no soy uno"
```

Switch

Funciona por casos.

```
Var numero = 1

switch (true) {
  case 1 :
    Console.log ("soy uno")
    break
  case 10 :
    Console.log ("soy un 10")
    break
    :
  defaut:
    console.log ("no soy nada").
```

→ break: prohibe hacer las otras validaciones.

→ defaut: cuando no se cumple ninguna condición.

Array

- Es un valor tipo objeto.
- Estructura de datos.
- Puedes hacer array dentro de array

Mutar array:

frutas.push("uvas") → Agrega al último lugar
frutas.pop("uvas") → Elimina elemento u. lugar
frutas.unshift("uvas") → Agrega al primer lugar

frutas.shift("uvas") → Elimina al primer lugar
frutas.indexOf("uvas") → Saber el index de un elemento en un array

Loops

- Recorrer arreglos con for.

```
for(var estudiante of estudiantes){
    saludarEstudiante(estudiante);
    }
```

estudiante → el array

estudiante se declara y se convierte en el index del array

estudiante = estudiantes[i]

Objetos

```
var carro = {
    marca : "Toyota"
    modelo : "Corolla"
}
```

carro.marca → Llama la función.

- Se puede que una de las propiedades del objeto sea una función.

```
var carro = {
    ⋮
    detalleDelAuto : function(){
        console.log(`Auto ${this.modelo} ${this.marca}`)
    }
}
```

this → Hace referencia al objeto padre, en este caso "carro"

- Función constructora.

```
function auto(marca,modelo,annio){
      this.marca = marca;
      this.modelo = modelo;
      this.annio = annio;
}

var autoNuevo = new auto ("Tesla", "model 3", 2020)
```

creo un objeto, basándome en otro objeto.

Métodos de recorridos Array de Objetos

```
var articulos = [
    { nombre: "Bici", costo: 3000 },
    { nombre: "TV", costo: 4000 },
    { nombre: "Ipod", costo: 9000 },
                ⋮
];
```

- Filter.

Metodo para recorrer el array

```
var articulos = articulos.filter (function (articulo) {
      return articulo.costo <= 5000
});
```

- Map.

Mapea el array

```
var articulos = articulos.map (function (articulo) {
      return articulo.nombre.
});
```

los trae por este parámetro

→ ["bici","TV","Ipod"]

- Find. (genera un nuevo array)

```
var encuentraArticulo = articulos.find(function(articulo){
    return articulo.nombre === "TV"
}
```

→ { nombre: "laptop", costo: 4000 }

- Foreach. (sobre el arreglo).

```
articulos.foreach(function(articulo){
    console.log(articulo.nombre);
});
```

→

```
Bici
TV
Ipod
```

- some() -> Hace una validación de verdadero o falso sobre una condición.