La Guía del Programador

Resumen hiperconcentrado de los conocimientos y conceptos más importantes

Variables

Las usamos **para almacenar tipos de datos y valores**. Tipos más comunes: string, number, boolean, object, array.

```
// Usamos let cuando sabemos que la variable puede redefinirse o cambiar su tipo
let suma = 20;
// Usamos const cuando sabemos que la variable no va a ser redefinida
const objeto = {
  propiedad: "contenido",
};
```

Condicionales

Los usamos para hacer nuestros algoritmos más inteligentes. Cuando una condición se cumple, se ejecuta un bloque de código, de lo contrario, puede ejecutarse otro.

```
if (suma > 19) {
  console.log("Suma es mayor que 19.");
} else {
  console.log("Suma es menor que 19.");
}
```

Alternativa con switch:

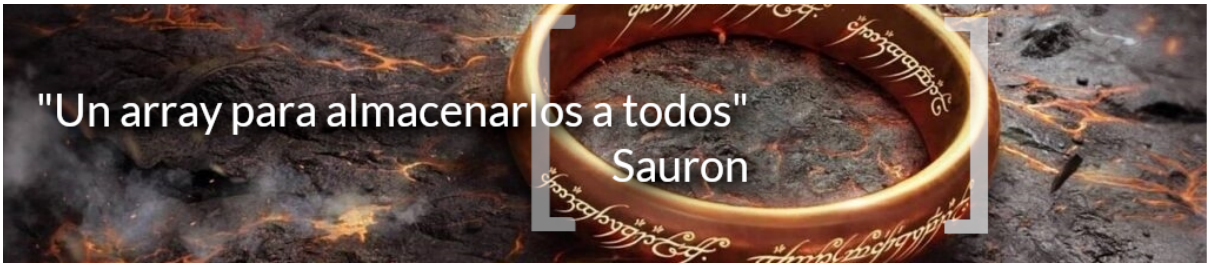
Funciones

Las funciones son extremadamente útiles y necesarias en programación, **nos permiten aislar ciertos algoritmos y porciones de código para poder reutilizarlos en cualquier parte de nuestra aplicación**.

```
function sumar(valor1, valor2) {
  // Me suma los dos valores recibidos como parámetros
  // y los devuelve con un return
  return valor1 + valor2;
}
let resultado1 = sumar(20, 20); // 40
let resultado2 = sumar(10, 10); // 20
```

Arrays

Los arrays son listas de variables, y estas variables pueden ser de cualquier tipo. **Básicamente una variable para almacenar variables**.



Los arrays tienen métodos MUY útiles que nos permiten trabajar con ellos de manera eficiente según el problema a resolver al que nos enfrentemos:

```
const listaVariada = ["Ricardo", "Ana", "Vicente", "María", 20, true, objeto];
// Con find buscamos elementos y lo almacenamos en una variable
let ana = listaVariada.find((elemento) => elemento == "Ana");
// indexOf devuelve el índice del elemento
let indiceRicardo = listaVariada.indexOf("Ricardo");
// Elimina uno o varios elemento elemento del array a partir de índice indicado
listaVariada.splice(indiceRicardo, 1);
// Pobre Ricky, lo volvemos a poner en el array con push
listaVariada.push("Ricardo");
```

Estos son algunos de los más usados, pero hay muchos más.

Ciclos

Los usamos **para recorrer listas (arrays) o repetir bloques de código X cantidad de veces**.

```
// For incremental
for (let i = 1; i < 11; i++) {
  let resultado = tabla * i;
  alert(tabla + " x " + i + " es igual a " + resultado);
}
// For facha
for (const item of mochila) {
  let indiceItem = mochila.indexOf(item);
}
```

Objetos

Los objetos son **una herramienta poderosa que nos permite agrupar muchos datos (propiedades) en una sola variable**, llamada objeto. Por ejemplo un producto:

```
const producto = {
    nombre: "Celular Samsung",
    stock: 10,
    precio: 100
}
```

Métodos

Son funciones internas del objeto.

objeto.metodo()

Constructor
El primer método que se llama automáticamente cuando se crea (instancia) un objeto desde una clase "molde". Es la encargada de recibir los parámetros y definir las propiedades del objeto (con this).

```
class Pokemon {
  constructor(nivel, energia) {
    this.nivel = nivel;
    this.energia = energia;
  }
}

const pikachu = new Pokemon(1, 10);
```

Alternativa para crear un objeto simplificado:

```
const pikachu = {
    nivel: 1,
    energia: 10
}
```

Esta alternativa no se puede usar como molde al no tener una clase, por lo que tampoco tiene método constructor.

Las dos opciones son válidas, solo que en algunos casos una opción puede resultar mejor que otra.

Si tenemos varios tipos de objetos, y estos comparten sus métodos y propiedades (ejemplo clase Pokémon y sus 150 Pokemones) lo ideal es que usemos una clase.

Pero en proyectos más pequeños es probable que con la versión simple nos alcance.

PRO TIP: Está bueno aclarar que internamente para JavaScript prácticamente todo es un objeto, es por eso que incluso hasta las variables string o number tienen métodos.

```
let numero = 20;
let numeroATexto = numero.toString() // de 20 a “20”
```

Y por supuesto los arrays, que tienen métodos muy útiles.

El poder del DOM

El DOM nos brinda **el poder de MANIPULAR los elementos de nuestro HTML de forma dinámica y en tiempo real, sin necesidad de recargar la página**.

```
// Creo un div de la nada (mágeco)
let div = document.createElement("div");
// Le agrego HTML dentro del div
div.innerHTML = "<h2>¡Hola Coder!</h2>";
// Añado el elemento como hijo de body con append, y aparece en el HTML
document.body.append(div);
```

Eventos

Es **la herramienta que tenemos para saber cuándo el usuario está interactuando con nuestro HTML**.

Con el método addEventListener le indicamos a JavaScript que esté atento a que cuando tal evento ocurra dentro de tal elemento, ejecute una función con las instrucciones que nos interese a nosotros.

```
const elemento = document.querySelector("#btnEnviar");
elemento.addEventListener("click", function () {
  // Contenido de la función, este bloque de código se va a ejecutar cuando se
  // dispare el evento "click" dentro del evento
  console.log("Hiciste click en el elemento #btnEnviar");
});
```

Storage

Gracias al storage tenemos la posibilidad de **almacenar variables y datos de forma PERSISTENTE en el navegador**.

```
// Ej de variable o dato que queremos almacenar
let variable = "Ricky";
// Con el método setItem guardamos datos en el storage. El primer parámetro es la clave (similar a un ID de una base de datos), y el segundo parámetro es el valor, el dato que queremos guardar
localStorage.setItem('miVariable1', variable);
// Con el método getItem obtenemos datos del storage. Con solo pasarle la clave que le asociamos al dato almacenado nos devuelve lo que guardamos
let datoAlmacenado = localStorage.getItem('miVariable1');
// Imprimo en consola el dato almacenado, que se va a imprimir de forma persisten incluso si cierro y abro la ventana
console.log(datoAlmacenado); // Ricky
```

Ahora bien, **si queremos guardar datos más complejos como objetos o arrays, debemos usar el poder de JSON**. Esto es debido a que el storage sólo permite almacenar datos en forma de string.

```
// Array vacío
let arrayProductos = [];
// Le agrego producto en forma de objeto
arrayProductos.push({
    nombre: "Arroz",
    precio: 100,
    cantidad: 1
});
// Interactuando con el storage
// Con el método stringify de JSON lo que hacemos en convertir nuestro array en un string en formato JSON, para poder guardarlo de forma COMPATIBLE con el storage
localStorage.setItem('carrito', JSON.stringify(arrayProductos));
// Y cuando necesitemos volver a utilizar el array, nos va a devolver un string. Un array u objeto en forma de string no nos sirve, por lo que usamos el método parse para convertir el string en su forma original
let arrayStorage = JSON.parse(localStorage.getItem('carrito'));
// Nos devuelve el array original
console.log(arrayStorage);
```

Operadores avanzados: Sugar Syntax

Son **simplemente alternativas a la sintaxis original, pero con menos código** (o más facheras, como nos gusta decir). Si bien no nos hace mejores programadores utilizarlas, sí es importante conocerlas por si el día de mañana nos toca interpretar código ajeno (algo muy probable en nuestras carreras).

Operador ternario

```
let nombre1 = "Ricky"; // Viene del prompt
let edad = 40; // Viene del prompt
// Sintaxis original
if (mayorEdad >= 18) {
  mayorEdad = true;
} else {
  mayorEdad = false;
}
// Alternativa con OPERADOR TERNARIO
let mayorEdad = edad >= 18 ? true : false;
```

Operador lógico OR

```
// Sintaxis original
if (carritoStorage) {
  this.productos = carritoStorage;
} else {
  this.productos = [];
}
// Operador lógico OR
this.productos = carritoStorage || [];
```

Variables “falsy”

Es importante entender cuándo JavaScript interpreta cuando una variable es “falsy” o no a la hora de usar condiciones y operadores lógicos. Acá un ejemplo simple que lo demuestra:

```
// Lista de variables Falsy
let variable;
variable = false;
variable = 0;
variable = "";
variable = null;
variable = undefined;
variable = NaN;

if (variable) {
  console.log("La variable es válida (true):", variable);
} else {
  console.log("No entró en el condicional (falsy):", variable);
}
```

El Camino del Programador:

Cómo enfocarse en aprender la lógica y en adquirir la mentalidad del programador

1. No hace falta entender TODO en profundidad y a la perfección.

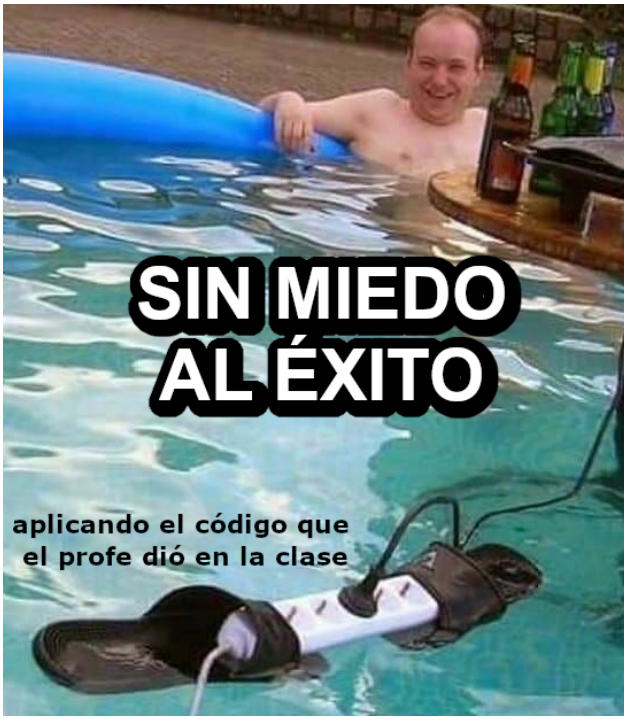
```
// Ejemplo: Tal vez no lo entienda a la perfección, pero vi en clase que con
// este método puedo buscar un elemento del array y cuando lo encuentra me lo
// devuelve en una variable, por ejemplo cuando necesito saber si un producto
// está en el carrito
enCarrito(nuevoProducto) {
 return productos.find((producto) => (producto.nombre == nuevoProducto.nombre));
}
```

Saber su utilidad es más que suficiente para empezar a utilizarlo cuando lo necesite. Con la práctica y el tiempo ya lo voy a entender en profundidad.

2. Se aprende jugando y picando código, no mirando.

Copiar código y modificarlo, cambiarle los nombres a las variables, probar cosas diferentes, trabajar y sortear los errores, debuggear con console.log(), ahí es cuando más se aprende.

No hay que tenerle miedo a romper el código o a los errores, porque es ahí donde se aprende de verdad.



3. Este curso es solo el comienzo de sus exitosas carreras, por lo que es REQUISITO y parte del camino como programadores aprender a usar las poderosas herramientas que tienen a su disposición.

- **Google**: Las problemas y las dudas que tienen, alguien ya las se las hizo antes que ustedes y las respuestas están a un tipeo y a un click de distancia.
- **YouTube**: Mucho material y muy bien explicado, desde las cosas más simples como variables y funciones a cosas más complejas como el manejo del DOM y los eventos.
- **ChatGPT**: TREMENDA ventaja que tenemos a favor de las generaciones de programadores anteriores. Un asistente personal que puede explicarnos en detalle lo que se nos ocurra; desdé el cómo funciona hasta el cómo hacerlo.

Estas son herramientas poderosísimas de las que se nutren no solo los programadores en formación como ustedes, sino también los programadores profesionales de alto rango (seniors).

Si logran adquirir esta mentalidad, no van a tener límites.

Tarea:

Aprender a aprender de Freddy Vega