



UNAHR
UNIVERSIDAD NACIONAL
DE HURLINGHAM

Introducción a JavaScript

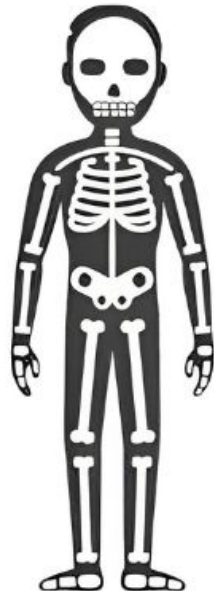
CÁTEDRA: CONSTRUCCIÓN DE INTERFACES DE USUARIO

Lenguajes para el desarrollo de páginas web

HTML



HTML the Skeleton



CSS



CSS the Skin



JS



Javascript the Brain



Lenguajes para el desarrollo de páginas web

HTML: Define la estructura y contenido de una página web.

CSS: Define la presentación y el diseño de una página web.

Javascript: Añade interactividad y comportamiento dinámico a una página web.



¿Qué es JavaScript?

JavaScript es un lenguaje interpretado, orientado a objetos, débilmente tipado y dinámico.



JavaScript



¿Realmente es Javascript un lenguaje interpretado?

Sí, y la razón es que el navegador lee línea por línea nuestro código, el cual le indica lo que tiene que ir haciendo, sin la necesidad de compilar. Todo esto es controlado por el motor de Javascript V8 del navegador.

Orientado a objetos

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo vamos a pensar en un coche para tratar de modelizarlo en un esquema de POO.

Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca.

Por tanto, pensar en objetos requiere analizar qué elementos vas a manejar en tus programas, tratando de identificar sus características y funcionalidades.

¿Por qué es débilmente tipado?

Porque los tipos de datos no están bien definidos en el lenguaje y permite, por ejemplo, operaciones entre números y letras. Esto sucede porque el lenguaje asume tipos de datos que no necesariamente fueron los que se querían representar. Se pueden hacer operaciones entre tipos distintos de datos (enteros con strings, booleanos con enteros, etc.).

Ejemplo:

```
4 + "7"; // 47
```

```
4 * "7"; // 28
```

```
2 + true; // 3
```

```
false - 3; // -3
```

¿Por qué decimos que Javascript es un lenguaje dinámico?

Corre directamente en la etapa de Runtime, sin una etapa de compilación previa. Esto permite probar nuestro código inmediatamente; pero también es lo que hace que los errores no se muestren sino hasta que se ejecuta el programa.

¿Por qué decimos que Javascript es un lenguaje dinámico?

JavaScript permite declarar (por ejemplo) variables cuyo valor (y tipo) sólo se conocerá al momento de su ejecución en función de las condiciones dadas al momento de correrlo en un entorno real. En cambio, los lenguajes estáticos no compilarán en código ejecutable a menos que todos los valores (o tipos de valores) se conozcan por adelantado.

Primeros pasos Javascript

alert es una función incorporada que se utiliza para mostrar un cuadro de diálogo de alerta con un mensaje especificado.

Este cuadro de diálogo es modal, lo que significa que bloquea la interacción con el resto de la página hasta que el usuario lo cierre haciendo clic en el botón "Aceptar".

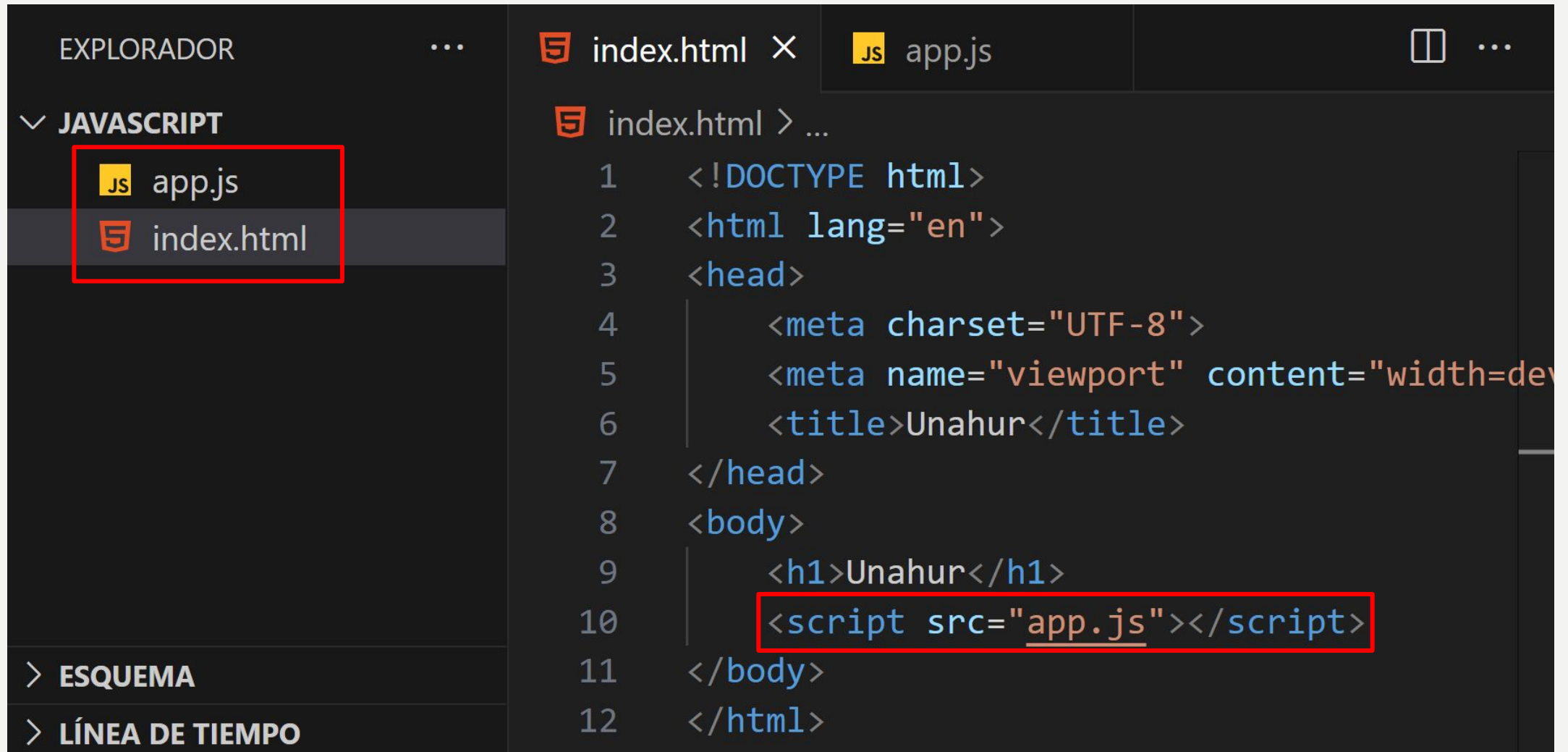
La sintaxis básica para usar alert es:

```
alert(mensaje);
```

Primeros pasos Javascript

```
alert("primero");
```

Primeros pasos Javascript



EXPLORADOR

... index.html X JS app.js

JS app.js

index.html

index.html > ...

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>Unahur</title>
7 </head>
8 <body>
9     <h1>Unahur</h1>
10    <script src="app.js"></script>
11 </body>
12 </html>
```

> ESQUEMA

> LÍNEA DE TIEMPO

Primeros pasos Javascript

Orden de ejecución

Cuando el navegador encuentra un bloque de JavaScript, generalmente lo ejecuta en orden, de arriba a abajo. Es importante el orden en el que colocamos el código.

```
alert("primero");  
alert("segundo");
```

Primeros pasos Javascript

JavaScript distingue entre mayúsculas y minúsculas y es muy exigente, por lo que debes ingresar la sintaxis exactamente como se muestra; de lo contrario, es posible que no funcione.

Primeros pasos Javascript

Valores por consola

Para mostrar valores por consola, se utiliza la función:
`console.log()`

Sintaxis Básica de JavaScript

Variables

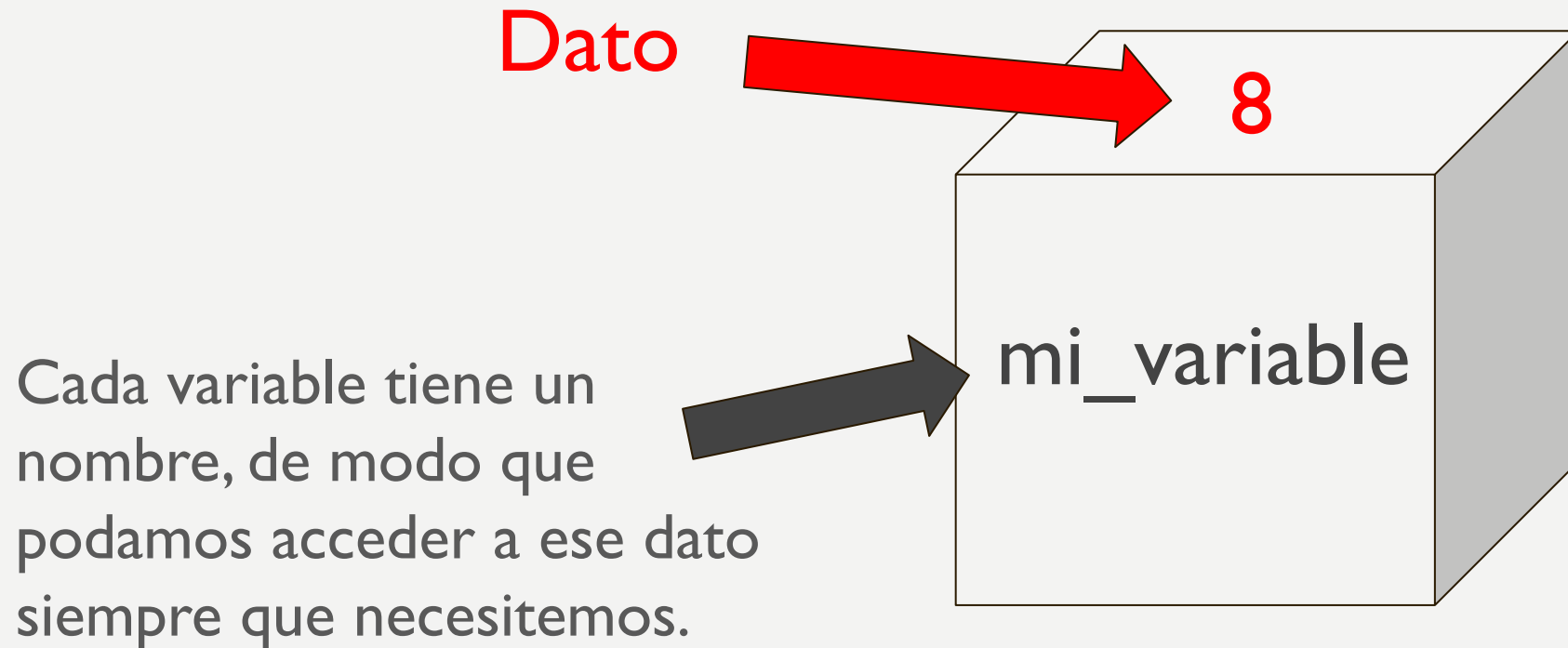
La mayoría del tiempo, una aplicación de JavaScript necesita trabajar con información.

Ejemplos:

- Una tienda en línea – La información puede incluir los bienes a la venta y un “carrito de compras”.
- Una aplicación de chat – La información puede incluir los usuarios, mensajes, y mucho más.

Variables

Una variable es un “almacén con un nombre” para guardar datos.



Variables

Entonces una variable es un espacio en memoria donde se almacena un dato, un espacio donde podemos guardar cualquier tipo de información que necesitemos. Podemos pensarla como una caja, donde almacenamos un dato. Esa caja tiene un nombre, para que más adelante podamos referirnos a la variable, recuperar el dato así como asignar un valor a la variable siempre que deseemos.

Variables

Para generar una variable en JavaScript, se usa la palabra clave **let**.

La siguiente declaración genera (en otras palabras: declara o define) una variable con el nombre “mensaje”:

```
let mensaje;
```

Ahora podemos introducir datos en ella al utilizar el operador de asignación =

```
mensaje = 'Hola'; // almacenar la cadena 'Hola' en la variable llamada  
mensaje
```

```
alert(mensaje); // muestra el contenido de la variable,
```

Ejercicio

Declara dos variables, una para almacenar tu nombre y otra para tu edad. Luego, imprime ambos valores en la consola.

Resolución

```
// Declaración de variables  
let nombre = "Juan";  
let edad = 25;
```

```
// Imprimir valores en la consola  
console.log("Nombre:", nombre);  
console.log("Edad:", edad);
```

Variables

El nombre únicamente puede incluir letras, dígitos, o los símbolos \$ y _.
El primer carácter no puede ser un dígito.

Ejemplos de nombres válidos:

```
let usuarioNombre;  
let test123;
```

Cuando el nombre contiene varias palabras, comúnmente se utiliza **camelCase**. Ejemplo: **nombreApellidoUsuario**.

Variables

Hay una **lista de palabras reservadas**, las cuales no pueden ser utilizadas como nombre de variable porque el lenguaje en sí las utiliza.

Por ejemplo: **let, class, return, y function** están reservadas.

El siguiente código nos da un error de sintaxis:

```
let let = 5; // no se puede nombrar "let" a una variable ¡Error!
```

```
let return = 5; // tampoco se le puede nombrar "return", ¡Error!
```


Constantes

Para declarar una variable constante (inmutable) use `const` en vez de `let`:

```
const tuCumple = '18.04.1996';
```

Las variables declaradas utilizando `const` se llaman “constantes”. No pueden ser alteradas. Al intentarlo causaría un error:

```
const tuCumple = '18.04.1996';  
tuCumple = '01.01.2001'; // ¡error, no se puede reasignar la constante!
```

Cuando un programador está seguro de que una variable nunca cambiará, puede declarar la variable con `const` para garantizar y comunicar claramente este hecho a todos.

Tipos de datos primitivos

En JavaScript, los tipos de datos primitivos son los tipos de datos más básicos que no se pueden descomponer en tipos más simples.

Los tipos de datos primitivos en JavaScript incluyen:

- **number (número):** Representa tanto números enteros como de punto flotante.
- **string (cadena de texto):** Representa una secuencia de caracteres.
- **boolean (booleano):** Representa un valor de verdad, que puede ser **true** o **false**.
- **null:** Representa la ausencia intencional de cualquier valor. Es un tipo de dato que tiene un solo valor: null
- **undefined:** Representa un valor no asignado. Una variable que ha sido declarada pero no inicializada tiene el valor undefined.

number

Ejemplos:

```
let entero = 42;
```

// Número entero

```
let flotante = 3.14;
```

// Número de punto flotante

```
let negativo = -10;
```

// Número negativo

```
let notANumber = NaN;
```

// Not-a-Number (resultado de una operación

matemática inválida)

```
let infinito = Infinity;
```

// Infinito positivo

```
let menosInfinito = -Infinity;
```

// Infinito negativo

string

Ejemplos:

```
let saludo = 'Hola, Mundo';           // Usando comillas simples
let nombre = "Juan";                  // Usando comillas dobles
let mensaje = `Hola, ${nombre}`;      // Usando comillas invertidas (template
literals) para interpolación de variables
```

boolean

Ejemplos:

```
let esVerdadero = true;  
let esFalso = false;
```

null y undefined

Ejemplos:

```
let valorNulo = null;
```

```
let sinDefinir;  
console.log(sinDefinir); // undefined
```

Ejercicio

1. Declara dos variables: admin y nombre.
2. Asigna el valor "John" a nombre.
3. Copia el valor de nombre a admin.
4. Muestra el valor de admin usando `console.log()`.

Resumen

Podemos declarar variables para almacenar datos al utilizar las palabra clave `var`, `let`, o `const`.

let – es la forma moderna de declaración de una variable.

var – es la declaración de variable de vieja escuela. Normalmente no lo utilizamos en absoluto.

const – es como `let`, pero el valor de la variable no puede ser alterado.

Las variables deben ser nombradas de tal manera que entendamos fácilmente lo que está en su interior.

Operadores

Los operadores son símbolos que se utilizan para realizar operaciones sobre valores y variables.

Los operadores permiten manipular los datos de diversas maneras, tales como realizar cálculos matemáticos, comparar valores, y combinar expresiones lógicas.

Operadores Aritméticos

Se utilizan para realizar operaciones matemáticas sobre números.

+ : Suma

- : Resta

***** : Multiplicación

/ : División

% : Módulo (resto de la división)

++ : Incremento

-- : Decremento

Ejemplo de operadores aritméticos

```
let a = 5;
```

```
let b = 2;
```

```
console.log(a + b); // 7
```

```
console.log(a - b); // 3
```

```
console.log(a * b); // 10
```

```
console.log(a / b); // 2.5
```

```
console.log(a % b); // 1
```

```
a++;
```

```
console.log(a); // 6
```

```
b--;
```

```
console.log(b); // 1
```

Operadores de Asignación

Se utilizan para asignar valores a las variables.

= :Asignación simple

+= :Asignación con suma

-= :Asignación con resta

***=** :Asignación con multiplicación

/= :Asignación con división

%= :Asignación con módulo

Ejemplo de operadores de asignación

```
let x = 10;
```

```
x += 5; // x = x + 5
```

```
console.log(x); // 15
```

```
x -= 3; // x = x - 3
```

```
console.log(x); // 12
```

```
x *= 2; // x = x * 2
```

```
console.log(x); // 24
```

```
x /= 4; // x = x / 4
```

```
console.log(x); // 6
```

```
x %= 5; // x = x % 5
```

```
console.log(x); // 1
```

Operadores de Comparación

Se utilizan para comparar dos valores y devolver un valor booleano (true o false).

== : Igualdad

!= : Desigualdad

=== : Igualdad estricta (incluye tipo)

!== : Desigualdad estricta (incluye tipo)

> : Mayor que

< : Menor que

>= : Mayor o igual que

<= : Menor o igual que

Ejemplo de operadores de comparación

```
let a = 5;  
let b = "5";
```

```
console.log(a == b); // true  
console.log(a === b); // false  
console.log(a != b); // false  
console.log(a !== b); // true  
console.log(a > 3); // true  
console.log(a < 3); // false  
console.log(a >= 5); // true  
console.log(a <= 5); // true
```

Operadores Lógicos

Se utilizan para combinar expresiones booleanas.

&& :AND lógico

|| :OR lógico

! :NOT lógico

Ejemplo de operadores lógicos

```
let a = true;  
let b = false;
```

```
console.log(a && b); // false  
console.log(a || b); // true  
console.log(!a); // false  
console.log(!b); // true
```

Operadores de Cadena (String)

Se utilizan para trabajar con cadenas de texto.

+ : Concatenación de cadenas

Ejemplo

```
let nombre = "Juan";  
let saludo = "Hola, " + nombre + "!";  
console.log(saludo); // Hola, Juan!
```

Ejercicio

Declara dos variables con valores numéricos. Realiza operaciones de suma, resta, multiplicación y división, y muestra los resultados en la consola.

Resolución

// Declaración de variables

```
let num1 = 10;
```

```
let num2 = 5;
```

// Operaciones

```
let suma = num1 + num2;
```

```
let resta = num1 - num2;
```

```
let multiplicacion = num1 * num2;
```

```
let division = num1 / num2;
```

// Imprimir resultados en la consola

```
console.log("Suma:", suma);
```

```
console.log("Resta:", resta);
```

```
console.log("Multiplicación:", multiplicacion);
```

```
console.log("División:", division);
```

Tipos de datos no primitivos

Los tipos de datos no primitivos se refieren a aquellos que no son valores básicos y pueden contener múltiples valores y funcionalidades.

A diferencia de los tipos de datos primitivos (como number, string, boolean, null y undefined), los tipos de datos no primitivos son objetos y arreglos que pueden almacenar colecciones de datos y más funcionalidades.

Arreglos (Array)

Un arreglo (array) es una **colección ordenada de elementos**, donde cada elemento tiene un índice numérico. Los arreglos son un tipo especial de objeto en JavaScript y pueden almacenar elementos de diferentes tipos de datos.

El índice comienza en cero. Es decir, el primer elemento de un array se encuentra en el índice cero, el segundo elemento en el índice uno y el tercer elemento en el índice dos, etc.

```
let nums = [1, 2, 3];  
nums[0] = 5;
```

```
console.log(nums); // [5, 2, 3]
```

Objeto

Un objeto es una colección de propiedades, donde cada propiedad es una asociación de clave-valor. Las claves son cadenas y los valores pueden ser de cualquier tipo de datos, incluidos otros objetos y funciones.

Los objetos son una parte fundamental del lenguaje y permiten estructurar y organizar datos de manera flexible y dinámica.

Objeto

Ejemplo:

```
let persona = {  
  nombre: "Juan",  
  edad: 30,  
  esEmpleado: true,  
  direccion: {  
    calle: "Calle Falsa",  
    numero: 123  
  }  
};
```

Objeto

Ejercicio: Creación de Objetos en JavaScript

Crea un objeto llamado tienda con las siguientes propiedades:

nombre: Una cadena que represente el nombre de la tienda (puede ser cualquier nombre).

direccion: Una cadena que represente la dirección de la tienda.

productos: Un array que contenga al menos 4 elementos.

Objeto

```
// Definición del objeto
const tienda = {
  nombre: "Ferretería ABC",
  direccion: "Calle Principal 123",
  productos: [
    { id: 1, nombre: "Martillo", precio: 10.99 },
    { id: 2, nombre: "Destornillador", precio: 5.49 },
    { id: 3, nombre: "Taladro", precio: 29.99 },
    { id: 4, nombre: "Sierra", precio: 15.99 }
  ],
};
```

Función

Una función es un bloque de código diseñado para realizar una tarea específica. Las funciones son objetos de primera clase, lo que significa que pueden ser asignadas a variables, pasadas como argumentos a otras funciones y devueltas por otras funciones.

Las funciones permiten encapsular lógica y reutilizar código de manera eficiente.

```
function saludar(nombre) {  
  return "Hola, " + nombre;  
}
```

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

Función

Las funciones también pueden ser definidas como expresiones y asignadas a variables:

```
let saludar = function(nombre) {  
  return "Hola, " + nombre;  
};
```

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

Función

Las funciones flecha (Arrow Functions) fueron introducidas en ES6.

Las funciones flecha proporcionan una sintaxis más corta.

```
let saludar = (nombre) => {  
  return "Hola, " + nombre;  
};
```

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

Creación de objeto

Ejemplo:

```
let persona = {  
  nombre: "Juan",  
  edad: 30,  
  esEmpleado: true  
};
```

Creación de objeto

Otra forma de crear un objeto es utilizando el constructor Object:

```
let persona = new Object();  
persona.nombre = "Juan";  
persona.edad = 30;  
persona.esEmpleado = true;
```