Variables y Tipos de Datos en JavaScript

©2025 Codefinity. All Terms Reserved.

codefinity

📖 Comprensión de las Variables en JavaScript

Las variables desempeñan un papel fundamental en la programación, permitiendo almacenar y reutilizar valores.

> **Nota**
>
> Una **variable** es como un contenedor que puede guardar diferentes tipos de datos, como números, texto o estructuras más complejas.

Antes de utilizar variables, deben ser definidas. Para definir una variable, utiliza la palabra clave `let` y elige un nombre para ella:

```
let variableName;
```

> **Nota**
>
> Los nombres de las variables deben seguir el estilo **camelCase**. **camelCase** significa que las palabras se escriben juntas sin espacios, y cada palabra (excepto la primera) comienza con mayúscula.

Por defecto, una nueva variable contiene el valor `undefined`, lo que indica que no se ha asignado ningún valor.

> **Nota**
>
> El valor `undefined` indica que aún no se ha asignado un valor a la variable.

Puedes asignar un valor a una variable utilizando el operador de asignación (`=`):

©2025 Codefinity. All Terms Reserved.

```
let x;
x = 5;
```

De manera alternativa, puedes asignar un valor al momento de definir la variable:

```
let x = 13;
```

Uso de variables

Las variables pueden utilizarse para representar diversos valores en el código y pueden reutilizarse varias veces, lo que reduce la repetición de código.

```
let word = "VariableText";  // The variable value is the
`"VariableText"`

console.log("ValueText");  // Print the `"ValueText"`
console.log(word);  // Print the variable value
```

Puede asignar y reasignar valores a las variables según sea necesario:

```
let numb;
numb = 100;
console.log(numb);

numb = 100 + 20;
console.log(numb);
```

©2025 Codefinity. All Terms Reserved.

Las variables son fundamentales en la programación porque permiten reutilizar el mismo valor muchas veces. Imagina un escenario en el que has escrito 1000 líneas de código utilizando un valor específico, pero cometiste un error tipográfico que aparece varias veces. En ese caso, tendrías que corregir cada aparición de esa palabra en tu código, lo que supondría una considerable pérdida de tiempo.

Ahora, analicemos un ejemplo similar con menos líneas:

```
console.log("Hello Word!");
console.log("I love this Word!");
console.log("I need to live in this Word!");
console.log("This Word is the best!");
```

En este ejemplo, el autor omitió accidentalmente la letra `l` en la palabra `World`. Para corregir este programa, solo es necesario corregir una aparición del error tipográfico.

Observa el siguiente ejemplo:

```
let x = "Word";

console.log("Hello " + x + "!");
console.log("I love this " + x + "!");
console.log("I need to live in this " + x + "!");
console.log("This " + x + " is the best!");
```

Puedes corregirlo cambiando una letra en el valor de la variable de `"Word"` a `"World"`. Corrige el ejemplo proporcionado por tu cuenta.

©2025 Codefinity. All Terms Reserved.

> **Nota**
>
> En el ejemplo anterior, utilizamos **concatenación de cadenas**. Profundizaremos en la concatenación de cadenas en la siguiente sección.

©2025 Codefinity. All Terms Reserved.

codefinity

🕮 Desafío: Definir la Variable

Tarea

1. Definir una variable llamada `myVar` y asignarle el valor `15`.
2. Imprimir la variable `myVar` en la consola.

```
___ ___ = ___;

console.log(___);
```

1. Utilizar la palabra clave `let` para declarar una variable.
2. Asignar el valor 15 a la variable utilizando el operador de asignación (`=`).
3. Utilizar la función `console.log()` para imprimir el valor de la variable en la consola.

```
let myVar = 15;

console.log(myVar);
```

©2025 Codefinity. All Terms Reserved.

codefinity

📖 Uso de Const para Constantes

Otra forma de definir una variable en JavaScript es utilizando la palabra clave `const`. La diferencia principal entre `let` y `const` es que las variables creadas con const no pueden cambiar su valor, mientras que el uso de la palabra clave `let` permite modificar el valor de la variable.

```
const myNumber = 10;
```

Comparemos el comportamiento de las variables usando `let` y `const`. Observa el siguiente ejemplo donde podemos cambiar el valor de la variable `a`:

```
// A variable changing
let a = 5;
console.log(a); // Print the initial value of `a`

a = 10;
console.log(a); // Print the updated value of `a`
```

En contraste, examinemos el comportamiento de la variable `b`. Se producirá un error: `TypeError: Assignment to a constant variable.`

©2025 Codefinity. All Terms Reserved.

```
// A constant cannot be changed
const b = 7;
console.log(b); // Print the initial value of `b`

// Attempting to reassign a `const` variable will result in an error
b = 8; // This line will throw an error: "TypeError: Assignment to
constant variable."
console.log(b);
```

Uso

Las constantes se utilizan como variables inmutables. Se puede definir una constante una vez y utilizarla varias veces.

Las constantes proporcionan integridad de datos y permiten una refactorización rápida.

> **Nota**
>
> **Refactorización** implica realizar cambios estructurales en el código, como modificar valores, nombres de variables/funciones y más.

Por ejemplo, considere `maxHeight` para los elementos de un sitio. Se puede lanzar una actualización del sitio cambiando la altura máxima de los elementos con solo una modificación en el código. Sin embargo, es importante tener en cuenta que no se puede cambiar la altura máxima durante la ejecución, lo que garantiza la integridad de los datos.

©2025 Codefinity. All Terms Reserved.

```
const maxHeight = 250;
console.log(maxHeight - 15);
console.log(maxHeight - 12);
console.log(maxHeight - 5);
console.log(maxHeight - 13);
console.log(maxHeight - 22);
console.log(maxHeight - 52);
```

©2025 Codefinity. All Terms Reserved.

codefinity

📖 Explorando los Tipos de Datos en JavaScript

Los datos pueden representarse de diversas maneras, y las operaciones que se realizan sobre los datos pueden variar según los tipos de datos.

> **Nota**
>
> El **tipo de dato** indica al intérprete cómo trabajar con los datos.

Veamos la diferencia en el comportamiento del intérprete:

```
// First case
console.log("123" + "123");

// Second case
console.log(123 + 123);
```

En el ejemplo anterior, se puede observar que el intérprete realiza operaciones diferentes para distintos tipos de datos.

typeof()

El operador `typeof()` devuelve una cadena que indica el tipo del valor del operando.

©2025 Codefinity. All Terms Reserved.

```
let a = 15;
console.log(typeof 23);
console.log(typeof a);

const b = "today";
console.log(typeof "word");
console.log(typeof b);
```

Number

El tipo de dato **number** se utiliza para cálculos, contadores, operaciones matemáticas y más.

A diferencia de otros lenguajes de programación, JavaScript utiliza el tipo **number** en lugar de tipos separados como **int** y **float**.

```
console.log(typeof(15.25));
console.log(typeof(211));

console.log(typeof(22 + 251));
console.log(typeof(26 / 342));
```

> **Nota**
>
> El operador `typeof` solo determina el tipo de dato del resultado, no las operaciones realizadas.

String

El tipo de dato **string** se utiliza para modificar, imprimir y transferir texto a otros programas.

©2025 Codefinity. All Terms Reserved.

```
let str = "Hello! I'm String, and I should help you to work with text!";
console.log(str);
```

Para identificar la cadena en el código, se deben usar comillas simples o dobles (por ejemplo, `'some text'` o `"some text"`).

```
console.log("text");
console.log('text');

console.log("console.log('text')");
console.log('console.log("text")');

console.log(typeof("10"));
```

> **Nota**
>
> - Elegir un estilo de comillas (`"texto"` o `'texto'`) para tu código o proyecto;
> - Puedes alternar entre estilos de comillas al usar `'` o `"` dentro del texto, como `"She hasn't hat"` o `'He says "Hi!"'`.

Booleano

El tipo de dato **booleano** se utiliza para operaciones lógicas. Tiene dos valores: `true` y `false`. Los booleanos se usan para verificar condiciones, como se describirá más adelante.

©2025 Codefinity. All Terms Reserved.

Los booleanos permiten controlar la ejecución del código y dirigirlo por diferentes caminos.

Para crear un valor booleano, utiliza los valores `true` o `false`:

```
console.log(true);
console.log(false);

console.log(typeof(true));
console.log(typeof(false));

console.log(25 > 15);
console.log(15 > 25);
```

©2025 Codefinity. All Terms Reserved.

🕮 Trabajando con Null en JavaScript

En JavaScript, el tipo **null** representa "nada" o la ausencia de datos. Se utiliza para indicar que una variable carece intencionadamente de un valor. El siguiente ejemplo demuestra que el tipo **null** no produce salida en la consola:

```
let variable = null

console.log("Some data 1")
console.log(variable)
console.log("Some data 2")
```

El ejemplo anterior demuestra que el tipo `null` no produce salida en la consola.

> **Nota**
>
> 1. `null` es diferente de `undefined`;
> 2. Se utiliza `null` cuando es necesario indicar la ausencia de datos o transmitir el concepto de "nada" a otra parte del programa.

Por ejemplo, imagina que estás trabajando en un juego donde necesitas describir los datos de un héroe. En algunos casos, el nombre del héroe puede ser desconocido o estar ausente. Si la variable que contiene el nombre del héroe no existe, intentar acceder a ella resultaría en un error. Utilizar el tipo null permite indicar que el héroe no tiene nombre y puede ser pasado a otra parte del programa.

©2025 Codefinity. All Terms Reserved.

codefinity

📖 Introducción a los Arreglos

La **matriz** es la estructura de datos más útil.

> **Nota**
>
> Una **estructura de datos** es un formato especializado para organizar y trabajar con una colección de datos. Una **matriz** es una colección de elementos donde cada elemento es un valor.

Creación de una matriz

Para crear una matriz, utilice corchetes `[]`:

```
const arr = [];
```

Esto crea una matriz vacía sin elementos. Para crear una matriz con elementos, coloque los elementos dentro de `[]` y sepárelos con comas (`,`):

```
const arr = [1, 2, 3, 4, 5];
```

> **Nota**
>
> El último elemento de la matriz no debe llevar una coma después de él.

Para imprimir una matriz, simplemente utilice `console.log()`:

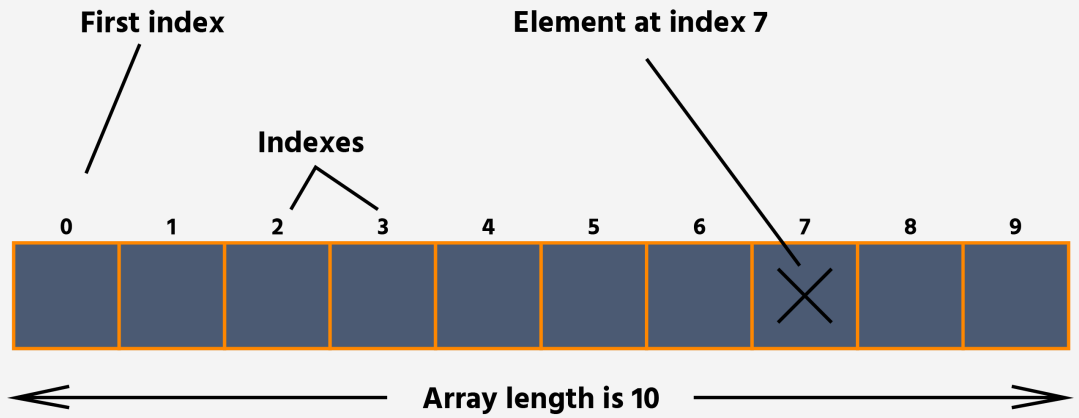
©2025 Codefinity. All Terms Reserved.

```
const arr = [1, 2, 3, 4, 5];

console.log(arr);
```

Índices

Cada elemento en un arreglo tiene un índice único.



> **Nota**
>
> Los índices comienzan desde **0**. El primer elemento tiene un índice de **0**, el segundo elemento tiene un índice de **1**, y así sucesivamente.

El acceso a los datos de un arreglo se realiza especificando el índice entre corchetes (`[index]`) después del nombre del arreglo:

```
arrayName[index]
```

©2025 Codefinity. All Terms Reserved.

```
const arr = [1, 2, 3, 4, 5];

const thirdElement = arr[2];  // retrieving the 3rd element

console.log(thirdElement);
```

Los arreglos pueden contener diferentes tipos de datos, incluyendo **number**, **string**, **boolean** y más.

```
const arr = [2441.23, "Hello!", false];

console.log(arr[0]);
console.log(arr[1]);
console.log(arr[2]);
```

Propiedad Length

La propiedad `length` es una propiedad incorporada de los arreglos que representa la cantidad de elementos en ese arreglo. Para utilizar esta propiedad, emplee la siguiente sintaxis:

```
arrayName.length
```

©2025 Codefinity. All Terms Reserved.

```
const arr1 = [10, 20, 30];
const arr2 = [15, 25, 35, 45, 55];

const x = arr1.length; // `x` variable represents the `arr1` length
const y = arr2.length; // `y` variable represents the `arr2` length

console.log(x);
console.log(y);
```

©2025 Codefinity. All Terms Reserved.

codefinity

📖 Métodos y Manipulación de Arreglos

Los arrays son versátiles para almacenar y recuperar datos. Recuperar datos utilizando corchetes `[]` se denomina **indexación**.

```
let arr = [1, 2, 3, 4, 5, 6];
arr[3] // This is indexing
```

Sin embargo, existen varios métodos disponibles para trabajar con arrays.

Agregar elementos

Existen diferentes formas de agregar elementos a un array.

Push

El método `push()` agrega un nuevo valor al final del array:

```
let arr = [1, 2, 3];

arr.push(4);
arr.push(5);
arr.push(6);

console.log(arr);
```

Unshift

El método `unshift()` funciona como el método `push()`, pero inserta el valor al principio del arreglo.

©2025 Codefinity. All Terms Reserved.

```
let arr = [1, 2, 3];
console.log("Array:", arr);

arr.unshift(222);  // Insert element at the start

console.log("Array:", arr);
```

Indexación

Es posible agregar un nuevo valor mediante **indexación**:

```
let arr = [1, 2];

arr[2] = 3;
arr[3] = 4;

console.log(arr);
```

La indexación permite asignar un valor a un índice específico, reasignar un valor anterior y realizar otras operaciones:

```
let arr = [1, 2, 3];

arr[0] = 4;

console.log("Array:", arr);
```

Para crear un nuevo elemento en el array sin errores, se puede utilizar el método `push(value)` o la expresión `arr[arr.length] = value`:

©2025 Codefinity. All Terms Reserved.

```
let myArray = [];

myArray[myArray.length] = "indexing";
console.log("After first indexing:", myArray);

myArray.push("pushing");
console.log("After first pushing:", myArray);

myArray[myArray.length] = "indexing";
console.log("After second indexing:", myArray);

myArray.push("pushing");
console.log("After second pushing:", myArray);
```

Eliminación de elementos

En ocasiones, puede ser necesario eliminar elementos de un array. Esto se puede realizar de diferentes maneras.

Pop

El método `pop()` elimina el último elemento de un array y permite guardarlo en otra variable:

©2025 Codefinity. All Terms Reserved.

```
let arr = [11, 22, 33, 44];
console.log("Array:", arr);

let x = arr.pop(); // Remove and save the last element in `arr` to
variable `x`

console.log("Popped element:", x);
console.log("Array now:", arr);
```

Shift

El método `shift()` funciona como `pop()`, pero elimina el **primer elemento** de un arreglo:

```
let arr = [11, 22, 33, 44, 55, 66];
console.log("Array:", arr);

let popped = arr.pop();  // Remove the last element

console.log("Popped:", popped);
console.log("Array:", arr);

let shifted = arr.shift();  // Remove the first element

console.log("Shifted:", shifted);
console.log("Array:", arr);
```

©2025 Codefinity. All Terms Reserved.

codefinity

🕮 Desafío: Realizar Operaciones con Arreglos

Tarea

1. Crear un arreglo llamado `myArray` con los siguientes datos: `["Heine", 25, "Bob", 16]`.
2. Eliminar el último elemento, `16`, del arreglo.
3. Agregar un nuevo elemento con el valor `21` al final del arreglo.
4. Reemplazar el **segundo** elemento con el valor `27` utilizando **indexación**.
5. Imprimir el arreglo actualizado en la consola.

```
let ___ = ["Heine", 25, "Bob", 16];
myArray.___();
myArray.___(21);
myArray[___] = 27;
console.log(___);
```

La salida debe ser:

```
Heine,27,Bob,21
```

1. Utilizar el método `pop()` para eliminar un elemento del final de un arreglo.
2. Utilizar el método `push()` para agregar un elemento al final de un arreglo.
3. Acceder al segundo elemento del arreglo usando el índice `1`.
4. Para mostrar los datos del arreglo en la consola, utilizar `console.log()` con el nombre del arreglo.

©2025 Codefinity. All Terms Reserved.

```
let myArray = ["Heine", 25, "Bob", 16]; // Step 1
myArray.pop();          // Step 2
myArray.push(21);       // Step 3
myArray[1] = 27;        // Step 4
console.log(myArray);   // Step 5
```

©2025 Codefinity. All Terms Reserved.