Estructuras de Datos

Ing. En Sistemas Computacionales – 3er Semestre agosto – diciembre 2023

JOSE ARTURO BUSTAMANTE LAZCANO

Contenido

[Semana 1 – Unidad 1 - Clasificación de las estructuras de datos 2](#_Toc144992636)

[Introducción a las Estructuras de datos 2](#_Toc144992637)

[Semana 2 – Unidad 1 – Tipos de datos Abstractos 3](#_Toc144992638)

[Tipos de datos abstractos 3](#_Toc144992639)

[Manejo de memoria 9](#_Toc144992640)

[Semana 3 – Unidad 1 – Cierre de la unidad 11](#_Toc144992641)

[Ejemplos de estructuras 11](#_Toc144992642)

[Semana 4 – Resumen de la unidad 1 14](#_Toc144992643)

[Tipos de datos 14](#_Toc144992644)

[Operadores aritméticos 15](#_Toc144992645)

[Concatenación de cadenas 16](#_Toc144992646)

[Métodos 17](#_Toc144992647)

[Objetos incorporados 19](#_Toc144992648)

[Instrucciones 20](#_Toc144992649)

[Revisar 22](#_Toc144992650)

# Semana 1 – Unidad 1 - Clasificación de las estructuras de datos

21, 22 y 24 de agosto.

## Introducción a las Estructuras de datos

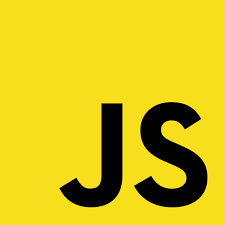
Las estructuras de datos son formas de organizar y almacenar datos en la memoria de una computadora de manera eficiente y accesible. Se pueden clasificar en varias categorías según su organización y funcionalidad.

Las estructuras de datos se pueden clasificar de varias maneras, pero una de las formas más comunes es por su organización. Las estructuras de datos lineales son aquellas en las que los datos están organizados en una secuencia, como una lista, una pila o una cola. Las estructuras de datos no lineales son aquellas en las que los datos no están organizados en una secuencia, como un árbol o un grafo.

Otra forma de clasificar las estructuras de datos es por su tamaño. Las estructuras de datos estáticas son aquellas en las que el tamaño de los datos es fijo, mientras que las estructuras de datos dinámicas son aquellas en las que el tamaño de los datos puede cambiar durante la ejecución del programa.

Es importante elegir la estructura de datos adecuada según los requisitos del problema y las operaciones que se realizan con los datos. Cada estructura de datos tiene sus propias ventajas y desventajas en términos de eficiencia y complejidad en las operaciones.

Por último, las estructuras de datos también se pueden clasificar por su tipo. Las estructuras de datos primitivas son aquellas que almacenan datos simples, como números, cadenas o booleanos. Las estructuras de datos complejas son aquellas que almacenan datos compuestos, como listas, árboles o grafos.



Conclusión: 🤔

# Semana 2 – Unidad 1 – Tipos de datos Abstractos

28, 29 y 31 de agosto.

## Tipos de datos abstractos

Los tipos de datos abstractos (ADT) son tipos de datos que definen un conjunto de operaciones que se pueden realizar sobre los datos, pero no especifican cómo se implementan los datos. Los ADT se utilizan para abstraer los detalles de la implementación de los datos de la lógica de la aplicación. Esto hace que el código sea más portable y mantenible, ya que los cambios en la implementación de los datos no requieren cambios en la lógica de la aplicación.

Los ADT se suelen implementar como clases en un lenguaje de programación orientado a objetos. Las clases definen los datos y las operaciones que se pueden realizar sobre ellos. Los objetos de las clases son instancias de los ADT.

Algunos ejemplos de ADT comunes son:

Listas: una lista es una colección de elementos ordenados. Las operaciones típicas que se pueden realizar sobre una lista son agregar, eliminar y buscar elementos.

Pilas: una pila es una colección de elementos ordenados en la que los elementos se agregan y eliminan al final de la pila.

Colas: una cola es una colección de elementos ordenados en la que los elementos se agregan al final de la cola y se eliminan del principio de la cola.

Diccionarios: un diccionario es una colección de pares de valores clave. Las operaciones típicas que se pueden realizar sobre un diccionario son agregar, eliminar y buscar elementos por clave.

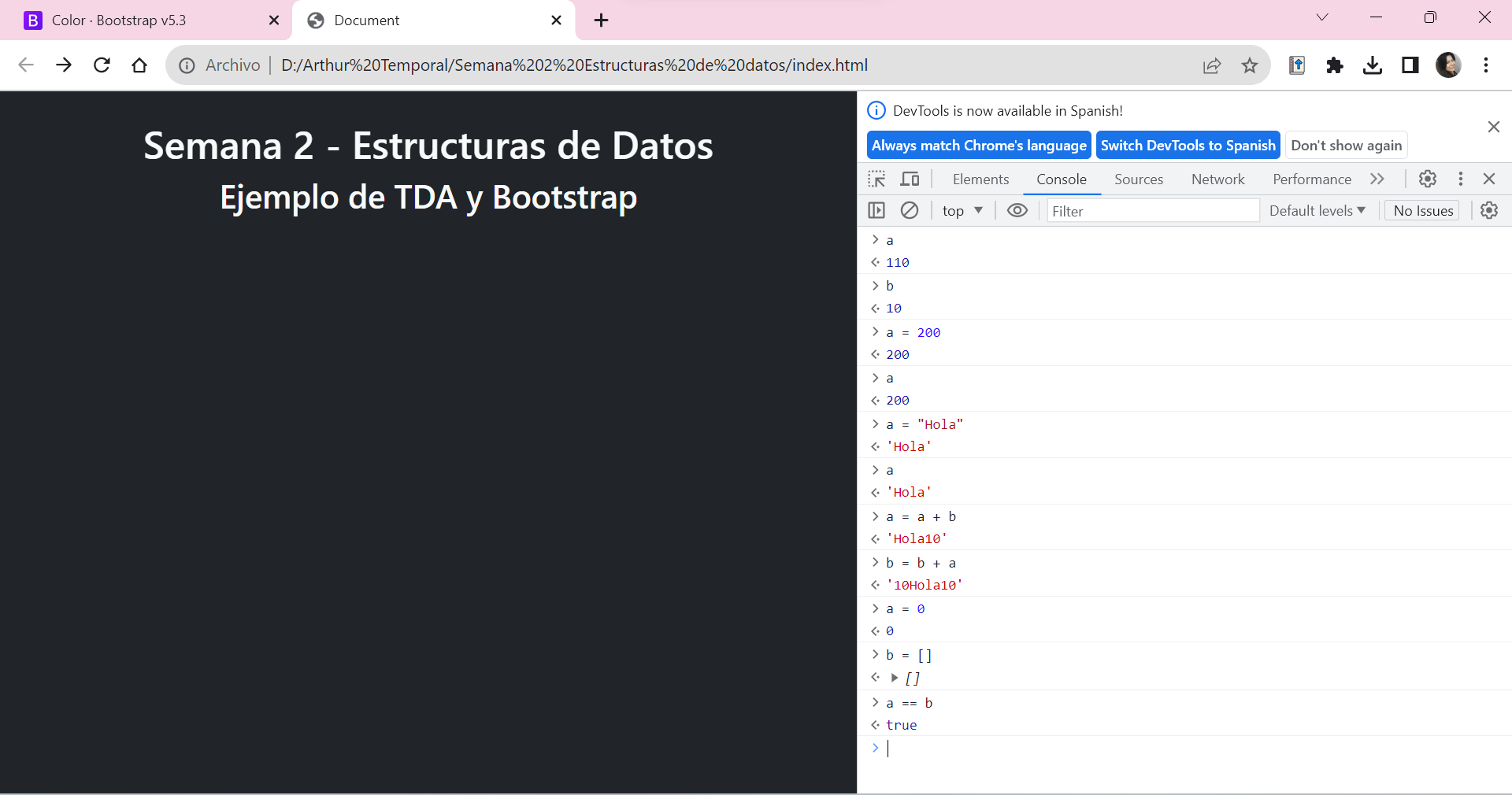
Los ADT se utilizan en una amplia gama de aplicaciones, incluyendo:

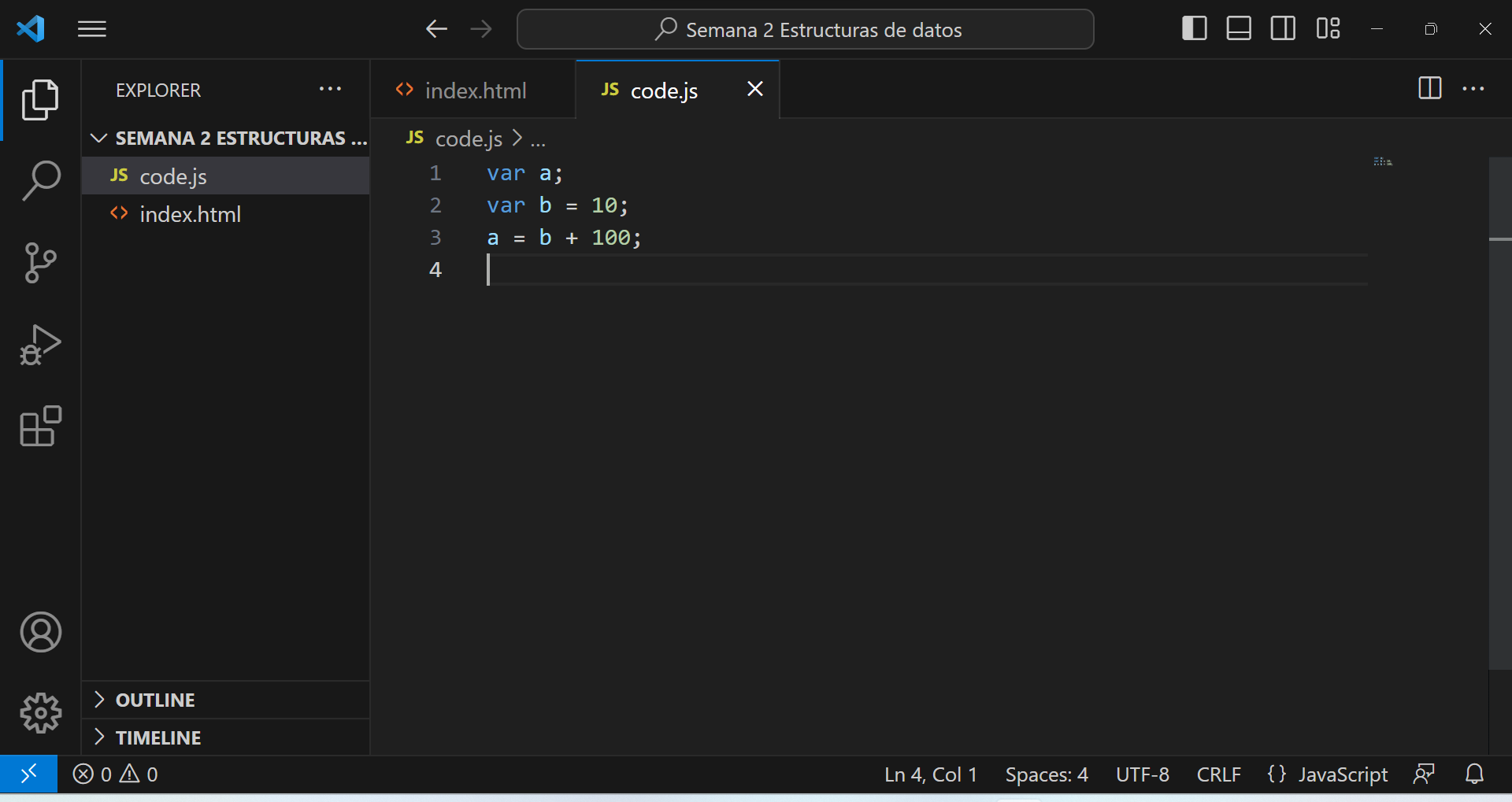
Programación de sistemas: los ADT se utilizan para representar datos abstractos en sistemas operativos, compiladores y otras aplicaciones de software de sistemas.

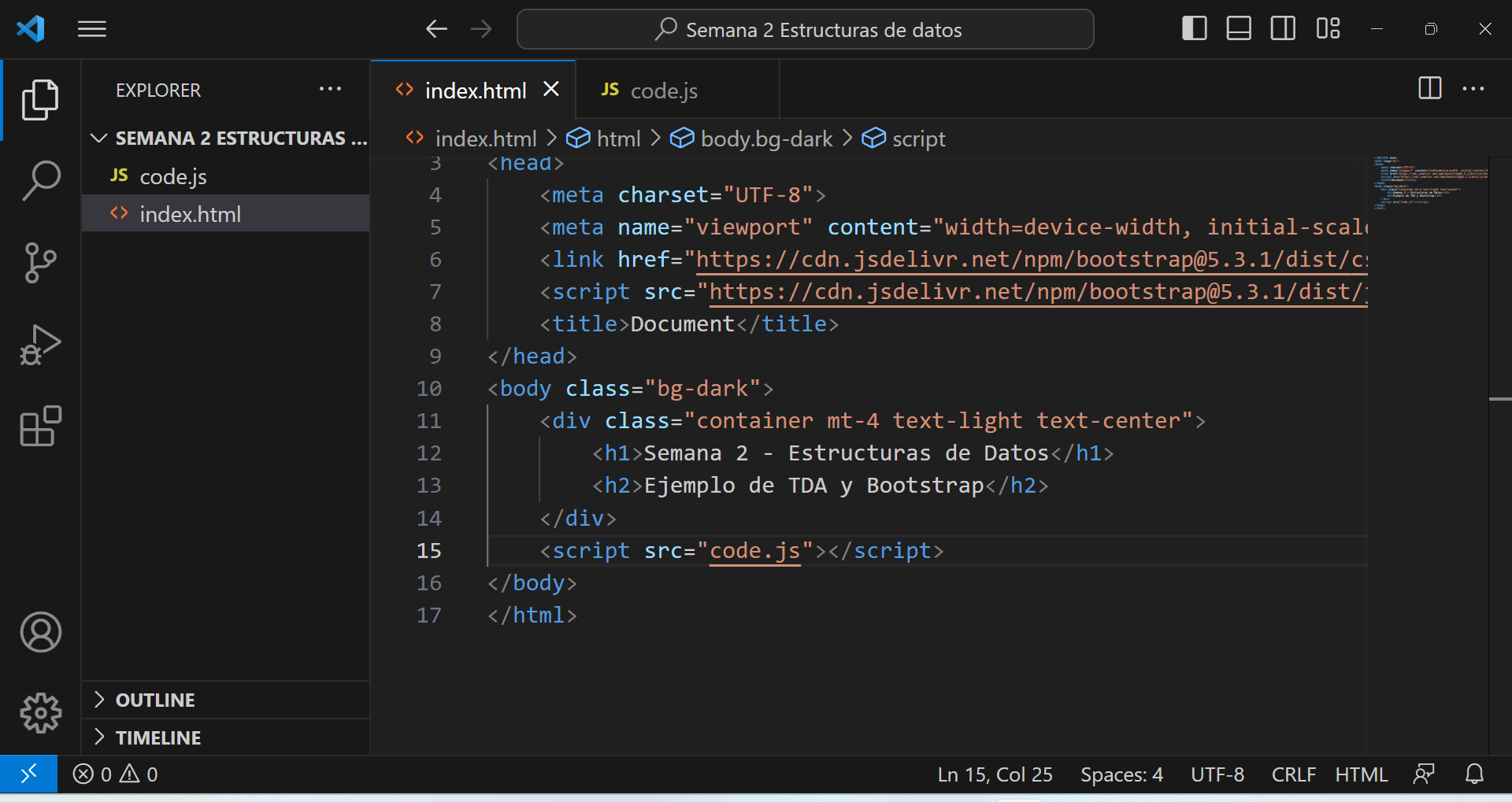
Programación de aplicaciones: los ADT se utilizan para representar datos abstractos en aplicaciones de usuario final, como procesadores de texto, hojas de cálculo y bases de datos.

Los ADT son una herramienta poderosa que puede ayudar a los programadores a escribir código más portable y mantenible.

Ejemplos de TDA’s







Notas:

Una variable de tipo numérico que es sumada con una variable de tipo carácter cambia su tipo de dato siempre a carácter

Las variables pueden cambiar el entorno de almacenamiento a: Variable, Arreglo y Objeto.

Las variables pueden cambiar de tipo de dato en cualquier momento.

Referencia: <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array>

Los arreglos son listas ordenadas que van de la posición 0 a la n por lo tanto su tamaño se calcula como n - 1

Ejemplos en el navegador:

a

110

b

10

a = 200

200

a

200

a = "Hola"

'Hola'

a

'Hola'

a = a + b

'Hola10'

b = b + a

'10Hola10'

a = 0

0

b = []

[]

a == b

true

a = []

[]

b = 0

0

b

0

b = 0.7

0.7

b = "Hola"

'Hola'

b = 10

10

b = []

[]

b = ["Manzana","Pera","Durazno","Mango"]

(4) ['Manzana', 'Pera', 'Durazno', 'Mango']0: "Manzana"1: "Pera"2: "Durazno"3: "Mango"length: 4[[Prototype]]: Array(0)at: ƒ at()concat: ƒ concat()constructor: ƒ Array()copyWithin: ƒ copyWithin()entries: ƒ entries()every: ƒ every()fill: ƒ fill()filter: ƒ filter()find: ƒ find()findIndex: ƒ findIndex()findLast: ƒ findLast()findLastIndex: ƒ findLastIndex()flat: ƒ flat()flatMap: ƒ flatMap()forEach: ƒ forEach()includes: ƒ includes()indexOf: ƒ indexOf()join: ƒ join()keys: ƒ keys()lastIndexOf: ƒ lastIndexOf()length: 0map: ƒ map()pop: ƒ pop()push: ƒ push()reduce: ƒ reduce()reduceRight: ƒ reduceRight()reverse: ƒ reverse()shift: ƒ shift()slice: ƒ slice()some: ƒ some()sort: ƒ sort()splice: ƒ splice()toLocaleString: ƒ toLocaleString()toReversed: ƒ toReversed()toSorted: ƒ toSorted()toSpliced: ƒ toSpliced()toString: ƒ toString()unshift: ƒ unshift()values: ƒ values()with: ƒ with()Symbol(Symbol.iterator): ƒ values()Symbol(Symbol.unscopables): {at: true, copyWithin: true, entries: true, fill: true, find: true, …}[[Prototype]]: Object

b = ["Mandarina"]

['Mandarina']0: "Mandarina"length: 1[[Prototype]]: Array(0)

b = 4

Un ejemplo de una estructura de datos en JavaScript es un array. Un array es una colección de datos del mismo tipo, que están indexados. Los arrays se pueden utilizar para almacenar una variedad de datos, como números, cadenas, objetos y funciones.

Aquí hay un ejemplo de cómo crear un array en JavaScript:

JavaScript

const myArray = [1, 2, 3, 4, 5];

Usa el código con precaución. Más información

Este array contiene cinco números enteros.

Los arrays se pueden manipular mediante una variedad de métodos, como push(), pop(), shift() y unshift(). Estos métodos se utilizan para agregar, eliminar y mover elementos en un array.

Aquí hay un ejemplo de cómo utilizar algunos de estos métodos:

JavaScript

// Add a new element to the end of the array

myArray.push(6);

// Remove the last element from the array

myArray.pop();

// Remove the first element from the array

myArray.shift();

// Add a new element to the beginning of the array

myArray.unshift(0);

Usa el código con precaución. Más información

Este código agregará un nuevo número entero al final del array, eliminará el último elemento del array, eliminará el primer elemento del array y agregará un nuevo elemento entero al principio del array.

Los arrays son una estructura de datos muy versátil que se puede utilizar para una variedad de tareas. Son una parte esencial de la programación en JavaScript.

Aquí hay algunos otros ejemplos de estructuras de datos en JavaScript:

Objetos: Un objeto es una colección de datos de diferentes tipos, que están indexados por claves. Los objetos se utilizan para almacenar datos complejos, como información de contacto o configuraciones.

Matrices: Una matriz es una estructura de datos similar a un array, pero que puede almacenar datos de diferentes tipos. Las matrices se utilizan para almacenar datos de gran tamaño.

Listas enlazadas: Una lista enlazada es una estructura de datos que almacena datos en forma de nodos. Los nodos están conectados entre sí por enlaces. Las listas enlazadas se utilizan para almacenar datos de forma eficiente.

Arboles binarios: Un árbol binario es una estructura de datos que almacena datos en forma de árbol. Los árboles binarios se utilizan para almacenar datos jerárquicos.

Estas son solo algunas de las muchas estructuras de datos disponibles en JavaScript. El tipo de estructura de datos que se utiliza depende de la tarea que se quiera realizar.

## Manejo de memoria

Las estructuras de datos son una forma de organizar datos de manera eficiente para que puedan ser manipulados de forma rápida y fácil. El manejo de memoria es el proceso de asignar y liberar memoria para las estructuras de datos.

Existen dos tipos principales de manejo de memoria: estático y dinámico.

Manejo de memoria estático

El manejo de memoria estático asigna memoria para las estructuras de datos en tiempo de compilación. Esto significa que el tamaño de la memoria asignada no puede cambiar durante la ejecución del programa.

El manejo de memoria estático es simple y eficiente, pero puede ser inflexible. Por ejemplo, si se necesita más memoria para una estructura de datos, no se puede asignar más memoria en tiempo de ejecución.

Manejo de memoria dinámico

El manejo de memoria dinámico asigna memoria para las estructuras de datos en tiempo de ejecución. Esto significa que el tamaño de la memoria asignada puede cambiar durante la ejecución del programa.

El manejo de memoria dinámico es más flexible que el manejo de memoria estático, pero puede ser menos eficiente. Por ejemplo, asignar y liberar memoria en tiempo de ejecución puede requerir más operaciones que asignar memoria en tiempo de compilación.

Relaciones entre estructuras de datos y manejo de memoria

Las estructuras de datos y el manejo de memoria están estrechamente relacionados. El tipo de manejo de memoria que se utiliza puede afectar al rendimiento y la eficiencia de las estructuras de datos.

Por ejemplo, las estructuras de datos que requieren grandes cantidades de memoria pueden beneficiarse del manejo de memoria dinámico. Esto se debe a que el manejo de memoria dinámico permite asignar más memoria a las estructuras de datos en tiempo de ejecución, si es necesario.

Por otro lado, las estructuras de datos que se utilizan con frecuencia pueden beneficiarse del manejo de memoria estático. Esto se debe a que el manejo de memoria estático puede asignar memoria para las estructuras de datos en tiempo de compilación, lo que puede mejorar el rendimiento.

En conclusión

El manejo de memoria es un aspecto importante del diseño y la implementación de estructuras de datos. El tipo de manejo de memoria que se utiliza puede afectar al rendimiento y la eficiencia de las estructuras de datos.

a = []

[]

a.push(100)

1

a

[100]

a.push(200)

2

a

(2) [100, 200]

a.push(300)

3

# Semana 3 – Unidad 1 – Cierre de la unidad

04 de Septiembre

## Ejemplos de estructuras

Para mostrar una variable de JavaScript en un documento HTML, puedes utilizar la manipulación del DOM (Document Object Model) para actualizar el contenido de un elemento HTML con el valor de la variable JavaScript. Aquí hay un ejemplo sencillo de cómo hacerlo:

Supongamos que tienes una variable en JavaScript llamada miVariable y quieres mostrar su valor en un elemento HTML con el id "resultado". Aquí está el código HTML y JavaScript correspondiente:

html

Copy code

<!DOCTYPE html>

<html>

<head>

<title>Mostrar Variable en HTML</title>

</head>

<body>

<!-- Un elemento donde mostrarás la variable -->

<p>El valor de miVariable es: <span id="resultado"></span></p>

<script>

// Tu variable JavaScript

var miVariable = "Hola, esto es un ejemplo.";

// Obtén una referencia al elemento donde deseas mostrar la variable

var elementoResultado = document.getElementById("resultado");

// Asigna el valor de la variable al contenido del elemento HTML

elementoResultado.textContent = miVariable;

</script>

</body>

</html>

En este ejemplo, hemos creado un elemento p con un span dentro de él con el id "resultado". Luego, en JavaScript, hemos obtenido una referencia a ese elemento utilizando getElementById y hemos asignado el valor de miVariable al contenido de ese elemento utilizando la propiedad textContent.

Cuando cargues este HTML en tu navegador, verás que el valor de miVariable se muestra en la página web. Puedes adaptar este ejemplo para mostrar cualquier variable en tu HTML según tus necesidades.

//Trabajando Variables

//Var de tipo entero

var entero = 10;

//Var de tipo Flotante

var real = 20.89;

//Var de tipo Cadena

var cadena = "Mi Saludo";

//Var de tipo Boleano

var banderaT = true;

var banderaF = false;

var elemento1 = document.getElementById("miEntero");

elemento1.textContent = entero;

var elemento2 = document.getElementById("miReal");

elemento2.textContent = real;

var elemento3 = document.getElementById("miCadena");

elemento3.textContent = cadena;

var elemento4 = document.getElementById("miBanderaT");

elemento4.textContent = banderaT;

var elemento5 = document.getElementById("miBanderaF");

elemento5.textContent = banderaF;

//Trabajando con arreglos

//Arreglo con datos constantes

var arregloTipo1 = [20,90,10,7,8,12];

//Arreglo con datos variados

var arregloTipo2 = ["uno",1,"dos",2,"tres",3];

function mostrarLista1(){

    var lista = document.getElementById("miLista1");

    for(var i=0; i<arregloTipo1.length; i++){

        var elementoLista = document.createElement("li");

        elementoLista.textContent = arregloTipo1[i];

        lista.appendChild(elementoLista);

    }

}

mostrarLista1();

function mostrarLista2(){

    var lista = document.getElementById("miLista2");

    for(var i=0; i<arregloTipo2.length; i++){

        var elementoLista = document.createElement("li");

        elementoLista.textContent = arregloTipo2[i];

        lista.appendChild(elementoLista);

    }

}

mostrarLista2();

# Semana 4 – Resumen de la unidad 1

11, 12 y 14

## Tipos de datos

Los tipos de datos son las clasificaciones que damos a los diferentes tipos de datos que utilizamos en la programación. En JavaScript, hay siete tipos de datos fundamentales:

Número : Cualquier número, incluidos los números con decimales: 4, 8, 1516, 23.42.

BigInt : cualquier número, mayor que 2 53 -1 o menor que -(2 53 -1), con n añadido al número: 1234567890123456n.

Cadena : cualquier agrupación de caracteres en su teclado (letras, números, espacios, símbolos, etc.) entre comillas simples: ' ... 'o comillas dobles " ... ", aunque preferimos las comillas simples. A algunas personas les gusta pensar en cuerda como una palabra elegante para referirse al texto.

Booleano : este tipo de datos solo tiene dos valores posibles: trueo false(sin comillas). Es útil pensar en los valores booleanos como interruptores de encendido y apagado o como respuestas a una pregunta de “sí” o “no”.

Nulo : este tipo de datos representa la ausencia intencional de un valor y está representado por la palabra clave null(sin comillas).

Indefinido : este tipo de datos se indica mediante la palabra clave undefined(sin comillas). También representa la ausencia de un valor aunque tiene un uso diferente al de null. undefinedsignifica que un valor dado no existe.

Símbolo : una característica más nueva del lenguaje, los símbolos son identificadores únicos, útiles en codificaciones más complejas. No hay necesidad de preocuparse por esto por ahora.

Objeto : Colecciones de datos relacionados.

Los primeros 6 de esos tipos se consideran tipos de datos primitivos . Son los tipos de datos más básicos del lenguaje. Los objetos son más complejos y aprenderá mucho más sobre ellos a medida que avance en JavaScript. Al principio, siete tipos pueden no parecer muchos, pero pronto observarás que el mundo se abre con posibilidades una vez que comiences a aprovechar cada uno de ellos. A medida que aprenda más sobre los objetos, podrá crear colecciones complejas de datos.

Pero antes de hacer eso, ¡vamos a familiarizarnos con las cadenas y los números!

console.log('Location of Codecademy headquarters: 575 Broadway, New York City');

console.log(40);

## Operadores aritméticos

La aritmética básica suele resultar útil a la hora de programar.

Un operador es un personaje que realiza una tarea en nuestro código. JavaScript tiene varios operadores aritméticos integrados , que nos permiten realizar cálculos matemáticos sobre números. Estos incluyen los siguientes operadores y sus símbolos correspondientes:

Agregar:+

Sustraer:-

Multiplicar:\*

Dividir:/

Resto:%

Los primeros cuatro funcionan como se puede adivinar:

console.log(3 + 4); // Prints 7

console.log(5 - 1); // Prints 4

console.log(4 \* 2); // Prints 8

console.log(9 / 3); // Prints 3

Tenga en cuenta que cuando console.log()la computadora evaluará la expresión dentro del paréntesis e imprimirá ese resultado en la consola. Si quisiéramos imprimir los caracteres 3 + 4, los envolveríamos entre comillas y los imprimiríamos como una cadena.

console.log(11 % 3); // Prints 2

console.log(12 % 3); // Prints 0

El operador restante, a veces llamado módulo , devuelve el número que queda después de que el número de la derecha se divide entre el número de la izquierda tantas veces como sea posible: es 11 % 3igual a 2 porque 3 cabe en 11 tres veces, dejando 2 como resto.

## Concatenación de cadenas

¡ Los operadores no son sólo para números! Cuando +se utiliza un operador en dos cadenas, agrega la cadena derecha a la cadena izquierda:

console.log('hi' + 'ya'); // Prints 'hiya'

console.log('wo' + 'ah'); // Prints 'woah'

console.log('I love to ' + 'code.')

// Prints 'I love to code.'

Este proceso de agregar una cadena a otra se llama concatenación . Observe que en el tercer ejemplo teníamos que asegurarnos de incluir un espacio al final de la primera cadena. La computadora unirá las cadenas exactamente, por lo que necesitábamos asegurarnos de incluir el espacio que queríamos entre las dos cadenas.

console.log('front ' + 'space');

// Prints 'front space'

console.log('back' + ' space');

// Prints 'back space'

console.log('no' + 'space');

// Prints 'nospace'

console.log('middle' + ' ' + 'space');

// Prints 'middle space'

Al igual que con las matemáticas normales, podemos combinar o encadenar nuestras operaciones para obtener un resultado final:

console.log('One' + ', ' + 'two' + ', ' + 'three!');

// Prints 'One, two, three!'

## Métodos

Recuerde que [los métodos](https://www.codecademy.com/resources/docs/javascript/methods?page_ref=catalog) son acciones que podemos realizar. Los tipos de datos tienen acceso a métodos específicos que nos permiten manejar instancias de ese tipo de datos. JavaScript proporciona varios métodos de cadena.

Llamamos o usamos estos métodos agregando una instancia con :

* un punto (el operador de punto)
* el nombre del método
* paréntesis de apertura y cierre

Por ejemplo 'example string'.methodName().

¿Esa sintaxis te parece un poco familiar? Cuando usamos console.log(), llamamos al .log()método del consoleobjeto. ¡ Veamos console.log()algunos métodos de cadenas reales en acción!

console.log('hello'.toUpperCase()); // Prints 'HELLO'  
console.log('Hey'.startsWith('H')); // Prints true

Veamos cada una de las líneas anteriores:

* En la primera línea, el [.toUpperCase()](https://www.codecademy.com/resources/docs/javascript/strings/toUpperCase)método se llama en la instancia de cadena 'hello'. El resultado se registra en la consola. Este método devuelve una cadena en letras mayúsculas: 'HELLO'.
* En la segunda línea, el [.startsWith()](https://www.codecademy.com/resources/docs/javascript/strings/startsWith)método se llama en la instancia de cadena 'Hey'. Este método también acepta el carácter 'H'como entrada o argumento entre paréntesis. Dado que la cadena 'Hey'comienza con la letra 'H', el método devuelve el valor booleano true.

Puede encontrar una lista de métodos de cadena integrados en la [documentación de JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/prototype) . Los desarrolladores utilizan la documentación como herramienta de referencia. Describe las palabras clave, los métodos y la sintaxis de JavaScript.

## Objetos incorporados

Además console, hay otros [objetos integrados en JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects) . Más adelante, creará sus propios [objetos](https://www.codecademy.com/resources/docs/javascript/objects) , pero por ahora estos objetos "integrados" están llenos de funciones útiles.

MathPor ejemplo, si desea realizar operaciones matemáticas más complejas que la aritmética, JavaScript tiene el objeto incorporado .

¡Lo mejor de los objetos es que tienen métodos! Llamemos al .random()método desde el objeto incorporado Math:

console.log(Math.random()); // Prints a random number between 0 and 1

En el ejemplo anterior, llamamos al .random()método agregando al nombre del objeto el operador de punto, el nombre del método y abriendo y cerrando paréntesis. Este método devuelve un número aleatorio entre 0 (inclusive) y 1 (exclusivo).

Para generar un número aleatorio entre 0 y 50, podríamos multiplicar este resultado por 50, así:

Math.random() \* 50;

El ejemplo anterior probablemente se evaluará como un decimal. Para asegurarnos de que la respuesta sea un número entero, podemos aprovechar otro Mathmétodo útil llamado Math.floor().

Math.floor()toma un número decimal y redondea hacia abajo al número entero más cercano. Puedes usar Math.floor()para redondear hacia abajo un número aleatorio como este:

Math.floor(Math.random() \* 50);

En este caso:

1. Math.random()genera un número aleatorio entre 0 y 1.
2. Luego multiplicamos ese número por 50, por lo que ahora tenemos un número entre 0 y 50.
3. Luego, Math.floor()redondea el número hacia abajo al número entero más cercano.

Si quisiera ver el número impreso en la terminal, aún necesitaría usar una console.log()declaración:

console.log(Math.floor(Math.random() \* 50)); // Prints a random whole number between 0 and 50

Para ver todas las propiedades y [métodos](https://www.codecademy.com/resources/docs/javascript/methods) del Mathobjeto, consulte [la documentación aquí](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math) .

### Instrucciones

**1 .**

Dentro de a console.log(), crea un número aleatorio con Math.random()y luego multiplícalo por 100.

Punto de control 2 superado

¿Atascado? Obtén un consejo

**2 .**

Ahora, use Math.floor()para hacer que la salida sea un número entero.

Dentro del que console.log()escribiste en el último paso, coloca el Math.random() \* 100código existente dentro del paréntesis de Math.floor().

Punto de control 3 superado

¿Atascado? Obtén un consejo

**3 .**

Encuentre un método en el [objeto JavaScriptMath](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math) que devuelva el número entero más pequeño mayor o igual a un número decimal.

Utilice este método con el número 43.8. Registre la respuesta en la consola.

Punto de control 4 superado

Pista

Utilice el .ceil()método para calcular el número entero más pequeño mayor o igual a 43.8.

**4 .**

Utilice la [documentación de JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number) para encontrar un método en el objeto integrado Numberque verifique si un número es un número entero.

Coloque el número 2017entre paréntesis del método y utilícelo console.log()para imprimir el resultado.

Punto de control 5 superado

Pista

Utilice el .isInteger()método del Numberobjeto para comprobar si 2017es un número entero.

Vea un ejemplo de cómo usar .isInteger()y registrar el resultado:

console.log(Number.isInteger(34.2)); // Prints false

## Revisar

Echemos un vistazo más a los conceptos que acabamos de aprender:

* Los datos se imprimen o registran en la consola, un panel que muestra mensajes, con console.log().
* Podemos escribir [comentarios](https://www.codecademy.com/resources/docs/javascript/comments) de una sola línea y //comentarios de varias líneas entre /\*y \*/.
* Hay 7 [tipos de datos](https://www.codecademy.com/resources/docs/javascript/data-types) fundamentales en JavaScript: [cadenas](https://www.codecademy.com/resources/docs/javascript/strings) , números, booleanos, nulos, indefinidos, símbolos y objetos.
* Los números son cualquier número sin comillas:23.8879
* Las cadenas son caracteres entre comillas simples o dobles:'Sample String'
* [Los operadores](https://www.codecademy.com/resources/docs/javascript/operators) aritméticos integrados incluyen +, -, \*, /y %.
* [Los objetos](https://www.codecademy.com/resources/docs/javascript/objects) , incluidas instancias de tipos de datos, pueden tener propiedades e información almacenada. Las propiedades se indican con un .después del nombre del objeto, por ejemplo: 'Hello'.length.
* Los objetos, incluidas instancias de tipos de datos, pueden tener [métodos](https://www.codecademy.com/resources/docs/javascript/methods) que realicen acciones. Los métodos se llaman agregando al objeto o instancia un punto, el nombre del método y paréntesis. Por ejemplo: 'hello'.toUpperCase().
* Podemos acceder a propiedades y métodos utilizando el .operador de punto.
* Los objetos integrados, incluido Math, son colecciones de métodos y propiedades que proporciona JavaScript.

Aquí hay algunos recursos más para agregar a su kit de herramientas:

* [Documentos de Codecademy: JavaScript](https://www.codecademy.com/resources/docs/javascript)
* [Espacios de trabajo de Codecademy: JavaScript](https://www.codecademy.com/workspaces/new)