

Contenido

[Semana 1 – Unidad 1 - Clasificación de las estructuras de datos 2](#_heading=h.gjdgxs)

[Semana 2 – Unidad 1 – Tipos de datos Abstractos 3](#_heading=h.30j0zll)

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

21, 22 y 24 de agosto.

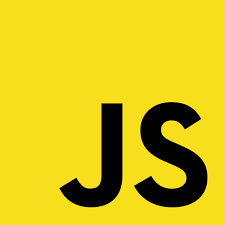
Las estructuras de datos son formas de organizar y almacenar datos en la memoria de una computadora de manera eficiente y accesible. Se pueden clasificar en varias categorías según su organización y funcionalidad.

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.

Contexto: 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.

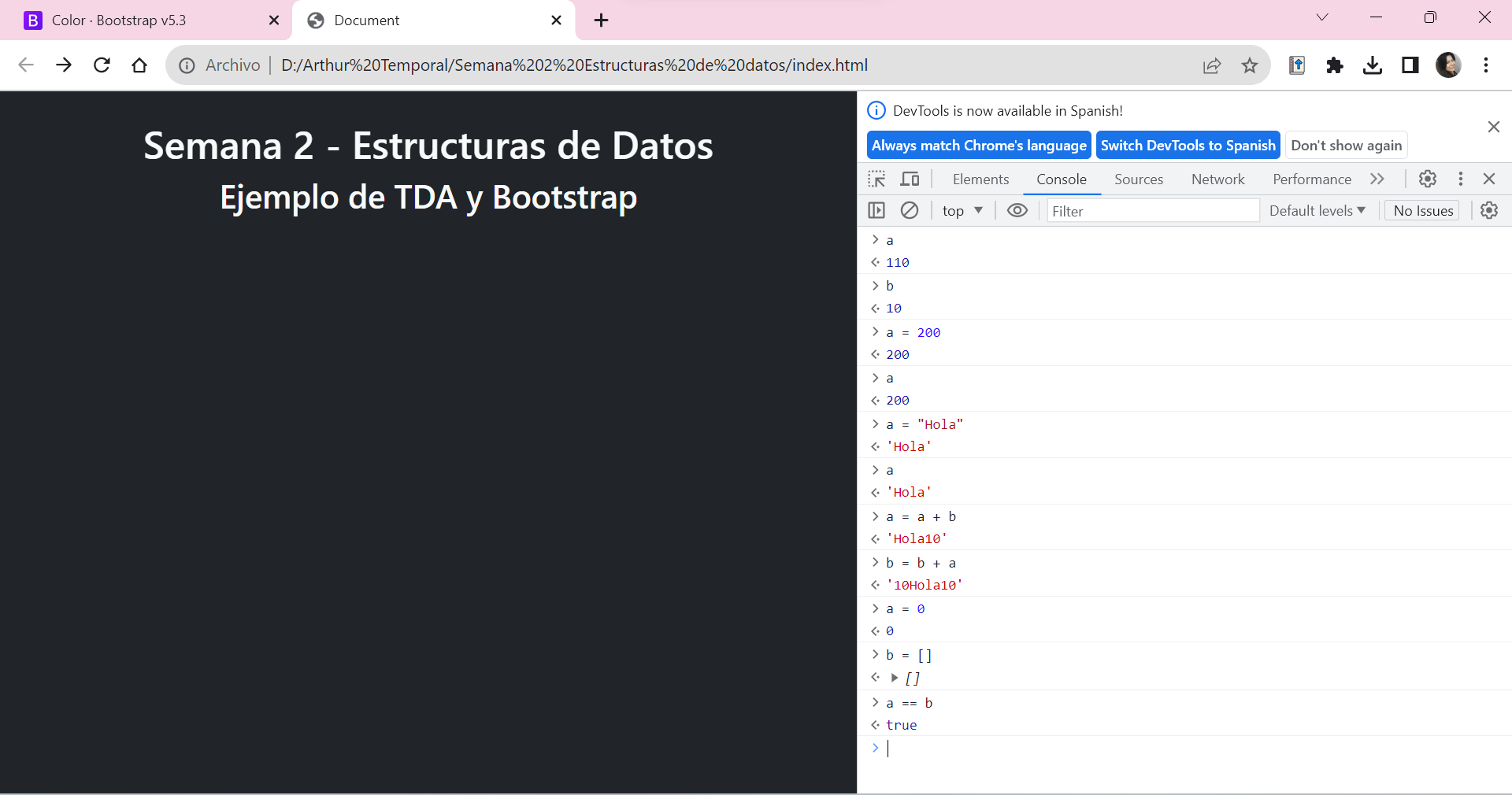
Los ADT se utilizan en una amplia gama de aplicaciones, incluyendo:

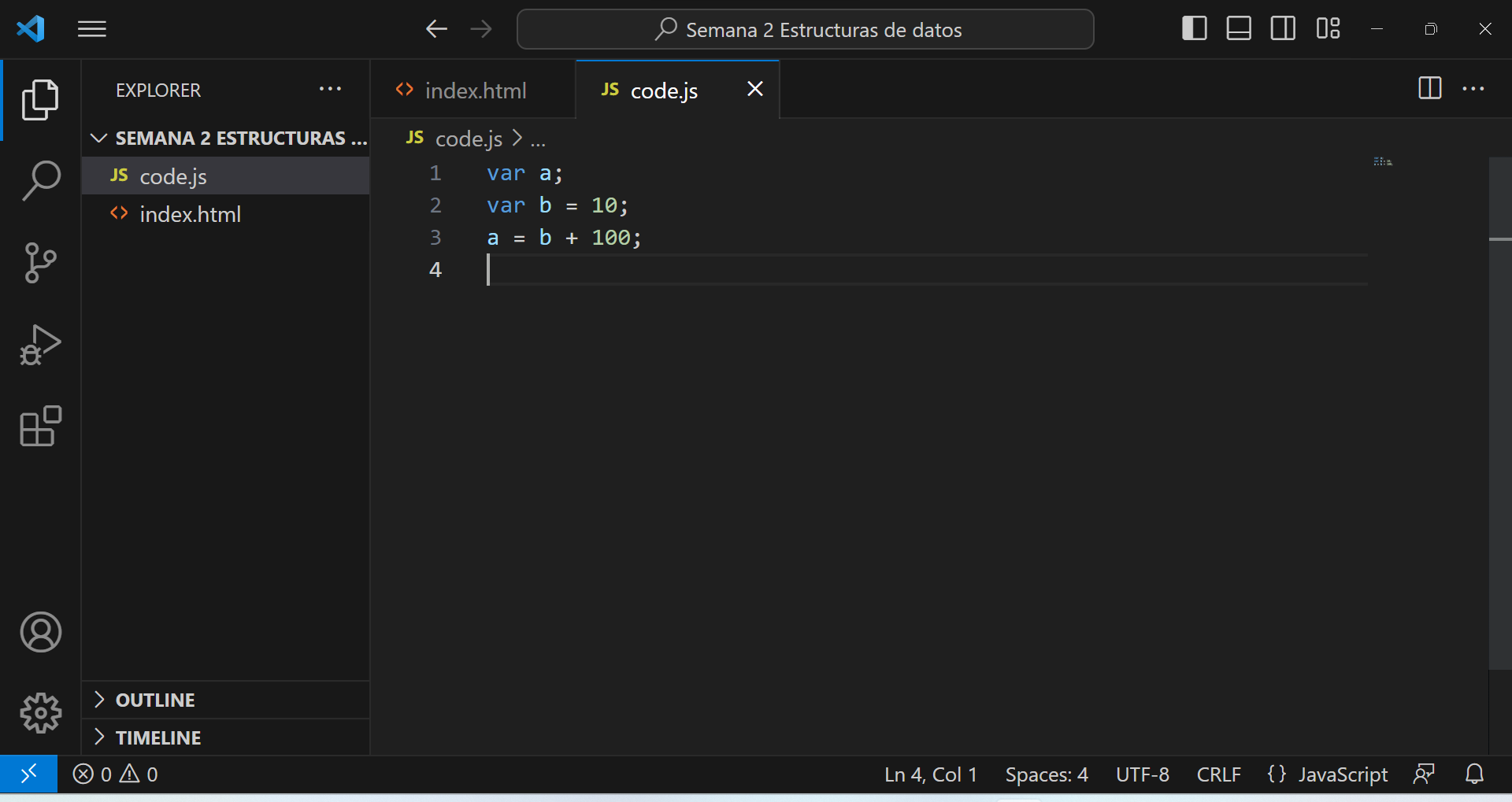
Programación de sistemas: los ADT se utilizan para representar datos abstractos en sistemas operativos, compiladores y otras aplicaciones de software de sistemas.

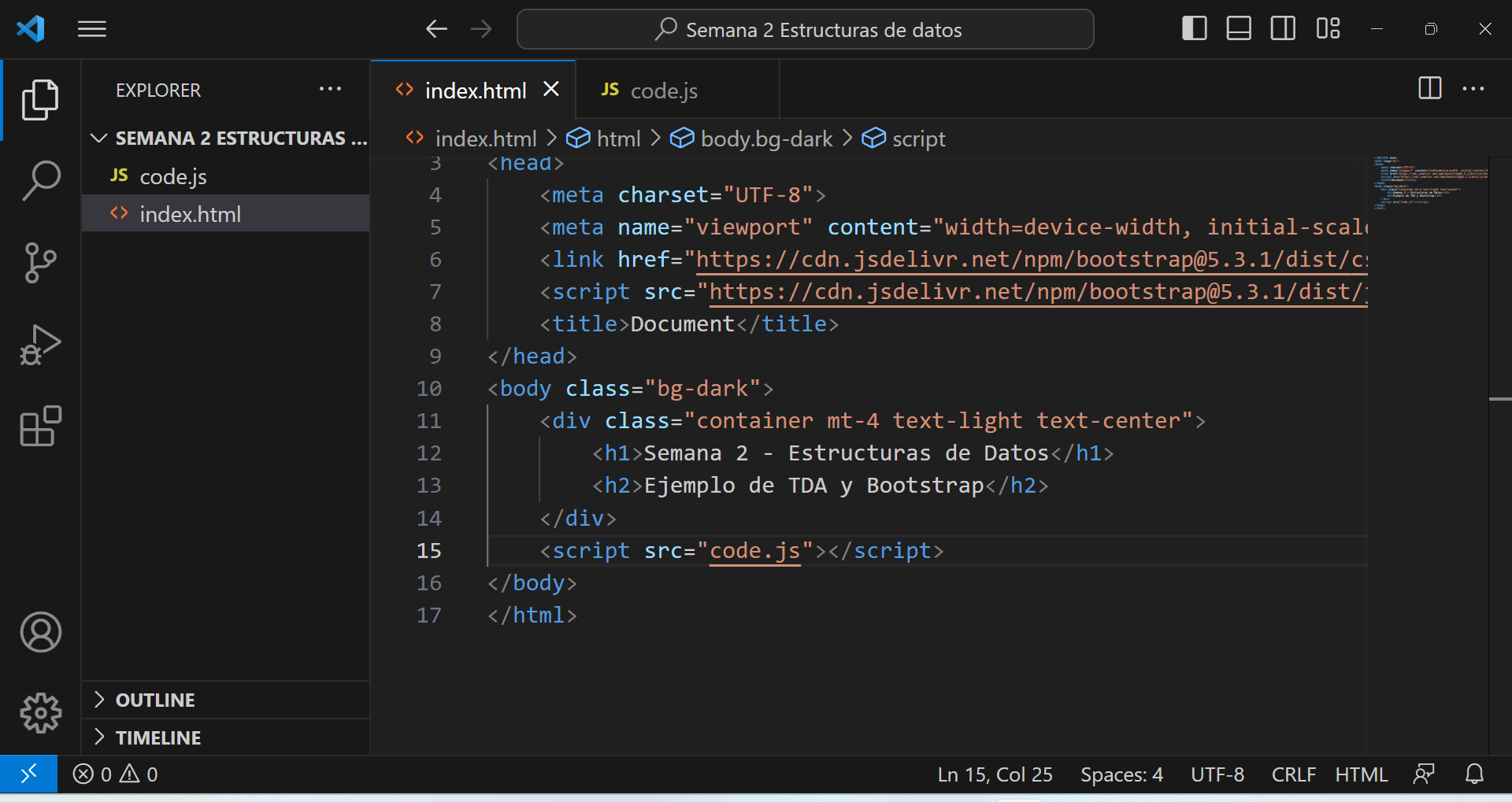
Programación de aplicaciones: los ADT se utilizan para representar datos abstractos en aplicaciones de usuario final, como procesadores de texto, hojas de cálculo y bases de datos.

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

1.3 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.

1.4 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.