


## Lista enlazada ejercicios

Más ejemplos de listas enlazadas

js

 Copiar código

```
// Lista enlazada Ejemplo new 1


// Creamos la clase nodo para cada nodo de la lista
class Node {
  // Cada nodo tiene 2 propiedades, valor y el puntero que indica el nodo que sigue
  constructor(val){
    this.val = val;
    this.next = null;
  }
}

// Creamos una clase para la lista enlazada
class listaEnlazada{
  // lleva 3 propiedades el encabezado, la cola y el tamaño
  constructor(){
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  // Primer método, MÉTODO PUSH toma un valor como parámetro y lo asigna al final de la lista
  Push(val){
    const newNode = new Node(val)
    if(!this.head){
      this.head = newNode;
      this.tail = this.head;
    }else{
      this.tail.next = newNode;
      this.tail = newNode;
    }
    this.size++;
    return this;
  }
}

const lista = new listaEnlazada();
lista.Push('a');
lista.Push('hola');
lista.Push('b');
lista.Push('mundo');
lista.Push('c');
console.log(lista);
```

js

 Copiar código

```

// Lista enlazada Ejemplo new 2
class Node {
  // Cada nodo tiene 2 propiedades, valor y el puntero que indica el nodo que sigue
  constructor(val){
    this.val = val;
    this.next = null;
  }
}
class listaEnlazada{
  // Lleva 3 propiedades el encabezado, la cola y el tamaño
  constructor(){
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  Push(val){
    const newNode = new Node(val)
    if(!this.head){
      this.head = newNode;
      this.tail = this.head;
    }else{
      this.tail.next = newNode;
      this.tail = newNode;
    }
    this.size++
    return this
  }

  // Sirve para saber la Cantidad de datos que tiene
  numDatos = () => {
    let contador = 0;
    let dato = this.head;
    if(!dato) return 0;
    else contador = 1;
    while(dato.next){
      dato = dato.next;
      contador++;
    }
    return contador;
  }

  // Este método sirve para buscar un dato en específico
  buscarDato = (valor) => {
    let dato = this.head;
    while(dato){
      if(dato.val === valor){
        return dato;
      }
      dato = dato.next;
    }
    return null;
  }

  // función para eliminar por la cola
  pop(){

```

```
if(!this.head) return undefined
let actual = this.head;
let newTail = actual;
while(actual.next){
  newTail = actual;
  actual = actual.next
}

this.tail = newTail;
this.tail.next = null;
this.size--
if(this.size === 0){
  this.head = null;
  this.tail = null;
}
return actual
}
}

const lista = new listaEnlazada();
lista.Push('a');
lista.Push('hola');
lista.Push('b');
lista.Push('mundo');
lista.Push('c');
lista.Push(23);
console.log(lista);
// Comprobando el número de datos
console.log(lista.numDatos());

//Comprobando el buscador
console.log(lista.buscarDato('c'));

// Comprobando la eliminación por la cola
console.log(lista.pop());
console.log(lista);
```

[Afianza conocimientos sobre 'Listas enlazadas'](#)


## Árbol binario

Los árboles binarios en JavaScript son estructuras de datos compuestas por nodos que pueden tener hasta dos hijos: un hijo izquierdo y un hijo derecho. Cada nodo contiene un valor y referencias a sus hijos. Estos árboles son útiles para representar relaciones jerárquicas, como estructuras de archivos o relaciones familiares. Las operaciones comunes en árboles binarios

incluyen la inserción, eliminación y búsqueda de nodos, así como recorridos para visitar los nodos en diferentes órdenes.

ISSUE 4.2024 MOD-2

js

 Copiar código

```
// Árbol binario
// Creamos la clase principal 'Nodo', con tres propiedades.
class Node {
  // 3 propiedades
  constructor(valor){
    this.valor = valor;
    this.izq = null;
    this.der = null;
  }
}

// Creamos la clase 'Árbol' con una sola propiedad
class Arbol{
  constructor(){
    this.raiz = null;
  }

  // Creamos un método a función de inserción a nuestra clase 'Arbol',
  Agregar(valor){
    const newNode = new Node(valor);
    if(this.raiz === null){
      this.raiz = newNode;
    }else{
      this.AgregarNode(this.raiz, newNode);
    }
  }

  // Creamos la función agregar a los nodos Izq o Der.
  AgregarNode(nodo, newNode){
    if(newNode.valor < nodo.valor){
      if(nodo.izq === null){
        nodo.izq = newNode;
      }else{
        this.AgregarNode(nodo.izq, newNode);
      }
    }else{
      if(nodo.der === null){
        nodo.der = newNode;
      }else{
        this.AgregarNode(nodo.der, newNode);
      }
    }
  }
}

// Función buscar
Buscar(valor){
  return this.BuscarNodo(this.raiz, valor)
}

// Función Buscar nodo
BuscarNodo(nodo, valor){
  if(nodo === null || nodo.valor === valor ){
    return nodo;
  }else if(valor < nodo.valor){
    return this.BuscarNodo(nodo.izq, valor)
  }else{
    return this.BuscarNodo(nodo.der, valor)
  }
}
```

```
    }  
  }  
}  
  
// comprobando la función Agregar()  
const newArbol = new Arbol();  
newArbol.Agregar("D");  
console.log(newArbol);  
newArbol.Agregar("B");  
console.log(newArbol);  
newArbol.Agregar("C");  
console.log(newArbol);  
newArbol.Agregar("a");  
console.log(newArbol);  
newArbol.Agregar("A");  
console.log(newArbol);  
newArbol.Agregar("c");  
console.log(newArbol);  
newArbol.Agregar("b");  
console.log(newArbol);  
  
// Comprobando la función BuscarNode()  
console.log(newArbol.Buscar("A"));  
console.log(newArbol.Buscar("Z"));
```

[Afianza conocimientos sobre 'Árbol binarios'](#)