


## Pilas Ejercicios

Lorem

js

 Copiar código

```
class stack {
  constructor() {
    this.head = null;
  }

  push(element) {
    const node = {
      data: element,
      next: this.head
    };
    this.head = node;
  }

  pop() {
    const node = this.head;
    this.head = node.next;
    return node.data;
  }
}

// Crear una nueva instancia de stack
const myStack = new stack();

// Agregar elementos a la pila
myStack.push('A');
myStack.push('B');
myStack.push('C');

// Mostrar la pila después de agregar elementos
console.log(myStack);


// Eliminar un elemento de la pila y mostrarlo
console.log(myStack.pop());

// Mostrar la pila después de eliminar un elemento
console.log(myStack);
```

## Colas

Una cola es una estructura de datos que sigue el principio "primero en entrar, primero en salir" (FIFO, por sus siglas en inglés). Puedes implementar una cola utilizando un array o creando una clase específica para ello. Las colas son útiles para gestionar tareas en un orden determinado, como el procesamiento de solicitudes en un servidor o el manejo de eventos en un sistema de eventos. Las operaciones básicas en una cola incluyen agregar elementos al final (enqueue) y eliminar elementos del principio (dequeue), mientras que otros métodos comunes son verificar si la cola está vacía y obtener su tamaño.

js

 Copiar código

```
class Queue{
  constructor(){
    this.queue = [];
  }

  enqueue(element){ // Agrega un elemento al final de la cola.
    this.queue.push(element);
    return this.queue;
  }

  dequeue(){ // Elimina y devuelve el elemento en el frente de la cola.
    return this.queue.shift();
  }

  peek(){ // Devuelve el elemento en el frente de la cola sin eliminarlo.
    return this.queue[0];
  }

  size(){
    return this.queue.length;
  }

  isEmpty(){ // Verifica si la cola está vacía.
    return this.queue.length === 0;
  }

  print(){
    return this.queue;
  }
}
```


```
const newCola = new Queue();
```

05.04.2024 MOD-2

```
newCola.enqueue(1);
newCola.enqueue(2);
newCola.enqueue(3);

console.log(newCola);
console.log(newCola.dequeue());
console.log(newCola);
console.log(newCola.peek());
console.log(newCola.size());
console.log(newCola.isEmpty());
console.log(newCola.print());
```

js

 Copiar código

```
// Ejemplo 2 - Cola con prioridad
let queue = [];
```

```
function enqueue(item, priority) {
  let node = {
    data: item,
    priority: priority,
    next: null
  };
}
```

```
if (queue.length === 0) {
  queue.head = node;
} else {
  let current = queue.head;
  let previous;
```

```
  while (current && current.priority >= node.priority) {
    previous = current;
    current = current.next;
  }
```

```
  if (!previous) {
    node.next = queue.head;
    queue.head = node;
  } else {
    node.next = current;
    previous.next = node;
  }
}
```

```
queue.length++;
}
```

```
function dequeue() {
  let node = queue.head;
  queue.head = node.next;
  queue.length--;
  return node.data;
}
```


```
enqueue('A', 1);
enqueue('B', 2);
enqueue('C', 3);
```

```
enqueue('D', 4);
enqueue('E', 5);
```

05.04.2024 MOD-2

```
console.log(dequeue()); // A
console.log(dequeue()); // B
console.log(dequeue()); // C
console.log(dequeue()); // D
console.log(dequeue()); // E
```

js

 Copiar código

```
// Ejercicio 3 - Usando una class
// Creamos la clase nodo.
class Node{
  constructor(value){ // Cada nodo tiene dos propiedades, valor y puntero que indica el nodo que
    this.value = value;
    this.next = null;
  }
}

// Creamos la clase para la cola
class Cola{
  constructor(){ // Tres propiedades 'first', 'last', size
    this.first = null;
    this.last = null;
    this.size = 0;
  }

  // creamos la función para agregar los elementos, recibe el valor y lo agrega al final
  enqueue(val){
    let newNode = new Node(val);

    if(!this.first){
      this.first = newNode;
      this.last = newNode;
    }else{
      this.last.next = newNode;
      this.last = newNode;
    }
    return ++ this.size;
  }

  // Creamos la función que elimina el valor inicial de la cola
  dequeue(){
    if(!this.first) return null;
    let tem = this.first;
    if(this.first === this.last){
      this.last = null;
    }

    this.first = this.first.next;
    this.size --
    return tem.value;
  }
}

const cola = new Cola;
cola.enqueue("Primero");
```

```
cola.enqueue("Segundo");
cola.enqueue("Tercero");
cola.enqueue("Cuarto");
cola.enqueue("Quinto");
console.log(cola)

console.log(cola.first);
console.log(cola.last);
console.log(cola.size);
console.log(cola.dequeue());

console.log(cola.first);
console.log(cola.last);
console.log(cola.size);
console.log(cola.dequeue());

console.log(cola.first);
console.log(cola.last);
console.log(cola.size);
console.log(cola.dequeue());

console.log(cola.first);
console.log(cola.last);
console.log(cola.size);
console.log(cola.dequeue());

console.log(cola.first);
console.log(cola.last);
console.log(cola.size);
console.log(cola.dequeue());

console.log(cola.first);
console.log(cola.last);
console.log(cola.size);
console.log(cola.dequeue());
```

[Afianza conocimientos sobre 'colas'](#)


## Listas enlazadas

Las listas enlazadas en JavaScript son estructuras de datos que consisten en nodos conectados mediante referencias. Cada nodo almacena un valor y una referencia al siguiente nodo. Son útiles para operaciones de inserción, eliminación y búsqueda, y son flexibles en su aplicación,

desde pilas hasta listas ordenadas, debido a su eficiencia en la inserción y eliminación de elementos.

CS101-2024 MOD-2

js

 Copiar código

```
// Creamos la clase nodo
class Node {
  constructor(data) {
    this.data = data;
    this.next = null;
  }
}

// Creamos la clase para la lista enlazada
class LinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }
  insertAtHead(data) {
    const newNode = new Node(data);
    newNode.next = this.head;
    this.head = newNode;
  }

  insertAtTail(data) {
    const newNode = new Node(data);
    if (!this.head) {
      this.head = newNode;
    }
    if (this.tail) {
      this.tail.next = newNode;
    }
    this.tail = newNode;
  }
}

const lista = new LinkedList;
lista.insertAtHead("Primer");
lista.insertAtHead("Segundo");
console.log(lista);


lista.insertAtTail("Tercero");
lista.insertAtTail("Cuarto");
console.log(lista)
```

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

## Tarea Lista enlazada

Corre el anterior ejercicio sobre lista enlazada completo, este lo puedes encontrar [aquí](#)

js

 Copiar código

```
// Cada 'Nodo' tendrá dos propiedades: 'datos' y 'siguiente'.
class Node {
  constructor(data) {
    this.data = data; // almacena los datos de ese nodo
    this.next = null; // almacena la referencia al siguiente nodo de la lista
  }
}

// Definimos la clase que representa nuestra lista enlazada
class LinkedList {
  constructor() {
    this.head = null; // almacena el primer nodo de la lista
    this.tail = null; // almacena el último nodo de la lista.
  }

  // Creamos la función que ingrese datos por la cabeza
  insertAtHead(data) {
    const newNode = new Node(data);
    newNode.next = this.head;
    this.head = newNode;
  }

  // Creamos la función que ingrese datos por la cola
  insertAtTail(data) {
    const newNode = new Node(data);
    if (!this.head) {
      this.head = newNode;
    }
    if (this.tail) {
      this.tail.next = newNode;
    }
    this.tail = newNode;
  }

  // Creamos la función que elimine los datos por la cabeza
  deleteFromHead() {
    if (!this.head) {
      return;
    }
    this.head = this.head.next;
  }

  // Creamos la función que elimine los datos por la cola
  deleteFromTail() {
    if (!this.tail) {
      return;
    }
    if (!this.head.next) {
      this.head = null;
    }
  }
}
```

```

        this.tail = null;
        return;
    }
    let current = this.head;
    while (current.next !== this.tail) {
        if (!current || !current.next) { // Aquí se produce el error
            return;
        }
        current = current.next;
    }
    current.next = null;
    this.tail = current;
}

```

// Creamos una función que busque datos

```

search(data) {
    let current = this.head;
    while (current) {
        if (current.data === data) {
            return current;
        }
        current = current.next;
    }
    return null;
}

```

// Función que recorre la lista enlazada

```

traverse() {
    let current = this.head;
    while (current) {
        console.log(current.data);
        current = current.next;
    }
}

```

// Función que invierte la lista enlazada

```

reverse() {
    let current = this.head;
    let previous = null;
    let next = null;
    while (current) {
        next = current.next;
        current.next = previous;
        previous = current;
        current = next;
    }
    this.head = previous;
}

```

// Función para ordenar la lista

```

sort() {
    let current = this.head;
    let minimum = null;
    let next = null;
    while (current) {
        minimum = current;
        next = current.next;
        while (next) {
            if (next.data < minimum.data) {
                minimum = next;
            }
            next = next.next;
        }
        if (minimum !== current) {
            const temp = current.data;

```



```
        current.data = minimum.data;
        minimum.data = temp;
    }
    current = current.next;
}
}

// Crear una nueva lista enlazada
const lista = new LinkedList();

// Insertar datos por la cabeza
lista.insertAtHead(1);
lista.insertAtHead(2);
lista.insertAtHead(3);
lista.insertAtHead(4);

// Imprimir la lista enlazada después de insertar datos por la cabeza
console.log("Lista después de insertar datos por la cabeza:");
lista.traverse();

// Insertar datos por la cola
lista.insertAtTail(5);
lista.insertAtTail(6);

// Imprimir la lista enlazada después de insertar datos por la cola
console.log("Lista después de insertar datos por la cola:");
lista.traverse();

// Eliminar datos por la cabeza
lista.deleteFromHead();

// Imprimir la lista enlazada después de eliminar datos por la cabeza
console.log("Lista después de eliminar datos por la cabeza:");
lista.traverse();

// Eliminar datos por la cola
lista.deleteFromTail();

// Imprimir la lista enlazada después de eliminar datos por la cola
console.log("Lista después de eliminar datos por la cola:");
lista.traverse();

// Buscar un dato en la lista enlazada
console.log("Buscar 2 en la lista:", lista.search(2));

// Invertir la lista enlazada
lista.reverse();
console.log("Lista después de invertir:");
lista.traverse();

// Ordenar la lista enlazada
lista.sort();
console.log("Lista después de ordenar:");
lista.traverse();
```

**NOTA**

El código original de la página presenta un error al ingresar datos por la cola, los toma como 'NULL'. Se ha modificado pero no esta funcionando el

agregar datos por la cola. Para que funcione de manera optima se debe dividir por método, es decir crear un nuevo nodo por cada método o función.

02.07.2024 MOD-2

Bryan Hernández | Telento Tech DWFSV2-42 | 2024

—