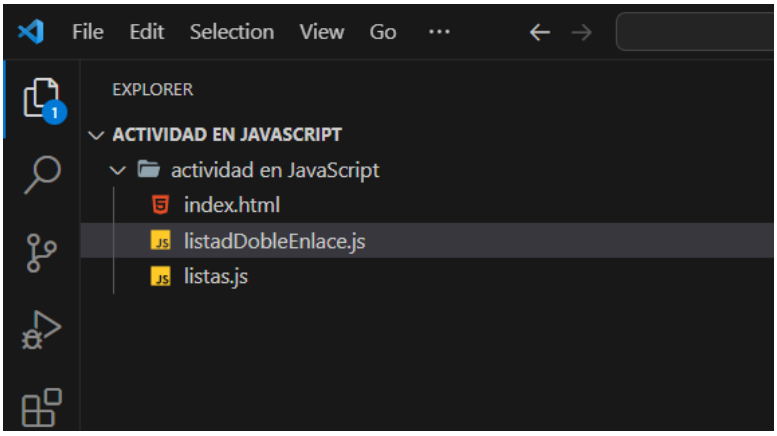
LISTAS SIMPLES EN JAVASCRIPT

Iniciamos con la creación de una carpeta que contenga un documento HTML y 2 JavaScript



Para el HTML se realiza el enlace con JavaScript

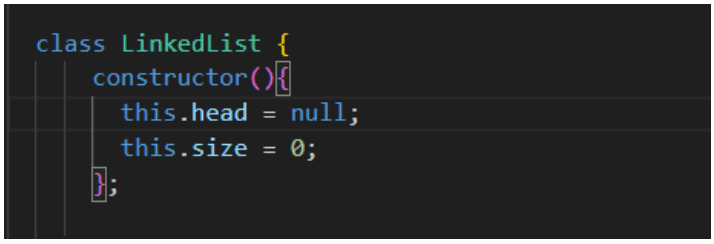
```html
<!DOCTYPE html>
<html>
    <head>

    </head>
    <body>
        ejercicios



        <script src="listas.js"></script> >

    </body>

</html>
```

Para el JavaScript:

1. Primeramente, se creará La clase Node representa un nodo de la lista enlazada. Tiene dos propiedades: data, que contiene el valor del nodo, y next, que contiene la referencia al siguiente nodo en la secuencia.

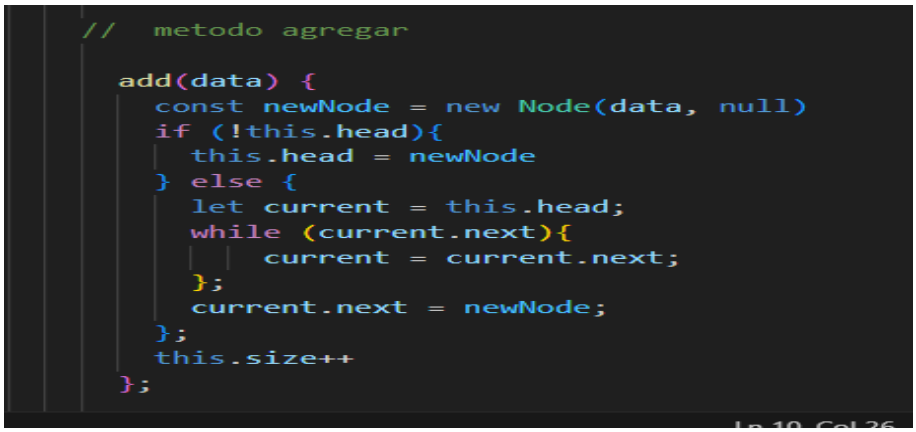
```javascript
class Node {
    constructor(data, next){
      this.data = data;
      this.next = next;
    };
```

2. Creación La clase LinkedList representa la lista enlazada. Tiene tres propiedades: head, que es una referencia al primer nodo de la lista; size, que es el número de nodos en la lista; y los métodos para manipular la lista.



```
class LinkedList {
    constructor(){
      this.head = null;
      this.size = 0;
    };
```

3. Luego de esto se crea el método agregar, El método add agrega un nuevo nodo al final de la lista. El método crea un nuevo nodo con el valor especificado y lo agrega a la lista. Si la lista está vacía, el nuevo nodo se convierte en el primer nodo de la lista, en este método se puede validar la implementación de if, else y while que hacen que se cumpla la función agregar.



```
    //   metodo agregar

      add(data) {
        const newNode = new Node(data, null)
        if (!this.head){
          this.head = newNode
        } else {
          let current = this.head;
          while (current.next){
              current = current.next;
          };
          current.next = newNode;
        };
        this.size++
      };
```

4. Lo siguiente es agregar en posiciones específicas, El método insertAt agrega un nuevo nodo en una posición específica de la lista. El método crea un nuevo nodo con el valor especificado y lo inserta en la posición especificada. Si la posición es 0, el nuevo nodo se convierte en el primer nodo de la lista. Como la anterior esta nos muestra el uso de if y else, pero se anexa un for que recorra las posiciones hasta llegar a la deseada.

```
// metodo agregar en posición especifica

  insertAt(data,index){
    if(index < 0 || index > this.size){
      return null
    };

    const newNode = new Node(data);
    let current = this.head;
    let previous;

    if (index === 0) {
      newNode.next = current;
      this.head = newNode;
    } else {
        for (let i = 0; i < index; i++){
          previous = current;
          current = current.next;
        };
        newNode.next = current;
      previous.next = newNode;

    };
    this.size++;
  };
```

5. Sigue la creación El método removeData que elimina el primer nodo de la lista que contiene el valor especificado. Si se encuentra el valor, el método elimina el nodo de la lista y devuelve su valor, aquí va a evaluar el valor que se encuentre en primera posición y se va a eliminar por medio de los diferentes condicionales y el ciclo while.

```
// metodo eliminar elemento o nodo

removeData(data){
  let current = this.head;
  let previous = null;

  while(current != null){
    if (current.data === data){
      if (!previous){
        this.head = current.next;
      } else{
        previous.next = current.next;
      };
      this.size--;
      return current.data;
    };
    previous = current;
    current = current.next;
  }
```

6. El método removeFrom elimina el nodo en una posición específica de la lista. Si la posición es 0, el primer nodo de la lista se elimina. Aquí en este método se pasa un parámetro donde se va a evaluar que posición se desea y el ciclo se recorra hasta llegar a ella con la finalidad de eliminar la posición deseada.

```
removeFrom(index){
    if  (index < 0 || index > this.size){
      return null
    };

    let current = this.head;
    let previous = null;

    if (index === 0){
      this.head = current.next;
    } else {
      for (let i = 0; i < index; i++){
        previous = current;
        current = current.next;
      };
      previous.next = current.next;
    };
    this.size--;
    return current.data;
};
```

7. El método print devuelve una cadena que representa los valores de los nodos en la lista. Este método lo que hace es mostrar por medio de un while lo que se encuentra en la lista, lo trae y da como resultada la lista de todos los nodos.

```
print(){
  if(!this.size){
    return null
  };
  let current = this.head;
  let result = '';
  while (current) {
    result += current.data += '';
    current = current.next;
  };
  result += 'X';
  return result;
};
```

8. El método `isEmpty` devuelve verdadero si la lista está vacía. Este método evalúa si esta lista se encuentra 0 y por medio del if va a retornar que esta vacio o que no lo esta.

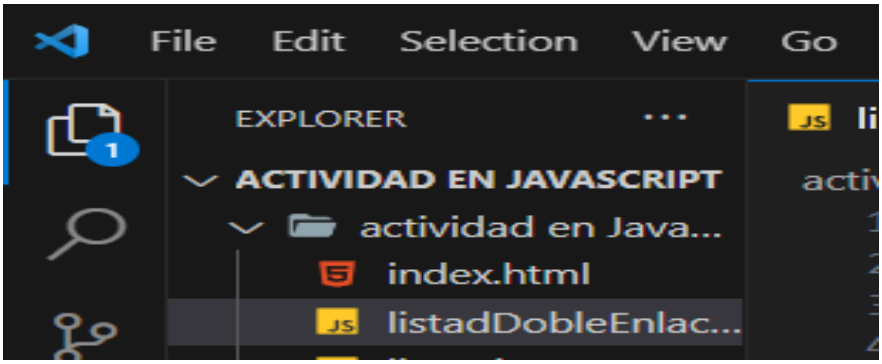
```
//  metodo que valida si la lista esta vacia
isEmpty(){
  if (this.size === 0){
    return true;
  }else{
    return false;
  }
};
```

9. Por último, El método getSize devuelve el número de nodos en la lista, este, método cuenta el tamaño de la lista y de vuelve cuantos elementos tiene esta.

```
// metodo para obtener el tamaño de la lista

  getSize(){

    return  this.size;
  };
};
```

LISTAS DOBLES ENLAZADAS

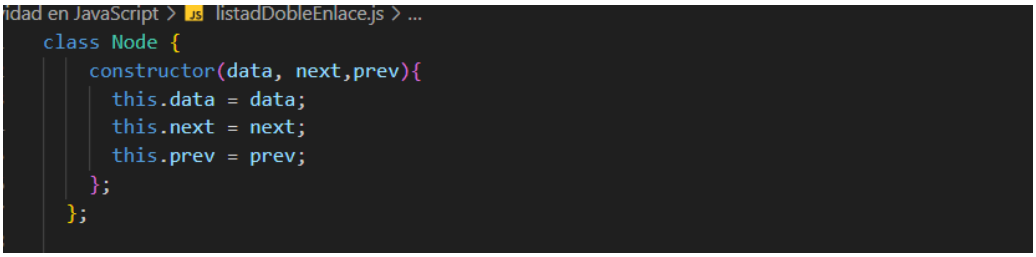
1. Para la creación de listas dobles se crea otro archivo JavaScript llamado listasDoblesEnlace.js



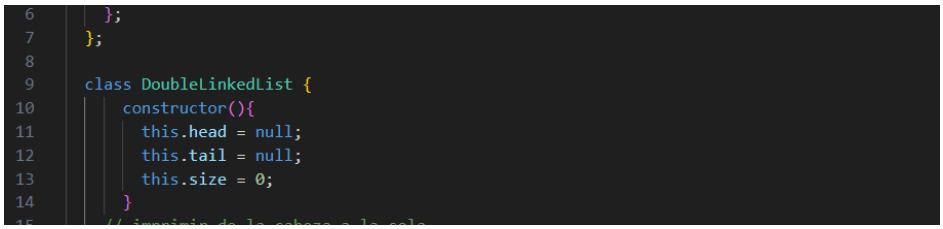
Para el HTML se realiza el enlace con JavaScript

```
    </head>
    <body>
        ejercicios



        <script src="listasDobleEnlace.js"></script> >
    </body>
</html>
```

2. Primeramente, la creación de La clase Node donde se representa un nodo de la lista doblemente enlazada. Tiene tres propiedades: data, que contiene el valor del nodo; next, que contiene la referencia al siguiente nodo en la secuencia; y prev, que contiene la referencia al nodo anterior en la secuencia.

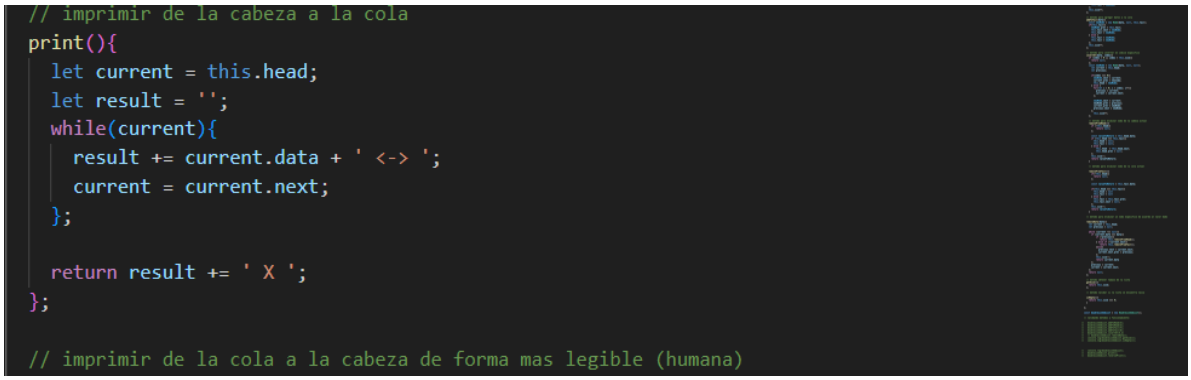


3. En segundo lugar, está la creación de La clase DoubleLinkedList representa la lista doblemente enlazada. Tiene tres propiedades: head, que es una referencia al primer nodo de la lista; tail, que es una referencia al último nodo de la lista; size, que es el número de nodos en la lista; y los métodos para manipular la lista.



4. De aquí en adelante se repite el proceso de creación de métodos

- El método PRINT devuelve una cadena que representa los valores de los nodos en la lista desde la cabeza hasta la cola.



- El método REVERSEPRINT devuelve una cadena que representa los valores de los nodos en la lista desde la cola hasta la cabeza.

```
// imprimir de la cola a la cabeza de forma mas legible (humana)
reversePrint(){
  let current = this.tail;
  let result = '';
  while(current) {
    result += current.data + ' <-> ';
    current = current.prev;
  };
  return result += ' X ';
};
```

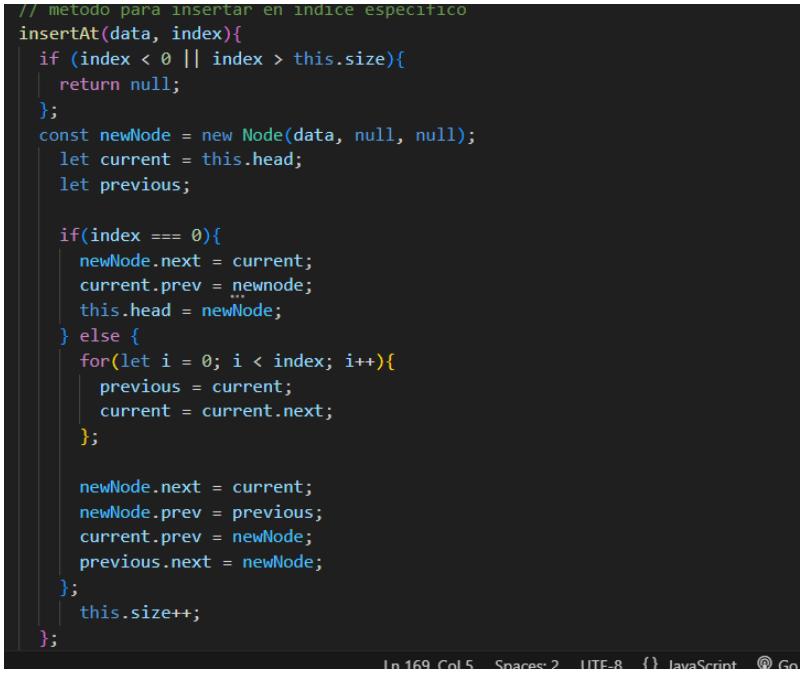
- El método ADDTOHEAD agrega un nuevo nodo al principio de la lista. El método crea un nuevo nodo con el valor especificado y lo agrega al principio de la lista.

```
addToHead(data){
  const newNode = new Node(data, this.head, null);

  if(this.head){
    newNode.next = this.head;
    this.head.prev = newNode;
    this.head = newNode;
  }else{
    this.head = newNode;
    this.tail = newNode;
  };
  this.size++;
};
```

- El método ADDTOTAIL agrega un nuevo nodo al final de la lista. El método crea un nuevo nodo con el valor especificado y lo agrega al final de la lista.

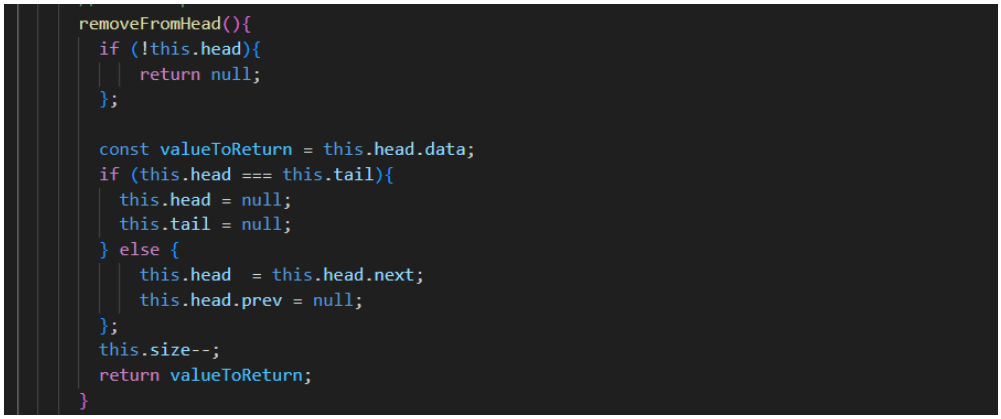
```
};

// metodo para agregar datos a la cola
addToTail(data){
  const newNode = new Node(data, null, this.tail);
  if(this.tail){
    newNode.prev = this.tail;
    this.tail.next = newNode;
    this.tail = newNode;
  } else {
    this.tail = newNode;
    this.tail = newNode;
  };
  this.size++;
};
```

- El método INSERTAT agrega un nuevo nodo en una posición específica de la lista. El método crea un nuevo nodo con el valor especificado y lo inserta en la posición especificada.



```
// metodo para insertar en indice especifico
insertAt(data, index){
  if (index < 0 || index > this.size){
    return null;
  };
  const newNode = new Node(data, null, null);
    let current = this.head;
    let previous;

    if(index === 0){
      newNode.next = current;
      current.prev = newnode;
      this.head = newNode;
    } else {
      for(let i = 0; i < index; i++){
        previous = current;
        current = current.next;
      };

      newNode.next = current;
      newNode.prev = previous;
      current.prev = newNode;
      previous.next = newNode;
    };
      this.size++;
  };
```

- El método REMOVEFROMHEAD elimina el primer nodo de la lista y devuelve su valor.



```
    removeFromHead(){
      if (!this.head){
          return null;
      };

      const valueToReturn = this.head.data;
      if (this.head === this.tail){
        this.head = null;
        this.tail = null;
      } else {
          this.head  = this.head.next;
          this.head.prev = null;
      };
      this.size--;
      return valueToReturn;
    }
```

- El método REMOVEFROMTAIL elimina el último nodo de la lista y devuelve su valor.

```
removeFromTail(){
  if(!this.head){
    return null;
  };

  const valueToReturn = this.tail.data;

  if(this.head === this.tail){
    this.head = null
    this.tail = null
  } else {
    this.tail = this.tail.prev;
    this.tail.next = null;
  };
  this.size--;
  return valueToReturn;
}

// metodo para eliminar un nodo especifico de acuerdo al
```

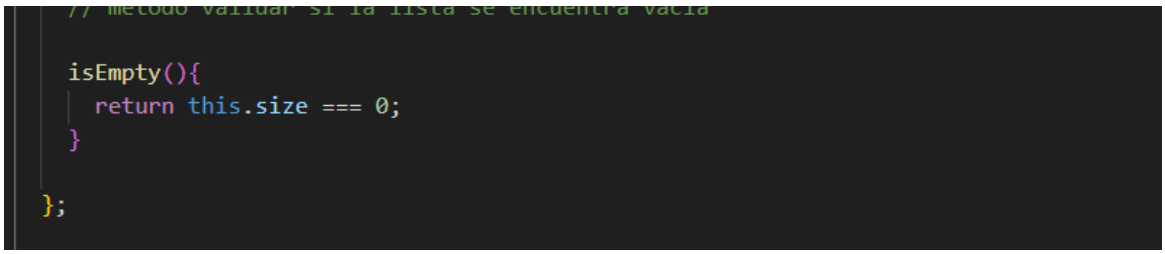
- El método REMOVEDATA elimina el primer nodo de la lista que contiene el valor especificado. Si se encuentra el valor, el método elimina el nodo de la lista y devuelve su valor.

```
34
35  removeData(data){
36    let current = this.head;
37    let previous = null;
38
39    while (current !== null){
40      if (current.data === data){
41          if (!previous){
42              return this.removeFromHead();
43          } else if (!current.next){
44              return this.removeFromTail();
45          }else{
46            previous.next = current.next;
47            current.next.prev = previous;
48          };
49          this.size--;
50          return current.data
51      };
52      previous = current;
53      current = current.next;
54    };
55    return null;
56  };
57
```

- El método GETSIZE devuelve el número de nodos en la lista.

```
// metodo obtener tamaño de la lista
getSize(){
  return this.size;
};
```

- El método ISEMPTY devuelve verdadero si la lista está vacía.



5. Finalmente, ya solo es correr el proyecto y verificar su funcionalidad.