Arrays Methods

concat (array0?, value1?, ... , valueN?)

The concat() method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

```
const letters = ['a', 'b', 'c'];
const numbers = [1, 2, 3];
letters.concat(numbers);
// result in ['a', 'b', 'c', 1, 2, 3]}
```

splice(start, deleteCount?, item1?, item2?, itemN?)

The splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements in place. To access part of an array without modifying it, see slice().

```
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
//output: ["Ja n", " Feb ", " Mar ch", " Apr il", " Jun - e"]
```

slice (start?, end?)

The slice() method returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array. The original array will not be modified.

```
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];
animals.slice(2, 4)
// output: ["ca mel ", " duc - k"]
```

Arrays Methods (cont)

shit ()

The shift() method removes the first element from an array and returns that removed element. This method changes the length of the array.

unshift (element0,element1)

The unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.

sort ((firstEl, secondEl) => { ... })

The sort() method sorts the elements of an array in place and returns the sorted array. The default sort order is ascending, built upon converting the elements into strings

If compareFunction(a, b) returns a value **> than 0**, sort b before a.

If compareFunction(a, b) returns a value **< than 0**, sort a before b.

If compareFunction(a, b) **returns 0**, a and b are considered equal.

filter ((element, index, array) => { ... })

The filter() method creates a new array with all elements that pass the test implemented by the provided function. callbackFn Function is a predicate, to test each element of the array. Return a value that coerces to true to keep the element, or to false otherwise.

```
let filtered = [12, 5, 8, 130, 44].fi lte r(value => value>=10)
// filtered is [12, 130, 44]
```

Arrays Methods (cont)

find ((element, index, array) => { ... })

The find() method returns the value of the first element in the provided array that satisfies the provided testing function. If no values satisfy the testing function, undefined is returned.

If you need the index of the found element in the array, use findIndex().

```
const array1 = [5, 12, 8, 130, 44];
const found = array1.fi nd( - element => element > 10);
// expected output: 12
```

join(separator?)

The join() method creates and returns a new string by concatenating all of the elements in an array (or an array-like object), separated by commas or a specified separator string. If the array has only one item, then that item will be returned without using the separator.

```
var a = ['Wind', 'Water', 'Fire'];
a.join();// 'Wind, Wat er, - Fire'
a.join(''); // 'WindW ate - rFire'
```

Linked List

```
class Node {
constr uct or( ele ment)
        {
                thi s.e lement = element;
                thi s.next = null;
                thi s.p revious = null;
        }
}
class LinkedList {
```

Linked List (cont)

```
        con str uctor()
       {
                thi s.head =
null; //root Node
                thi s.tail =
null; // last element of the
list
                thi s.size = 0;
       }

    //Adds element to the
begining of the list. Similar to
Array.u nshift
    //O(1)
      add Hea d(v alue) {
        const newNode = new
Node(v alue);
        new Nod e.next =
this.head;
       if (this.h ead) {
            thi s.h ead.pr -
evious = newNode;
       } else {
            thi s.tail =
newNode;
       }
        thi s.head = newNode; //
update head
        thi s.size += 1;
        return newNode;
   }
        //Adds element to the
end of the list (tail)
        //O(1)
        add Tai l(v alue) {
        const newNode = new
Node(v alue);
       if (this.h ead) {
            new Nod e.p revious
= this.tail;
            thi s.t ail.next =
newNode;
            thi s.tail =
newNode;
```

Linked List (cont)

```
       } else {
            thi s.head =
newNode;
            thi s.tail =
newNode;
       }
        thi s.size += 1;
        return newNode;
   }

       // insert element at the
position index of the list
        ins ert At( ele ment,
index)
       {
if (index < 0 || index >
this.size)
return consol e.l og( " Please
enter a valid index." );
else {
// creates a new node
var node = new Node(e lem ent);
var curr, prev;
curr = this.head;
// add the element to the
// first index
if (index == 0) {
node.next = this.head;
this.head = node;
} else {
curr = this.head;
var it = 0;
// iterate over the list to find
// the position to insert
while (it < index) {
```

Linked List (cont)

```
it++;
prev = curr;
curr = curr.next;
}
// adding an element
node.next = curr;
prev.next = node;
}
this.s ize++;
}
         }
    //R emoves element from the
start of the list (head/ root).
    //R untime O(1)
      rem ove Head() {
        const head = this.head;
       if (head) {
            thi s.head=
head.next;
           if (this.h ead) {
                thi s.h ead.pr -
evious = null;
           } else {
                thi s.tail =
null;
           }
            thi s.size -= 1;
       }
        return head && head.v -
alue;
   }
      //R emoves element to the
end of the list. Similar to
Array.pop
```

Linked List (cont)

```
      //R untime: O(1)
    rem ove Tail() {
        const tail = this.tail;
       if (tail) {
            thi s.tail =
tail.p rev ious;
           if (this.t ail) {
                thi s.t ail.next
= null;
           } else {
                thi s.head =
null;
           }
            thi s.size -= 1;
       }
        return tail && tail.v -
alue;
   }
     // removes an element from
the specified location
      rem ove Fro m(i ndex)
     {
       if (index < 0 || index >=
this.size)
                return consol -
e.l og( " Please Enter a valid
index");
       else {
               var curr, prev,
it = 0;
               curr = this.head;
               prev = curr;

               // deleting first
element
               if (index === 0)
{
                        thi -
s.head = curr.next;
               } else {
                       //
iterate over the list to the
                       //
position to removce an element
```

Linked List (cont)

```
                           while
(it < index) {
                                   -
it++;
                               p
rev = curr;
                               c
urr = curr.next;
                           }

                           // remove
the element
                            pre -
v.next = curr.next;
                }
                 thi s.s ize--;

                // return the
remove element
                 return curr.e -
lement;
         }
     }
       // removes a given
element from the list
        rem ove Ele men t(e -
lement)
       {
var current = this.head;
var prev = null;
// iterate over the list
while (current != null) {
// comparing element with
current
// element if found then remove
the
// and return true
if (curre nt.e lement ===
element) {
if (prev == null) {
this.head = curren t.next;
} else {
prev.next = curren t.next;
}
```

Linked List (cont)

```
this.s ize--;
return curren t.e lement;
}
prev = current;
current = curren t.next;
}
return -1;
       }
       // finds the index of
element
        ind exO f(e lement)
     {
       var count = 0;
       var current = this.head;

       // iterae over the list
        while (current != null)
{
               // compare each
element of the list
               // with given
element
               if (curre nt.e -
lement === element)
                        return
count;
                cou nt++;
                current =
curren t.next;
       }

       // not found
        return -1;
     }

        isE mpty()
     {
        return this.size == 0;
     }
```

Linked List (cont)

```
}
```

Queue with Array

```
//FIFO(First in First Out)
class Queue {
       // Array is used to
implement Queue
        con str uctor() {
                thi s.items =
[];
       }
        enq ueu e(e lem ent){
//O(1)
            thi s.i tem s.p -
ush (el ement);
       }
          //R emoves an element
from the front of a queue
(items[0])
        deq ueue(){ // O(n)
            return this.i tem -
s.s hift();
       }
        peek(){
       // return the front
element from the queue
        return this.i tem s[0];
       }
        isE mpty(){
         // return true if queue
is empty
        return this.i tem s.l -
ength == 0;
     }
}
```

Queue with Linked List

```
class Queue {
    con str uctor() {
        thi s.items = new
Linked List();
   }
      //Add element to the queue
      //R untime: O(1)
    enq ueu e(item) {
        thi s.i tem s.a ddT -
ail (item);
        return this;
   }
  //R emove element from the
queue
  //R untime: O(1)
    deq ueue() {
        return this.i tem s.r -
emo veH ead();
   }
   get size() {
        return this.i tem -
s.size;
   }
    isE mpty() {
        return !this.i te ms.s -
ize;
   }
```

Stack with Array

```
//LIFO(Last in First Out) AND
FILO(First in Last Out)
class Stack {
       // Array is used to
implement stack
        con str uctor() {
                thi s.items =
[];
       }
```

Stack with Array (cont)

```
        pus h(e lem ent){ //O(1)
            thi s.i tem s.p -
ush (el ement);
       }
        pop(){ // O(1)
              return this.i -
tem s.p op();
       }
        peek(){
       // return the top most
element from the stack
        return this.i tem s[t -
his.it ems.length - 1];
       }
        isE mpty(){
         // return true if stack
is empty
        return this.i tem s.l -
ength == 0;
     }
}
```