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: ["Jan", "Feb", "Mar-
ch", "April", "June"]
```

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: ["camel", "duck"]
```

shit ()

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

Arrays Methods (cont)

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].filter(value => value>=10)
// filtered is [12, 130, 44]
```

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.find(-
element => element > 10);
// expected output: 12
```

Arrays Methods (cont)

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,Water,Fire'
a.join(''); // 'WindWaterFire'
```

Linked List

```
class Node {
constructor(element)
    {
        this.element = element;
        this.next = null;
        this.previous = null;
    }
}
class LinkedList {
    constructor()
    {
        this.head = null; //root
Node
        this.tail = null; //
last element of the list
        this.size = 0;
    }

  //Adds element to the begining
of the list. Similar to Array.u-
nshift
  //O(1)
   addHead(value) {
    const newNode = new Node(v-
alue);
    newNode.next = this.head;
    if (this.head) {
      this.head.previous =
newNode;
    } else {
```


Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Linked List (cont)

```
      this.tail = newNode;
    }
    this.head = newNode; //
update head
    this.size += 1;
    return newNode;
  }
    //Adds element to the end of
the list (tail)
    //O(1)
    addTail(value) {
    const newNode = new Node(v-
alue);
    if (this.head) {
      newNode.previous =
this.tail;
      this.tail.next = newNode;
      this.tail = newNode;
    } else {
      this.head = newNode;
      this.tail = newNode;
    }
    this.size += 1;
    return newNode;
  }

    // insert element at the
position index of the list
    insertAt(element, index)
    {
if (index < 0 || index >
this.size)
return console.log("Please enter
a valid index.");
else {
// creates a new node
var node = new Node(element);
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) {
it++;
prev = curr;
curr = curr.next;
}
// adding an element
node.next = curr;
prev.next = node;
}
this.size++;
}
     }
  //Removes element from the
start of the list (head/root).
  //Runtime O(1)
   removeHead() {
    const head = this.head;
    if (head) {
      this.head= head.next;
      if (this.head) {
        this.head.previous =
null;
      } else {
        this.tail = null;
      }
      this.size -= 1;
    }
    return head && head.value;
  }
   //Removes element to the end
of the list. Similar to
Array.pop
   //Runtime: O(1)
  removeTail() {
    const tail = this.tail;
    if (tail) {
      this.tail = tail.previous;
      if (this.tail) {
        this.tail.next = null;
      } else {
        this.head = null;
      }
      this.size -= 1;
    }
    return tail && tail.value;
  }
   // removes an element from
the specified location
   removeFrom(index)
   {
    if (index < 0 || index >=
this.size)
        return console.log("-
Please Enter a valid index");
    else {
        var curr, prev, it = 0;
        curr = this.head;
        prev = curr;

        // deleting first
element
        if (index === 0) {
            this.head =
curr.next;
        } else {
            // iterate over the
list to the
            // position to
removce an element
            while (it < index) {
```


Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Linked List (cont)

```
                it++;
                prev = curr;
                curr =
curr.next;
            }

            // remove the
element
            prev.next =
curr.next;
        }
        this.size--;

        // return the remove
element
        return curr.element;
     }
   }
    // removes a given element
from the list
    removeElement(element)
    {
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 (current.element === element)
{
if (prev == null) {
this.head = current.next;
} else {
prev.next = current.next;
}
this.size--;
return current.element;
}
prev = current;
current = current.next;
}
```

Linked List (cont)

```
return -1;
    }
    // finds the index of
element
    indexOf(element)
   {
    var count = 0;
    var current = this.head;

    // iterae over the list
    while (current != null) {
        // compare each element
of the list
        // with given element
        if (current.element ===
element)
            return count;
        count++;
        current = current.next;
    }

    // not found
    return -1;
   }

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

Queue with Array

```
//FIFO(First in First Out)
class Queue {
    // Array is used to
implement Queue
    constructor() {
        this.items = [];
    }
    enqueue(element){ //O(1)
      this.items.push(element);
    }
```

Queue with Array (cont)

```
     //Removes an element from
the front of a queue (items[0])
    dequeue(){ // O(n)
      return this.items.shift();
    }
    peek(){
    // return the front element
from the queue
    return this.items[0];
    }
    isEmpty(){
     // return true if queue is
empty
    return this.items.length ==
0;
   }
}
```

Queue with Linked List

```
class Queue {
  constructor() {
    this.items = new Linked-
List();
  }
   //Add element to the queue
   //Runtime: O(1)
  enqueue(item) {
    this.items.addTail(item);
    return this;
  }
 //Remove element from the queue
 //Runtime: O(1)
  dequeue() {
    return this.items.removeH-
ead();
  }
  get size() {
    return this.items.size;
  }
  isEmpty() {
```


Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Queue with Linked List (cont)

```
    return !this.items.size;
  }
```

Stack with Array

```
//LIFO(Last in First Out) AND FILO(First in
Last Out)
class Stack {
    // Array is used to implement stack
    constructor() {
        this.items = [];
    }
    push(element){ //O(1)
      this.items.push(element);
    }
    pop(){ // O(1)
       return this.items.pop();
    }
    peek(){
    // return the top most element from the
stack
    return this.items[this.items.length - 1];
    }
    isEmpty(){
     // return true if stack is empty
    return this.items.length == 0;
   }
}
```


Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com