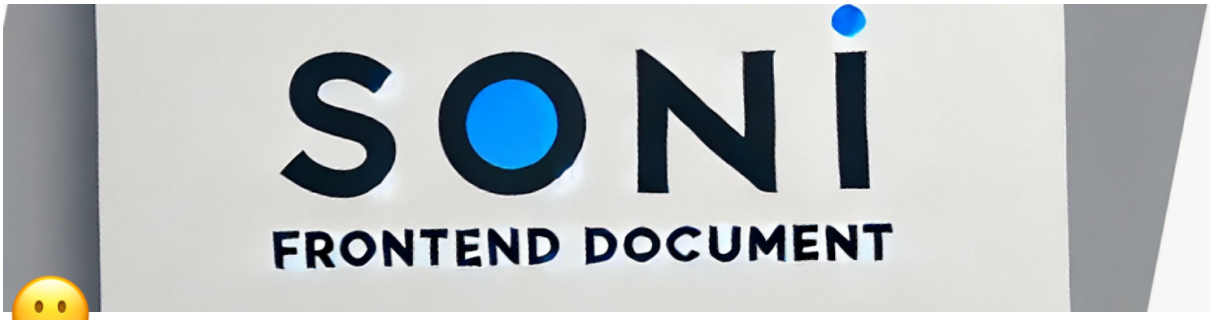


Soni Frontend Document

Telegram to Join: here

Basic

▼ Chain calculator

```
Question
Asked in Publicis sepient, Meesho
Level -> Easy

How would you implement a calculator class with methods for addition,
subtraction, and multiplication, supporting method chaining?
calculator.add(3).multiply(4).subtract(5).getValue()

class Calculator {
    // write code here
}

const calculator = new Calculator(2);
console.log(calculator.add(3).multiply(4).subtract(5).getValue()); //15
```

It's first question - Go ahead, make your brain sweat😅 a little before the solution spoils all your fun.

```
// Solution
class Calculator {
    constructor(initialValue = 0) {
        this.value = initialValue;
    }
```

```
    add(amount) {
        this.value += amount;
        return this; // Enable method chaining
    }

    subtract(amount) {
        this.value -= amount;
        return this; // Enable method chaining
    }

    multiply(factor) {
        this.value *= factor;
        return this; // Enable method chaining
    }

    getValue() {
        return this.value;
    }
}
// this will return same instance of the Calculator, will allow method
// chaining

const calculator = new Calculator(2);
console.log(calculator.add(3).multiply(4).subtract(5).getValue());
```

▼ Promises in sequence

```
Question
Asked in Sumologic, forward network
Level -> Easy

How would you implement a function to execute an array of asynchronous tasks
sequentially, collecting both resolved values and errors?


const createAsyncTask = () => {
    const randomVal = Math.floor(Math.random() *10)
    return new Promise((resolve, reject) =>{
        setTimeout(() => {
            if(randomVal > 5) {
                resolve(randomVal)
            }else{
                reject(randomVal)
            }
        },randomVal*100)
```

```
    })
}

const tasks = [
    createAsyncTask,
    createAsyncTask,
    createAsyncTask,
    createAsyncTask,
    createAsyncTask
]

const taskRunnerIterative = () => {
    //write code here
}
const taskRunnerRecursion = () => {
    //write code here
}

taskRunnerIterative(tasks, (result, err) => console.log(result, err))
taskRunnerRecursion(tasks, (result, err) => console.log(result, err))
```

Man, you are just one loop away! Go ahead and give it a shot, or are you just here to steal my solution🧐?

```
// Solution
// Approach 1
const taskRunnerIterative = async(tasks, cb) => {
    const result = [];
    const error = [];
    for(let task of tasks) {
        try{
                // wait until promise to resolved
            const successTask = await task()
            result.push(successTask)
        } catch(e) {
            error.push(e)
        }
    }
    cb(result, error)
}

// Approch 2
const taskRunnerRecursion = (tasks, cb) => {
    const result = [];
    const error = [];

    const helper = (ptr = 0) => {
```

```
        if(ptr === tasks.length) {
            cb(result, error)
            return;
        }
        tasks[ptr]().then((num) => {
            result.push(num)
        }).catch((num) => {
            error.push(num)
        }).finally(() =>{
            helper(++ptr)
        })
    }
    helper()
}

taskRunnerIterative(tasks, (result, err) => console.log(result, err))
taskRunnerRecursion(tasks, (result, err) => console.log(result, err))
```

▼ Pipe and compose

```
Questions
Asked in Tekion
Level -> Easy

Pipe
Create a pipe function that takes a series of functions and executes them
from left to right on an input value.

Compose
Create a compose function that takes a series of functions and executes them
from right to left on an input value

const pipe = (...fns) => {
    // code here
};

const compose = (...fns) => {
  //code here
};

const add5 = (x) => x + 5;
const multiply2 = (x) => x * 2;
const subtract3 = (x) => x - 3;
const toString = (x) => `${x}`;

console.log("Pipe");
```

```
const result1 = pipe(add5, multiply2, subtract3)(10);// (10 + 5) * 2 - 3 = 27
console.log(result1);

const result2 = pipe(toString, add5)(5);  // "5" + 5 = "55"
console.log(result2);

console.log("Compose");

const result3 = compose(add5, multiply2, subtract3)(10);// (10 - 3) * 2 + 5 =
19
console.log(result3);

const result4 = compose(toString, add5)(5);  // "5" + 5 = "55"
console.log(result4);
```

Pipe it up and compose your thoughts before checking the solution⚠️

```
// Solution
const pipe = (...fns) => {
  return function (x) {
    return fns.reduce((v, f) => f(v), x);
  };
};

const compose = (...fns) => {
  return function (x) {
    return fns.reduceRight((v, f) => f(v), x);
    // You can reverse fn array and use reduce
  };
};
```

▼ Array polyfills

```
Question
Level -> Easy
** Very frequently asked question **
// Reduce is important among all

Write custom polyfill for map, reduce, filter, every ?
```

I know you might be thinking, 'Why reinvent the wheel?' 🔁

But this is exactly what interviewers want. 🤦

So, go ahead—try it yourself! 👍

```
// Solutions

//Map
Array.prototype.myMap = function(callback) {
  if (typeof callback !== 'function') {
    throw new TypeError(callback + ' is not a function');
  }

  const result = [];
  for (let i = 0; i < this.length; i++) {
      result.push(callback(this[i], i, this));
  }
  return result;
};

//Filter
Array.prototype.myFilter = function(callback) {
  if (typeof callback !== 'function') {
    throw new TypeError(callback + ' is not a function');
  }

  const result = [];
  for (let i = 0; i < this.length; i++) {
    if (callback(this[i], i, this)) {
      result.push(this[i]);
    }
  }
  return result;
};

//Reduce
Array.prototype.myReduce = function(callback, initialValue) {
  if (typeof callback !== 'function') {
    throw new TypeError(callback + ' is not a function');
  }

  let accumulator = initialValue !== undefined ? initialValue : this[0];
  const startIndex = initialValue !== undefined ? 0 : 1;

  for (let i = startIndex; i < this.length; i++) {
      accumulator = callback(accumulator, this[i], i, this);
  }
  return accumulator;
};

//Every
Array.prototype.myEvery = function(callback) {
  if (typeof callback !== 'function') {
```

```
    throw new TypeError(callback + ' is not a function');
  }

  for (let i = 0; i < this.length; i++) {
    if (!callback(this[i], i, this)) {
      return false;
    }
  }
  return true;
};

const arr = [1, 2, 3, 4, 5];

// Using myMap
const doubled = arr.myMap(x => x * 2);
console.log(doubled); // [2, 4, 6, 8, 10]

// Using myFilter
const evenNumbers = arr.myFilter(x => x % 2 === 0);
console.log(evenNumbers); // [2, 4]

// Using myReduce
const sum = arr.myReduce((acc, curr) => acc + curr, 0);
console.log(sum); // 15

// Using myEvery
const allPositive = arr.myEvery(x => x > 0);
console.log(allPositive); // true
```

▼ Prototype and prototype inheritance

```
Question
Asked in servicenow and AngleOne
Level -> Easy

Explain how prototypal inheritance works in JavaScript
and how a child inherits properties from a parent using prototype.
```

It's a bit tricky 🤯 if you're only familiar with class-based syntax, but don't mess up your brain—just look at the solution🤓.

```
//Solution

// This can be easily done using a class-based but to judge your prototype
// knowledge such questions can come.

//parent function
function Person(name) {
  this.name = name;
}
Person.prototype.hello = function() {
  return `Hello ${this.name}`;
};

//child function
function Developer(name, title) {
  Person.call(this, name);
  this.title = title;
}

// Override person prototype in Developer's prototype
Developer.prototype = Object.create(Person.prototype);

 // Reset the constructor property of Developer's prototype
Developer.prototype.constructor = Developer;

// Now you can add any methods in developer prototype
Developer.prototype.getTitle = function() {
  return this.title;
};


const obj = new Developer("Alice", "Software Engineer");
console.log(obj.hello()); // Output: Hello Alice
console.log(obj.getTitle()); // Output: Software Engineer

// Learn more here about - https://www.youtube.com/watch?v=CpmE5twq1h0
// https://youtu.be/wstwjQ1yqWQ?si=gXOhIy4v_9NELs-j
```

▼ Call, apply, bind

```
Question
Asked in Jio and Expedia
Level -> Easy
```

```
Write custome polyfill for call, apply and bind method ?
```

	call()	apply()	bind()
Execution	At the time of binding	At the time of binding	At the time when we execute the return function
Parameter	any number of arguments one by one.	Array []	array and any number of arguments
Is Return Function	Yes, it returns and calls the same function at that time of binding.	Yes, it returns and calls the same function at that time of binding.	Yes, it returns a new function or copy of the function. Which we can use whenever we want. It's like a loaded gun.

```
//Solution

//Polyfill for Call
Function.prototype.myCall = function (context, ...args) {
  // Ensure arguments are correct
  if (typeof this !== "function") {
    throw new TypeError("myCall must be called on a function");
  }

  context = context !== null && context !== undefined ? Object(context) :
  globalThis;

  const uniqueSymbol = Symbol(); // it shoud be unique
  context[uniqueSymbol] = this;

  // Call the function with the provided arguments
  const result = context[uniqueSymbol](...args);

  // Remove the unique property to avoid side effects
  delete context[uniqueSymbol];

  return result;
};

// Testing myCall
function greet(name) {
  return `Hello, ${name}`;
}

const person = { name: "John" };
console.log(greet.myCall(person, "Alice"));
```

```
//Polyfill for Apply
Function.prototype.myApply = function (context, args) {
  if (typeof this !== "function") {
    throw new TypeError("myApply must be called on a function");
  }

  // Ensure context is an object
  context = context !== null && context !== undefined ? Object(context) :
  globalThis;

  // Ensure args is an array
  if (!Array.isArray(args)) {
    throw new TypeError("The second argument must be an array");
  }

  const uniqueSymbol = Symbol();
  context[uniqueSymbol] = this;

  // Call the function with arguments spread out (if any)
  const result = context[uniqueSymbol](...args);

  delete context[uniqueSymbol];

  return result;
};

function sum(a, b) {
  return a + b;
}

const context = {};
console.log(sum.myApply(context, [5, 10]));
```

```
//Polyfill for Bind
Function.prototype.myBind = function (context, ...boundArgs) {
  if (typeof this !== "function") {
    throw new TypeError("myBind must be called on a function");
  }

  // Ensure context is an object
  context = context !== null && context !== undefined ? Object(context) :
  globalThis;

  const fn = this;

  return function (...args) {
    return fn.apply(context, [...boundArgs, ...args]);
  };
};
```

```
function multiply(a, b) {
  return a * b;
}

const boundMultiply = multiply.myBind(null, 2);
console.log(boundMultiply(3));
console.log(boundMultiply(4));
```

▼ Flatten Array

```
Question
Very frequently asked question
Level -> Easy

Write custom function for Array.flat() using both recursive and iterative
approaches.

const flattenRecursive = (arr) => {
     //code here
};

const flattenIterative = (arr) => {
    // code here
};



Follow up
// Write a function to flatten a nested array up to a given depth
const flattenRecursiveWithDepth = (arr) => {
    // code here
};
```

Flatten your brainwaves🏄 before you flatten the array, give it a shot🚀 before stealing my solution!

```
//Solution

// Recursive Approach without Depth
const flattenRecursive = (arr) => {
    if (!Array.isArray(arr)) {
        throw new Error("Input must be an array");
    }
    const result = [];
```

```
    for (const ele of arr) {
        if (Array.isArray(ele)) {
            result.push(...flattenRecursive(ele));
        } else {
            result.push(ele);
        }
    }
    return result;
};

const resultRecursive = flattenRecursive(
[[[[0]], [1]], [[[2], [3]]], [[4], [5]]]
); // [0, 1, 2, 3, 4, 5]
console.log(resultRecursive, "Recursive Result");



// Iterative Approach without Depth
const flattenIterative = (arr) => {
    if (!Array.isArray(arr)) {
        throw new Error("Input must be an array");
    }
    const stack = [...arr];
    const result = [];
    while (stack.length) {
        const ele = stack.pop();
        if (Array.isArray(ele)) {
            stack.push(...ele);
        } else {
            result.push(ele);
        }
    }
    return result.reverse();
};

// Test case for Iterative Approach
const resultIterative = flattenIterative([[[[0]], [1]], [[[2], [3]]], [[4],
[5]]]); // [0, 1, 2, 3, 4, 5]
console.log(resultIterative, "Iterative Result");
```

```
//solution
// Recursive Approach with Depth
const flattenRecursiveWithDepth = (arr, depth) => {
  if (!Array.isArray(arr)) {
    throw new TypeError("The first argument must be an array.");
  }

  let result = [];
```

```
  if (depth === 0) return arr;

  for (let ele of arr) {
    if (Array.isArray(ele) && depth > 0) {
      result.push(...flattenRecursiveWithDepth(ele, depth - 1));
    } else {
      result.push(ele);
    }
  }

  return result;
};

const result = flattenRecursiveWithDepth(
            [[[[[0]]], [1]], [[[2], [3]]], [[4], [5]]], 1
);
console.log(result);
```

▼ Basic Debouncing

```
Question
Level -> Easy
Asked in Flipkart, intuit

Implement a debouncing function in JavaScript that delays the execution
of a given function until after a specified wait time has passed since the
last call.

function debounce(fn, delay) {
    //code here
}

const fn = debounce((message) => {
  console.log(message);
}, 300);

// Simulate rapid function calls
fn("Hello");
fn("Hello, World!");
fn("Debounced!"); // Only this should log after 300ms

setTimeout(() => {
    fn("Debounced twice");
},400)

// output
```

```
Debounced!
Debounced twice
```

While practicing, it's okay to delay—but in an interview, the interviewer might debounce your chances of success!

So, try solving it yourself first :)

```
//Solution
function debounce(fn, delay) {
  let timeout;

  return function(...args) {
    const context = this;
    clearTimeout(timeout);

    // Set a new timeout to call the function after the delay
    timeout = setTimeout(() => {
      fn.apply(context, args);
    }, delay);
  };
}
```

▼ Basic Throttling

```
Question
Level -> Easy
Asked in sumo logic, hotstar

Implement basic throttling funtion in js?

const throttleFnTimeBased = (fn, delay) => {
    //code here
}
//Test
const throttledFunction = throttleFnTimeBased((msg) => {
  console.log(msg, Date.now());
}, 2000);

throttledFunction("Call 1");  // Executes immediately
throttledFunction("Call 2");  // Throttled
throttledFunction("Call 3");  // Throttled

setTimeout(() => throttledFunction("Call 4"), 1100);
// Executes after 1.1 seconds
setTimeout(() => throttledFunction("Call 5"), 900);
// throttle
```

```
setTimeout(() => throttledFunction("Call 6"), 2100);
// Executes after 2.1 seconds
```

If you don't get this right in interview, the interviewer might throttle your hopes of a callback😆.
Try it first!

```
//Solution
const throttleFnTimeBased = (fn, delay) => {
  let lastExecuted = null;
  let timerId = null;
  return function (...args) {
    if (!lastExecuted) {
      fn.apply(this, args);
      lastExecuted = Date.now();
    } else {
        // remove previous timer
      clearTimeout(timerId);

      // create new timer remaning time
      timerId = setTimeout(() => {
        if (Date.now() - lastExecuted >= delay) {
          fn.apply(this, args);
          lastExecuted = Date.now();
        }
      }, delay - (Date.now() - lastExecuted));
    }
  };
};
```

▼ Event emitter

```
Question
Asked in BookmyShow and Dp world
Level -> Easy

How does the EventEmitter class handle multiple event subscriptions and
allow unsubscribing from individual events?

class EventEmitter {
    constructor() {}

    // Subscribe to an event
    subscribe(eventName, callback) {}
```

```
    // Emit an event
    emit(eventName, ...args) {}
}

// Example usage:
const emitter = new EventEmitter();

const subscription = emitter.subscribe("modify", (link) => {
    console.log(`Modified: ${link}`);
});

emitter.emit("modify", "test@gmail.com");
subscription.remove();

// No event will get published as it is removed
emitter.emit("modify", "test@gmail.com");

// No event found
emitter.emit("noEventfount", "test@gmail.com");
```

Preparing for interviews? Then mastering this question is a must to **emit success!**😎

Publish a 'resignation letter**' event** to your manager and the entire hierarchy simultaneously—and enjoy the chaos!

So, **try it yourself first** 😉

```
//Solution
class EventEmitter {
    constructor() {
        this._eventSubscriptions = new Map();
    }

    // Subscribe to an event
    subscribe(eventName, callback) {
            // type check
        if (typeof callback !== "function") {
            throw new TypeError("Callback should be a function");
        }

                // if event already exist
        if (!this._eventSubscriptions.has(eventName)) {
            this._eventSubscriptions.set(eventName, new Map());
        }

                //for unique idetifier, you can use Date.now() also
        const subscriptionId = Symbol();
```

```
        const subscriptions = this._eventSubscriptions.get(eventName);
        subscriptions.set(subscriptionId, callback);

        return {
            remove: () => {
                if (!subscriptions.has(subscriptionId)) {
                    throw new Error("Subscription has already removed");
                }
                subscriptions.delete(subscriptionId);
            }
        };
    }

    // Emit an event
    emit(eventName, ...args) {
        const subscriptions = this._eventSubscriptions.get(eventName);
        if (!subscriptions) {
            throw new Error("No event found")
        }

        subscriptions.forEach(callback => callback(...args));
    }
}
```

Medium

▼ Find the matching element in the DOM

```
Question
Level ->> Medium
Asked in Swiggy
Find the matching element in the DOM?

<!Doctype html>
<html lang="en">
  <head>
    <title>Find matching element</title>
  </head>
  <body>
  <div id="container1">
  <div>
```

```
      <div>
          <span id="span-id">Test1</span>
          <span id="span-id-2">Test2</span>
      </div>
  </div>
</div>

<div id="container2">
  <div>
      <div>
          <span>Test2</span>
      </div>
  </div>
</div>
    <script src="./index.js"></script>
  </body>
</html>


const findMatchingElement = (container1, container2, targetElement) => {
    //code here
}

// When Target element exists in container1
const positiveResult = findMatchingElement(
  document.getElementById("container1"),
  document.getElementById("container2"),
  document.getElementById("span-id")
);
console.log("Positive Result:", positiveResult.textContent);
//Test2

// When Target element does not exist in container2
const negativeResult = findMatchingElement(
  document.getElementById("container1"),
  document.getElementById("container2"),
  document.getElementById("span-id-2")
);
console.log("Negative Result:", negativeResult);
```

```
// Solution
    const findMatchingElement = (container1, container2, targetElement) => {
      // If the targetElement is directly found in container1, return the
      // corresponding element in container2
      if (container1 === targetElement) return container2;

      const children1 = Array.from(container1.children);
      const children2 = Array.from(container2.children);

      for (let i = 0; i < children1.length; i++) {
```

```
            if (children1[i] && children2[i]) {
                const result = findMatchingElement(
                    children1[i], children2[i], targetElement
                );
                if (result) return result;
            }
        }

        // If no match is found, return null
        return null;
    };
```

▼ Sort Array

```
Question
Asked in Freashworks
Level ->> Medium

Write custom polyfill for Sort ?
```

The moment you write `arr.sort()`, you might as well pack up your things 😆

I know, sorting an array might seem like the easiest thing in the world, but this is the exact problem that got me kicked out of interviews twice.

So, go ahead and give it a try—no pressure!

```
//Solution

const sortArray = function(nums) {
    function quickSort(l, h) {
        if (l >= h) return; // Base case: invalid range

        const index = partition(l, h); // Partition the array
        quickSort(l, index - 1);       // Recursively sort left part
        quickSort(index, h);           // Recursively sort right part
    }

    function partition(l, h) {
        const pivot = nums[Math.floor((l + h) / 2)]; // Choose pivot element
        while (l <= h) {
            while (nums[l] < pivot) l++; // move left pointer
            while (nums[h] > pivot) h--; // Move right pointer
            if (l <= h) {
                [nums[l], nums[h]] = [nums[h], nums[l]]; // Swap elements
```

```
                l++;
                h--;
            }
        }
        return l;
    }

    quickSort(0, nums.length - 1); // Start QuickSort
    return nums; // Return the sorted array
};

const arr = [21,31,45,56,43]
console.log(sortArray(arr))

//you can also go for merge sort
```

▼ Object Flatten

```
Question
Asked in Fractal
Level ->> Medium

Write a function flattenObject that takes a nested object and converts it
into a flat object,where keys represent the path to each value in the
original object.
The function should handle nested objects, arrays, and primitive types and
null.

//Input
const user = {
  name: "Vishal",
  age: null,
  address: {
    primary: {
      house: "109",
      street: {
        main: "21",
        cross: null,
      },
    },
    secondary: null,
  },
  phones: [
    { type: "home", number: "1234567890" },
    { type: "work", number: null },
  ],
  preferences: null,
```

```
};

//output
{
  user_name: "Vishal",
  user_age: null,
  user_address_primary_house: "109",
  user_address_primary_street_main: "21",
  user_address_primary_street_cross: null,
  user_address_secondary: null,
  user_phones_0_type: "home",
  user_phones_0_number: "1234567890",
  user_phones_1_type: "work",
  user_phones_1_number: null,
  user_preferences: null
}

const flattenObject = (obj, prefix = "", result = {}) => {
    //code here
}
```

it's just recursion, and you've alredy solved it! time to test your skills

```
// Solution
const flattenObject = (obj, prefix = "", result = {}) => {
  // Iterate over the keys of the object
  for (const key in obj) {
    if (obj.hasOwnProperty(key)) {
      const newKey = prefix ? `${prefix}_${key}` : key;
      if (Array.isArray(obj[key])) {
        // Handle arrays first, flatten each element with index
        obj[key].forEach((item, index) => {
          flattenObject(item, `${newKey}_${index}`, result);
        });
      } else if (typeof obj[key] === "object" && obj[key] !== null) {
        // Recursively flatten nested objects
        flattenObject(obj[key], newKey, result);
      } else {
        // Assign primitive values or `null` directly to the result
        result[newKey] = obj[key];
      }
    }
  }
  return result;
};
console.log(flattenObject({user}))
```

▼ [Array] Dispatch event on push

```
Question
Level ->> Medium
Asked in PharmEasy

Attach event on push element in an array ?
```

```
//Solution
const originalPush = Array.prototype.push;

// Modify the push method
Array.prototype.push = function (...args) {
  const result = originalPush.apply(this, args);
  if (this.onPush) {
    this.onPush(args);
  }
  return result;
};

Array.prototype.setPushCb = function (callback) {
  if (typeof callback === 'function') {
    this.onPush = callback;  // Store the callback function on the array
  } else {
    throw new TypeError('Callback must be a function');
  }
};

// Test
const arr = [];
arr.setPushCb((items) => {
  console.log('Items pushed:', items);
});

arr.push(1);
arr.push(2, 3);
```

▼ Deep clone object

```
Question
Level ->> Medium
Asked in Myntra, IBM

Implement polyfill for deep cloning OBject
```

```
function deepClone(args) {
    // code here
 }


// Test example
const original = {
  name: 'Alice',
  age: 30,
  address: {
    city: 'Wonderland',
    country: 'Fantasy'
  },
  hobbies: ['reading', 'biking'],
  birthDate: new Date(1994, 5, 24),
  regexTest: /abc/i
};

// Follow up: Adding a circular reference
original.circularRef = original;

const cloned = deepClone(original);

// Test the deep clone behavior
console.log(cloned);

// Check if circular reference is handled correctly
console.log(cloned.circularRef === cloned); // true
console.log(cloned.circularRef !== original.circularRef); // true

console.log(cloned.address !== original.address); // true
console.log(cloned.hobbies !== original.hobbies); // true
console.log(cloned.birthDate !== original.birthDate); // true
console.log(cloned.regexTest !== original.regexTest); // true

console.log(cloned.address.city === original.address.city); // true
```


```
//Solution
// Make sure you cover bases cases like object, Array, string , number, null
initally then discuss on Regex, Dateand circular depedency

function deepClone(obj, seen = new WeakMap()) {
```

```
  if (obj === null || typeof obj !== 'object') {
    return obj;
  }

  // Handle circular references
  if (seen.has(obj)) {
    return seen.get(obj);
  }

  // Create a new object or array based on the type
  const clone = Array.isArray(obj) ? [] : {};

  // Store the clone to handle circular references
  seen.set(obj, clone);

  // Iterate through all keys of the object or array
  for (const key in obj) {
    if (obj.hasOwnProperty(key)) {
      const value = obj[key];

      if (value instanceof Date) {
        clone[key] = new Date(value);
      } else if (value instanceof RegExp) {
        clone[key] = new RegExp(value.source, value.flags);
      } else if (typeof value === 'object') {
        clone[key] = deepClone(value, seen);
      } else {
        // Direct copy for primitive values
        clone[key] = value;
      }
    }
  }

  return clone;
}
```

▼ JSON.Stringify

```
Question
Asked in Thoughtspot, Multiplier
Level ->> Medium

Write custom function for JSON.stringify

const obj = {
  name: "John",
  age: 30,
```

```
  city: "New York",
  addr: ["chandpol", "avv"],
  myUndefined: undefined,
  myNull: null,
  circularRef: null,
  nested: {
    name: "Nested",
    valid: true,
  },
  fn: () => {}
};

function myStringify( your args) {
    //code here
}

//output

{
    "name":"John",
    "age":30,
    "city":"New York",
    "addr":["chandpol","avv"],
    "myNull":null,
    "circularRef":null,
    "nested":{"name":"Nested","valid":true}
}
```


```
// Solution

//  In interview don't handle all data type in one shot, you can start with
//  string number object array then go for handling circular depedency
```

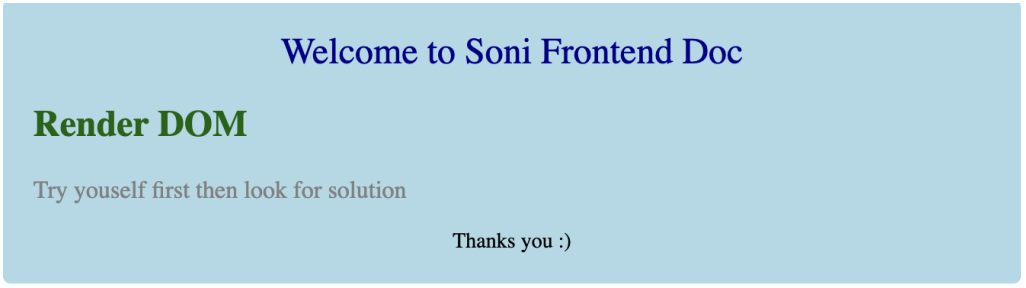
```
//  handle null properly, beacause typeof null === object


function myStringify(value, seen = new WeakSet()) {
  if (value === null || value === undefined || typeof value === "symbol") {
    return "null";
  }
  if (typeof value === "string") {
    return `"${value}"`;
  }
  if (typeof value === "number" || typeof value === "boolean") {
    return `${value}`;
  }

  if (typeof value === "function") {
    return undefined;
  }

  if (Array.isArray(value)) {
    let arrayResult = value
      .map((item) => (
      myStringify(item, seen) === undefined ? "null" : myStringify
      (item, seen))).join(",");
    return `[${arrayResult}]`;
  }

  if (typeof value === "object") {
    if (seen.has(value)) {
      throw new Error("Circular reference detected");
    }
    seen.add(value);

    let objResult = Object.entries(value)
      .filter(([key, val]) => typeof val !== "function" && val !== undefined)
      .map(([key, val]) => `"${key}":${myStringify(val, seen)}`)
      .join(",");

    return `{${objResult}}`;
  }

  throw new Error(`Unsupported data type: ${typeof value}`);
}

console.log(myStringify(obj))

// What if there is circular dependency ?
obj.circularRef = obj;

try {
  const output = myStringify(obj);
  console.log(output);
```

```
} catch (error) {
  console.error(error.message);
}
```

▼ React DOM render

```
Question
Asked in Meta
Level ->> Medium

How does the renderDom function create and append DOM elements
based on the dom object structure


const dom = {
  type: 'section',
  props: {
      id: 'section-1',
      class: 'main-section',
      style: 'background-color: lightblue; padding: 20px;
              border-radius: 5px;'
  },
  children: [
      {
          type: 'header',
          children: 'Welcome to Soni Frontend Doc',
          props: {
              style: 'font-size: 24px; color: darkblue; text-align: center;'
          }
      },
      {
          type: 'article',
          children: [
              {
                  type: 'h2',
                  children: 'Render DOM',
                  props: { style: 'color: darkgreen;' }
              },
              {
                  type: 'p',
                  children: 'Try youself first then look for solution',
                  props: { style: 'font-size: 16px; color: grey;' }
              }
          ]
      },
      {
          type: 'footer',
```

```
            children: 'Thanks you :)',
            props: {
                style: 'text-align: center; font-size: 14px; color: black;'
              }
        }
    ]
};
```




```
//Solution

// index.html
<html lang="en">
  <head>
    <title>Render Dom</title>
  </head>
  <body>
      <div id="root"></div>
    <script src="./index.js"></script>
  </body>
</html>

//index.js
const rootEle = document.getElementById("root");

const renderDom = ({ type, props, children }) => {
  // Edge cases
  if (!type) return null;
  const ele = document.createElement(type);

  // Set attributes and inline styles
  if (props) {
    Object.entries(props).forEach(([key, value]) => {
      if (key === 'style') {
        ele.style.cssText = value;
      } else {
        ele.setAttribute(key, value);
      }
    });
  }
```

```
  // Render children
  if (Array.isArray(children)) {
    children.forEach(child => ele.appendChild(renderDom(child)));
  } else if (typeof children === "string") {
    ele.textContent = children;
  }

  return ele;
};

if (rootEle) {
  rootEle.appendChild(renderDom(dom));
}
```

▼ Retry promises N times

```
Questions
Level ->> Medium
Asked in Agoda , Tekion

How can you implement a retry mechanism for fetching data?

function retryPromise(fn, retries = 3, delay = 1000) {
    // write code here
}

// mock fetch data
const fetchData = () => {
  return new Promise((resolve, reject) => {
    // Simulate a request that might fail
    const success = Math.random() > 0.5; // 50% chance of success
    console.log(success, "success")
    if (success) {
      resolve("Data fetched successfully!");
    } else {
      reject("Failed to fetch data");
    }
  });
};

retryPromise(fetchData, 3, 1000)
  .then((result) => console.log(result))
  .catch((error) => console.log(error));
```

Make sure you solve this question by yourself—unless you're excited about waiting six months for your next retry!

```
// Solution
function retryPromise(fn, retries = 3, delay = 1000) {
    // return promise
  return new Promise((resolve, reject) => {
    const attempt = (n) => {
      fn()
        .then(resolve)
        .catch((error) => {
          if (n <= 1) {
            reject(error); // If no retries left, reject the promise
          } else {
            setTimeout(() => {
              attempt(n - 1); // Retry the function after a delay
            }, delay);
          }
        });
    };
    attempt(retries); // Start the first attempt with the given retries
  });
}
```

▼ Extended version of Event emitter

```
Question
Asked in Coursera recently
Level ->> Medium

class Events {
    constructor() {
        this._subscriptions = new Map();
    }

    // Function to subscribe to an event
    subscribe(name, callback) {
        // Implement subscription logic
    }

    // Function to subscribe to an event once
    subscribeOnce(name, callback) {
        // Implement one-time subscription logic
    }

    // Function to subscribe to an event once asynchronously
```

```
    async subscribeOnceAsync(name) {
        // Implement async one-time subscription logic
    }

    // Function to publish an event
    publish(name, data) {
        // Implement event publishing logic
    }

    // Function to publish all events
    publishAll(data) {
        // Implement logic for publishing all events
    }
}

//Test
const events = new Events();

// Subscription
const newUserNewsSubscription = events.subscribe("new-user",
 function (payload) {
    console.log(`Sending Q1 News to: ${payload}`);
});

events.publish("new-user", "Jhon");
console.log("--------");

// Adding another subscription
const newUserNewsSubscription2 = events.subscribe("new-user",
function (payload) {
    console.log(`Sending Q2 News to: ${payload}`);
});

events.publish("new-user", "Doe");
console.log("--------");

// Removing a subscription
newUserNewsSubscription.remove();
events.publish("new-user", "Foo");
console.log("--------");

// Publish all events
events.publishAll("FooBar");
console.log("--------");

// Subscribe once
events.subscribeOnce("new-user", function (payload) {
    console.log(`I am invoked once ${payload}`);
});

events.publish("new-user", "Foo Once");
```

```
console.log("--------");

events.publish("new-user", "Foo Twice");
console.log("--------");

// Subscribe once asynchronously
events.subscribeOnceAsync("new-user").then(function (payload) {
    console.log(`I am invoked once asynchronously: ${payload}`);
});

events.publish("new-user", "Foo Once Async");
console.log("--------");


// Output
Sending Q1 News to: Jhon
--------
Sending Q1 News to: Doe
Sending Q2 News to: Doe
--------
Sending Q2 News to: Foo
--------
Sending Q2 News to: FooBar
--------
Sending Q2 News to: Foo Once
I am invoked once Foo Once
--------
Sending Q2 News to: Foo Twice
--------
Sending Q2 News to: Foo Once Async
--------
I am invoked once asynchronously: Foo Once Async
```

```
//Solution

class Events {
    constructor() {
        this._subscriptions = new Map(); // Stores all event subscriptions.
    // 🚨 dont create differnt map to handle subscribeOnce &
    //subscribeOnceAsync
    }


    subscribe(name, callback) {
        if (typeof callback !== "function") {
            throw new TypeError("Callback should be a function");
        }
```

```
        if (!this._subscriptions.has(name)) {
            this._subscriptions.set(name, []);
        }

        const subscription = { callback };
        this._subscriptions.get(name).push(subscription);

        return {
            remove: () => {
                const eventSubscriptions = this._subscriptions.get(name);
                const index = eventSubscriptions.findIndex(
                    sub => sub.callback === callback
                );
                if (index !== -1) {
                    eventSubscriptions.splice(index, 1);
                    // Remove the subscription
                }
            }
        };
    }


    subscribeOnce(name, callback) {
        const removeOnce = this.subscribe(name, (payload) => {
            callback(payload);
            removeOnce.remove(); // Remove the subscription after it runs
        });
    }


    async subscribeOnceAsync(name) {
        return new Promise((resolve) => {
            const removeOnce = this.subscribe(name, (payload) => {
                resolve(payload);
                removeOnce.remove(); // Remove the subscription after it runs
            });
        });
    }

    publish(name, data) {
        const eventSubscriptions = this._subscriptions.get(name);
        if (!eventSubscriptions) {
            return;
        }

        eventSubscriptions.forEach(sub => sub.callback(data));
    }

    publishAll(data) {
        this._subscriptions.forEach((eventSubscriptions) => {
            eventSubscriptions.forEach(sub => sub.callback(data));
```

```
        });
    }
}
```

▼ Promise.all

```
Question
Level ->> Medium
// Very frequently asked question
Asked in Salesforce recently


function myPromiseAll(taskList) {
    //code here
}


// Success case
const successTasks = [
  new Promise((resolve) => setTimeout(() => resolve('Task 1'), 1000)),
  new Promise((resolve) => setTimeout(() => resolve('Task 2'), 500)),
  new Promise((resolve) => setTimeout(() => resolve('Task 3'), 200)),
  "Test",
  3
];

myPromiseAll(successTasks)
  .then((result) => console.log(result))  // Output: ['Task 1', 'Task 2', 'Ta
sk 3',
   "Test", 3]
  .catch((error) => console.error(error));

// Error case
const errorTasks = [
  new Promise((resolve) => setTimeout(() => resolve('Task 1'), 1000)),
  new Promise((resolve, reject) => setTimeout(() => reject('Error'), 500))
];

myPromiseAll(errorTasks)
  .then((result) => console.log(result))
  .catch((error) => console.error(error)); // Output: Error
```

```
// Solution
function myPromiseAll(taskList) {
  const result = new Array(taskList.length);
  let completed = 0;
```

```
  return new Promise((resolve, reject) => {
    for (let i = 0; i < taskList.length; i++) {
      Promise.resolve(taskList[i]) // Important*
        .then((data) => {
          result[i] = data;  // Ensure result is stored in the correct order
          completed++;

          if (completed === taskList.length) {
            resolve(result);
          }
        })
        .catch((error) => {
          reject(error);  // Reject immediately if any promise fails
        });
    }
  });
}
```

▼ Promise.race

```
Question
Level ->> Medium
function myPromiseRace(taskList) {
    //code here
}

const successRaceTasks = [
  new Promise((resolve) => setTimeout(() => resolve('Task 1'), 1000)),
  new Promise((resolve) => setTimeout(() => resolve('Task 2'), 500)),
  new Promise((resolve) => setTimeout(() => resolve('Task 3'), 200)),
];

myPromiseRace(successRaceTasks)
  .then((result) => console.log(result)) // Output: "Task3"
  .catch((error) => console.error(error));


const errorRaceTasks = [
  new Promise((resolve, reject) => setTimeout(() => reject('Error 1'), 500)),
  new Promise((resolve) => setTimeout(() => resolve('Task 2'), 1000)),
  new Promise((resolve, reject) => setTimeout(() => reject('Error 2'), 200))
];

myPromiseRace(errorRaceTasks)
  .then((result) => console.log(result))
  .catch((error) => console.error(error)); // Output: "Error 2"
```

```
// Solution
// Custom Promie.Race
function myPromiseRace(taskList) {
  return new Promise((resolve, reject) => {
    for (let i = 0; i < taskList.length; i++) {
      Promise.resolve(taskList[i])
        .then(resolve)  // Resolve as soon as the first promise resolves
        .catch(reject); // Reject as soon as the first promise rejects
    }
  });
}

const successRaceTasks = [
  new Promise((resolve) => setTimeout(() => resolve('Task 1'), 1000)),
  new Promise((resolve) => setTimeout(() => resolve('Task 2'), 500)),
  new Promise((resolve) => setTimeout(() => resolve('Task 3'), 200)),
];

myPromiseRace(successRaceTasks)
  .then((result) => console.log(result)) // Output: "Task3"
  .catch((error) => console.error(error));


const errorRaceTasks = [
  new Promise((resolve, reject) => setTimeout(() => reject('Error 1'), 500)),
  new Promise((resolve) => setTimeout(() => resolve('Task 2'), 1000)),
  new Promise((resolve, reject) => setTimeout(() => reject('Error 2'), 200))
];

myPromiseRace(errorRaceTasks)
  .then((result) => console.log(result))
  .catch((error) => console.error(error)); // Output: "Error 2"
```

▼ Promise.any

```
Question
Level ->> Medium
function myPromiseAny(taskList) {
    //code here
}


const successTasks = [
  new Promise((_, reject) => setTimeout(() => reject("Error Task 1"), 1000)),
  new Promise((resolve) => setTimeout(() => resolve("Task 2"), 500)),
  new Promise((resolve) => setTimeout(() => resolve("Task 3"), 200)),
];
```

```
myPromiseAny(successTasks)
  .then((result) => console.log(result)) // Output: "Task 3"
  .catch((error) => console.error(error));


const failureTasks = [
  new Promise((_, reject) => setTimeout(() => reject("Error Task 1"), 1000)),
  new Promise((_, reject) => setTimeout(() => reject("Error Task 2"), 500)),
];

myPromiseAny(failureTasks)
  .then((result) => console.log(result))
  .catch((error) => console.error(error));
  // Output: AggregateError ErrorAll promise were rejeted
```

```
// Solution
function myPromiseAny(taskList) {
  let rejectionCount = 0;
  const errors = [];

  return new Promise((resolve, reject) => {
    if (taskList.length === 0) {
        // Reject immediately if input is empty
      reject(new AggregateError([], "All promises were rejected"));
    }

    for (let i = 0; i < taskList.length; i++) {
      Promise.resolve(taskList[i])
        .then(resolve) // Resolve as soon as any promise is fulfilled
        .catch((error) => {
          errors[i] = error;
          rejectionCount++;
          if (rejectionCount === taskList.length) {
              // Reject if all promises are rejected
            reject(new AggregateError(errors, "All promises were rejected"));
          }
        });
    }
  });
}
```

▼ Promise.allSettle

```
Question
Level ->> Medium
function myPromiseAllSettled(taskList) {
    //code here
}

const mixedTasks = [
  new Promise((resolve) => setTimeout(() => resolve('Task 1'), 1000)),
  new Promise((_, reject) => setTimeout(() => reject('Error Task 2'), 500)),
  "Immediate Value",
  new Promise((resolve) => setTimeout(() => resolve('Task 3'), 200))
];

myPromiseAllSettled(mixedTasks)
  .then((result) => console.log(result));

// Output
// [
//   { status: "fulfilled", value: "Task 1" },
//   { status: "rejected", reason: "Error Task 2" },
//   { status: "fulfilled", value: "Immediate Value" },
//   { status: "fulfilled", value: "Task 3" }
// ]
```

```
// Solution
function myPromiseAllSettled(taskList) {
  const result = new Array(taskList.length);
  let completed = 0;

  return new Promise((resolve) => {
    for (let i = 0; i < taskList.length; i++) {
      Promise.resolve(taskList[i])
        .then((value) => {
          result[i] = { status: "fulfilled", value };
        })
        .catch((reason) => {
          result[i] = { status: "rejected", reason };
        })
        .finally(() => {
          completed++;
          if (completed === taskList.length) {
            resolve(result); // Resolve when all promises settle
          }
        });
    }
  });
}
```

▼ ClearAllTimeout

```
Question
Asked in Dream11
Level ->> Medium

Write a polyfill for clearAllTimeout that tracks and clears all timeouts set
using setTimeout.
```

```
// Test
const timeout1 = setTimeout(() => console.log("Timeout 1"), 10000);
const timeout2 = setTimeout(() => console.log("Timeout 2"), 3000);
const timeout3 = setTimeout(() => console.log("Timeout 3"), 4000);

setTimeout(() => {
  console.log("Clearing all timeouts...");
  clearAllTimeout();
}, 5000); // This should clear all timeouts before they execute

// Timeout 2
// Timeout 3
```

```
//Solution
// we are storing original function in a varibale and overriding setTimeOut
// with custom functionality

//Using IIFE
(function() {
  let timeouts = []

  // set original setTimeout in a variable
  const originalSetTimeout = this.setTimeout;

  // modify original setTimeOut
  this.setTimeout = function(callback, delay, ...args) {
    const timeoutId = originalSetTimeout(callback, delay, ...args);
    timeouts.push(timeoutId);  // Store the timeout ID
    return timeoutId;
  };

  // Define clearAllTimeout to clear all timeouts
  this.clearAllTimeout = function() {
    timeouts.forEach((value, timeoutId) => {
      this.clearTimeout(timeoutId); // Clear each timeout
    });
    timeouts = []
```

```
  };
})();
```

▼ Memoization

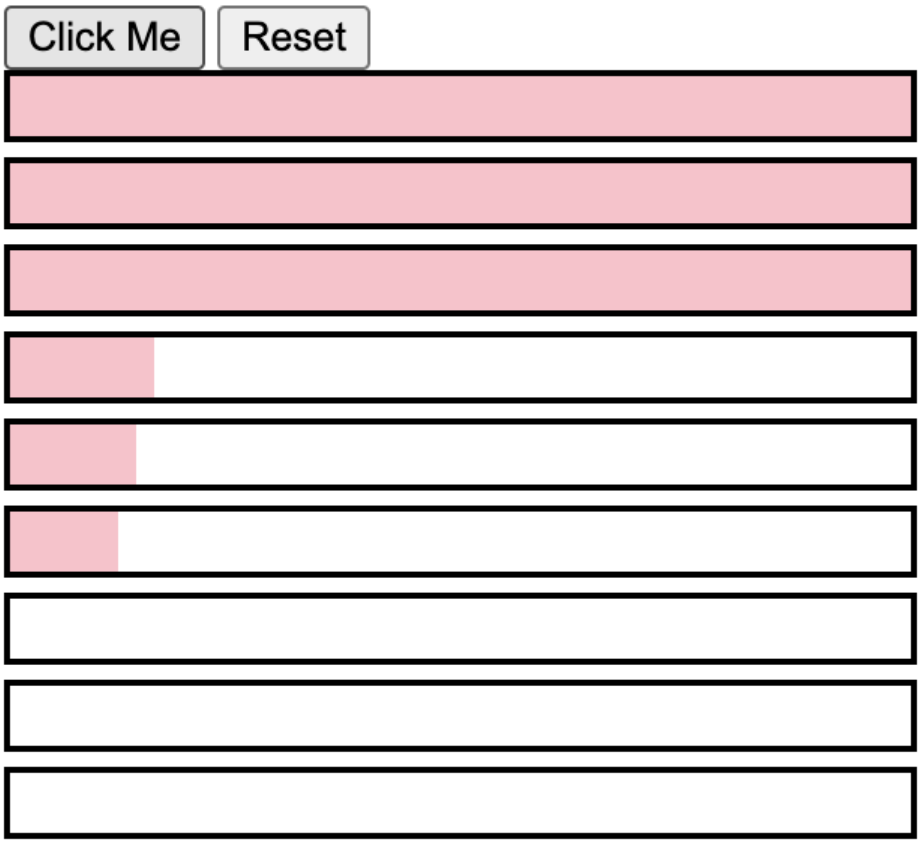
```
Question
Level ->> Medium
Asked in Rippling, Forward network


Write a memoization function ?

// Intially start with number data type ->  then string  ->  Object  ->
// Sort object

const memoization = (fn) => {
    // write you code here
}

// Example Function
const exampleFunction = (a, b) => {
  console.log("Function executed");
  return a + b;
};

// Memoized Function
const memoizedExample = memoization(exampleFunction);

console.log(memoizedExample(1, 2)); // Cache miss
console.log(memoizedExample(1, 2)); // Cache hit

console.log(memoizedExample("hello", "world")); // Cache miss
console.log(memoizedExample("hello", "world")); // Cache hit


// Follow up question
const exampleObj = (obj1, num) => {
  console.log("Function executed");
  return obj1.a + obj1.b + obj1.c + num;
};

const memoizedExampleObj = memoization(exampleObj);

// Test Cases with Objects
console.log(memoizedExampleObj({ a: 1, b: 2, c: 3 }, 6)); // Cache miss
console.log(memoizedExampleObj({ c: 3, b: 2, a: 1 }, 6));
// Cache hit (key order normalized)
```

```
console.log(memoizedExampleObj({ a: 1, b: 2, c: 3 }, 10)); // Cache miss
console.log(memoizedExampleObj({ a: 1, b: 2, c: 3 }, 10)); // Cache hit
```

```
// Solution
const memoization = (fn) => {
  const cache = new Map();

  // Serialize arguments to create a unique key
  const serialize = (value) => {
    if (typeof value === "object" && value !== null) {
      // Normalize the object by sorting its keys
      const sortedKeys = Object.keys(value).sort();
      return `{${sortedKeys.map((key) => `"${key}":${JSON.stringify
      (value[key])}`).join(",")}}`;
    }
    return JSON.stringify(value); // Handle numbers, strings, etc.
  };

  return function (...args) {
    const cacheKey = args.map(serialize).join("|");

    if (cache.has(cacheKey)) {
      console.log(`Cache hit for: ${cacheKey}`);
      return cache.get(cacheKey);
    }

    console.log(`Cache miss for: ${cacheKey}`);
    const result = fn.apply(this, args);
    cache.set(cacheKey, result);
    return result;
  };
};
```

▼ Async Progress bar

```
Quesiton
Level ->> Medium
Asked in Coursera, Uber

// How can you implement a progress bar system that supports multiple
// concurrent progress animations
// with a limit on active animations at a time?
```




```
//Solution

//HTML
<!doctype html>
<html lang="en">
  <head>
    <title>Progress App</title>
  </head>
  <body>
    <button id="btn">Click Me</button>
    <button id="reset-btn">Reset</button>
    <div id="progress-bar"></div>
  </div>
    <script src="./main.js"></script>
  </body>
</html>


//javascript
const progressBarEle = document.getElementById('progress-bar');
const btn = document.getElementById('btn');
const resetBtnbtn = document.getElementById("reset-btn");
```

```
function createProgressBarElement() {
    const wrapper = document.createElement("div");
    Object.assign(wrapper.style, {
        height: "20px",
        width: "300px",
        border: "2px solid black",
        marginBottom: "5px",
    });

    const progress = document.createElement("div");
    Object.assign(progress.style, {
        width: "0%",
        height: "20px",
        background: "pink",
    });

    wrapper.appendChild(progress);
    progressBarEle.appendChild(wrapper);

    return progress;
}

function ProgressBarManager(maxConcurrent) {
    let activeCount = 0;
    const queue = [];
    let currentIndex = 0;

    function animateProgressBar(element) {
        activeCount++;
        let progress = 0;

        const intervalId = setInterval(() => {
            progress++;
            element.style.width = `${progress}%`;

            if (progress >= 100) {
                clearInterval(intervalId);
                activeCount--;
                runNextProgressBar();
            }
        }, 10);
    }

    function runNextProgressBar() {
        if (currentIndex < queue.length && activeCount < maxConcurrent) {
            const element = queue[currentIndex++];
            animateProgressBar(element);
        }
    }

    function addProgressBar() {
```

```
        const progressBarElement = createProgressBarElement();
        queue.push(progressBarElement);
        runNextProgressBar();
    }

    function resetProgressBars() {
        queue.length = 0;
        currentIndex = 0;
        activeCount = 0;
        progressBarEle.innerHTML = "";
    }

    return { addProgressBar, resetProgressBars };
}

const progressBarManager = ProgressBarManager(3);

btn.addEventListener("click", () => {
    progressBarManager.addProgressBar();
});

resetBtnbtn.addEventListener("click", () => {
  progressBarManager.resetProgressBars();
});
```

▼ GroupBy Polyfill Loadsh

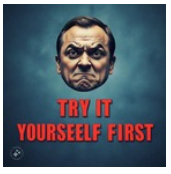
```
Question
Level ->> Medium
Asked in messho
Implement _.gropuBy from Loadsh library


function groupBy(collection, property) {
    //code here
}

// Test with invalid input
const result1 = groupBy(1);
console.log(result1); // Output: {}

// Group by a custom function
const result2 = groupBy([6.1, 2.4, 2.7, 6.8], Math.floor);
console.log(result2);
// Output: { "2": [2.4, 2.7], "6": [6.1, 6.8] }

// Group by string property (length of the string)
```

```
const result3 = groupBy(["one", "two", "three"], "length");
console.log(result3);
// Output: { "3": ["one", "two"], "5": ["three"] }

// Group by deep property path
const result4 = groupBy([{ a: { b: { c: 1 } } }, { a: { b: { c: 2 } } }],
 "a.b.c");
console.log(result4);
// Output: { "1": [{ a: { b: { c: 1 } } }], "2": [{ a: { b: { c: 2 } } }] }
```

```
//Solution

function groupBy(collection, property) {
  // Ensure the collection is valid
  if (typeof collection !== 'string' && !Array.isArray(collection) ) {
    return {};
  }

  // Helper function to get key
  const getKey = (item) => {
    if (typeof property === 'function') {
      return property(item);
    } else if (typeof property === 'string') {
      // Resolve deep property paths (e.g., "p.q.r")
      const keys = property.split('.');
      let value = item;
      for (const key of keys) {
        value = value[key];
      }
      return value;
    }
  };

  // Initialize the output object
  const output = {};

  // Iterate through the collection and group items by key
  for (const item of collection) {
    const key = getKey(item);

    // Create a new group if the key doesn't exist
    if (!output[key]) {
      output[key] = [];
    }

    // Add the item to the corresponding group
    output[key].push(item);
  }
  return output;
}
```

▼ Currying

```
Question
Asked in Pubmatic
Level ->> Medium/hard

How would you implement a function for infinite currying that accumulates
values passed in successive calls and returns the result when called without
arguments?

function currying(fn) {
        //code here
}

curry(1)(2)(3)(4)()




// Follow up Question

// Implement a currying function that allows partial application of arguments
// for a given multi-parameter function?

function currying(fn) {
    //code here
}

curriedMultiply(1)(2)(3)(4)
curriedMultiply(1, 2)(3, 4)
curriedMultiply(1)(2, 3)(4)
```




```
//Solution

function curry(args1) {
    return function (args2) {
        if(!args2) {
```

```
            return args1
        }else{
            return curry(args1 + args2)
        }
    }
}

const result = curry(1)(2)(3)(4)()
console.log(result)



function currying(fn) {
    return function curried(...args) {
        if (args.length >= fn.length){
        //checking if argument length is matching
            return fn.apply(this, args);
        } else {
            return function (...args2) {
                return curried.apply(this, args.concat(args2));
            };
        }
    };
}

function multiply(a, b, c, d) {
    return a * b * c * d;
}

const curriedMultiply = currying(multiply);

console.log(curriedMultiply(1)(2)(3)(4));
console.log(curriedMultiply(1, 2)(3, 4));
console.log(curriedMultiply(1)(2, 3)(4));
```

▼ Execute task in parallel with resolved dependency

```
Question
Level ->>> Hard
Asked in Rippling, Uber and Amazon

How can you resolve task dependencies in a directed acyclic graph
 and execute tasks in parallel with a concurrency limit?

 function taskA(done) {
  console.log("Task A Completed");
  done();
```

```
}
function taskB(done) {
  setTimeout(() => {
    console.log("Task B Completed");
    done();
  }, 2000);
}
function taskC(done) {
  setTimeout(() => {
    console.log("Task C Completed");
    done();
  }, 200);
}
function taskD(done) {
  console.log("Task D Completed");
  done();
}
function taskE(done) {
  console.log("Task E Completed");
  done();
}

const asyncGraph = {
  e: {
    dependency: ["c", "d"],
    task: taskE,
  },
  c: {
    task: taskC,
  },
  d: {
    dependency: ["a", "b"],
    task: taskD,
  },
  a: {
    task: taskA,
  },
  b: {
    task: taskB,
  },
};
// e must be resolved before c & d
// c no dependency
// d must be resolved before a & b
// a no dependency
// b no depedency

// first resolve dependency and then run task in parallel.
// taskOrder -> pass an array with resolved dependency
executeTasksInParallel(taskOrder, asyncGraph, 2).then(() => {
  console.log("All tasks completed.");
```

```
});

Output
// Task A Completed
// Task C Completed
// Task D Completed
// Task E Completed
// Task B Completed
// All tasks completed.
```

```
//Solution

//first try to resolve dependency using topologic sorting
function resolveDependencies(graph) {
  const graphNodes = Object.keys(graph);
  const adjList = new Map();
  const inDegree = new Map();
  const topologicalOrder = [];

  // Build adjacency list and in-degree map
  for (const node of graphNodes) {
    const { dependency } = graph[node] || {};
    for (const dep of dependency || []) {
      const neighbors = adjList.get(dep) || [];
      neighbors.push(node);
      adjList.set(dep, neighbors);
    }
    inDegree.set(node, (dependency ? dependency.length : 0));
  }

  // Perform topological sort
  const queue = [];
  for (const node of graphNodes) {
    //start adding nodes which has no dependency
    if (inDegree.get(node) === 0) {
      queue.push(node);
    }
  }

  while (queue.length > 0) {
    const current = queue.shift();
    topologicalOrder.push(current);

    const neighbors = adjList.get(current) || [];
    // look around it dependent neighbour
    for (const neighbor of neighbors) {
      inDegree.set(neighbor, inDegree.get(neighbor) - 1);
      if (inDegree.get(neighbor) === 0) {
        queue.push(neighbor);
      }
```

```
    }
  }

  return topologicalOrder;
}

//Then execute task in parallel
function executeTasksInParallel(order, graph, limit = 2) {
  let activeTasks = 0;
  let index = 0;

  return new Promise((resolve) => {
    const results = [];

    function executeNext() {
      if (index >= order.length && activeTasks === 0) {
        resolve(results); // All tasks are done
        return;
      }

      while (index < order.length && activeTasks < limit) {
        const currentTask = order[index];
        index++;
        activeTasks++;

        graph[currentTask].task(() => {
          console.log(`${currentTask} completed.`);
          activeTasks--;
          executeNext(); // Check for the next task
        });
      }
    }

    executeNext();
  });
}

const taskOrder = resolveDependencies(asyncGraph)
executeTasksInParallel(taskOrder, asyncGraph, 2).then(() => {
  console.log("All tasks completed.");
});
```

Advance

▼ Debouncing leading and trailing spaces

```
Question
Level ->>> Hard
Asked in mindtickle

// Follow up question with leading and traling spaces

function debounce(func, wait, option = {leading: false, trailing: true}) {
    //code here
}

// Test Case 1: {leading: false, trailing: true} (default case)
const logMessage1 = debounce((message) => {
  console.log(message);
}, 1000, { leading: false, trailing: true });

logMessage1("A");
logMessage1("B");
logMessage1("C");
setTimeout(() => logMessage1("D"), 2000);
setTimeout(() => logMessage1("E"), 2500);
setTimeout(() => logMessage1("F"), 3000);
setTimeout(() => logMessage1("G"), 3500);

// Expected Output: "C", "G"



// Test Case 2: {leading: true, trailing: true}
const logMessage2 = debounce((message) => {
  console.log(message);
}, 1000, { leading: true, trailing: true });

logMessage2("A");
logMessage2("B");
logMessage2("C");
setTimeout(() => logMessage2("D"), 2000);
setTimeout(() => logMessage2("E"), 2500);
setTimeout(() => logMessage2("F"), 3000);
setTimeout(() => logMessage2("G"), 3500);

// Expected Output: "A", "C", "D", "G"
```

```
// Test Case 3: {leading: true, trailing: false}
const logMessage3 = debounce((message) => {
  console.log(message);
}, 1000, { leading: true, trailing: false });

logMessage3("A");
logMessage3("B");
logMessage3("C");
setTimeout(() => logMessage3("D"), 2000);
setTimeout(() => logMessage3("E"), 2500);
setTimeout(() => logMessage3("F"), 3000);
setTimeout(() => logMessage3("G"), 3500);

// Expected Output: "A" "D"
```

```
//Solution
function debounce(func, wait, option = {leading: false, trailing: true}) {
  let timerId = null;
  let lastArgs = null;
  // if both leading and trailing are false then do nothing.
  if(!option.leading && !option.trailing) return () => null;

  return function debounced(...args) {
    // if timer is null and leading is true
    if(!timerId && option.leading) {
      func.apply(this, args);
    } else {
      lastArgs = args;
    }
    // clear timer so that next call is exactly after `wait` time
    clearTimeout(timerId);
    timerId = setTimeout(() => {
      // invoke if traling and  lastArgs is present
      if(option.trailing && lastArgs) func.apply(this, lastArgs);

      // Resent varibale
      lastArgs = null;
      timerId = null;
    }, wait);
  }
}
```

▼ MapLimit

```
Question
Level ->>> Hard
```

```
Asked in Uber and Rippling

Question link -
https://leetcode.com/discuss/interview-experience/2074287/uber-frontend-phone
-screen

inputs: An array containing the inputs to be processed.
maxlimit: The maximum number of concurrent operations that can be executed
at a time.
iterateeFn: An asynchronous function that is called for each input to
generate the corresponding output. It accepts two arguments:
input: The current input being processed.
callback: A function that is invoked when the processing of the input is
complete. It receives the processed output as its argument.
callback: A function that is called once all inputs have been processed.
It receives an array of the final outputs.

function getNameById(id, callback) {
    const randomRequestTime = Math.floor(Math.random() * 100) + 2000;
    console.log("randomRequestTime", randomRequestTime, id);

    setTimeout(() => {
      callback("User" + id)
    }, randomRequestTime);
  }

  function mapLimit(inputs, maxLimit, iterateeFn, callback) {
          // write solution
  }

  mapLimit([1,2,3,4,5], 3, getNameById, (allResults) => {
    console.log('output:', allResults)
    // ["User1", "User2", "User3", "User4", "User5"]
  })
```

```
//Solution

function getNameById(id, callback) {
  // simulating async request
  const randomRequestTime = Math.floor(Math.random() * 100) + 4000;
  console.log("randomRequestTime", randomRequestTime, id);

  setTimeout(() => {
    callback("User" + id)
  }, randomRequestTime);
}
```

```
function mapLimit(inputs, maxLimit, iterateeFn, callback) {
  let currentPtr = 0;
  let currentLimit = 0;
  const result = []

  const processNext = (id) => {
      result.push(id);
      currentLimit--;

      if(result.length == inputs.length  ) {
          callback(result)
      }
      if(currentPtr < inputs.length  && currentLimit < maxLimit) {
          iterateeFn(inputs[currentPtr++], processNext);
          currentLimit++
      }
  }

  while(currentPtr < inputs.length  && currentLimit < maxLimit) {
      iterateeFn(inputs[currentPtr++], processNext)
      currentLimit++;
  }
}

mapLimit([1,2,3,4,5], 2, getNameById, (allResults) => {
  console.log('output:', allResults)
})
```

▼ Cancelable Promise

```
Question
Asked in Thoughspot
Level ->>> Hard

// How can you implement a Promise.cancelable utility to allow
// cancellation of a promise with a custom error?

Promise.cancelable = (promise) => {
    //code here
}

// Test
const asyncTask1 = new Promise((resolve) => {
  setTimeout(() => {
    resolve("Task 1 completed");
  }, 500);
});
```

```
const asyncTask2 = new Promise((resolve) => {
  setTimeout(() => {
    resolve("Task 2 completed");
  }, 3000);
});

const cancelableTask1 = Promise.cancelable(asyncTask1);
const cancelableTask2 = Promise.cancelable(asyncTask2);

cancelableTask1
  .then((result) => console.log(result))
  .catch((error) => console.error(error));

  cancelableTask2
  .then((result) => console.log(result))
  .catch((error) => console.error(error));

// Cancel the task after 1 second
setTimeout(() => {
  cancelableTask1.cancel();
  cancelableTask2.cancel()
}, 1000);

// Output
// Task 1 completed
// ERROR! CanceledPromiseError: Promise has been canceled
```

Try to solve it yourself first☝️, or it might cancel your hopes of acing that interview!

Give it a try before looking at the solution!

```
// Solution
// we are building wrapper on existing promise and function cancel

class CanceledPromiseError extends Error {
  constructor() {
    super("Promise has been canceled");
    this.name = "CanceledPromiseError";
  }
}

Promise.cancelable = (promise) => {
  let isCanceled = false;

  const wrappedPromise = new Promise((resolve, reject) => {
    promise.then(
      (value) => {
        if (isCanceled) {
          reject(new CanceledPromiseError());
```

```
        } else {
          resolve(value);
        }
      },
      (error) => {
        reject(error); // Propagate any errors from the original promise
      }
    );
  });

  wrappedPromise.cancel = () => {
    isCanceled = true;
  };

  return wrappedPromise;
};
```

▼ TypeHead search using LRU cache

```
// Question
Level ->>> Hard
Asked in Salesforce, powerplay


How can we efficiently implement a typeahead cache system with LRU (Least
Recently Used)eviction to store and retrieve search suggestions?


class TypeAheadCache {
    constructor(capacity) {}

    async fetchSuggestions(query) {
        // Fetch reuslt from getSuggestionsFromAPI
    }

    async getSuggestionsFromAPI(query) {
        // mock server
        return new Promise(resolve => {
            setTimeout(() => {
                resolve(query);
            }, 500);
        });
    }
}

// Test
const typeAhead = new TypeAheadCache(2); // Limit cache to 2 items
```

```
async function testTypeAhead() {
   await typeAhead.fetchSuggestions('apple');  // Fetch from API
    await typeAhead.fetchSuggestions('banana'); // Fetch from API
    await typeAhead.fetchSuggestions('apple');  // Cache hit
    await typeAhead.fetchSuggestions('cherry'); // Fetch from API
    await typeAhead.fetchSuggestions('banana'); // Fetch from API
    await typeAhead.fetchSuggestions('date');   // Fetch from API
    await typeAhead.fetchSuggestions('apple');  //  Fetch from API
    await typeAhead.fetchSuggestions('date');  //  Cache hit
}

testTypeAhead();
```

```
// Solution
// Reference - https://leetcode.com/problems/lru-cache

class LRUCache {
    constructor(capacity) {
        this.cache = new Map();
        this.capacity = capacity;
    }

    get(key) {
        if (!this.cache.has(key)) return -1;

        // Move the accessed key to the end to mark it as recently used
        const value = this.cache.get(key);
        this.cache.delete(key);
        this.cache.set(key, value);

        return value;
    }

    put(key, value) {
        if (this.cache.has(key)) {
            this.cache.delete(key);
        }

        // If the cache exceeds capacity, remove the first entry
        if (this.cache.size >= this.capacity) {
            this.cache.delete(this.cache.keys().next().value);
        }

        // Insert item at end
        this.cache.set(key, value);
    }
}

class TypeAheadCache {
```

```
    constructor(capacity) {
        this.cache = new LRUCache(capacity);
    }

    async fetchSuggestions(query) {
        // Check if query exists in cache
        const cachedResult = this.cache.get(query);
        if (cachedResult !== -1) {
            console.log('Cache hit for:', query);
            console.log(this.cache.cache.keys());
            console.log("-------------------------------");
            return cachedResult;
        }

        // Simulate an API call or database search for the suggestions
        console.log('Fetching from API for:', query);
        const results = await this.getSuggestionsFromAPI(query);

        // Cache the result
        this.cache.put(query, results);
        console.log(this.cache.cache.keys());
        console.log("-------------------------------");

        return results;
    }

    async getSuggestionsFromAPI(query) {
        // Simulate a delay and return some dummy suggestions
        return new Promise(resolve => {
            setTimeout(() => {
                resolve(query);
            }, 500);
        });
    }
}
```

▼ Doc comparison

```
Question
level ->>> Hard
Asked in Google


// How can you compare two deeply nested JSON objects to identify their
// differences,ensuring that each differing key is represented with a
```

```
// "from and to" for string, number, and object data types?

// If key is not present in one of obj then print "EMPTY" (see hobbies and
// country)

const doc1 = {
  name: "John",
  age: 12,
  address: {
    city: "Boston",
    zip: "10001",
    country: "USA",
  },
  phone: "987-654-3210",
  friends: {
    friend1: { name: "Alice", age: 30 },
    friend2: { name: "Bob", age: 25 }
  },
  hobbies: ["table tennis"]
};

const doc2 = {
  name: "John",
  age: 14,
  address: {
    city: "New York",
    zip: "10001",
    country: "Canada",
  },
  phone: "123-456-7890",
  friends: {
    friend1: { name: "Alice", age: 30 },
    friend2: { name: "Bob", age: 26 }
  },
  country: "India"
};

function getDifference(doc1, doc2) {
   // write code here
}

// const difference = getDifference(doc1, doc2);

// Output
{
  age: { from: 12, to: 14 },
  address: {
    city: { from: 'Boston', to: 'New York' },
    country: { from: 'USA', to: 'Canada' }
  },
  phone: { from: '987-654-3210', to: '123-456-7890' },
```

```
  friends: { friend1: {}, friend2: { age:  {from: 25, to: 26} } },
  hobbies: { from: [ 'table tennis' ], to: 'EMPTY' },
  country: { to: 'India', from: 'EMPTY' }
}
```

```
// Solution
function compareObjects(obj1, obj2 ) {
    const difference = {};
    const keys = new Set([...Object.keys(obj1), ...Object.keys(obj2)]);

  const compareKeys = (key, obj1, obj2, parent) => {
    if (!(key in obj1)) {
      // Key added in obj2
      parent[key] = { to: obj2[key], from: "EMPTY" };
    } else if (!(key in obj2)) {
      // Key removed from obj1
      parent[key] = { from: obj1[key], to: "EMPTY" };
    } else if (typeof obj1[key] === 'object' && obj1[key] !== null &&
      typeof obj2[key] === 'object' && obj2[key] !== null) {
      // Recurse if both values are objects
      parent[key] = compareObjects(obj1[key], obj2[key]);
    } else if (obj1[key] !== obj2[key]) {
      parent[key] = { from: obj1[key], to: obj2[key] };
    }
  };
  keys.forEach(key => compareKeys(key, obj1, obj2, difference));

  return difference;
}

const diff = compareObjects(doc1, doc2)
```