Interview Questions

03 March 2020 11:42

LkedIn, Round 1, Telephonic

LinkedIn

Explain Event bubbling

Prototype vs Classical Inheritance

Create a tooltip with accessibility

```
function parent(a) {
    function inner() { return a; }
    this.prop = function () { return a; }
}
parent.prototype = {
    prop2 = function () { return a }
}
```

1. What is the return value of each function?
2. Changes to make sure each one returns correct value of parameter 'a'?

Merge and sort an array based on property and return a new array.

Supply AI

Deep comparison of two objects.

UI Path

Find missing number in an array.

Prefix search TRIE

JavaScript inheritance using functions

VM Ware

```
1.  Context values
    Name = 'global name';
    Obj = {
          name: "Andhra Pradesh"
         Arrow:  () =>{
           Console.log(this.name)
         }
         Normal () =>{
         Console.log(this.name)
         }
    }
```

2. Spread operator, mutable , deep and shallow copy
3. Type coercion in object key.
4. Array rotation
5. Box sizing and padding.
6. Em and relative position parent
7. Benefits of CSSs preprocessor
8. Specificity calculations.
9. Deep copy vs shallow copy.

GoldMan

N student array and 2 step iteration find the remaining element.

C2FO

Function Currying.	
✓	Promise all
☐	Rxjs Map Pipe implementation
✓	Angular guards
☐	Angular Lifecycle
☐	TypeScript Generics
☐	TypeScript Functions
☐	Local minima maxima

☐	BFT with print in the next line
☐	Array from 1 to n. with one additional element between 1 to n. Find the additional element

Flipkart
1. If styles are at the bottom 2. Const arr [promise1,promise2,promise3]; Run so that output of promise 1 is used in promise 2. 3. O(1) operation stack. 4. Nested object currying and function execution . 5. Copy an object recursively 6. Immutability. System design Suggestions - Implement stock ticker average problem - User experience for infinite photo albums.(by using bi-directions infinite scroll Sliding window algo) - Redux familiarity - Data flow - Virtual DOM and Real DOM comparison - Whatsapp product improvements - image handling, message deletion, contacts save UI System design Design a news feed app. 7. Feedback: Weak Poor 1. Round 1. a. Need to prioritize functionality. 2. Round 2. a. Correct approach wrong direction. b. Requires lots of hints. c. Couldn't do simple recursive copying. 3. Product sense. a. Weak fundamentals, Product sense, API design, Requirement gathering. b. Good Listener.

Oracle
Round 1.
Nested tree
Round 2.
1. Symbol 2. Promise
Round 3
1. Recursive value 2. Cost and Weight and weight limit bag. Find maximum value. (KNAPSACK) 3. Tree.

Rubirik

// Generate data center level aggregated stats of protected vs. unprotected applications
 	
// BLACKBOX START
 	
// The following mock helper method returns protection stats for a given application type in a given data center.

// In real implementation this will be an ajax call to \`${datacenterUrl}/${applicationType}\` or something like that
const getApplicationProtectionStats = (datacenterUrl, applicationType)=> new Promise((resolve) => {
setTimeout(() => {
resolve({

```
     protectedApps: Math.floor(Math.random() * 100),
     unprotectedApps: Math.floor(Math.random() * 100)
   });
  }, 1000);
});

// BLACKBOX END

const applicationTypes = [
  'VMware',
  'MSSQL',
  'AHV'
];

const datacenters = [
  {name: 'San Francisco', url: 'https://www.rubrik.com/sanfrancisco/stats'},
  {name: 'Bangalore', url: 'https://www.rubrik.com/bangalore/stats'},
  {name: 'Amsterdam', url: 'https://www.rubrik.com/amsterdam/stats'}
];

const combineStats = (datacenter, appTypes) =>{
   return Promise.all(appTypes.map(app => getApplicationProtectionStats(datacenter.url,app)))
  .then(res =>{
     const val = res.reduce((acc,curr)=>{
       acc.protectedApps = acc.protectedApps? acc.protectedApps+  curr.protectedApps
:curr.protectedApps;
       acc.unprotectedApps = acc.unprotectedApps? acc.unprotectedApps +  curr.unprotectedApps
:curr.unprotectedApps
       return acc;
     },{});
     return { [datacenter.name] : val};
   })
}

Promise.all(datacenters.map(datacenter =>{
  return combineStats(datacenter,applicationTypes)
})).then(stats =>datacenterStats(stats));

const datacenterStats = (stats)=>{
     const result = stats.reduce((acc,curr)=>{
       Object.assign(acc,curr)
       return acc;
     },{});
     resultCallback(result);
}

const resultCallback = (aggregateObj) =>{
   console.log('resultCallback', aggregateObj);
};
```

247 ai

1. Input Songs Id's : a, e, f, b, c

Output : c, a, b, e, f

Two constraints

1. Songs should not duplicate till all songs are played
2. There should not be pattern in shuffle.

2. State management and data consistency.

Nike

1. Merge two sorted arrays with condition of even is always greater than odd.
2. Unique pairs of sum in an array.