JavaScript Interview Questions

Section 1: Core Javascript Concepts

- Mapping Users to Get Usernames
  - Q1: Write code to get an array of names from given array of users

```
const users = [
  { id: 1, name: "Jack", isActive: true },
  { id: 2, name: "John", isActive: true },
  { id: 3, name: "Mike", isActive: false },
];
// Result
// ['Jack', 'John', 'Mike']
```

- Difference between null and undefined
  - Q1: What will be logged in this example?

```
let var1;
console.log(var1);
console.log(typeof var1);
```

  - Q2: What will be logged in this example?

```
let var2 = null;
console.log(var2);
console.log(typeof var2);
```

- Hoisting
  - Q1: What will be logged here?

```
console.log(foo);
foo = 1;
```

  - Q2: What will be logged here?

```
console.log(foo);
var foo = 2;
```

  - Q3: What will be logged here?

```
var foo;
foo = 3;
console.log(foo);
```

- Closures
  - Q1: Create a counter function which has increment and getValue functionality

```
const counter = privateCounter();
console.log(counter.getValue()); // 0
counter.increment();
console.log(counter.getValue()); // 1
```

  - Q2: Create a function which stores a secret string inside which is not accessible but is returned only when we call this function.

```
const getSecret = privateSecret();
console.log(getSecret()); // 'secret'
```

- Currying
  - Q1: Write a function which helps to achieve multiply(a)(b) and

returns multiplication of a and b

```
console.log(multiply(2)(3)); // 6
```

  - Q2: Create a curry function

```
const curriedSum = curry((a, b, c) => a + b + c);
const partiallyCurriedSum = curriedSum(1);
console.log(partiallyCurriedSum(2, 3)); // 6
```

- Adding Elements to the Array
  - Q1: Write a function which get's an array and an element and returns an array with this element at the end.

```
const numbers = [1, 2];
const newNumbers = append(numbers, 3);
console.log(newNumbers, numbers); // [1,2,3]
```

- Concatenating Arrays
  - Q2: Write a function which can concatenate 2 arrays

```
const arr1 = [1];
const arr2 = [2, 3];
const result = mergeArrays(arr1, arr2);
console.log(result, arr1, arr2); // [1,2,3]
```

- Check if User With Such Name Exists
  - Q1: Write a function which accepts a list of users and a name to check if such user exists in the array.

```
const users = [
  { id: 1, name: "Jack", isActive: true },
  { id: 2, name: "John", isActive: true },
  { id: 3, name: "Mike", isActive: false },
];
console.log(isNameExists("Jack", users)); // true
```

- Remove All Duplicates in the Array
  - Q1: Write a function which removes all duplicates from the array.

```
console.log(uniqueArr([1, 1, 2])); // [1,2]
```

- Sorting the array
  - Q1: Sort the array of numbers
  - Q2: Sort an array of objects by author's lastname

```
const books = [
  { name: "Harry Potter", author: "Joanne Rowling" },
  { name: "Warcross", author: "Marie Lu" },
  { name: "THe Hunger Games", author: "Suzanne Collins" },
];
```

- Writing Range Function
  - Q1: Write a function which implements a range

```
console.log(range(1, 50)); // [1,2,3,4,5...,50]
```

- Writing Shuffle Function
  - Q1: Write a shuffle function which mixes the elements

```
console.log(shuffleItems([1, 2]));
```

- Find the Number of Occurences of Minimum Value in List
  - Q1: Find the number of occurrences of minimum value in the list of

numbers

- This
  - Q1: What will be logged here?
    ```
    function getItem() {
      console.log(this);
    }
    getItem();
    ```
  - Q2: What will be logged here?
    ```
    const item = {
      title: "Ball",
      getItem() {
        console.log(this);
      },
    };
    item.getItem();
    ```
  - Q3: What will be logged here?
    ```
    class Item {
      title = "Ball";
      getItem() {
        console.log(this);
      }
    }
    const item = new Item();
    item.getItem();
    ```
  - Q4: What will be logged here?
    ```
    class Item {
      title = "Ball";
      getItem() {
        [1, 2, 3].map(function (item) {
          console.log(this);
        });
      }
    }
    const item = new Item();
    item.getItem();
    ```
- Classes
  - Q1: Design a class for employee which takes id and name in during construction of object and has a salary property
    ```
    const employee = new Employee(1, "Jack");
    employee.setSalary(1000);
    ```
  - Q2: Design a class for manager which is employee and can have a department property.
    ```
    const manager = new Manager(1, "Jack");
    manager.setSalary(1000);
    manager.setDepartment("Development");
    console.log(manager.getDepartment());
    ```

- Prototypes
  - Q1: Design the same classes as in previous question but by using only JavaScript prototypes and not class keyword.
- I've Failed Interview. What's Next?
- Modules
  - Q1: Create an ES6 module with function getName, getSurname and default export getFullname.
  - Q2: Create the same with commonJS module
  - Q3: What is the differences between ES6 modules and CommonJS modules?
- Implement Debounce Function
  - Q1: Create a debounce function

```
const saveInput = (name) => console.log("saveInput", name);
const processChange = debounce(saveInput, 2000);
processChange("foo");
processChange("foo");
processChange("foo");
processChange("foo");
processChange("foo");
```

- Implement Throttle Function
  - Q1: Create a throttle function

```
const saveInput = (name) => console.log("saveInput", name);
const processChange = throttle(saveInput, 2000);
processChange("foo");
setTimeout(() => {
  processChange("foo");
}, 1000);
setTimeout(() => {
  processChange("foo");
}, 1200);
setTimeout(() => {
  processChange("foo");
}, 2400);
processChange("foo");
processChange("foo");
```

Section 2: Working with DOM

- Highlight All Words Over 8 Chars With Yellow
  - Q1: Highlight all of the words in markup over 8 characters long in the paragraph text (with a yellow background for example)
- Add a Link
  - Q1: Add a link "Back to source" after a paragraph tag which goes to https://forcemipsum.com in the markup
- Split Each Sentence to a Separate Line
  - Q1: Split each new sentence on to a separate line in the paragraph

text. A sentence can be assumed to be a string of text terminated with a period (.)

- Event Delegation
  - Q1: Implement a click on todo item which has a high performance.

```
<ul class="todo-app">
  <li class="item">Walk the dog</li>
  <li class="item">Pay bills</li>
  <li class="item">Make dinner</li>
  <li class="item">Code for one hour</li>
</ul>
```

Asynchronous Javascript

- Xml HTTP Request
  - Q1: Write an example of fetching data with XMLHttpRequest
- Fetch API
  - Q1: Write an example of fetching data using fetch API
- Basic Callback
  - Q1: Write an asynchronous function which executes callback after finishing it's asynchronous task

```
asyncFn((message) => {
  console.log("callback", message);
});
```

  - Q2: What problem does callback solve?
- Parallel Async Array
  - Q1: Execute the given list of asynchronous functions in parallel and return the results as an array to the callback

```
const asyncFn1 = (callback) => {
  setTimeout(() => {
    callback(1);
  }, 3000);
};
const asyncFn2 = (callback) => {
  setTimeout(() => {
    callback(2);
  }, 2000);
};
const asyncFn3 = (callback) => {
  setTimeout(() => {
    callback(3);
  }, 1000);
};
asyncParallel([asyncFn1, asyncFn2, asyncFn3], (result) => {
  console.log(result); // 1, 2, 3
});
```

- Convert Callback to Promise

  - Q1: Create a promise function to be able to use callback function via promise approach
- Map Data in Promises
  - Q1: You have 2 functions which return promises. Map data from getUsers and getUserStatuses to get array of users with id, name, isActive

```
const users = [
  { id: 1, name: "Jack" },
  { id: 2, name: "John" },
  { id: 3, name: "Mike" },
];
const userStatuses = [
  { id: 1, isActive: true },
  { id: 2, isActive: true },
  { id: 3, isActive: false },
];
const getUsers = () => {
  return new Promise((resolve) => {
    resolve(users);
  });
};
const getUserStatuses = () => {
  return new Promise((resolve) => {
    resolve(userStatuses);
  });
};
```

- Rewrite Mapping Data in Async Await
  - Q1: You have 2 functions which return promises. Map data from users and userStatuses to get array of users with id, name, isActive (you take data from the previous task)

```
const getUsers = () => {
  return new Promise((resolve) => {
    resolve(users);
  });
};
const getUserStatuses = () => {
  return new Promise((resolve) => {
    resolve(userStatuses);
  });
};
```

- Design Request Manager
  - Q1: Design an utility which takes URL and a value for attempts which will attempt to make a fetch request. If on failure it tries again with increasing delay for number of times which user has requested.

```
requestManager("http://foo.com", {}, 3).then((response) => {
  console.log(response);
```

```
});
```

Comparison Functions

- Implement Shallow Comparison
  - Q1: Design a shallow comparison function
- Implement Deep comparison
  - Q1: Design a deep comparison function
- Create Memoization Function
  - Q1: Design a memorization function which adds 10 to provided value and takes it from cache if it was already calculated

Tasks Asked Only on Interview

- Fibonacci
  - Q1: Design a function which returns a fibonacci sequence value
- Palindrome
  - Q1: Write a function which checks if string is a palidrome
- Anagram
  - Q1: Write a function which checks if string is an anagram
- Finding vowels
  - Q1: Write a function which counts vowels in a string
- Convert to Title Case
  - Q1: Write a function to convert a string to title case
- Convert the Time Input Given in 12 Hours Format to 24
  - Q1: Write a function which can convert the time input given in 12 hours format to 24 hours format
- Mapping Data
  - Q1: Map data to frontend format. The main element is location key and we need to map all data to it.

```
const loc = [
  {
    location_key: [32, 22, 11],
    autoassign: 1,
  },
  {
    location_key: [41, 42],
    autoassign: 1,
  },
```

```
];
const bulkConfig = [
  {
    dataValues: {
      config_key: 100,
    },
  },
  {
    dataValues: {
      config_key: 200,
    },
  },
];
// Result
// [
//   {
//     config_key: 100,
//     location_key: 32,
//     autoassign: 1
//   },
//   {
//     config_key: 100,
//     location_key: 22,
//     autoassign: 1
//   },
//   ....
// ]
```

- Replace Parameters in URL
  - Q1: Write a function to replace parameters in url

```
const initalUrl = "/posts/:postId/comments/:commentId";
const resultUrl = replaceParamsInUrl(initalUrl, [
  { from: "postId", to: "1" },
  { from: "commentId", to: "3" },
]); // /posts/1/comments/3
```

- Validation Messages
  - Q1: Format backend validation message to frontend format

```
const backendErrors = {
  email: {
    errors: [{ message: "Can't be blank" }],
  },
  password: {
    errors: [
      { message: "Must contain symbols in different case" },
```

```
      { message: "Must be at least 8 symbols length" },
    ],
  },
  passwordConfirmation: {
    errors: [{ message: "Must match with password" }],
  },
};
// Result
// [
//   "Email: Can't be blank",
//   "Password: Must contain symbols in different case, Must be at least 8 symbols
//   "PasswordConfirmation: Must match with password"
// ]
```

- Nested List
  - Q1: Transform flat list to nested list

```
const flatList = [
  {
    id: 1,
    name: "lvl 1 item 1",
    parentId: null,
  },
  {
    id: 2,
    name: "lvl 1 item 2",
    parentId: null,
  },
  {
    id: 3,
    name: "lvl 2 item 3",
    parentId: 1,
  },
  {
    id: 4,
    name: "lvl 3 item 4",
    parentId: 3,
  },
  {
    id: 5,
    name: "lvl 2 item 5",
    parentId: 2,
  },
];

//  Result
```

```
//  [
//   {
//     id: 1,
//     children: [
//       {
//         id: 3,
//         children: [
//           {id: 4, children: []}
//         ]
//       }
//     ]
//   },
//   {
//     id: 2,
//     children: [
//       {
//         id: 5,
//         children: []
//       }
//     ]
//   }
// ]
```

Knowing JS inside out is huge — but senior devs also need to think about systems, not just code.

That's a big focus in my Middle to Senior Bootcamp. We prep for real interviews, design better apps, and sharpen senior-level skills. If that sounds good, here's more info.