



# Questions for JavaScript Interview (with Answers)

## **1. Write a code to demonstrate promises.**

Let's assume, if you're creating a web application that sends data requests to a server, you must handle those requests asynchronously to keep the application responsive to user interactions.

Promises are an important part of this type of task because they let you handle operations that happen at different times in a clean and effective way.

Function returning promise

```
1  const fetchData = () => {  
2      return new Promise((resolve, reject) => {  
3          fetch('https://jsonplaceholder.typicode.com/todos/1')  
4              .then(response => response.json())  
5              .then(data => resolve(data))  
6              .catch(error => reject(error));  
7      });  
8  };  
9  
10 fetchData()  
11   .then(data => console.log(data))  
12   .catch(error => console.error(error));  
13
```

On success, Resolving promise with  
data

On failure, Rejecting promise

On success, this block will be executed

On failure, this block will be executed

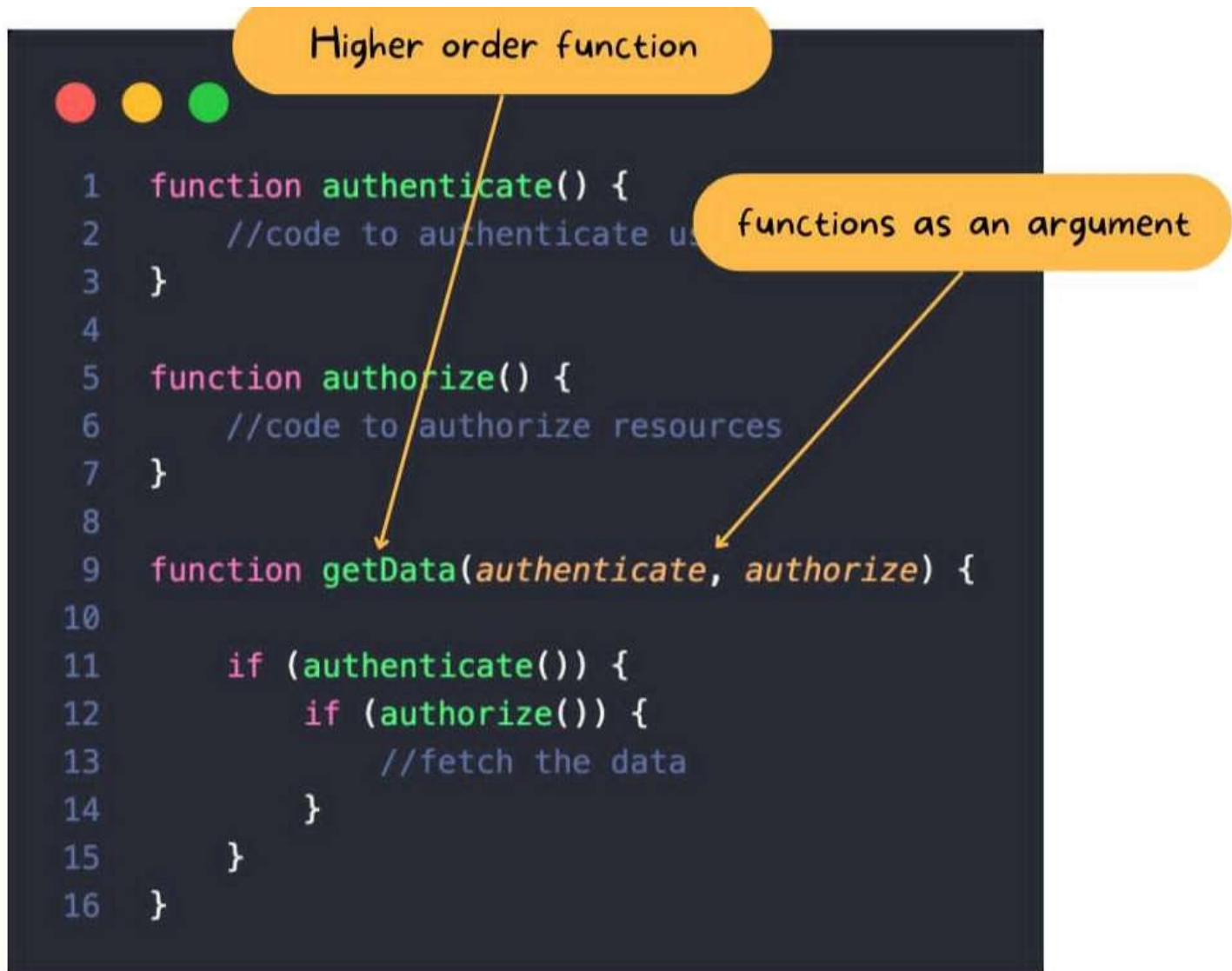
## **2. What are higher order functions?**

Higher order are functions that either accept one or more functions as arguments or returns a function.

Let's take an example,

When a user sends a request to access data, you have to first authenticate them, then authorize them to access the data.

So code will be as follows:



```
1  function authenticate() {  
2      //code to authenticate user  
3  }  
4  
5  function authorize() {  
6      //code to authorize resources  
7  }  
8  
9  function getData(authenticate, authorize) {  
10  
11      if (authenticate()) {  
12          if (authorize()) {  
13              //fetch the data  
14          }  
15      }  
16  }
```

Higher order function

functions as an argument

In the above example, `getData` is a higher-order function as it takes two functions as arguments.

### **3. What is object destructuring?**

Object destructuring is a way to extract properties from an object and assign them to variables. It makes working with objects simpler and easier to read.

Let's take an example to understand it.

User Profile Object

```
1  const userProfile = {
2    name: 'Alex',
3    email: 'alex@example.com',
4    phone: '555-123-4567'
5  };
6
7  const { name, email } = userProfile;
8
9  console.log(name);
10 //Output: Alex
11
12 console.log(email);
13 //Output: alex@example.com
```

Extracting name and email only and  
assigning them to new variables



## 4. What is the difference between synchronous and asynchronous code?

Synchronous Code	Asynchronous Code
Executes sequentially.	Executes out of order, not waiting for one operation to complete before starting the next.
Can lead to a blocking behavior in applications, potentially making the UI unresponsive if a task takes a long time.	Non-blocking behavior, allowing other tasks to run simultaneously.
<div>Example</div> <pre>const result = someFunction(); console.log(result); console.log('After function');</pre>	<div>Example</div> <pre>fetch('https://api.example.com/data')   .then(response =&gt; response.json())   .then(data =&gt; console.log(data))   .catch(error =&gt; console.error(error)); console.log('After fetch call');</pre>
Not well-suited for tasks that can take an unpredictable amount of time, such as network requests or reading from disk.	Ideal for tasks like network requests, timers, and other operations that shouldn't block the main thread.
<b>Common Methods/Functions:</b> Array.forEach(), Math.max(), etc.	<b>Common Methods/Functions:</b> setTimeout(), fetch(), XMLHttpRequest, Promises, async/await.



## 5. List out and explain mouse events

1. **click**: Occurs when the mouse clicks on an element (mouse button pressed and released).
2. **dblclick**: Happens when the mouse double-clicks on an element.
3. **mousedown**: Triggers when the mouse button is pressed down over an element.
4. **mouseup**: Fires when the mouse button is released over an element.
5. **mousemove**: Occurs when the mouse is moving while over an element.
6. **mouseover**: Triggered when the mouse comes over an element.
7. **mouseout**: Fires when the mouse leaves an element.
8. **mouseenter**: Similar to mouseover, but it does not bubble and does not react to the mouse coming from a child element.
9. **mouseleave**: Similar to mouseout, but it does not bubble and does not react to the mouse going to a child element.
10. **contextmenu**: Occurs when the right button of the mouse is clicked (before the context menu is displayed).