1. **What is a promise in JavaScript?**
2. A callback function
3. An object representing a future value or completion of an async operation
4. A synchronous operation
5. None of the above
6. **Which method is used to handle a resolved promise?**
7. catch()
8. then()
9. finally()
10. reject()
11. **What is the default state of a promise when created?**
12. Fulfilled
13. Rejected
14. Pending
15. Resolved
16. **Which method is used to handle a rejected promise?**
17. then()
18. reject()
19. catch()
20. resolve()
21. **What does Promise.resolve(value) do?**
22. Creates a promise that is immediately fulfilled with value
23. Creates a pending promise
24. Rejects a promise
25. None of the above
26. **Which of the following is true for .then() in a promise?**
27. It must return a value or another promise
28. It is optional and only for rejected cases
29. It cannot chain another .then()
30. None of the above
31. **What happens when a promise is resolved?**
32. It remains in a pending state
33. It transitions to the fulfilled state and executes .then() handlers
34. It transitions to the rejected state
35. It stops execution
36. **What is printed in the console?** javascript Promise.resolve('Hello').then(res => console.log(res));
37. No output
38. Hello
39. Error
40. Undefined
41. **How many .then() handlers can a promise chain?**
42. Only one
43. Two
44. As many as needed
45. None
46. **What happens when you return a value from .then()?**
    1. It resolves to another promise
    2. It is passed to the next .then() handler
    3. It is ignored
    4. An error occurs
47. **What is the output of the following code?**

* const promise = new Promise((resolve, reject) => {  
   resolve('First');  
  });  
  promise  
   .then(res => {  
   console.log(res);  
   return 'Second';  
   })  
   .then(res => console.log(res));
  1. First then Second
  2. Only First
  3. Only Second
  4. No output

1. **What is printed in the console?**

* Promise.resolve('Start')  
   .then(res => {  
   console.log(res);  
   throw new Error('Something went wrong');  
   })  
   .catch(err => console.log(err.message));
  1. Start and Something went wrong
  2. Start only
  3. Something went wrong only
  4. Error

1. **What is logged in the following code?**

* const promise = new Promise((resolve, reject) => {  
   resolve('Hello');  
  });  
  promise  
   .then(res => {  
   console.log(res);  
   return new Promise((resolve, reject) => resolve('World'));  
   })  
   .then(res => console.log(res));
  1. Hello then World
  2. World then Hello
  3. Only Hello
  4. Only World

1. **What happens if a .catch() is added at the end of a fulfilled promise chain?**
   1. The chain will break
   2. The .catch() will be ignored
   3. It will handle any error thrown in the chain
   4. It causes a syntax error
2. **What does this code output?**

* const promise = new Promise((resolve, reject) => {  
   resolve('Step 1');  
  });  
  promise  
   .then(res => {  
   console.log(res);  
   return 'Step 2';  
   })  
   .then(res => {  
   console.log(res);  
   throw new Error('Oops!');  
   })  
   .catch(err => console.log(err.message));
  1. Step 1, Step 2, Oops!
  2. Step 1 then Oops!
  3. Only Step 1
  4. Error

1. **What will the following code output?**

* const promise = new Promise((resolve, reject) => {  
   resolve('First');  
  });  
  promise  
   .then(res => console.log(res))  
   .catch(err => console.log('Caught:', err));
  1. First
  2. Caught: Error
  3. Both First and Caught: Error
  4. No output

1. **What will be logged here?**

* Promise.reject('Error')  
   .then(res => console.log(res))  
   .catch(err => console.log(err))  
   .then(() => console.log('Completed'));
  1. Error then Completed
  2. Only Error
  3. Only Completed
  4. Error

1. **What happens in this scenario?**

* const promise = Promise.resolve('Data');  
  promise.then(res => console.log(res));  
  console.log('End');
  1. Data then End
  2. End then Data
  3. No output
  4. Error

1. **What will be the result of this code?**

* Promise.resolve('Start')  
   .then(res => {  
   console.log(res);  
   return 'Next';  
   })  
   .then(res => console.log(res))  
   .catch(err => console.log('Caught:', err));
  1. Start then Next
  2. Start then Caught: Error
  3. Only Start
  4. Error

1. **What does this produce?**

* const promise = Promise.reject('Failure');  
  promise.catch(err => {  
   console.log(err);  
   return 'Recovered';  
  }).then(res => console.log(res));
  1. Failure then Recovered
  2. Recovered
  3. Only Failure
  4. Error

1. **What is logged here?**

* Promise.resolve()  
   .then(() => {  
   console.log('First');  
   return Promise.resolve();  
   })  
   .then(() => console.log('Second'));
  1. First then Second
  2. Only First
  3. Only Second
  4. Error

1. **What will this code log?**

* const promise = new Promise((resolve, reject) => {  
   resolve('Done');  
  });  
  promise  
   .then(res => {  
   console.log(res);  
   return Promise.reject('Error in Chain');  
   })  
   .catch(err => console.log(err));
  1. Done then Error in Chain
  2. Only Done
  3. Only Error in Chain
  4. Error

1. **What happens here?**

* const promise = Promise.resolve('Resolved');  
  promise  
   .then(() => Promise.reject('Failed'))  
   .catch(err => console.log(err));
  1. Failed
  2. Error
  3. No output
  4. Resolved

1. **What is printed in the console?**

* Promise.resolve('A')  
   .then(res => {  
   console.log(res);  
   return 'B';  
   })  
   .then(res => console.log(res))  
   .catch(() => console.log('Error'));
  1. A then B
  2. A then Error
  3. Only A
  4. Error

1. **What is logged here?**

* const promise = Promise.resolve('Start');  
  promise.then(res => {  
   console.log(res);  
   return Promise.resolve('Middle');  
  }).then(res => console.log(res));
  1. Start then Middle
  2. Middle then Start
  3. Only Start
  4. Error

1. **What does this code log?**

* Promise.resolve('Hello')  
   .then(() => {  
   console.log('World');  
   throw new Error('Oops');  
   })  
   .catch(err => console.log(err.message));
  1. World and Oops
  2. Only World
  3. Only Oops
  4. Error

1. **What is printed in the following code?**

* Promise.reject('Error')  
   .catch(err => {  
   console.log(err);  
   throw new Error('Another Error');  
   })  
   .catch(err => console.log(err.message));
  1. Error then Another Error
  2. Only Error
  3. Only Another Error
  4. Error

1. **What happens in this scenario?**

* const promise = new Promise((resolve, reject) => {  
   reject('Failed');  
  });  
  promise.catch(err => {  
   console.log(err);  
   return 'Recovered';  
  }).then(res => console.log(res));
  1. Failed then Recovered
  2. Only Failed
  3. Only Recovered
  4. Error

1. **What does this produce?**

* const promise = Promise.resolve('Step 1');  
  promise  
   .then(res => {  
   console.log(res);  
   return new Promise((resolve, reject) => {  
   resolve('Step 2');  
   });  
   })  
   .then(res => console.log(res));
  1. Step 1 then Step 2
  2. Only Step 1
  3. Only Step 2
  4. Error

1. **What will this code output?**

* const promise = new Promise((resolve, reject) => {  
   reject('Rejected');  
  });  
  promise  
   .catch(err => {  
   console.log(err);  
   return 'Recovered';  
   })  
   .then(res => console.log(res));
  1. Rejected then Recovered
  2. Only Rejected
  3. Only Recovered
  4. Error

Answers:

b) An object representing a future value or completion of an async operation

b) then()

c) Pending

c) catch()

a) Creates a promise that is immediately fulfilled with value

a) It must return a value or another promise

b) It transitions to the fulfilled state and executes .then() handlers

b) Hello

c) As many as needed

b) It is passed to the next .then() handler

a) First then Second

a) Start and Something went wrong

a) Hello then World

c) It will handle any error thrown in the chain

a) Step 1, Step 2, Oops!

a) First

a) Error then Completed

b) End then Data

a) Start then Next

a) Failure then Recovered

a) First then Second

a) Done then Error in Chain

a) Failed

a) A then B

a) Start then Middle

a) World and Oops

a) Error then Another Error

a) Failed then Recovered

a) Step 1 then Step 2

a) Rejected then Recovered