**1. What will the output of the following code snippet be?**

async function test() {  
 return "Hello";

}  
console.log(test());

1. Hello
2. undefined
3. Promise {: “Hello”}
4. Error

Ans: a(Hello)

**2. Which of the following correctly invokes an async function?**

async function fetchData() {  
 return "Data";  
}

1. fetchData();
2. await fetchData();
3. fetchData().then();
4. Both a and c

Ans: d(Both a and c)

**3. What is the output of the following code?**

async function getNumber() {  
 return 42;  
}  
console.log(await getNumber());

1. 42
2. undefined
3. SyntaxError
4. Promise {: 42}

Ans: c(SyntaxError)

**4. What will happen if await is used outside an async function?**

const data = await fetch("https://example.com");

1. SyntaxError
2. Promise is returned
3. Works normally
4. ReferenceError

Ans: a(SyntaxError)

1. **Which of the following is true about async functions?**  
   a) They always return a Promise  
   b) They can only be used with await  
   c) They block the main thread  
   d) They cannot throw errors

Ans: a(They always return a Promise)

1. **What does await do in an async function?**  
   a) Waits for the promise to resolve or reject  
   b) Converts a Promise to synchronous behavior  
   c) Blocks the main thread until completion  
   d) Both a and b

Ans: d(Both a and b)

**7. What is the output of the following code?**

async function example() {  
 return await 10;  
}  
example().then(console.log);

1. 10
2. undefined
3. Error
4. Promise {: 10}

Ans: a(10)

**8. What happens when an error occurs inside an async function?**

async function errorFunction() {  
 throw new Error("Oops!");  
}  
errorFunction().catch(console.log);

1. The program crashes
2. Error is returned as a rejected Promise
3. Error is ignored
4. Undefined behavior

Ans: b(Error is returned as a rejected Promise)

**9. What is the output of the following code?**

async function delayedLog() {  
 console.log("Before delay");  
 await new Promise(resolve => setTimeout(resolve, 1000));  
 console.log("After delay");  
}  
delayedLog();

1. Before delay, After delay (simultaneously)
2. Before delay (immediately), After delay (after 1 second)
3. Before delay, After delay
4. After delay, Before delay

Ans: b(Before delay (immediately), After delay (after 1 second))

1. **How can you handle a rejected Promise in an async function?**  
   a) Using try-catch  
   b) Using .catch()  
   c) Using Promise.resolve()  
   d) Both a and b

Ans: d(Both a and b)

**11. What is the output of this code?**

async function fetchData() {  
 return Promise.resolve("Done");  
}  
const result = await fetchData();  
console.log(result);

1. Done
2. Promise {: “Done”}
3. undefined
4. Error

Ans: d(Error)

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

async function main() {  
 console.log(await Promise.resolve("Resolved!"));  
}  
main();

1. Resolved!
2. undefined
3. Error
4. Promise {: “Resolved!”}

Ans: a

**13. How does await handle rejected Promises by default?**  
a) Logs the error  
b) Throws an exception  
c) Converts it to undefined  
d) Returns the rejected Promise

Ans: b(Throws an exception)

**14. What is the output of this code?**

async function test() {  
 console.log("1");  
 await null;  
 console.log("2");  
}  
test();  
console.log("3");

1. 1, 2, 3
2. 1, 3, 2
3. 3, 1, 2
4. 1, 2, undefined

Ans: b(1, 3, 2)

1. **What happens when await is used with a non-Promise value?**  
   a) Converts it into a resolved Promise  
   b) Throws a TypeError  
   c) Skips the line  
   d) Returns undefined

Ans: a(Converts it into a resolved Promise)

**16. What is the output of the following code?**

async function sequence() {  
 console.log("Start");  
 await new Promise(resolve => setTimeout(resolve, 1000));  
 console.log("End");  
}  
sequence();  
console.log("Outside");

1. Start, Outside, End
2. Start, End, Outside
3. Outside, Start, End
4. End, Start, Outside

Ans: a(Start, Outside, End)

1. **How do you return multiple resolved values from an async function?**  
   a) Use an array  
   b) Use an object  
   c) Chain Promises  
   d) All of the above

Ans: d

1. **Which is true about chaining Promises with async functions?**  
   a) .then can still be used  
   b) .catch must be used for error handling  
   c) await removes the need for .then  
   d) All of the above

Ans:d

**19. What is the behavior of await when used with a Promise that never resolves?**  
a) Blocks forever  
b) Throws an error  
c) Returns undefined  
d) Causes a memory leak

Ans: a

**20. What is the result of calling await on a resolved Promise in a try block?**

async function main() {  
 try {  
 const result = await Promise.resolve("Success");  
 console.log(result);  
 } catch (error) {  
 console.log(error);  
 }  
}  
main();

1. Success
2. undefined
3. Error: Unhandled Promise Rejection
4. SyntaxError

Ans: a