Promise interview questions

provide list of most frequently asked promise questions and its output/answers and explanation 5 easy, 5 medium, 10 hard level

Here is **the most complete and interview-ready list** of **Promises interview questions** with **actual outputs and clear explanations**—exactly as asked:

✅ 5 EASY PROMISE INTERVIEW QUESTIONS

1. What is the output?

```
js

console.log("A");

Promise.resolve().then(() => console.log("B"));

console.log("C");
```

Output

```
css

A
C
B
```

Explanation

- `A` → normal synchronous log.
- `C` → synchronous.
- `B` → microtask (promise `.then()`) runs **after all sync code**.

---

2. Output of Promise constructor

```
js

new Promise((resolve, reject) => {
  console.log("Inside executor");
  resolve();
});
console.log("Outside");
```

Output

```
scss

Inside executor
Outside
```

Explanation

- Promise executor runs **immediately and synchronously**.
- `.then()` is async, but here no `.then()`.

---

3. Basic chain

```
js

Promise.resolve(10)
  .then(val => val + 5)
  .then(val => console.log(val));
```

Output

```

15
```

Explanation

- Each `.then()` receives result of previous step.

---

4. Rejection handling

```
js

Promise.reject("Error")
  .catch(err => console.log(err));
```

Output

```
javascript

Error
```

---

5. What is the output?

```js
Promise.resolve()
  .then(() => console.log("then"));

console.log("log");
```

Output

```bash
log
then
```

✅ 5 MEDIUM PROMISE QUESTIONS

---

6. Promise chain with return Promise

```js
Promise.resolve(5)
  .then(x => {
    return Promise.resolve(x * 2);
  })
  .then(x => console.log(x));
```

Output

```
10
```

Explanation

- Returning a Promise from **`.then()` flattens the chain** automatically.

---

7. Missing return

```js
Promise.resolve(1)
  .then(x => {
    x + 1;     // No return
  })
  .then(x => console.log(x));
```

Output

```
javascript

undefined
```

Explanation

- If no return, **`.then()`** returns **`undefined`**.

---

8. What is printed?

```
js

Promise.resolve()
  .then(() => {
    console.log("1");
    return "A";
  })
  .then(x => {
    console.log(x);
  });
```

Output

```
css

1
A
```

---

9. Catch then then

```
js

Promise.reject("X")
  .catch(err => {
    console.log(err);
    return "Recovered";
  })
  .then(x => console.log(x));
```

Output

```
nginx

X
Recovered
```

---

10. Promise inside setTimeout

```js
setTimeout(() => console.log("timeout"), 0);

Promise.resolve().then(() => console.log("promise"));

console.log("sync");
```

Output

```bash
sync
promise
timeout
```

Explanation

- Microtask (promise) → before macrotask (setTimeout).

✅ 10 HARD PROMISE INTERVIEW QUESTIONS

---

11. Promise chain with errors

```js
Promise.resolve(1)
  .then(x => {
    throw new Error("Fail");
  })
  .catch(err => {
    console.log("Caught");
    return 10;
  })
  .then(x => console.log(x));
```

Output

```nginx
Caught
10
```

---

12. Double rejection

```js
Promise.reject(1)
  .catch(() => {
    throw 2;
```

```
  })
  .catch(x => console.log(x));
```

Output

```
2
```

---

13. What is the output? (Tricky async ordering)

```
js

console.log("A");

setTimeout(() => console.log("B"), 0);

Promise.resolve().then(() => console.log("C"));

console.log("D");
```

Output

```
css

A
D
C
B
```

---

14. Multiple resolves (ignored after first)

```
js

const p = new Promise((resolve, reject) => {
  resolve(1);
  resolve(2);
  reject(3);
});

p.then(x => console.log(x));
```

Output

```
1
```

Explanation

- Promise can settle **only once**.
- Additional resolve/reject is ignored.

15. Nested promises

```
js

Promise.resolve(1)
  .then(x => {
    console.log(x);
    return Promise.resolve(2);
  })
  .then(x => console.log(x));
```

Output

```


1
2
```

16. Promise.all

```
js

Promise.all([
  Promise.resolve("A"),
  Promise.resolve("B"),
  Promise.resolve("C"),
]).then(console.log);
```

Output

```
css

[ 'A', 'B', 'C' ]
```

17. Promise.race

```
js

Promise.race([
  new Promise(res => setTimeout(() => res("slow"), 100)),
  Promise.resolve("fast"),
]).then(console.log);
```

Output

```
nginx

fast
```

18. Chained async hell

```
js

Promise.resolve()
  .then(() => {
    console.log("1");
    return Promise.resolve();
  })
  .then(() => {
    console.log("2");
    return Promise.reject("Err");
  })
  .catch(err => {
    console.log("3");
  })
  .then(() => {
    console.log("4");
  });
```

Output

```
1
2
3
4
```

---

19. Finally chain

```
js

Promise.resolve("A")
  .finally(() => console.log("final"))
  .then(x => console.log(x));
```

Output

```
cpp

final
A
```

Explanation

- **`finally()`** does **not** receive value
- does not stop the chain

---

20. Microtask queue ordering

```js
Promise.resolve().then(() => console.log("P1"));
Promise.resolve().then(() => console.log("P2"));

queueMicrotask(() => console.log("M1"));

console.log("S");
```

Output

```nginx
S
P1
P2
M1
```

Explanation

Order inside microtask:

1. promise `.then()` callbacks
2. queueMicrotask

Want even deeper?

I can also provide:

✅ 10 super-advanced Promise questions
✅ Real interview questions from MAANG/Top startups
✅ Coding questions using promises (like promise throttle, promise pool)
Just tell me!