Q1. Simple Object Context

```
const laptop = {
    brand: "Dell",
    getBrand: function() {
        return this.brand;
    }
};
const myBrand = laptop.getBrand();
console.log(myBrand);
```

Q2. Basic Promise Flow

```
console.log(1);
Promise.resolve().then(() => {
    console.log(2);
});
console.log(3);
```

Q3. The Broken Chain

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

Q4. Global vs. Local Scope

```
var status = "Offline";

const server = {
    status: "Online",
    getStatus: function() {
        return this.status;
    }
};

console.log(server.getStatus());
```

Q5. Math in Promises

```
Promise.resolve(10)
    .then((num) => num * 2)
    .then((result) => console.log(result));
```

Q6. The "Lost" Context

```
const user = {
    name: "Alex",
    printName() {
        console.log(this.name);
    }
};

const print = user.printName;
print();
```

Q7. Event Loop Basic Race

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

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

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

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

Q8. Arrow Function Pitfall

```
const group = {
    title: "Developers",
    getTitle: () => {
        console.log(this.title);
    }
};

group.getTitle();
```

Q9. Chaining Returns

```
Promise.resolve(5)
    .then((val) => {
        console.log(val);
        return val + 5;
    })
    .then((val) => console.log(val));
```

Q10. Catch and Continue

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

Q11. The Nested Timeout

```
console.log('A');

setTimeout(() => {
    console.log('B');
}, 0);

Promise.resolve().then(() => {
    console.log('C');
    Promise.resolve().then(() => console.log('D'));
});

console.log('E');
```

Q12. Explicit Binding (Call/Apply)

```
const agent = {
    id: 101
};
```

```
function showId() {
    console.log(this.id);
}

showId.call(agent);
showId.apply(null);
```

Q13. Promise.all Failure

```
Promise.all([
    Promise.resolve("Success 1"),
    Promise.reject("Error 1"),
    Promise.resolve("Success 2")
])
.then(res => console.log("Result:", res))
.catch(err => console.log("Caught:", err));
```

Q14. The "Callback" Context Trap

```
const player = {
    score: 50,
    updateScore() {
        setTimeout(function() {
            console.log(this.score);
        }, 100);
    }
};

player.updateScore();
```

Q15. Throwing Inside a Chain

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

```
        return 10;
    })
    .then(x => console.log(x));
```

Q16. Async Function Order

```
async function foo() {
    console.log("A");
    await Promise.resolve();
    console.log("B");
}

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

Q17. The "Finally" Gotcha

```
Promise.resolve("Done")
    .finally(() => {
        console.log("Cleanup");
        return "Modified?";
    })
    .then(res => console.log(res));
```

Q18. Variable Hoisting & Promises

```
console.log(a);
var a = 5;

Promise.resolve().then(() => {
    console.log(a);
});

a = 10;
```

Q19. Microtask vs Macrotask Interleaving

```
setTimeout(() => console.log("T1"), 0);

Promise.resolve().then(() => {
   console.log("P1");
   setTimeout(() => console.log("T2"), 0);
});

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

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

Q20. Object Method Assigned to Class

```
class Manager {
   constructor(name) {
      this.name = name;
   }

   print = () => {
      console.log(this.name);
   }
}

const m = new Manager("Sarah");
const p = m.print;
p();
```