Async/Await & Promise Cheat Sheet

Promise Basics

Creating a Promise

javascript

```
const promise = new Promise((resolve, reject) => {
  // Async operation
  if (success) {
    resolve(value);
  } else {
    reject(error);
  }
});
```

Promise Methods

javascript

```
// Handle success/error
promise
  .then(value => { /* handle success */ })
  .catch(error => { /* handle error */ })
  .finally(() => { /* always executes */ });

// Multiple promises
Promise.all([promise1, promise2]) // Resolves when ALL complete
  .then(values => console.log(values));

Promise.race([promise1, promise2]) // Resolves when FIRST completes
  .then(value => console.log(value));

Promise.allSettled([promise1, promise2]) // Waits for ALL to settle
  .then(results => console.log(results));

Promise.any([promise1, promise2]) // Resolves when FIRST succeeds
  .then(value => console.log(value));
```

Async/Await

Basic Syntax

javascript

```
async function myFunction() {
  try {
    const result = await promise;
    return result;
  } catch (error) {
    console.error(error);
  }
}
```

Error Handling

javascript

```
// Try/catch
async function fetchData() {
  try {
    const response = await fetch(url);
    const data = await response.json();
    return data;
  } catch (error) {
    console.error('Failed:', error);
  }
}

// .catch() alternative
async function fetchData() {
  const response = await fetch(url).catch(handleError);
  return response.json();
}
```

Common Patterns

Sequential Execution

javascript

```
async function sequential() {
  const result1 = await task1();
  const result2 = await task2(result1);
  return result2;
}
```

Parallel Execution

javascript

```
async function parallel() {
  const [result1, result2] = await Promise.all([task1(), task2()]);
  return { result1, result2 };
}
```

Loop with Async/Await

javascript

```
// Sequential loop
for (const item of items) {
  await processItem(item);
}

// Parallel loop
await Promise.all(items.map(item => processItem(item)));
```

Useful Promise Utilities

Timeout Wrapper

javascript

```
function withTimeout(promise, timeoutMs) {
  return Promise.race([
    promise,
    new Promise((_, reject) =>
      setTimeout(() => reject(new Error('Timeout')), timeoutMs)
    )
  ]);
}
```

Retry Pattern

javascript

```
async function retry(fn, retries = 3, delay = 1000) {
  try {
    return await fn();
  } catch (error) {
    if (retries === 0) throw error;
    await new Promise(resolve => setTimeout(resolve, delay));
    return retry(fn, retries - 1, delay * 2);
  }
}
```

Common Gotchas

1. **Don't forget await**:

   javascript

   ```
   // Wrong
   const data = fetch(url); // Returns Promise, not data

   // Correct
   const data = await fetch(url);
   ```

2. **Handle errors properly**:

   javascript

   ```
   // Unhandled rejection
   async function risky() {
     throw new Error('Oops!');
   }

   // Handled
   risky().catch(error => console.error(error));
   ```

3. **Use Promise.all for parallel operations**:

   javascript

   ```
   // Slow (sequential)
   const a = await getA();
   const b = await getB();

   // Fast (parallel)
   const [a, b] = await Promise.all([getA(), getB()]);
   ```

Quick Reference

Pattern	Promise	Async/Await
Create	`new Promise()`	`async function()`
Success	`.then()`	`await`
Error	`.catch()`	`try/catch`
Finally	`.finally()`	`try/finally`
Parallel	`Promise.all()`	`await Promise.all()`

This cheat sheet covers the most common patterns. Remember: **async functions always return Promises**, and **await can only be used inside async functions**!