JavaScript Promises and async/await Cheat Sheet

This cheat sheet covers the basic principles of JavaScript promises and the async/await syntax.

PROMISES

The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.
A promise has three states:

Pending: Initial state, neither fulfilled nor rejected.

Fulfilled: The operation was completed successfully.

Rejected: The operation failed.

Promise

Creates a new Promise object. The constructor is primarily used to wrap functions that do not already support promises.

```
new Promise((resolve, reject) => {
  setTimeout(() => resolve(), 2000);
});
```

Promise.prototype.then()

The .then() method of a JavaScript Promise object can be used to get the eventual result of the asynchronous operation.

```
asyncOperation().then(result => console.log(result));
```

Promise.prototype.catch()

The information for the rejection of the promise is available to the handler supplied in the .catch() method.

```
asyncOperation().catch(err => console.log(err));
```

Promise.prototype.finally()

The handler is called when the promise is settled, whether fulfilled or rejected.

```
asyncOperation().finally(() => console.log('async operation ended!'));
```

Promise.resolve()

Returns a promise that resolves to the value given to it.

```
Promise.resolve(15).then(console.log);
```

Promise.reject()

Returns a promise that rejects with an error given to it.

```
Promise
  .reject(new Error('This is an error!'))
  .catch(console.log);
```

Promise.all([...promises])

Wait for all promises to be resolved, or for any to be rejected.

```
Promise
  .all([promise1, promise2])
  .then(([val1, val2]) => console.log(val1, val2));
```

Promise.allSettled([...promises])

Wait until all promises have settled (each may resolve or reject).

```
Promise
  .allSettled([promise1, promise2])
  .then(results => {
    results.forEach(result => console.log(result.status));
  });
```

Promise.any([...promises])

Takes an iterable of Promise objects and as soon as one of the promises in the iterable fulfills, returns a single promise that resolves with the value from that promise.

```
Promise
  .any([promise1, promise2])
  .then(value => console.log(value));
```

Promise.race([...promises])

Wait until any of the promises is resolved or rejected. The difference with .any is that the outer promise can be rejected if an internal promise gets rejected.

```
Promise
  .race([promise1, promise2])
  .then(value => console.log(value));
```

ASYNC / AWAIT

The async...await syntax in ES6 offers a new way to write more readable and scalable code to handle promises.

Async functions

An async function is a function declared with the async keyword, and the await keyword is permitted within them. Calling an async function always results in a promise.

```
async function asyncOperation(...params) {
  // function code
}
```

Async functions return statements

(async () ⇒ value)()
Returning a value from an async function will always resolve to this value.

```
const getName = async () => "Red Hat";

getName().then(console.log); // output: Red Hat
```

(async () ⇒ throw err)()
Throwing an error from an async function will always reject to that error.

```
const throwError = async () => throw Error("Error...");

throwError().reject(console.error);
```

Await keyword

You can await a promise using the await keyword.

```
(async () => {
  const data = await asyncOperation();
  console.log(data);
})();
```

Note: Top-level await is not yet supported. You can only use the await keyword inside an async function.

Async/Await error handling

You can use **try/catch** blocks to catch rejections from an async function (keep in mind there is also the promises API available to catch errors).

```
const main = async () => {
  try {
    const value = await asyncOperation();
    console.log(value);
  } catch (err) {
    console.log(err);
  }
};

main();
```