Promises, promises

Maker

```
function getSomething() {
  return new Promise(function (resolve, reject) {
    // do something, then call...
    resolve('resolved'); //if successful
    reject('rejected'); //if NOT
  });
}
```

Receiver

```
getSomething()
  .then((data) => {
    console.log(data); // when successful
    return format(data);
    // to chain, ALWAYS return a value
  })
  .then(console.log)
  .catch((err) => {
     // if NOT successful
     console.error(`ERROR: ${err}`);
  });
```

Important Static Methods

```
Promise.all( [promise1, promise2, promise3] ).then().catch();

Promise.allSettled( [promise1, promise2, promise3] ).then().catch();
```

```
Promise.race( [promise1, promise2, promise3] ).then().catch();

Promise.any( [promise1, promise2, promise3] ).then().catch();
```

Author: **Dan Cielos**

Website: dancielos.com

Async / Await

```
const getWeather = async function () {
  // writing async code like a synchronous code
  try {
    const data = await fetch(FETCH_URL);
    const res = await data.json();
    return res;
  } catch (err) {
    return new Error(`Something went wrong.
${err}`);
  }
};
```

```
const getWeatherThen = function () {
  // using the .then() method
  return fetch(FETCH_URL)
    .then((data) => data.json())
    .then((res) => res)
    .catch((err) => new Error(`Something went
wrong. ${err}`));
};
```

```
const weather = await getWeather();
console.log({ weather });

const weather2 = await getWeatherThen();
console.log({ weather2 });
```

```
// if you want to run async codes simultaneously
const getAddress = async function () {
     const [street, city, province, postalCode] =
await Promise.all([
          getStreet(),
          getCity(),
          getProvince(),
          getPostalCode(),
     ]);

     return `${street} ${city}, ${province},
${postalCode}`;
};
```

→ `.then()` vs. `async/await` are generally subjective and I 100% prefer the `async/await`.

→ In this example, the `.then()` looks shorter and simpler. However, the `async/await` is more readable and manageable as it's written like a synchronous code.

Author: **Dan Cielos**

Website: dancielos.com

Order of execution

```
console.log('Hello, World!'); // 1

Promise.resolve('1st promise').then((data) => {
      console.log(data); // 3
});

setTimeout(() => {
      console.log('START of timeout with rejected promise'); // 5
      Promise.reject('Rejected Promise').catch(console.error); // 7
      console.log('END of timeout with rejected promise'); // 6
}, 0);

setTimeout(() => {
      console.log('Timeout callback'); // 8
}, 0);

Promise.resolve('2nd promise').then((data) => {
      console.log(data); // 4
});


console.log('End of code'); // 2
```

Author: **Dan Cielos**

Website: dancielos.com