# Asynchronous JavaScript

## Coding Challenge #1

In this challenge you will build a function `'whereAmI'` which renders a country **only** based on GPS coordinates. For that, you will use a second API to geocode coordinates. So in this challenge, you'll use an API on your own for the first time 😄

**Your tasks:**

**PART 1**

1. Create a function `'whereAmI'` which takes as inputs a latitude value (`'lat'`) and a longitude value (`'lng'`) (these are GPS coordinates, examples are in test data below).
2. Do "reverse geocoding" of the provided coordinates. Reverse geocoding means to convert coordinates to a meaningful location, like a city and country name. Use this API to do reverse geocoding: https://geocode.xyz/api. The AJAX call will be done to a URL with this format: https://geocode.xyz/52.508,13.381?geoit=json. Use the `fetch` API and promises to get the data. Do **not** use the `'getJSON'` function we created, that is cheating 😉
3. Once you have the data, take a look at it in the console to see all the attributes that you received about the provided location. Then, using this data, log a message like this to the console: *"You are in Berlin, Germany"*
4. Chain a `.catch` method to the end of the promise chain and log errors to the console
5. This API allows you to make only 3 requests per second. If you reload fast, you will get this error with code 403. This is an error with the request. Remember, `fetch()` does **not** reject the promise in this case. So create an error to reject the promise yourself, with a meaningful error message

**PART 2**

6. Now it's time to use the received data to render a country. So take the relevant attribute from the geocoding API result, and plug it into the countries API that we have been using.
7. Render the country and catch any errors, just like we have done in the last lecture (you can even copy this code, no need to type the same code)

**Test data:**

- Coordinates 1: 52.508, 13.381 (Latitude, Longitude)
- Coordinates 2: 19.037, 72.873
- Coordinates 3: -33.933, 18.474

**GOOD LUCK 😀**

## Coding Challenge #2

**For this challenge you will actually have to watch the video!** Then, build the image loading functionality that I just showed you on the screen.

**Your tasks:**

Tasks are not super-descriptive this time, so that you can figure out some stuff by yourself. Pretend you're working on your own 😉

**PART 1**

1. Create a function `'createImage'` which receives `'imgPath'` as an input. This function returns a promise which creates a new image (use `document.createElement('img')`) and sets the `.src` attribute to the provided image path
2. When the image is done loading, append it to the DOM element with the `'images'` class, and resolve the promise. The fulfilled value should be the image element itself. In case there is an error loading the image (listen for the `'error'` event), reject the promise
3. If this part is too tricky for you, just watch the first part of the solution

**PART 2**

4. Consume the promise using `.then` and also add an error handler
5. After the image has loaded, pause execution for 2 seconds using the `'wait'` function we created earlier
6. After the 2 seconds have passed, hide the current image (set `display` CSS property to `'none'`), and load a second image (**Hint:** Use the image element returned by the `'createImage'` promise to hide the current image. You will need a global variable for that 😉)
7. After the second image has loaded, pause execution for 2 seconds again
8. After the 2 seconds have passed, hide the current image

**Test data:** Images in the `img` folder. Test the error handler by passing a wrong image path. Set the network speed to “Fast 3G” in the dev tools Network tab, otherwise images load too fast

**GOOD LUCK** 😀

## Coding Challenge #3

**Your tasks:**

**PART 1**

1. Write an `async` function `'loadNPause'` that recreates Challenge #2, this time using `async/await` (only the part where the promise is consumed, reuse the `'createImage'` function from before)
2. Compare the two versions, think about the big differences, and see which one you like more
3. Don't forget to test the error handler, and to set the network speed to "Fast 3G" in the dev tools Network tab

**PART 2**

1. Create an async function `'loadAll'` that receives an array of image paths `'imgArr'`
2. Use `.map` to loop over the array, to load all the images with the `'createImage'` function (call the resulting array `'imgs'`)
3. Check out the `'imgs'` array in the console! Is it like you expected?
4. Use a promise combinator function to actually get the images from the array 😉
5. Add the `'parallel'` class to all the images (it has some CSS styles)

**Test data Part 2:** `['img/img-1.jpg', 'img/img-2.jpg', 'img/img-3.jpg']`. To test, turn off the `'loadNPause'` function

**GOOD LUCK** 😀