JONAS.IO
SCHMEDTMANN

# CODING CHALLENGES FOR ALL SECTIONS

## THE COMPLETE JAVASCRIPT COURSE

@JONASSCHMEDTMAN

# Table of Contents

# Instructions

- The Complete JavaScript Course contains over 25 coding challenges for you to practice the concepts we learn in each section, on your own. **This is a fundamental part of your learning journey** 🤓
- This document contains all coding challenges in this course, so that you can read the challenge text without watching the video (I still recommend watching the videos because I might give some additional clues)
- Some challenges require some starting code, which you can get from this document instead of copying it from the videos
- The solution of each challenge is in the videos themselves, so you still have to go watch them 😉
- Some of the later challenges are quite difficult on purpose (after all, they are called challenges, right? 😅). If one of them is too difficult, don't worry, this is completely normal! Just start watching the solution, then pause the video, and keep working on your own. Struggling is part of the journey, but you can do it 🚀

# JavaScript Fundamentals – Part 1

## Coding Challenge #1

Mark and John are trying to compare their BMI (Body Mass Index), which is calculated using the formula:
`BMI = mass / height ** 2 = mass / (height * height)` (mass in kg and height in meter).

**Your tasks:**

1. Store Mark's and John's mass and height in variables
2. Calculate both their BMIs using the formula (you can even implement both versions)
3. Create a Boolean variable `'markHigherBMI'` containing information about whether Mark has a higher BMI than John.

**Test data:**

- Data 1: Marks weights 78 kg and is 1.69 m tall. John weights 92 kg and is 1.95 m tall.
- Data 2: Marks weights 95 kg and is 1.88 m tall. John weights 85 kg and is 1.76 m tall.

**GOOD LUCK** 😀

## Coding Challenge #2

Use the BMI example from Challenge #1, and the code you already wrote, and improve it.

**Your tasks:**

1. Print a nice output to the console, saying who has the higher BMI. The message is either *"Mark's BMI is higher than John's!"* or *"John's BMI is higher than Mark's!"*
2. Use a template literal to include the BMI values in the outputs. Example: *"Mark's BMI (28.3) is higher than John's (23.9)!"*

**Hint:** Use an `if/else` statement 😉

**GOOD LUCK** 😀

## Coding Challenge #3

There are two gymnastics teams, **Dolphins** and **Koalas**. They compete against each other 3 times. The winner with the highest average score wins a trophy!

**Your tasks:**

1. Calculate the average score for each team, using the test data below
2. Compare the team's average scores to determine the winner of the competition, and print it to the console. Don't forget that there can be a draw, so test for that as well (draw means they have the same average score)
3. **Bonus 1**: Include a requirement for a minimum score of 100. With this rule, a team only wins if it has a higher score than the other team, and the same time a score of at least 100 points. **Hint:** Use a logical operator to test for minimum score, as well as multiple else-if blocks 😉
4. **Bonus 2:** Minimum score also applies to a draw! So a draw only happens when both teams have the same score and both have a score greater or equal 100 points. Otherwise, no team wins the trophy

**Test data:**

- Data 1: Dolphins score 96, 108 and 89. Koalas score 88, 91 and 110
- Data Bonus 1: Dolphins score 97, 112 and 101. Koalas score 109, 95 and 123
- Data Bonus 2: Dolphins score 97, 112 and 101. Koalas score 109, 95 and 106

**GOOD LUCK** 😀

## Coding Challenge #4

Steven wants to build a very simple tip calculator for whenever he goes eating in a restaurant. In his country, it's usual to tip 15% if the bill value is between 50 and 300. If the value is different, the tip is 20%.

**Your tasks:**

1. Calculate the tip, depending on the bill value. Create a variable called `'tip'` for this. It's not allowed to use an if/else statement 😅 (If it's easier for you, you can start with an `if/else` statement, and then try to convert it to a ternary operator!)
2. Print a string to the console containing the bill value, the tip, and the final value (`bill + tip`). Example: *"The bill was 275, the tip was 41.25, and the total value 316.25"*

**Test data:**

- Data 1: Test for bill values 275, 40 and 430

**Hints:**

- To calculate 20% of a value, simply multiply it by 20/100 = 0.2
- Value X is between 50 and 300, if it's `>= 50 && <= 300` 😉

**GOOD LUCK** 😀

# JavaScript Fundamentals – Part 2

## Coding Challenge #1

Back to the two gymnastics teams, the Dolphins and the Koalas! There is a new gymnastics discipline, which works differently.
Each team competes 3 times, and then the average of the 3 scores is calculated (so one average score per team).
A team **only** wins if it has at least **double** the average score of the other team.
Otherwise, no team wins!

**Your tasks:**

1. Create an arrow function `'calcAverage'` to calculate the average of 3 scores
2. Use the function to calculate the average for both teams
3. Create a function `'checkWinner'` that takes the average score of each team as parameters (`'avgDolhins'` and `'avgKoalas'`), and then logs the winner to the console, together with the victory points, according to the rule above. Example: *"Koalas win (30 vs. 13)"*
4. Use the `'checkWinner'` function to determine the winner for both Data 1 and Data 2
5. Ignore draws this time

**Test data:**

- Data 1: Dolphins score 44, 23 and 71. Koalas score 65, 54 and 49
- Data 2: Dolphins score 85, 54 and 41. Koalas score 23, 34 and 27

**Hints:**

- To calculate average of 3 values, add them all together and divide by 3
- To check if number A is at least double number B, check for `A >= 2 * B`. Apply this to the team's average scores 😉

**GOOD LUCK** 😀

## Coding Challenge #2

Steven is still building his tip calculator, using the same rules as before: Tip 15% of the bill if the bill value is between 50 and 300, and if the value is different, the tip is 20%.

**Your tasks:**

1. Write a function `'calcTip'` that takes any bill value as an input and returns the corresponding tip, calculated based on the rules above (you can check out the code from first tip calculator challenge if you need to). Use the function type you like the most. Test the function using a bill value of 100
2. And now let's use arrays! So create an array `'bills'` containing the test data below
3. Create an array `'tips'` containing the tip value for each bill, calculated from the function you created before
4. **Bonus**: Create an array 'total' containing the total values, so the `bill + tip`

**Test data:** 125, 555 and 44

**Hint:** Remember that an array needs a value in each position, and that value can actually be the returned value of a function! So you can just call a function as array values (so don't store the tip values in separate variables first, but right in the new array) 😉

**GOOD LUCK** 😀

## Coding Challenge #3

Let's go back to Mark and John comparing their BMIs! This time, let's use objects to implement the calculations! Remember: `BMI = mass / height ** 2 = mass / (height * height)` (mass in kg and height in meter)

**Your tasks:**

1. For each of them, create an object with properties for their full name, mass, and height (Mark Miller and John Smith)
2. Create a `'calcBMI'` method on each object to calculate the BMI (the same method on both objects). Store the BMI value to a property, and also return it from the method
3. Log to the console who has the higher BMI, together with the full name and the respective BMI. Example: *"John's BMI (28.3) is higher than Mark's (23.9)!"*

**Test data:** Marks weights 78 kg and is 1.69 m tall. John weights 92 kg and is 1.95 m tall.

**GOOD LUCK** 😀

## Coding Challenge #4

Let's improve Steven's tip calculator even more, this time using loops!

**Your tasks:**

1. Create an array `'bills'` containing all 10 test bill values
2. Create empty arrays for the tips and the totals (`'tips'` and `'totals'`)
3. Use the `'calcTip'` function we wrote before (no need to repeat) to calculate tips and total values (`bill + tip`) for every bill value in the bills array. Use a `for` loop to perform the 10 calculations!

**Test data:** 22, 295, 176, 440, 37, 105, 10, 1100, 86 and 52

**Hints:** Call `‘calcTip ‘`in the loop and use the push method to add values to the tips and totals arrays 😉

**Bonus:**

4. **Bonus:** Write a function `'calcAverage'` which takes an array called `'arr'` as an argument. This function calculates the average of all numbers in the given array. This is a **difficult** challenge (we haven't done this before)! Here is how to solve it:
    4.1. First, you will need to add up all values in the array. To do the addition, start by creating a variable `'sum'` that starts at 0. Then loop over the array using a `for` loop. In each iteration, add the current value to the `'sum'` variable. This way, by the end of the loop, you have all values added together
    4.2. To calculate the average, divide the sum you calculated before by the `length` of the array (because that's the number of elements)
    4.3. Call the function with the `'totals'` array

**GOOD LUCK** 😀

# Developer Skills & Editor Setup

## Coding Challenge #1

Given an array of forecasted maximum temperatures, the thermometer displays a string with the given temperatures. **Example:** `[17, 21, 23]` will print *"... 17ºC in 1 days ... 21ºC in 2 days ... 23ºC in 3 days ..."*

**Your tasks:**

1. Create a function `'printForecast'` which takes in an array `'arr'` and logs a string like the above to the console. Try it with both test datasets.
2. Use the problem-solving framework: Understand the problem and break it up into sub-problems!

**Test data:**

- Data 1: `[17, 21, 23]`
- Data 2: `[12, 5, -5, 0, 4]`

**GOOD LUCK** 😀

# JavaScript in the Browser: DOM and Events

## Coding Challenge #1

Implement a game rest functionality, so that the player can make a new guess!

**Your tasks:**

1. Select the element with the `'again'` class and attach a click event handler
2. In the handler function, restore initial values of the `'score'` and `'secretNumber'` variables
3. Restore the initial conditions of the message, number, score and guess input fields
4. Also restore the original background color (#222) and number width (`15rem`)

**GOOD LUCK** 😀

# Data Structures, Modern Operators and Strings

## Coding Challenge #1

We're building a football betting app (soccer for my American friends 😅)!
Suppose we get data from a web service about a certain game (`'game'` variable on next page). In this challenge we're gonna work with that data.

**Your tasks:**

1. Create one player array for each team (variables `'players1'` and `'players2'`)
2. The first player in any player array is the goalkeeper and the others are field players. For Bayern Munich (team 1) create one variable (`'gk'`) with the goalkeeper's name, and one array (`'fieldPlayers'`) with all the remaining 10 field players
3. Create an array `'allPlayers'` containing all players of both teams (22 players)
4. During the game, Bayern Munich (team 1) used 3 substitute players. So create a new array (`'players1Final'`) containing all the original team1 players plus *'Thiago'*, *'Coutinho'* and *'Perisic'*
5. Based on the `game.odds` object, create one variable for each odd (called `'team1'`, `'draw'` and `'team2'`)
6. Write a function (`'printGoals'`) that receives an arbitrary number of player names (**not** an array) and prints each of them to the console, along with the number of goals that were scored in total (number of player names passed in)
7. The team with the lower odd is more likely to win. Print to the console which team is more likely to win, **without** using an `if/else` statement or the ternary operator.

**Test data for 6.:** First, use players *'Davies'*, *'Muller'*, *'Lewandowski'* and *'Kimmich'*. Then, call the function again with players from `game.scored`

**GOOD LUCK** 😀

```
const game = {
  team1: 'Bayern Munich',
  team2: 'Borrussia Dortmund',
  players: [
    [
      'Neuer',
      'Pavard',
      'Martinez',
      'Alaba',
      'Davies',
      'Kimmich',
      'Goretzka',
      'Coman',
      'Muller',
      'Gnarby',
      'Lewandowski',
    ],
    [
      'Burki',
      'Schulz',
      'Hummels',
      'Akanji',
      'Hakimi',
      'Weigl',
      'Witsel',
      'Hazard',
      'Brandt',
      'Sancho',
      'Gotze',
    ],
  ],
  score: '4:0',
  scored: ['Lewandowski', 'Gnarby', 'Lewandowski',
  'Hummels'],
  date: 'Nov 9th, 2037',
  odds: {
    team1: 1.33,
    x: 3.25,
    team2: 6.5,
  },
};
```

## Coding Challenge #2

Let's continue with our football betting app! Keep using the `'game'` variable from before.

**Your tasks:**

1. Loop over the `game.scored` array and print each player name to the console, along with the goal number (Example: *"Goal 1: Lewandowski"*)
2. Use a loop to calculate the average odd and log it to the console (We already studied how to calculate averages, you can go check if you don't remember)
3. Print the 3 odds to the console, but in a nice formatted way, exactly like this:
   *Odd of victory Bayern Munich: 1.33*
   *Odd of draw: 3.25*
   *Odd of victory Borrussia Dortmund: 6.5*
   Get the team names directly from the game object, don't hardcode them (except for *"draw"*). **Hint:** Note how the odds and the game objects have the same property names 😉
4. **Bonus:** Create an object called `'scorers'` which contains the names of the players who scored as properties, and the number of goals as the value. In this game, it will look like this:
   ```
   {
     Gnarby: 1,
     Hummels: 1,
     Lewandowski: 2
   }
   ```

**GOOD LUCK** 😀

## Coding Challenge #3

Let's continue with our football betting app! This time, we have a map called `'gameEvents'` (see below) with a log of the events that happened during the game. The values are the events themselves, and the keys are the minutes in which each event happened (a football game has 90 minutes plus some extra time).

**Your tasks:**

1. Create an array `'events'` of the different game events that happened (no duplicates)
2. After the game has finished, is was found that the yellow card from minute 64 was unfair. So remove this event from the game events log.
3. Compute and log the following string to the console: *"An event happened, on average, every 9 minutes"* (keep in mind that a game has 90 minutes)
4. Loop over `'gameEvents'` and log each element to the console, marking whether it's in the first half or second half (after 45 min) of the game, like this:
   *[FIRST HALF] 17: ⚽ GOAL*

**GOOD LUCK** 😀

```
const gameEvents = new Map([
  [17, '⚽ GOAL'],
  [36, '🔁 Substitution'],
  [47, '⚽ GOAL'],
  [61, '🔁 Substitution'],
  [64, '🔶 Yellow card'],
  [69, '🔴 Red card'],
  [70, '🔁 Substitution'],
  [72, '🔁 Substitution'],
  [76, '⚽ GOAL'],
  [80, '⚽ GOAL'],
  [92, '🔶 Yellow card'],
]);
```

## Coding Challenge #4

Write a program that receives a list of variable names written in underscore_case and convert them to camelCase.
The input will come from a `textarea` inserted into the DOM (see code below to insert the elements), and conversion will happen when the button is pressed.

**Test data (pasted to `textarea`, including spaces):**

```
underscore_case
 first_name
Some_Variable
  calculate_AGE
delayed_departure
```

**Should produce this output (5 separate `console.log` outputs):**

```
underscoreCase      ✅
firstName           ✅✅
someVariable        ✅✅✅
calculateAge        ✅✅✅✅
delayedDdeparture   ✅✅✅✅✅
```

**Hints:**

- Remember which character defines a new line in the `textarea` 😉
- The solution only needs to work for a variable made out of 2 words, like a_b
- Start without worrying about the ✅. Tackle that only after you have the variable name conversion working 😉
- This challenge is difficult on purpose, so start watching the solution in case you're stuck. Then pause and continue!

**Afterwards, test with your own test data!**

**GOOD LUCK** 😀

```
document.body.append(document.createElement('textarea'));
document.body.append(document.createElement('button'));
```

# A Closer Look at Functions

## Coding Challenge #1

Let's build a simple poll app!
A poll has a question, an array of options from which people can choose, and an array with the number of replies for each option. This data is stored in the starter `'poll'` object below.

**Your tasks:**

1. Create a method called `'registerNewAnswer'` on the `'poll'` object. The method does 2 things:
   1.1. Display a `prompt` window for the user to input the number of the selected option. The `prompt` should look like this:
   *What is your favourite programming language?*
   *0: JavaScript*
   *1: Python*
   *2: Rust*
   *3: C++*
   *(Write option number)*
   1.2. Based on the input number, update the `'answers'` array property. For example, if the option is 3, increase the value **at position** 3 of the array by 1. Make sure to check if the input is a number and if the number makes sense (e.g. answer 52 wouldn't make sense, right?)
2. Call this method whenever the user clicks the *"Answer poll"* button.
3. Create a method `'displayResults'` which displays the poll results. The method takes a string as an input (called `'type'`), which can be either *'string'* or *'array'*. If type is *'array'*, simply display the results array as it is, using `console.log()`. This should be the default option. If type is *'string'*, display a string like *"Poll results are 13, 2, 4, 1".*
4. Run the `'displayResults'` method at the end of each `'registerNewAnswer'` method call.
5. **Bonus:** Use the `'displayResults'` method to display the 2 arrays in the test data. Use both the *'array'* and the *'string'* option. Do **not** put the arrays in the poll object! So what should the this keyword look like in this situation?

**Test data for bonus:**

- Data 1: `[5, 2, 3]`
- Data 2: `[1, 5, 3, 9, 6, 1]`

**Hints:** Use many of the tools you learned about in this and the last section 😉

**GOOD LUCK** 😀

```
const poll = {
  question: "What is your favourite programming language?",
  options: ["0: JavaScript", "1: Python", "2: Rust", "3:
  C++"],
  // This generates [0, 0, 0, 0]. More in the next section!
  answers: new Array(4).fill(0),
};
```

## Coding Challenge #2

This is more of a *thinking* challenge than a *coding* challenge 🤓

**Your tasks:**

1. Take the IIFE below and at the end of the function, attach an event listener that changes the color of the selected h1 element (`'header'`) to blue, each time the body element is clicked. Do **not** select the h1 element again!
2. And now explain to **yourself** (or someone around you) **why** this worked! Take all the time you need. Think about **when** exactly the callback function is executed, and what that means for the variables involved in this example.

```
(function () {
  const header = document.querySelector('h1');
  header.style.color = 'red';
})();
```

**GOOD LUCK** 😀

# Working With Arrays

## Coding Challenge #1

Julia and Kate are doing a study on dogs. So each of them asked 5 dog owners about their dog's age, and stored the data into an array (one array for each). For now, they are just interested in knowing whether a dog is an adult or a puppy.
A dog is an adult if it is at least 3 years old, and it's a puppy if it's less than 3 years old.

**Your tasks:**

Create a function `'checkDogs'`, which accepts 2 arrays of dog's ages (`'dogsJulia'` and `'dogsKate'`), and does the following things:

1. Julia found out that the owners of the **first** and the **last two** dogs actually have cats, not dogs! So create a shallow copy of Julia's array, and remove the cat ages from that copied array (because it's a bad practice to mutate function parameters)
2. Create an array with both Julia's (corrected) and Kate's data
3. For each remaining dog, log to the console whether it's an adult (*"Dog number 1 is an adult, and is 5 years old"*) or a puppy (*"Dog number 2 is still a puppy 🐶"*)
4. Run the function for both test datasets

**Test data:**

- Data 1: Julia's data `[3, 5, 2, 12, 7]`, Kate's data `[4, 1, 15, 8, 3]`
- Data 2: Julia's data `[9, 16, 6, 8, 3]`, Kate's data `[10, 5, 6, 1, 4]`

**Hints:** Use tools from all lectures in this section so far 😉

**GOOD LUCK** 😀

## Coding Challenge #2

Let's go back to Julia and Kate's study about dogs. This time, they want to convert dog ages to human ages and calculate the average age of the dogs in their study.

**Your tasks:**

Create a function `'calcAverageHumanAge'`, which accepts an arrays of dog's ages (`'ages'`), and does the following things in order:

1. Calculate the dog age in human years using the following formula: if the dog is `<= 2` years old, `humanAge = 2 * dogAge`. If the dog is `> 2` years old, `humanAge = 16 + dogAge * 4`
2. Exclude all dogs that are less than 18 human years old (which is the same as keeping dogs that are at least 18 years old)
3. Calculate the average human age of all adult dogs (you should already know from other challenges how we calculate averages 😉)
4. Run the function for both test datasets

**Test data:**

- Data 1: `[5, 2, 4, 1, 15, 8, 3]`
- Data 2: `[16, 6, 10, 5, 6, 1, 4]`

**GOOD LUCK** 😀

## Coding Challenge #3

Rewrite the `'calcAverageHumanAge'` function from Challenge #2, but this time as an arrow function, and using chaining!

**Test data:**

- Data 1: `[5, 2, 4, 1, 15, 8, 3]`
- Data 2: `[16, 6, 10, 5, 6, 1, 4]`

**GOOD LUCK** 😀

## Coding Challenge #4

Julia and Kate are still studying dogs, and this time they are studying if dogs are eating too much or too little.
Eating too much means the dog's current food portion is larger than the recommended portion, and eating too little is the opposite.
Eating an okay amount means the dog's current food portion is within a range 10% above and 10% below the recommended portion (see hint).

**Your tasks:**

1. Loop over the `'dogs'` array containing dog objects, and for each dog, calculate the recommended food portion and add it to the object as a new property. Do **not** create a new array, simply loop over the array. Forumla: `recommendedFood = weight ** 0.75 * 28`. (The result is in grams of food, and the weight needs to be in kg)
2. Find Sarah's dog and log to the console whether it's eating too much or too little. **Hint:** Some dogs have multiple owners, so you first need to find Sarah in the owners array, and so this one is a bit tricky (on purpose) 🤓
3. Create an array containing all owners of dogs who eat too much (`'ownersEatTooMuch'`) and an array with all owners of dogs who eat too little (`'ownersEatTooLittle'`).
4. Log a string to the console for each array created in 3., like this: *"Matilda and Alice and Bob's dogs eat too much!"* and *"Sarah and John and Michael's dogs eat too little!"*
5. Log to the console whether there is any dog eating **exactly** the amount of food that is recommended (just `true` or `false`)
6. Log to the console whether there is any dog eating an **okay** amount of food (just `true` or `false`)
7. Create an array containing the dogs that are eating an **okay** amount of food (try to reuse the condition used in 6.)
8. Create a shallow copy of the `'dogs'` array and sort it by recommended food portion in an ascending order (keep in mind that the portions are inside the array's objects 😉)

**Hints:**

- Use many different tools to solve these challenges, you can use the summary lecture to choose between them 😉
- Being within a range 10% above and below the recommended portion means: `current > (recommended * 0.90) && current < (recommended * 1.10)`. Basically, the current portion should be between 90% and 110% of the recommended portion.

**Test data:**

```
const dogs = [
  { weight: 22, curFood: 250, owners: ['Alice', 'Bob'] },
  { weight: 8, curFood: 200, owners: ['Matilda'] },
  { weight: 13, curFood: 275, owners: ['Sarah', 'John'] },
  { weight: 32, curFood: 340, owners: ['Michael'] },
];
```

**GOOD LUCK** 😀

# Object Oriented Programming (OOP)

## Coding Challenge #1

**Your tasks:**

1. Use a constructor function to implement a `'Car'`. A car has a `'make'` and a `'speed'` property. The `'speed'` property is the current speed of the car in km/h
2. Implement an `'accelerate'` method that will increase the car's speed by 10, and log the new speed to the console
3. Implement a `'brake'` method that will decrease the car's speed by 5, and log the new speed to the console
4. Create 2 `'Car'` objects and experiment with calling `'accelerate'` and `'brake'` multiple times on each of them

**Test data:**

- Data car 1: *'BMW'* going at 120 km/h
- Data car 2: *'Mercedes'* going at 95 km/h

**GOOD LUCK** 😀

## Coding Challenge #2

**Your tasks:**

1. Re-create Challenge #1, but this time using an ES6 class (call it `'CarCl'`)
2. Add a getter called `'speedUS'` which returns the current speed in mi/h (divide by 1.6)
3. Add a setter called `'speedUS'` which sets the current speed in mi/h (but converts it to km/h before storing the value, by multiplying the input by 1.6)
4. Create a new car and experiment with the `'accelerate'` and `'brake'` methods, and with the getter and setter.

**Test data:**

- Data car 1: *'Ford'* going at 120 km/h

**GOOD LUCK** 😀

## Coding Challenge #3

**Your tasks:**

1. Use a constructor function to implement an Electric Car (called `'EV'`) as a **child "class"** of `'Car'`. Besides a make and current speed, the `'EV'` also has the current battery charge in % (`'charge'` property)
2. Implement a `'chargeBattery'` method which takes an argument `'chargeTo'` and sets the battery charge to `'chargeTo'`
3. Implement an `'accelerate'` method that will increase the car's speed by 20, and decrease the charge by 1%. Then log a message like this: *'Tesla going at 140 km/h, with a charge of 22%'*
4. Create an electric car object and experiment with calling `'accelerate'`, `'brake'` and `'chargeBattery'` (charge to 90%). Notice what happens when you `'accelerate'`! **Hint:** Review the definiton of polymorphism 😉

**Test data:**

- Data car 1: *'Tesla'* going at 120 km/h, with a charge of 23%

**GOOD LUCK** 😀

## Coding Challenge #4

**Your tasks:**

1. Re-create Challenge #3, but this time using ES6 classes: create an `'EVCl'` child class of the `'CarCl'` class
2. Make the `'charge'` property private
3. Implement the ability to chain the `'accelerate'` and `'chargeBattery'` methods of this class, and also update the `'brake'` method in the `'CarCl'` class. Then experiment with chaining!

**Test data:**

- Data car 1: *'Rivian'* going at 120 km/h, with a charge of 23%

**GOOD LUCK** 😀

# 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** 😀