**FONTEND ADVANCED CODING CHALLENGES**

**JAVASCRIPT MODULE II**

**DATA STRUCTURES, MODERN OPERATORS AND STRINGS**

**CODING CHALLENGE PART 5**

**Coding Challenge #1**

We're building a football betting app !

Suppose we get data from a web service about a certain game ('game' variable mentioned below) In this challenge we're going to 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

*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
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

}

**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:*

*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

delayedDeparture

**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

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

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

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

**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

**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

**Coding Challenge #3**

**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%

**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.
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**

1. 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.
2. 7. Render the country and catch any errors

**Test data:**

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