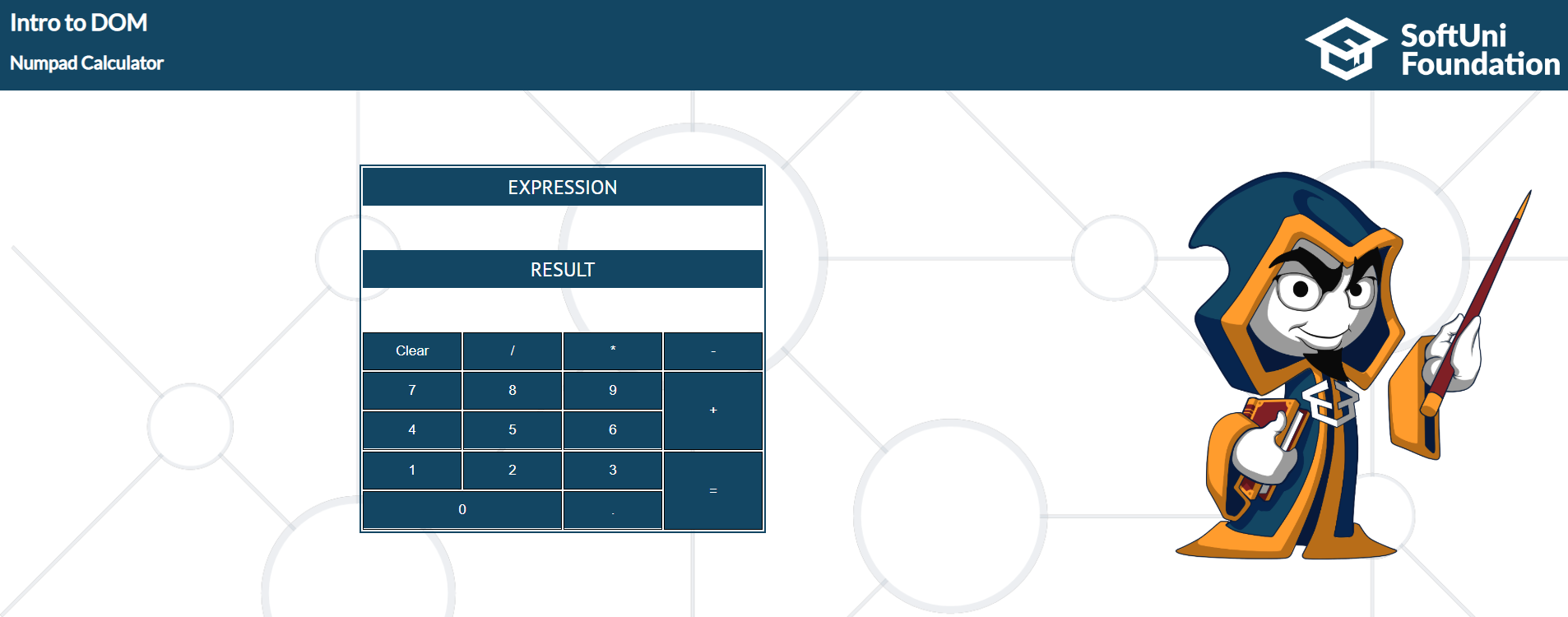
# More Exercises: Intro to DOM

Problems for more exercises for the ["JavaScript Fundamentals" course @ SoftUni](https://softuni.bg/trainings/2247/js-fundamentals-january-2019). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1427>

## Numpad calculator

In this problem, you should **create a JS functonality** on this calculator to work as expected:

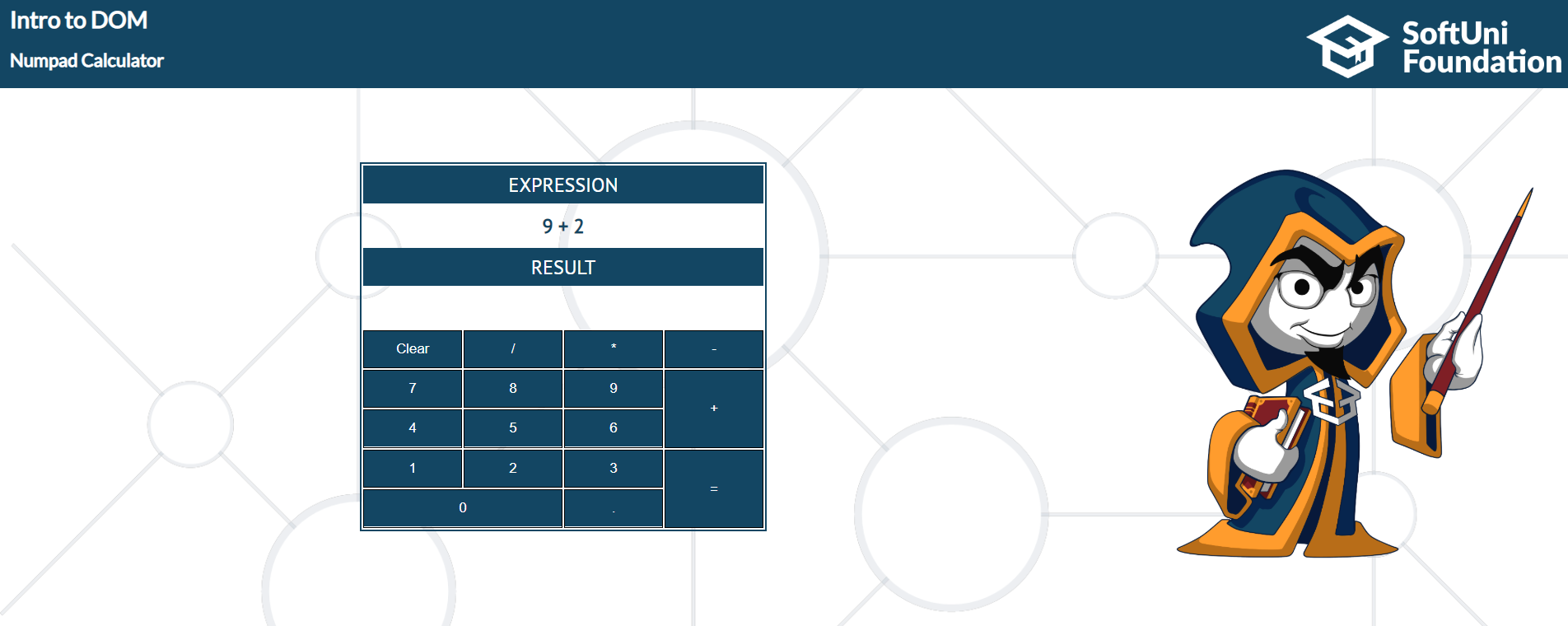


When one of **the buttons is clicked, its value** should be visualized into the "**Expression**" **field** (**#expressionOutput**).

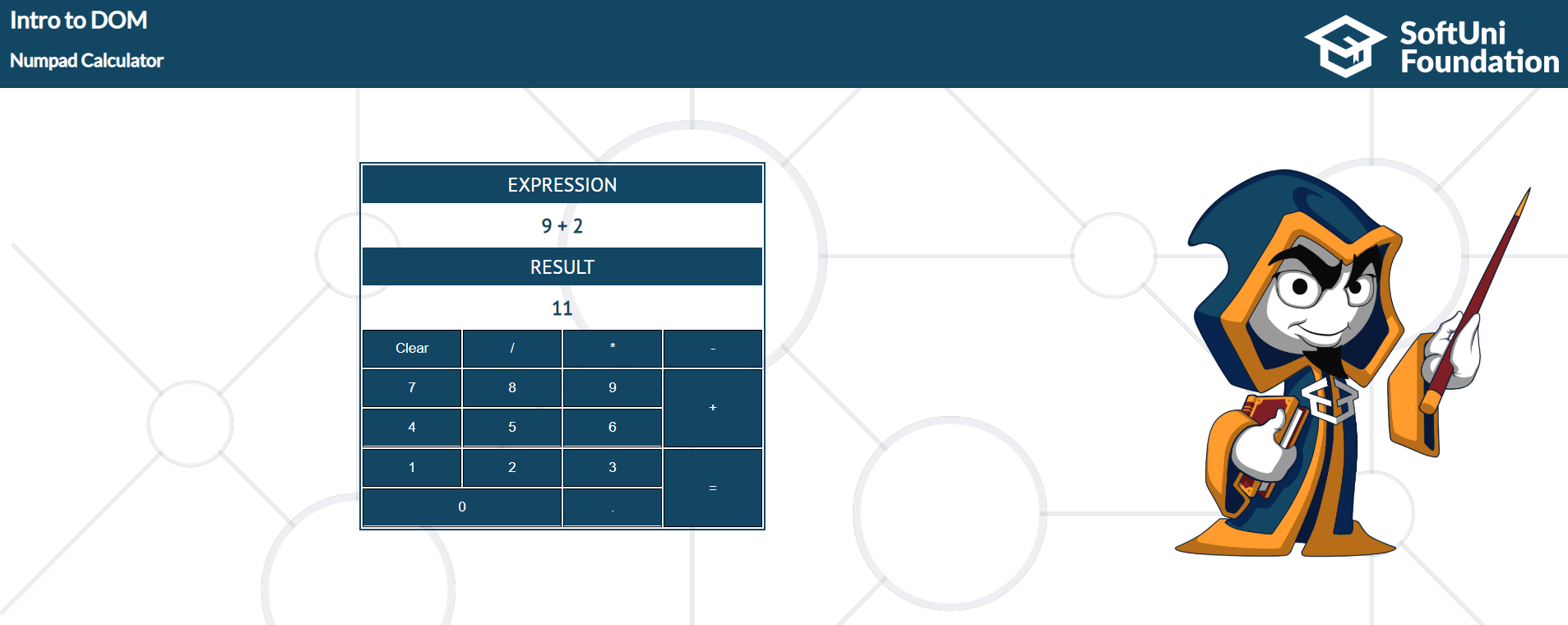
For instance, if we click on the button with value **9**, the expected result will be:



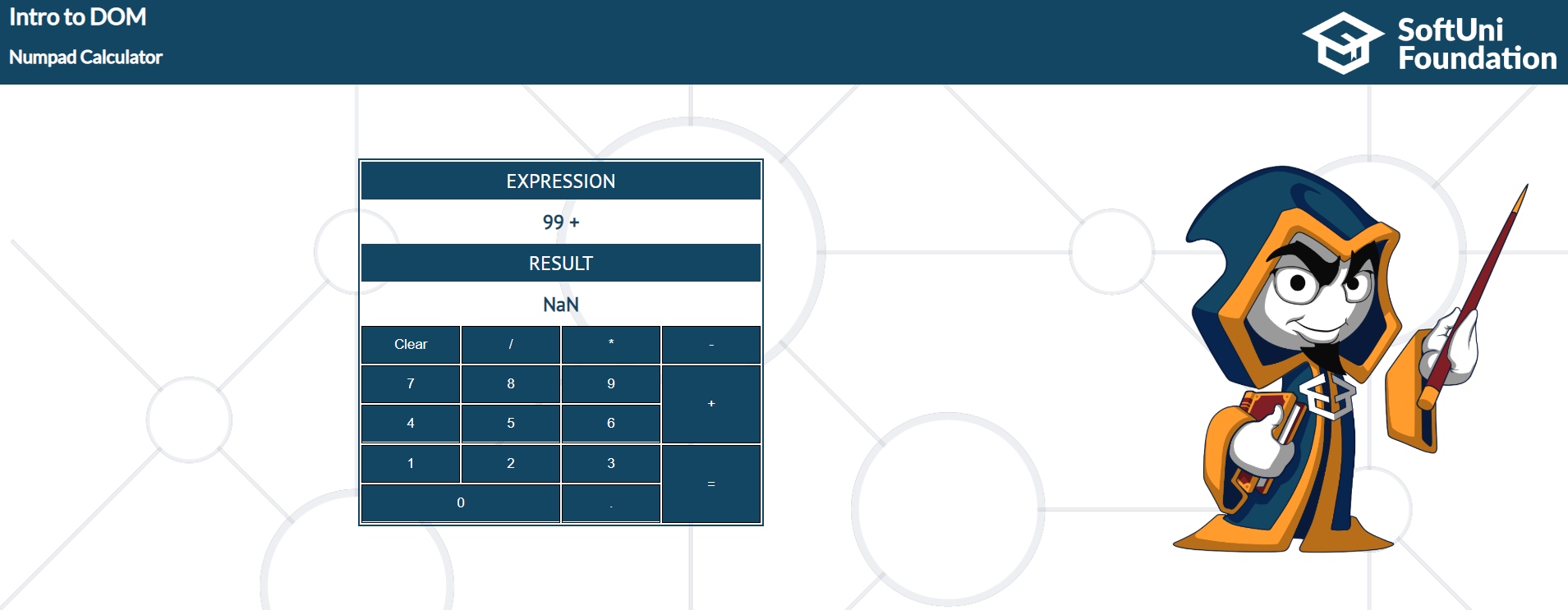
If the current Expression field (**#expresisonOutput**) contains the whole math expression (**left operand**, **operator** and **right operand**): **Example: 9 + 2**



When we press the **equal sign "=", the result of that expression** should appear in the **Result** field (**#resultOutput**).

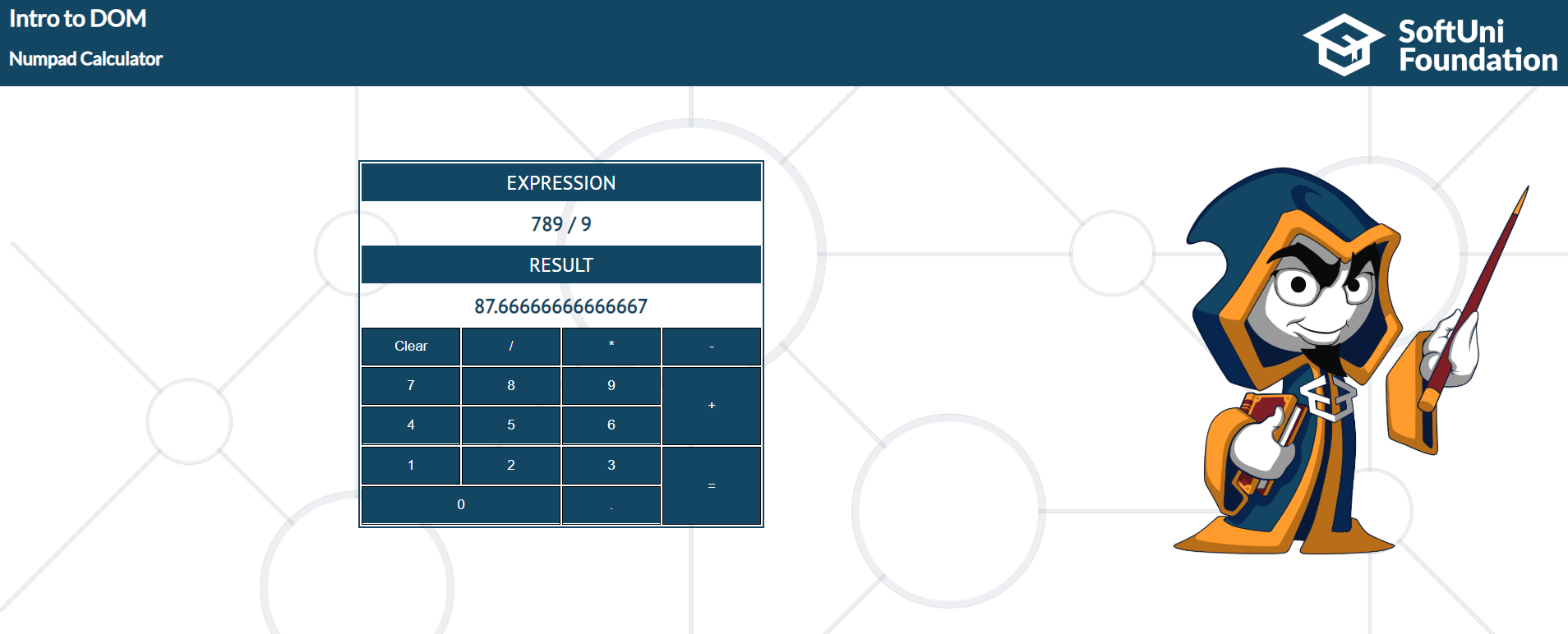


Otherwise, if we try something like "**99 +**" (**without second/right operand**) and we hit equal sign "=" after that, the expected result should be:

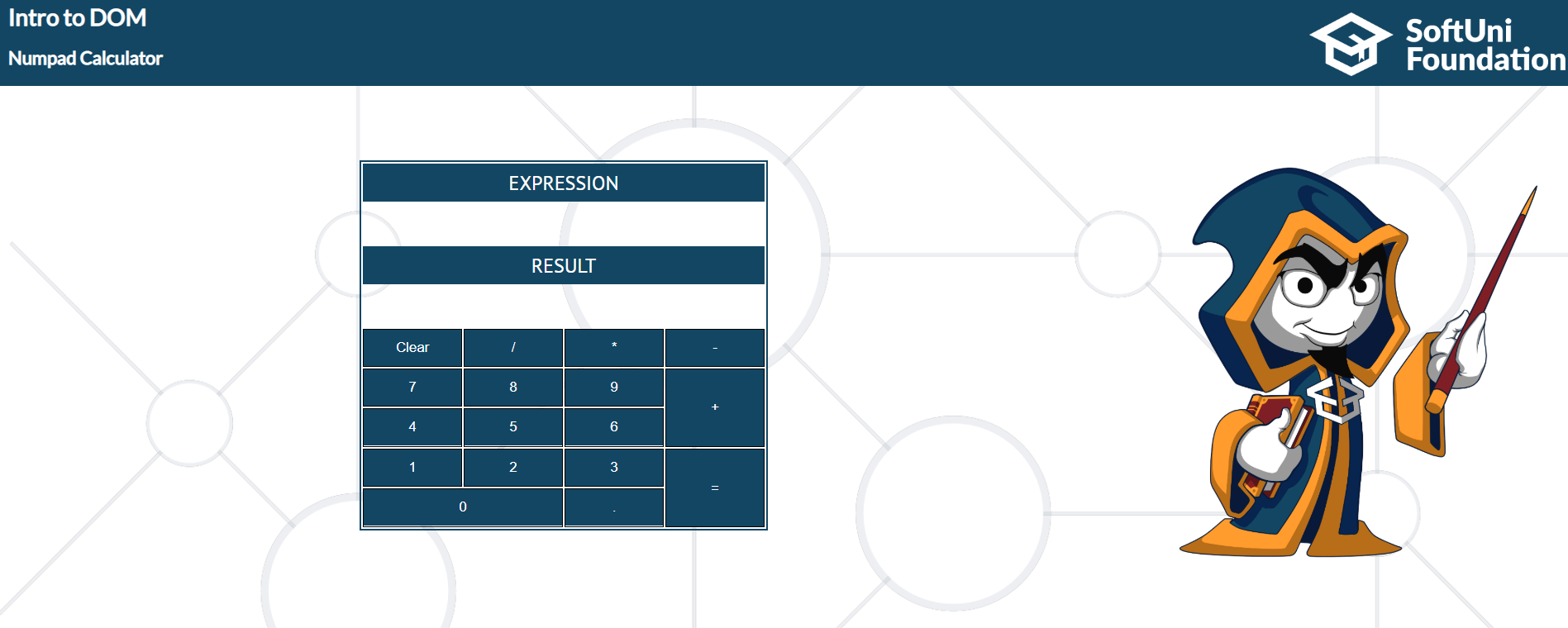


The "**Clear**" button **clears both fields Expression and Result (#expressionOutput and #resultOutput).**

For instance, if we have:

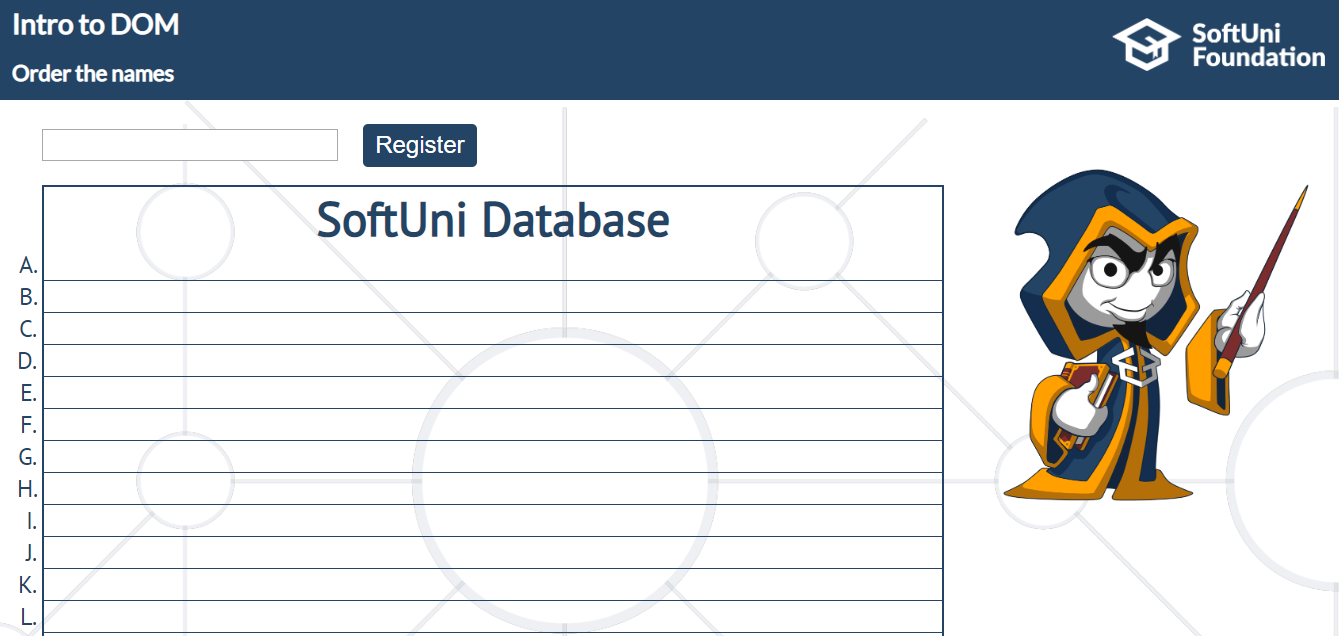


And we hit "**Clear**", the expected result, should be:



## Order the names

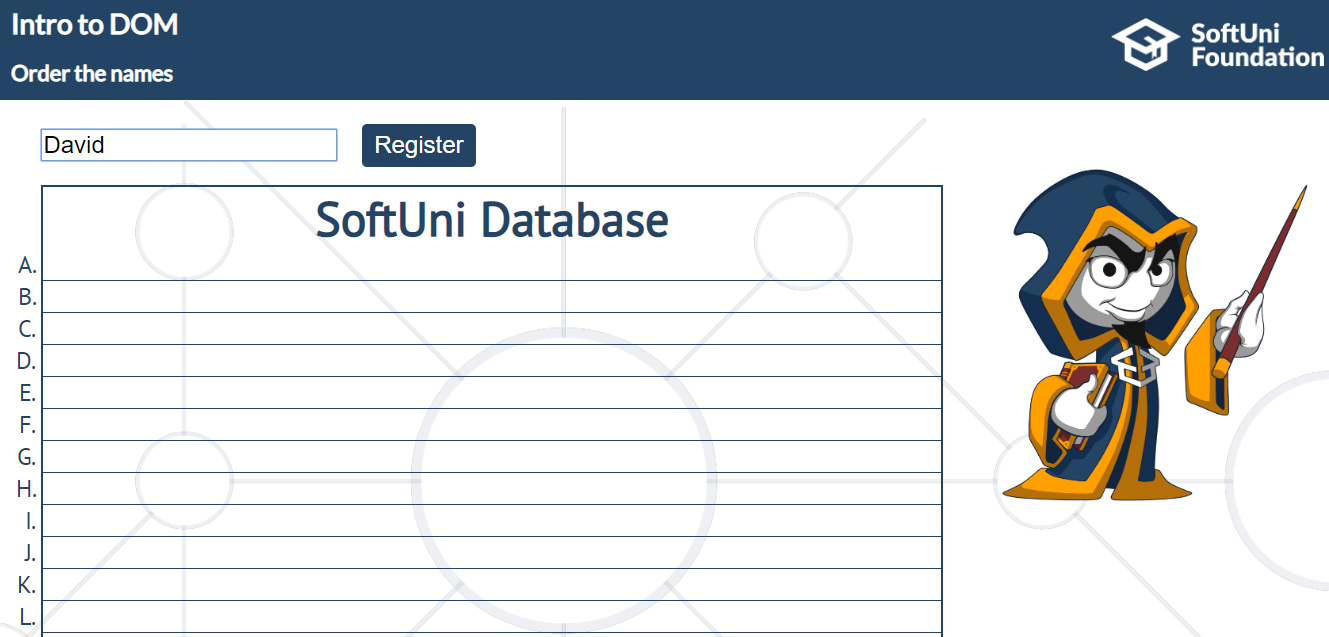
In this problem, you should **create a JS functonality that orders the names alphabetically.**



The input field receives a **name of a student**. When the "**Register**" button is **clicked**. You should

add this student name in the SoftUni Database, while **keeping** the **alphabetial order** **in it**.

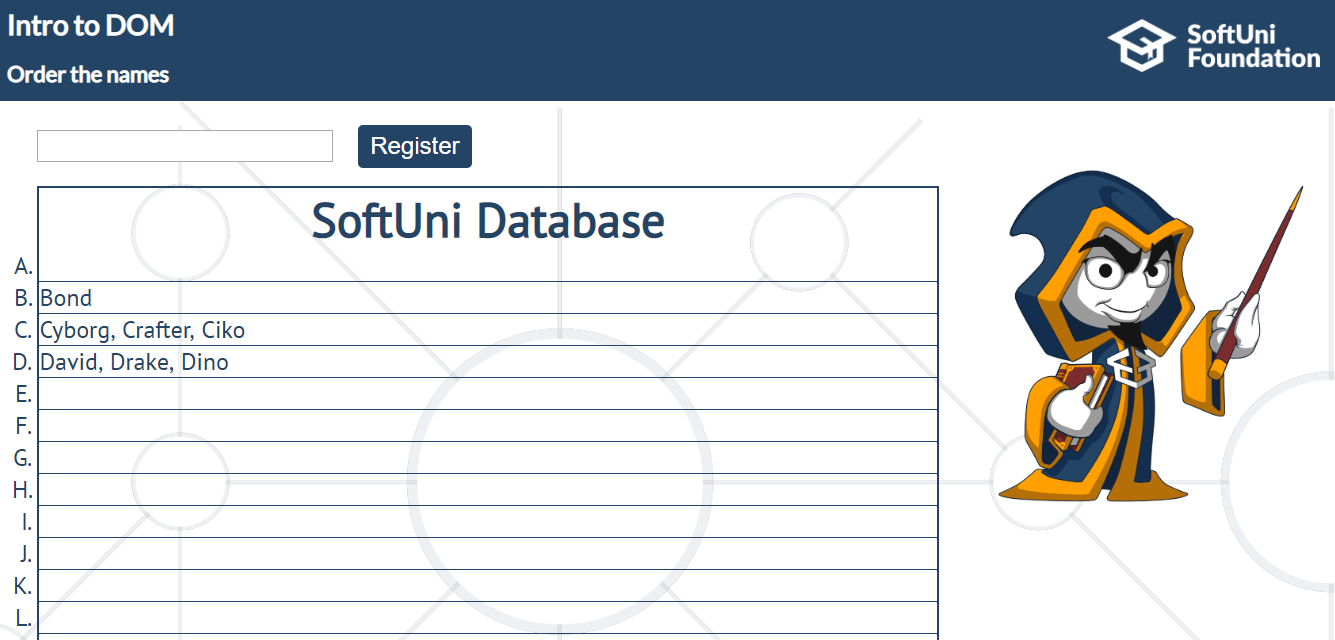
For instance, if we register **David,** his name should appear in the **D** column.





If you **receive more than one name with the same starting letter**, you should **join all names** by

comma and space ("**,** ").



## Robot race

In this problem, you should **create a JS functonality** that **simulates a race** between these three

"robots" and finds out which one is the **winner**.



The first input receives only the **robot name** (“softuni”, “google” or “facebook”);

The second input receives the **direction** and **distance** (forward 50 or backward 30);

When "Do It" **button is clicked** and the input fields are filled with **correct information**, the given robot should move depending on the direction and the given distance.

**Note**: After every robot move, you should **message** the "audience" what has happened.

If the current robot moves less than 80% forward, you should print **`${name} move ${distance} forward`.**

The first one that hits 80% forward is the winner and you should print**: ${name.toUpperCase()} WIN THE RACE!**. Also, you should keep track of who has **finished** **second** and **third**!

And having finished, a robot **cannot move** **backwards** because he has already finished the race.

If one of the robots moves backwards and the calculated distance is **greater** or **equal** to **0**, you should print:

**${name} was distracted and he got behind with ${distance} meters.**

If the current robot is at the beginning or if you want to move the robot **backwards more** than it has been moved forward, you should print:

**${name} can't move so backward**

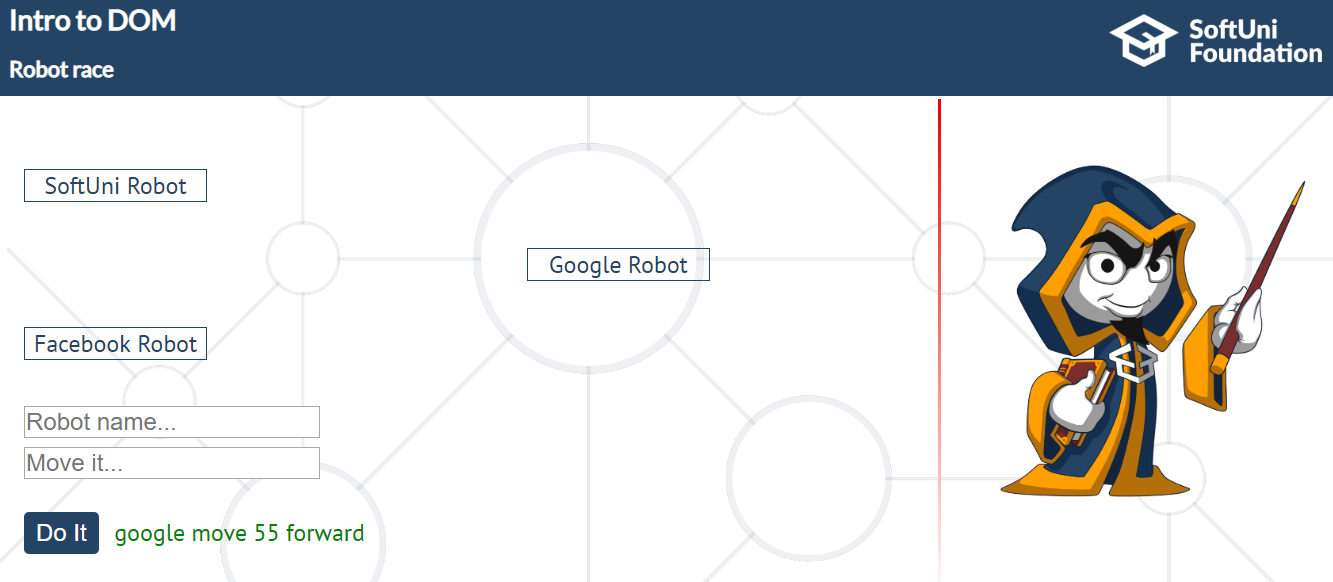
Otherwise, if you try to move a robot more than 80% or it has already finished and you try to move it again forward, you should print: **${name} can't move so forward.**

**The messages should appear into the span element.**

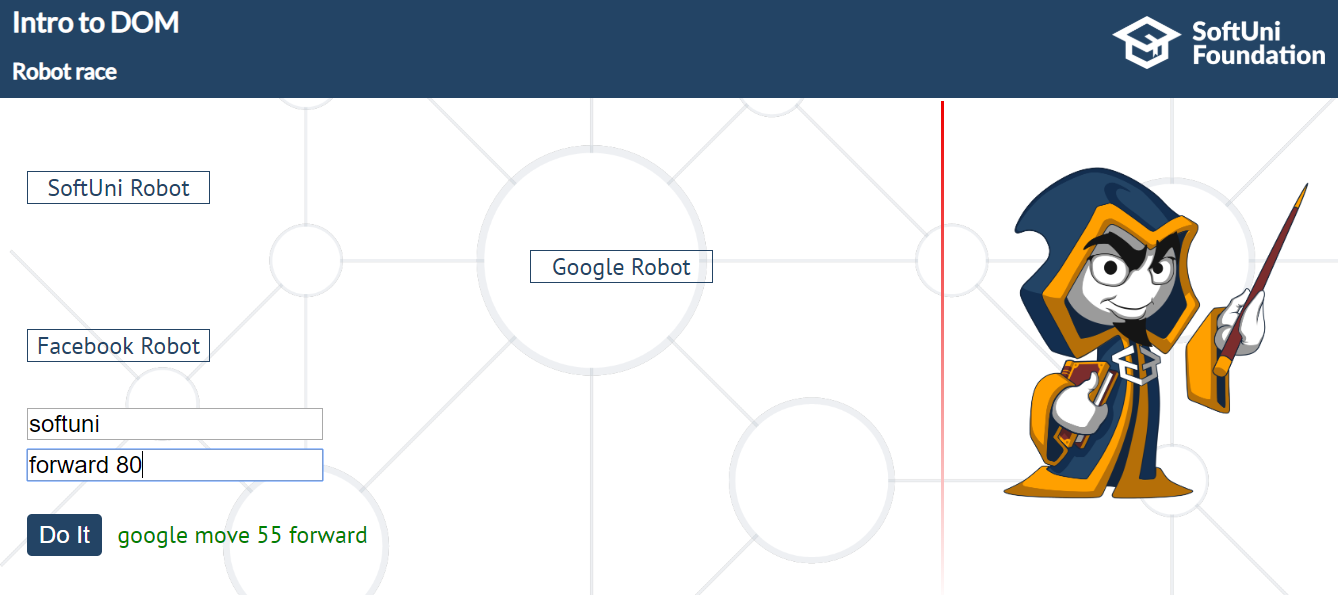
**Example**

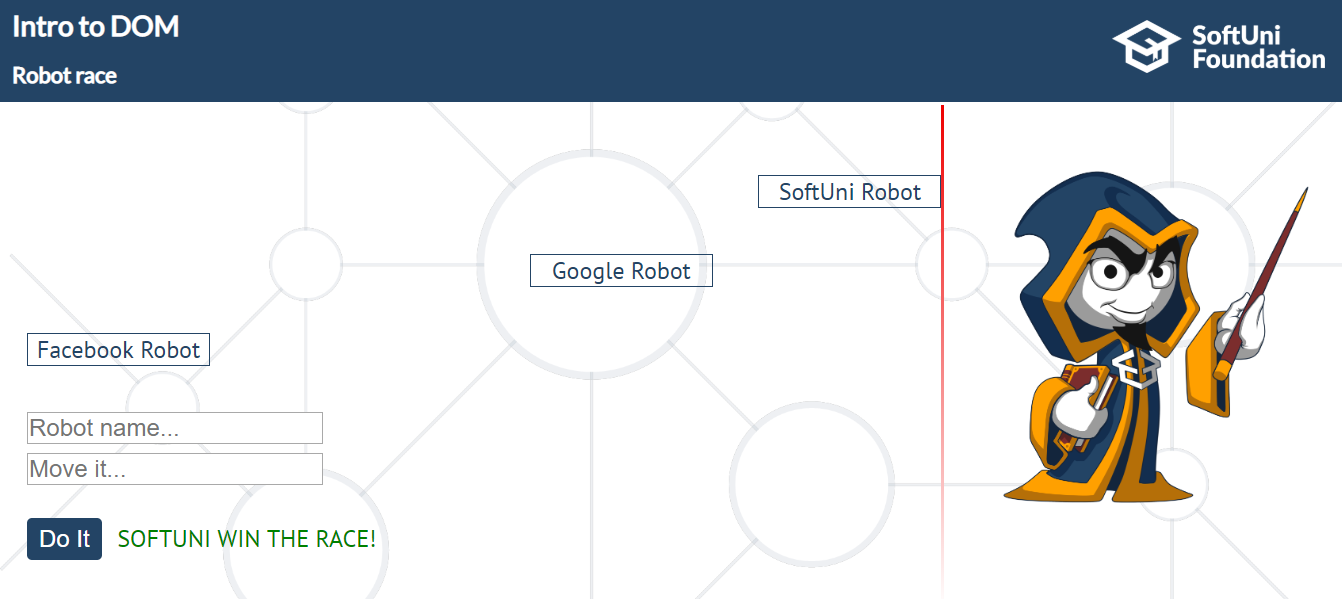
Here we move **google** **forward 55**:



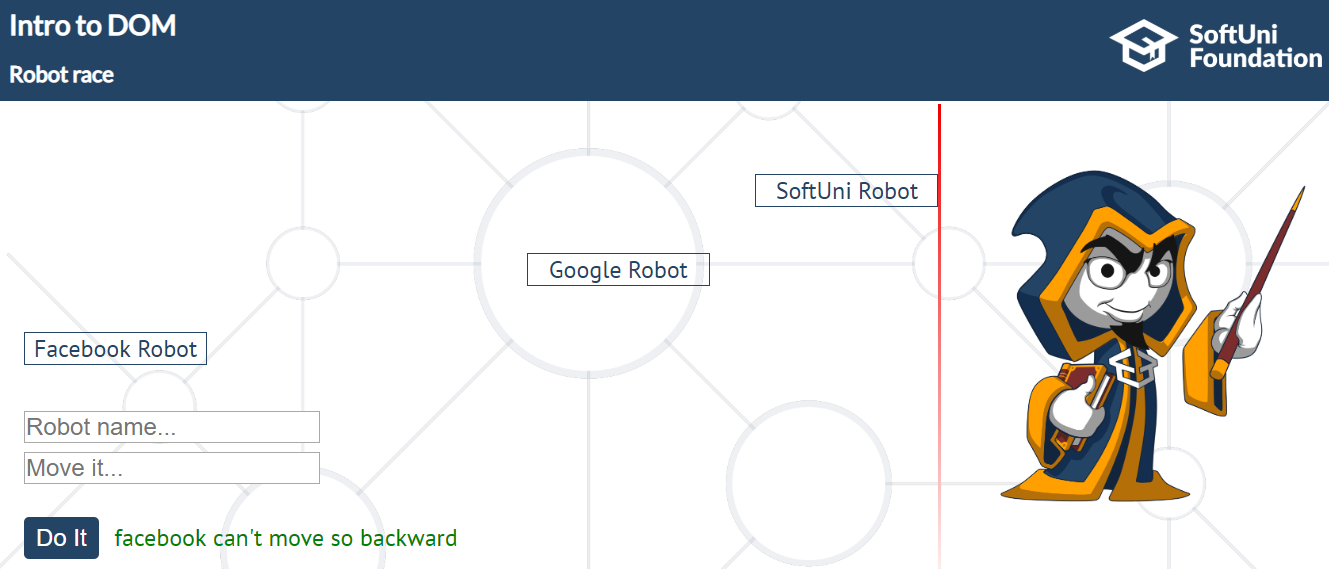


Here we move **softuni** **forward** **80**.

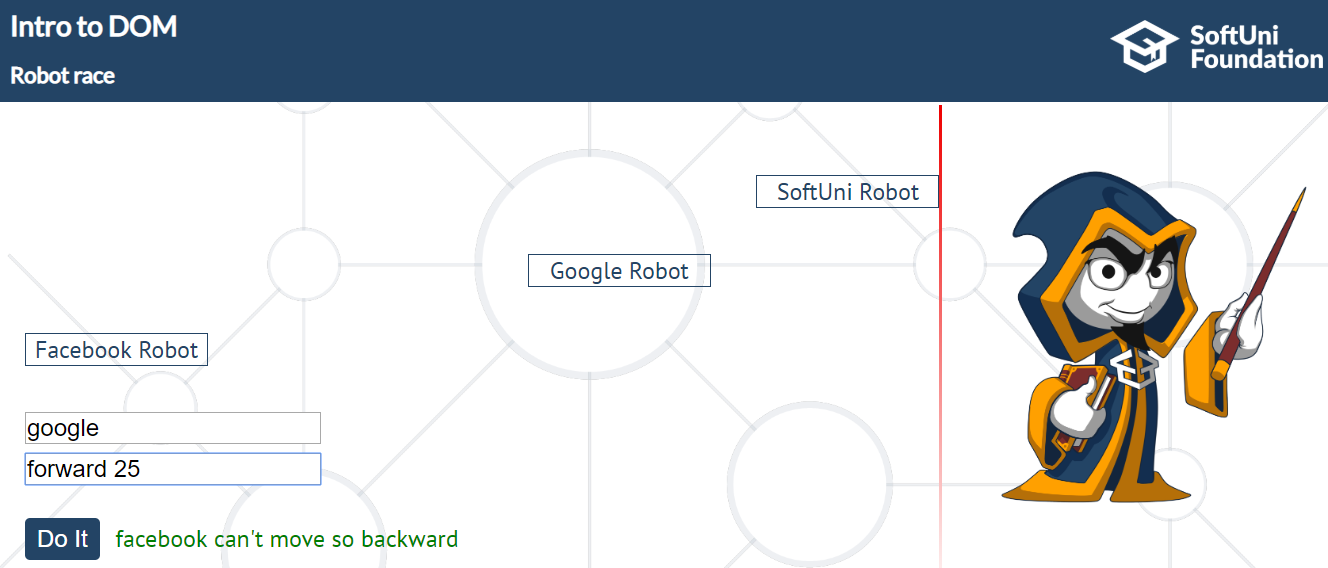


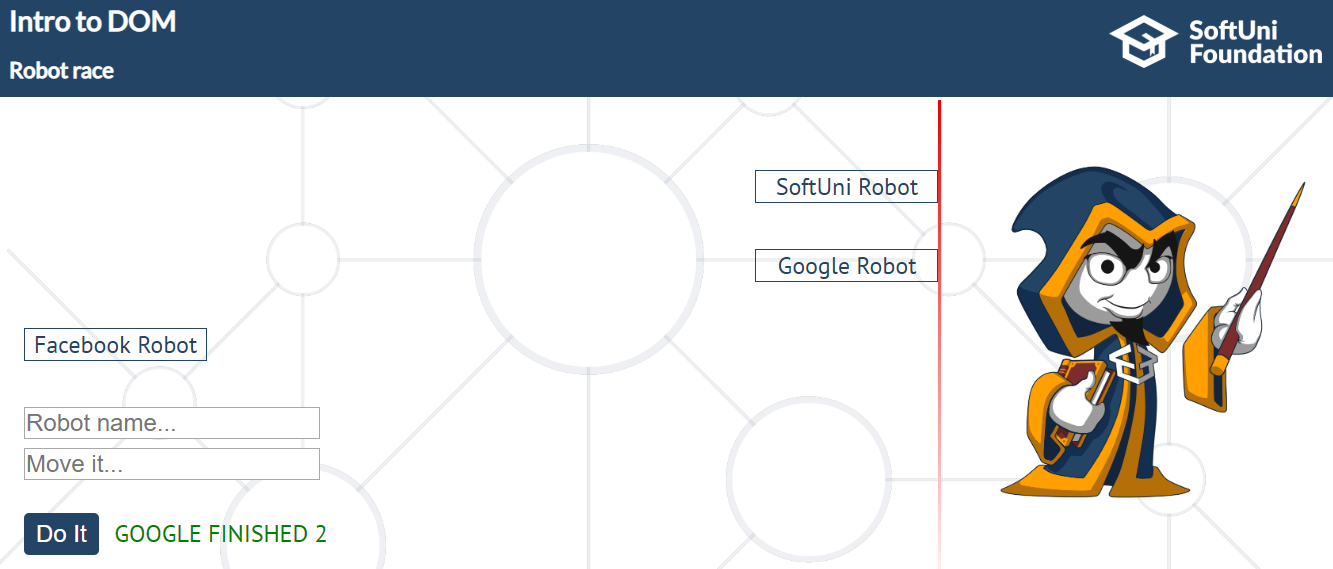


Here we move "facebook" backward 10



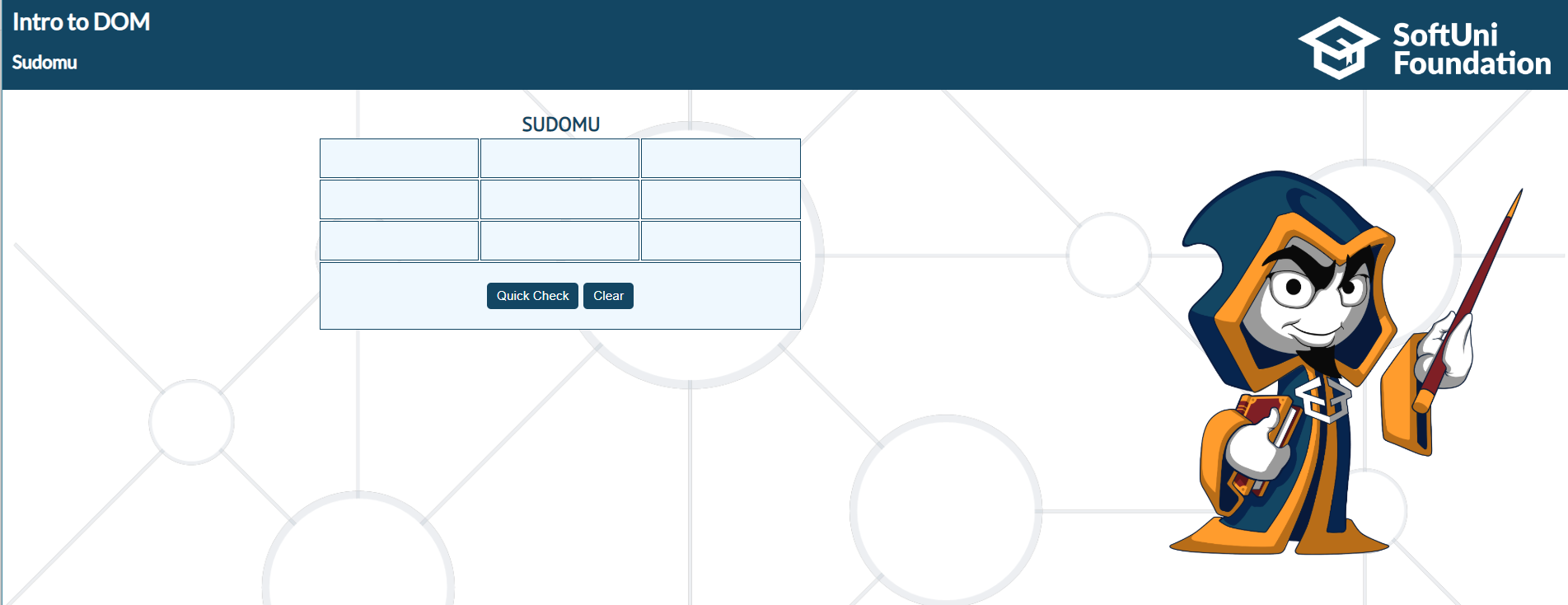
And here we move "google" 25 forward more





## Sudomu

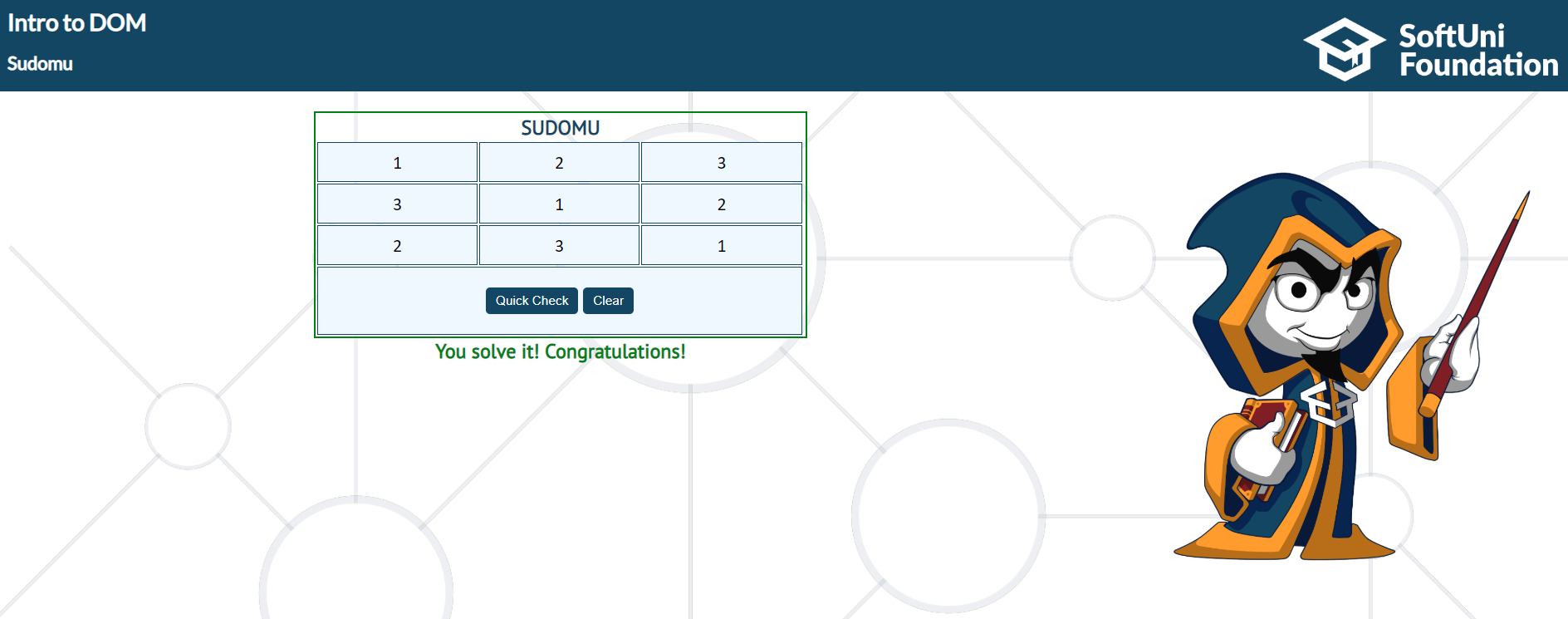
In this problem, you should **create a JS functonality** **of SUDOMU (Sudoku into the DOM).**



The rules are simple and they are **the same** as the **typical sudoku game**. –

<https://sudoku.bg/pravila/>

If the table is filled with the **right numbers**, and the "**Quick Check**" button is **clicked**, the

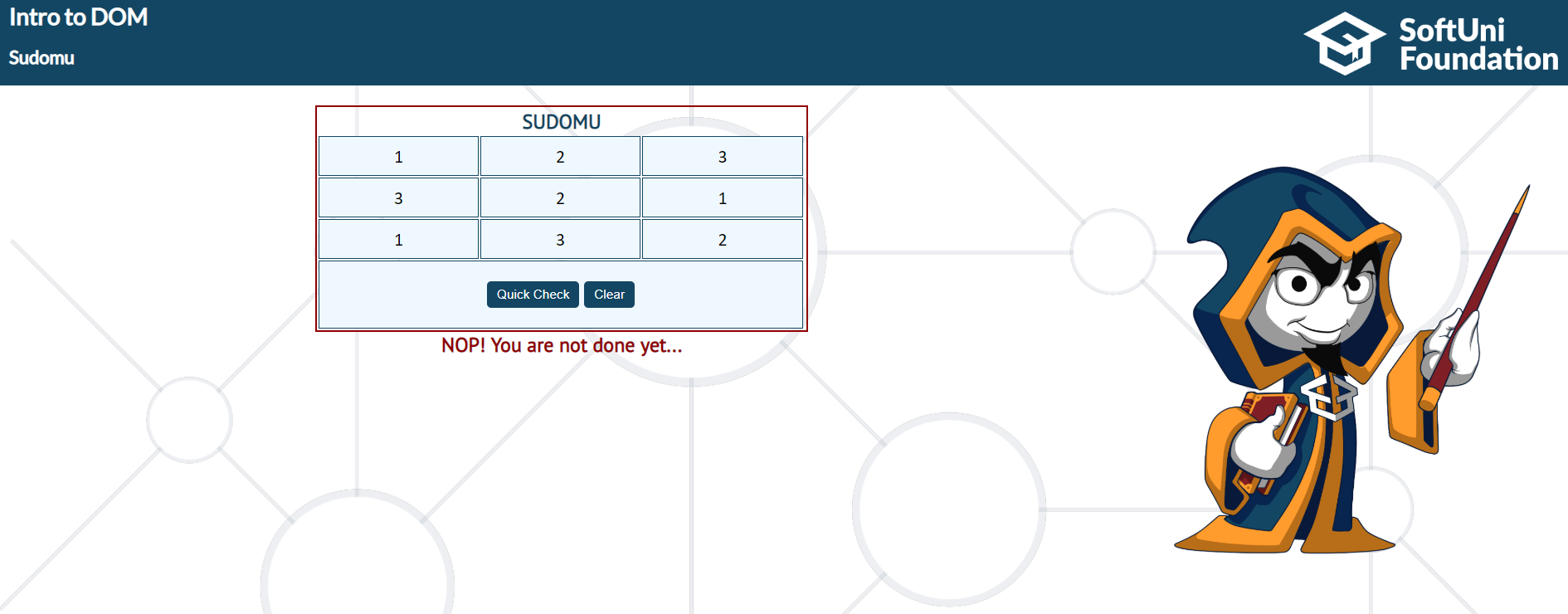
expected result should be:  


The table borer should be changed to: "**2px solid green".** The **text content** of the **paragraph**

inside the **div** with an **id** "**check**" must be "**You solve it! Congratulations!**", and text color of that

div must be **green!**

Otherwise, when the filled table **does not solve** **the sudomu,** the result should be:



The table border should be changed to: "**2px solid darkred**". The **text content** of the **paragraph**

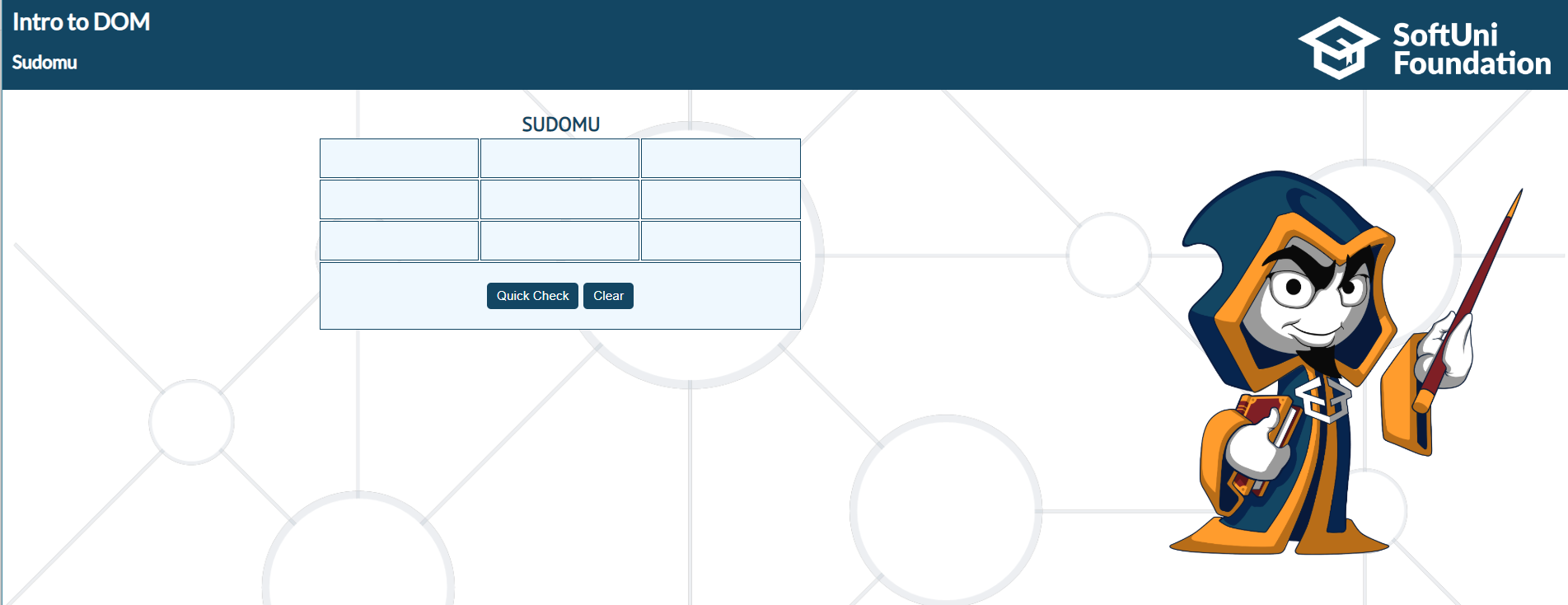
inside the **div** with an **id** "**check**" must be "**NOP! You are not done yet…**", and text color of that

div must be **darkred!**

**The Clear button clears the whole** **SUDOMU (removes all numbers in there) and the paragraph**

**which contains the messages. It also removes the table border! The expected result must be**

**the same as the one from the first picture, at the beginning.**



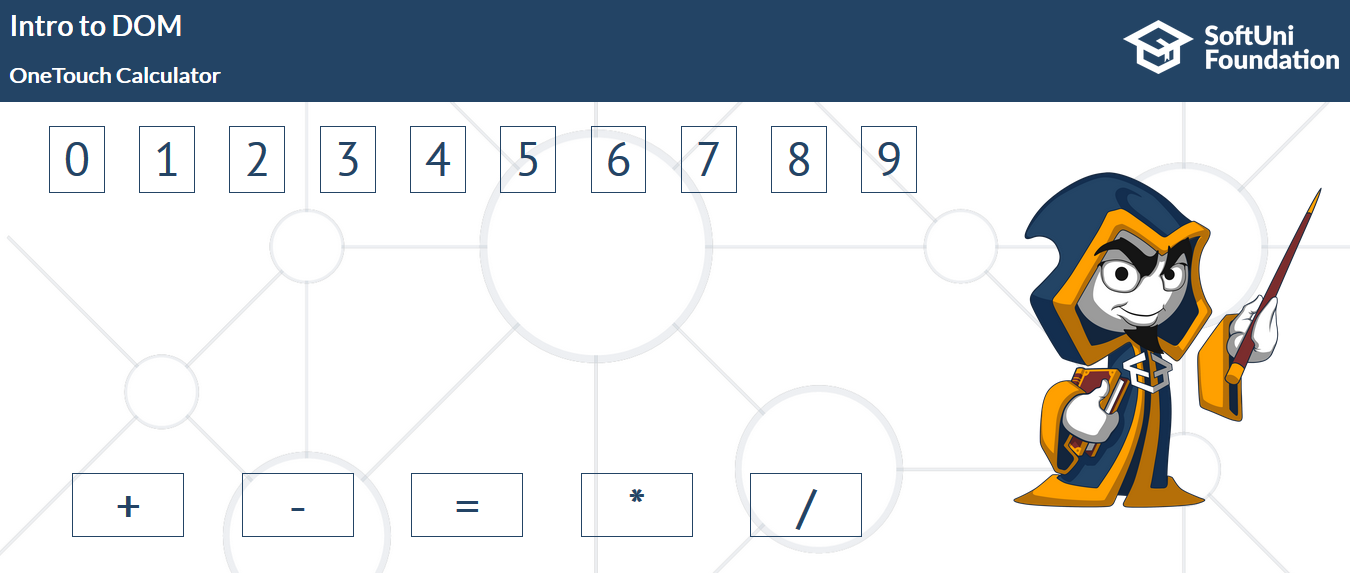
If you can **solve** this problem, you may want to try a **9x9 SUDOMU.** You can find it in the **lecture**

**resources.**

## \* OneTouch Caluclator

***Note: The task is not included in the judge system!***

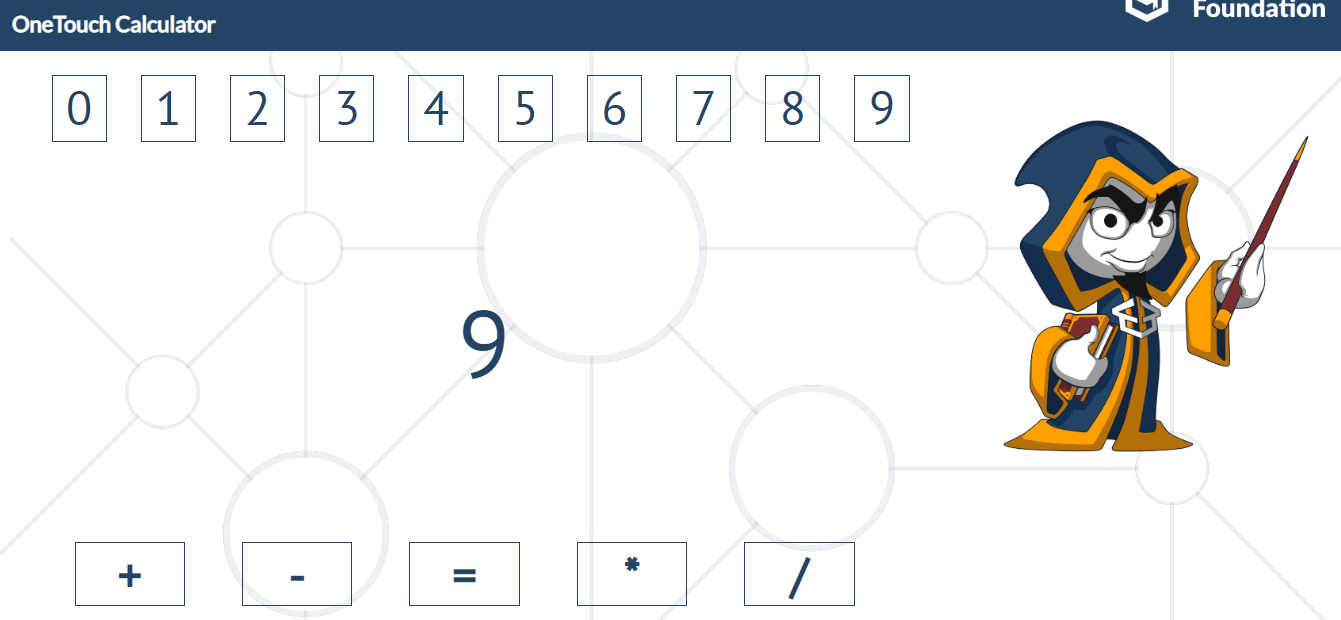
In this problem, you should **create a JS functonality** of a **calculator.**



Here, you should do deal with 2 **on events. (mouseover and click).**

**The digits** on **the top** side should **have a mouseover event**.When the mouse hovers one of the divs which holds a digit inside, the current digit should appear in the middle. (inside the paragraph which has an **id** **result**).

**For instance**, if we pass through the **div containing the digit** **9**, the expected result should be:



After that, if we pass through the **div** that contains the **digit 6** twice in a row, the expected result should be:



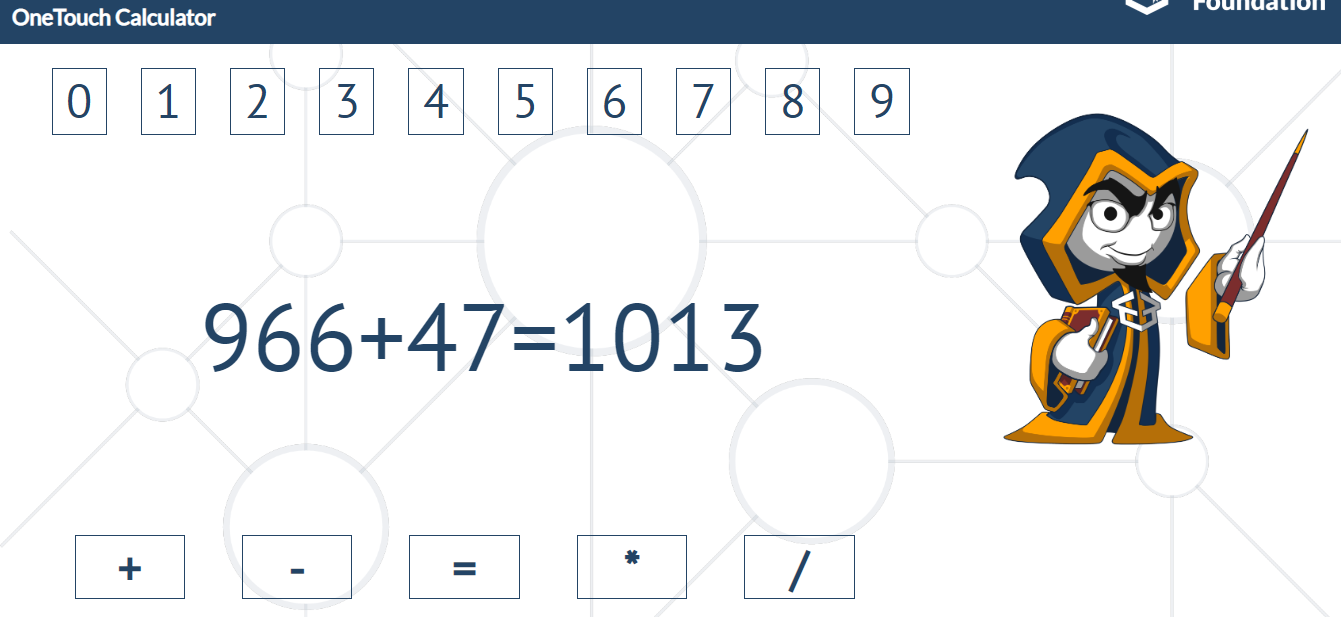
If that works fine, the **next step** is to **add on click events** to the **div elements**, which are located at the **bottom side** and contain only **mathematical operators** (+ - = \* /). **The expected functionality** is the **same** as above. When one of the operators is **clicked**, the **current operator** should **appear** in the **middle** (**inside that paragraph which has an id result**).

**For instance**, if we use the current example and we click on the **plus operator** (+), the expected result should be:



If we combine these two functionalities, we will successfully create a basic mathematical expression.

Тhe main condition of this problem to work is to have:   
First of all **– there should be one number, followed by** one of theoperators (+ - \* /) and **one more number** after the **operator**. When we hit the **equal sign** **(=),** **the** **result** of this **expression** should be added after the current text of the **result paragraph.**



**If the current condition is not met, nothing should happen!**

Otherwise, when we add the result of that expression, you should use the **setTimeOut function** to **clear the result paragraph after 5000 ms**, for the **next use** of our calculator.

After 5000 ms, the result should be:

