Exercises: Conditional Statements and Loops

Problems for exercises and homework for the "JavaScript Fundamentals" course @ SoftUni. Submit your solutions in the SoftUni judge system at https://judge.softuni.bg/Contests/300.

In this homework you are supposed to **write program logic** using expressions, conditions and loops in JavaScript. You will practice working with arithmetic operators, **expressions** and calculations, using **conditional statements** (**if**, **if-else**, multiple **if-else-if-else**-… and **switch-case**) and working with **loop statements** (like **for**, **while**, **do-while** and **for-of**) and **nested loops**, combined with conditional statements and calculations.

1. Biggest of 3 Numbers

Write a JS function that finds the **biggest of 3 numbers**.

The **input** comes as array of strings, where each element holds a number.

The **output** is the biggest from the input numbers.

Examples

Input	Output
5 -2 7	7

Input	Output
130 5 99	130

Input	Output
43 43.2 43.1	43.2

Input	Output
5 5 5	5

Input	Output
-10 -20 -30	-10

Hints

Read the input and parse them to numbers:

```
function biggestNumber(input) {
    let num1 = Number(input[0]);
    let num2 = // TODO
    let num3 = // TODO
}
```

Note: If you don't parse the input strings to numbers and you compare them by **if**-condition, your code will compare the string values instead the number values and thus 5 might be bigger than 11.

You may use **Math.max(num1, num2, num3)** to find the biggest number. It will automatically convert strings to numbers, so parsing the input elements to number might not be obligatory. Your code might look like this:

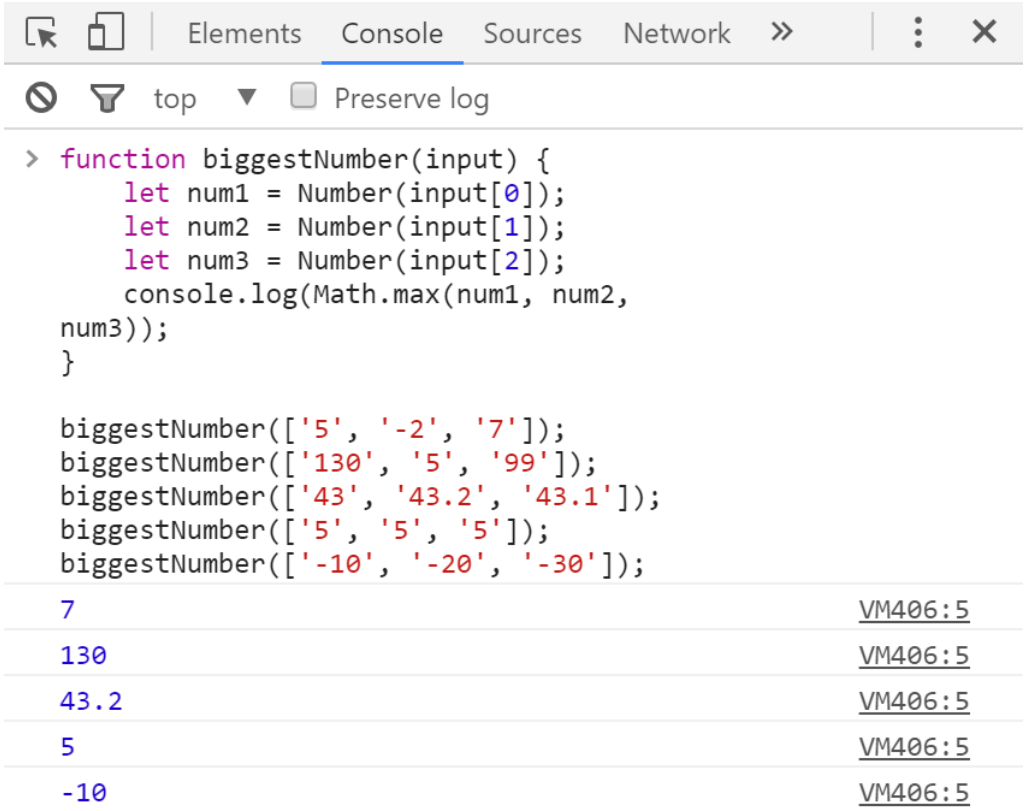
```
function biggestNumber(input) {
    let num1 = Number(input[0]);
    let num2 = // TODO
    let num3 = // TODO
    console.log(Math.max(num1, num2, num3));
}
```

Test the above code **locally** by invoking the **biggestNumber()** function like this:

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

```
biggestNumber(['5', '-2', '7']);
biggestNumber(['130', '5', '99']);
biggestNumber(['43', '43.2', '43.1']);
```

Test the above code **locally** in your Web browser:



Finally, submit your code in the judge system: https://judge.softuni.bg/Contests/Practice/Index/300.

Note: the above code might be shortened like this:

```
function biggestNumber([num1, num2, num3]) {
    console.log(Math.max(num1, num2, num3));
}
```

The above code takes the input as array of 3 string variables **num1**, **num2** and **num3**. Then the function **Math.max(…)** converts its string arguments to numbers and returns the biggest of them.

You may shorten further your solution by making it an **arrow function** like the shown below:

```
([num1, num2, num3]) => console.log(Math.max(num1, num2, num3));
```

The **judge** system will **accept** the above solution as correct.

In the judge you can also **return the expected result** instead of printing it at the console, so the following solution will also be accepted by the judge, as well:

```
([num1, num2, num3]) => Math.max(num1, num2, num3)
```

The above code could be even further shortened like this:

```
(nums) => Math.max(nums[0], nums[1], nums[2])
```

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

The judge system will accept the above arrow function as correct solution, passing all the tests.

Enjoy!

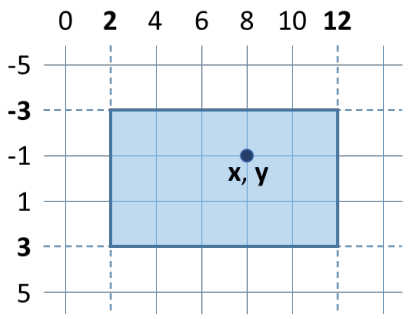
2. Point in Rectangle

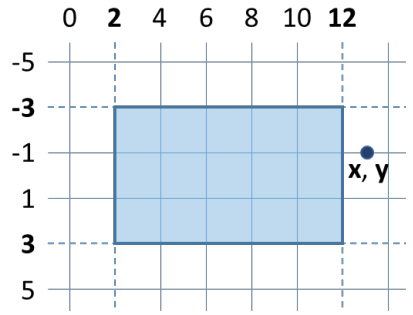
Write a JS function that takes as input 6 numbers **x**, **y**, **xMin**, **xMax**, **yMin**, **yMax** and prints whether the point {**x**, **y**} is **inside** the rectangle or **outside** of it. If the point is at the rectangle **border**, it is considered **inside**.

The **input** comes as array of strings. Its holds the string representations of **x**, **y**, **xMin**, **xMax**, **yMin**, **yMax**. All numbers will in the range [-1 000 000 … 1 000 000]. It is guaranteed that **xMin** < **xMax** and **yMin** < **yMax**.

The **output** is a single line holding "**inside**" or "**outside**".

Examples

Input	Output	Figure
8 -1 2 12 -3 3	inside	

Input	Output	Figure
12.5 -1 2 12 -3 3	outside	

Hints

First write a JS function to **read the input numbers x**, **y**, **xMin**, **xMax**, **yMin** and **yMax** from the array of 6 strings, passed as input parameter **input**. The numbers should be taken from the array elements **input[0]**, **input[1]**, … and should be parsed from string to number as follows:

```
function pointInRect(input) {
    let x = Number(input[0]);
    let y = Number(input[1]);
    let xMin = Number(input[2]);
    let xMax = Number(input[3]);
    let yMin = Number(input[4]);
    let yMax = Number(input[5]);
    // TODO
}
```

The above code could be significantly shortened by using a JS language feature called "**destructuring assignment**" + **mapping** the input string array to an array of numbers through the function **Number** like this:

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

```
function pointInRect(input) {
    let [x, y, xMin, xMax, yMin, yMax] = input.map(Number);
    // TODO
}
```

Next, write **if-else** statement to check whether the point {**x**, **y**} is inside the rectangle. A **point is inside a rectangle** if and only if both of the following conditions are fulfilled:

- its **x** coordinate is between **xMin** and **xMax**
- its **y** coordinate is between **yMin** and **yMax**

The code may look like this:

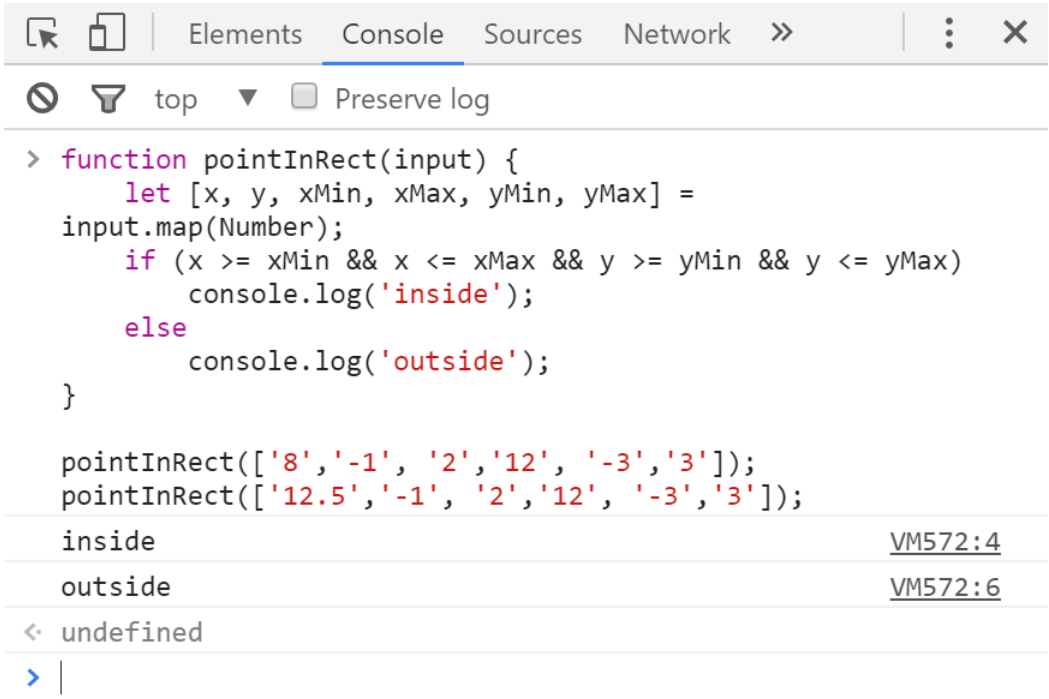
```
if (x >= xMin && x <= xMax && y >= yMin && y <= yMax)
    console.log('inside');
else
    console.log('outside');
```

Now **test the function locally** on your computer. Invoke the above function like this:

```
pointInRect(['8','-1', '2','12', '-3','3']);
pointInRect(['12.5','-1', '2','12', '-3','3']);
```

Note that the function takes its input as **array of 6 strings** (not numbers, not string holding number, not an array of numbers). You may test your code in your IDE, in the console (with Node.js) or in your Web browser's JS console.

The screenshot below shows the above function, invoked in **Chrome JavaScript Console**:

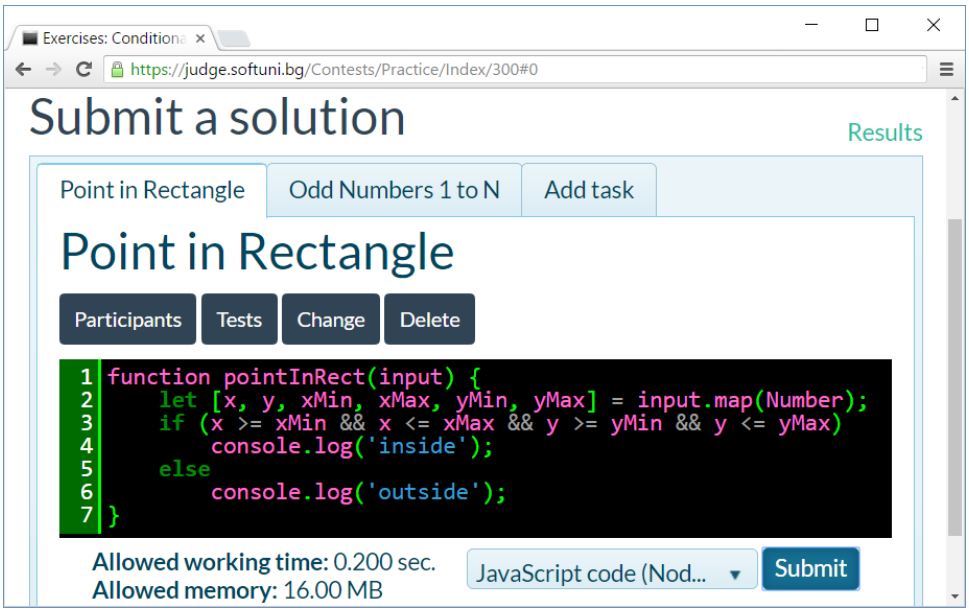


The screenshot below shows the above code in the **Node.js REPL console**:

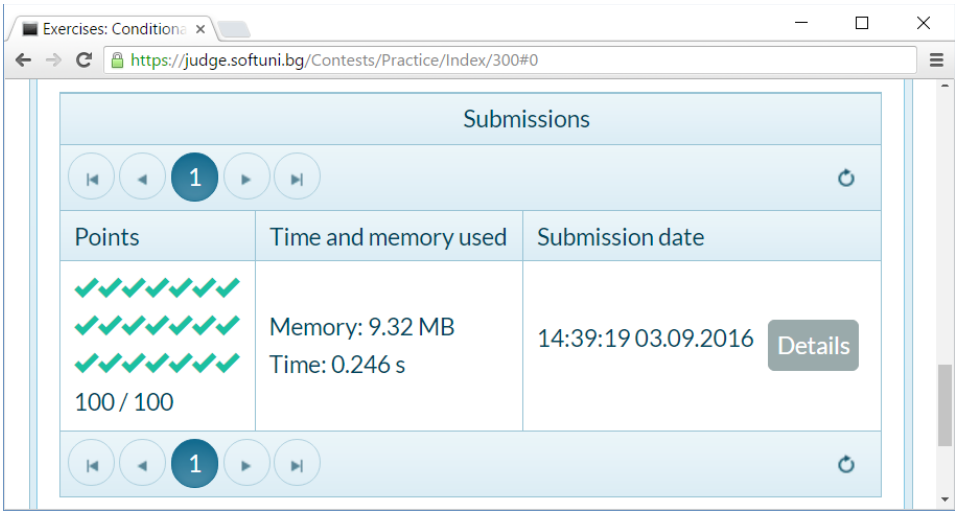
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

```
Command Prompt - node
>node
> function pointInRect(input) {
...     let [x, y, xMin, xMax, yMin, yMax] = input.map(Number);
...     if (x >= xMin && x <= xMax && y >= yMin && y <= yMax)
...         console.log('inside');
...     else
...         console.log('outside');
... }
undefined
>
> pointInRect(['8','-1', '2','12', '-3','3']);
inside
undefined
> pointInRect(['12.5','-1', '2','12', '-3','3']);
outside
undefined
>
```

Now you are ready to submit your solution in the **judge system**. Open the judge contest for this homework and submit your code: https://judge.softuni.bg/Contests/Practice/Index/300. It should look like this:



The judge system should **accept your solution** as correct:



© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

3. Odd Numbers 1 to N

Write a JS function that reads an integer **n** and prints all **odd numbers** from **1** to **n**.

The **input** comes as array of strings. Its first element holds the string representation of **n**. The number **n** will be an integer in the range [1 … 100 000].

The **output** should hold the expected odd numbers, each at a separate line.

Examples

Input	Output
5	1 3 5

Input	Output
4	1 3

Input	Output
7	1 3 5 7

Hints

First write a JS function to read the input number **n** from an array of strings, passed as parameter **input**. The number **n** should be taken from the first array element **input[0]** and should be parsed from string to number:

```
function nums(input) {
    let n = Number(input[0]);
    // TODO: print all odd numbers in range [1..n]
}
```

Next, write a **for**-loop from **1** to **n** with **step 2**:

```
for (let i=1; i<=n; i+=2) {

}
```

Finally, print the number **i** at each iteration of the **for**-loop. The entire function should look like this:

```
function nums(input) {
    let n = Number(input[0]);
    for (let i=1; i<=n; i+=2) {
        console.log(i);
    }
}
```

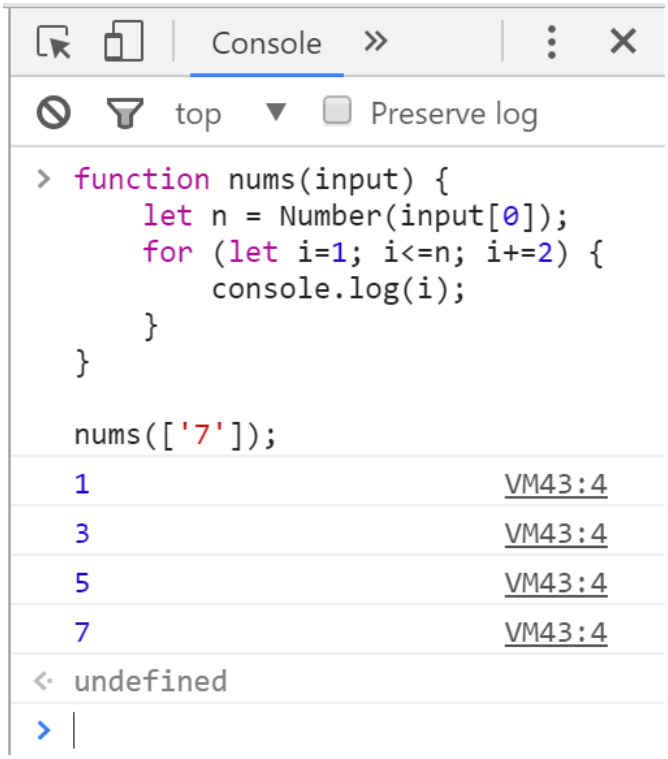
Now **test the function locally** on your computer. Invoke the above function like this:

```
nums(['7']);
```

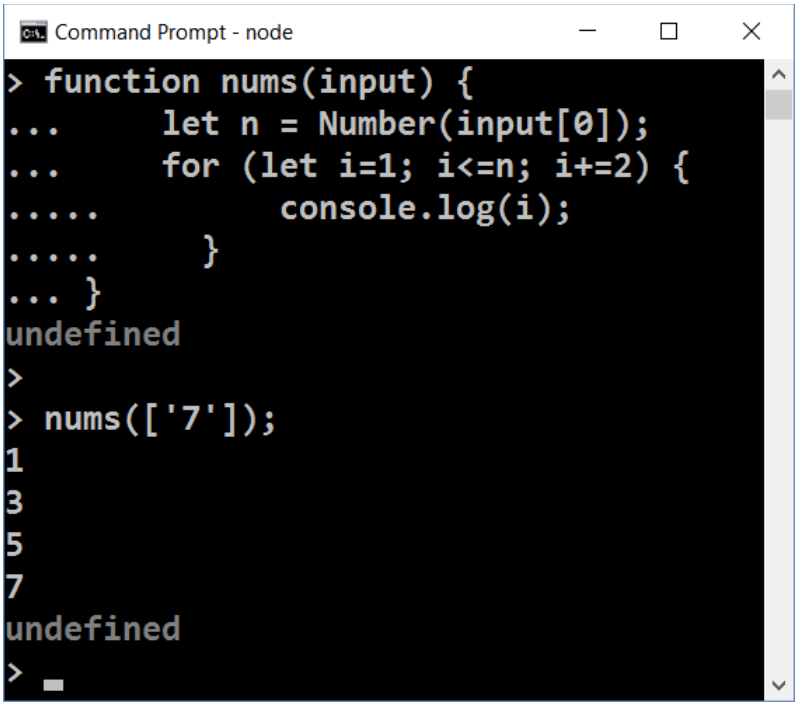
Note that the function takes its input as **array of strings** (not a string, not a number, not an array of numbers). You may test your code in the console (with Node.js) or in your Web browser's JS console.

The screenshot below shows the above function, invoked in **Chrome JavaScript Console**:

SOFTWARE UNIVERSITY FOUNDATION © Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

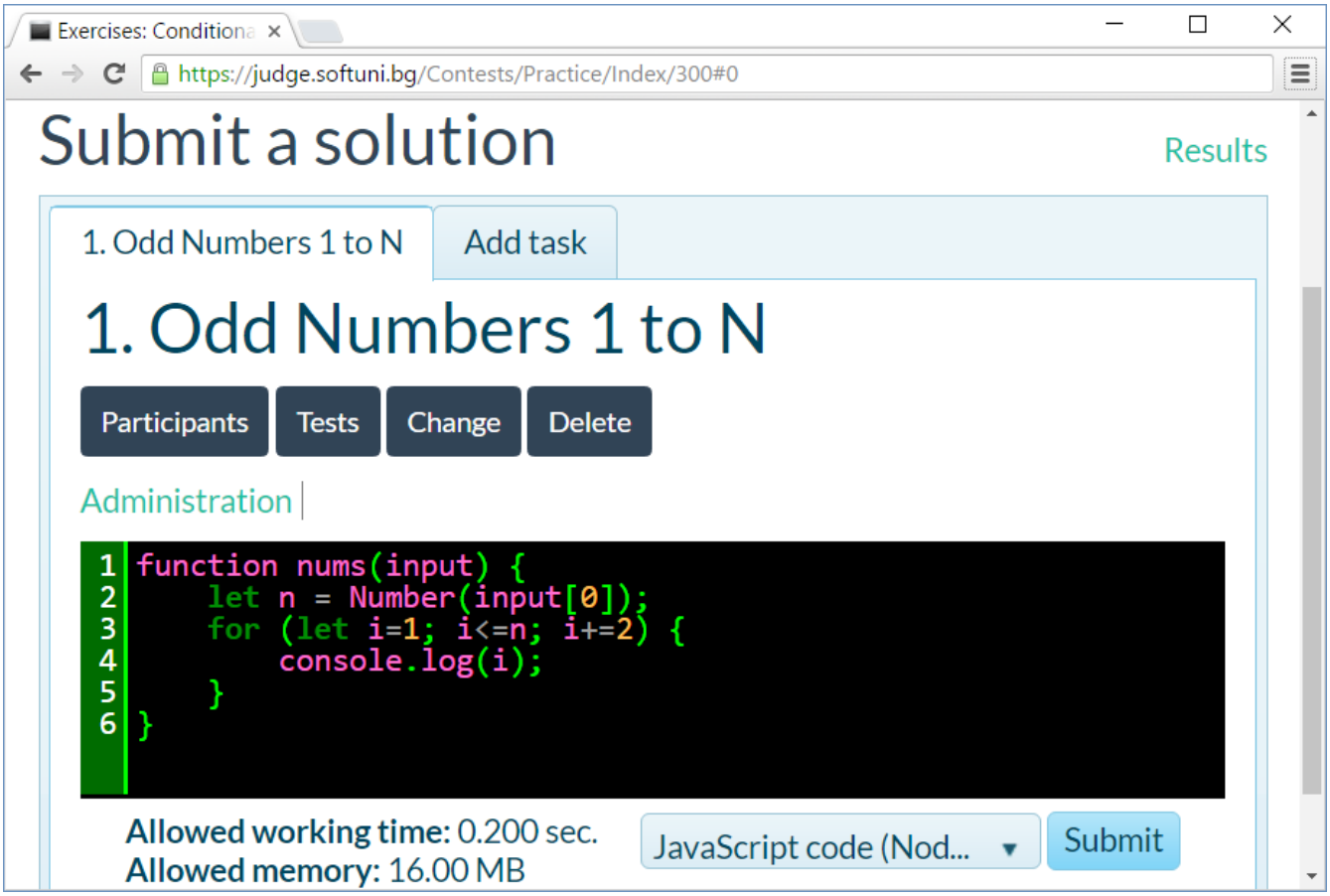


The screenshot below shows the above code in the **Node.js REPL console**:

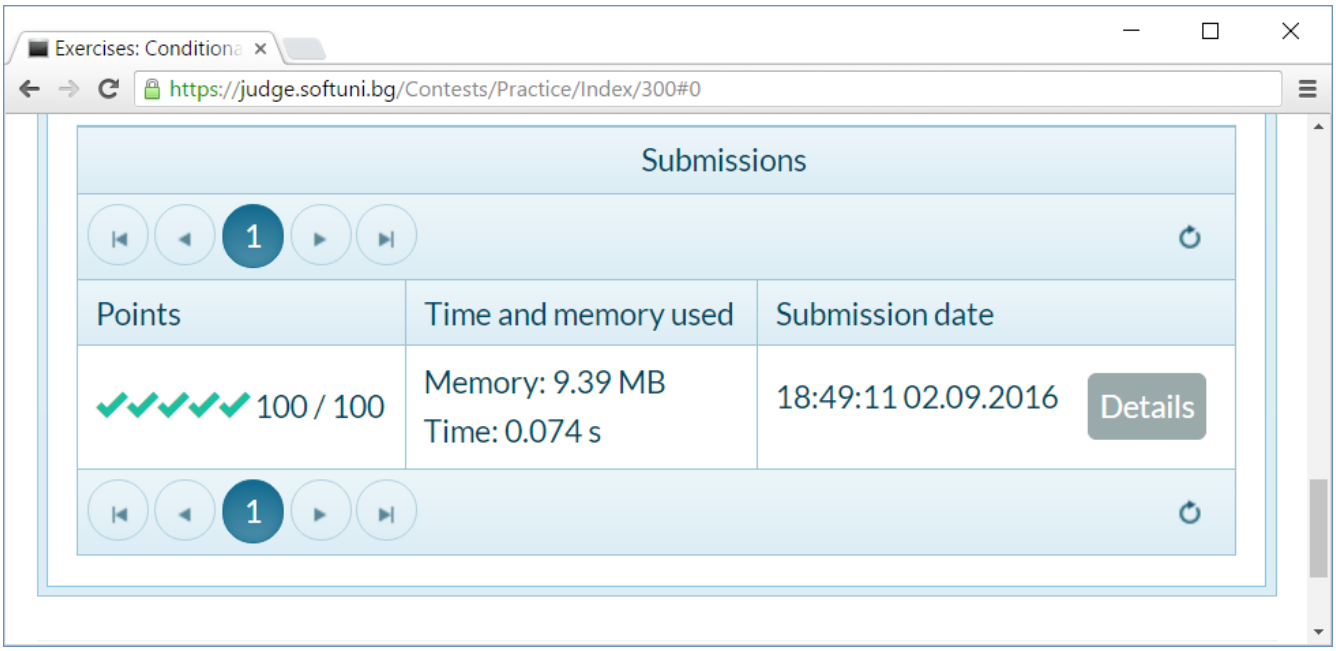


Now you are ready to submit the problem in the **judge system**. Open the judge contest for this homework and submit your code: https://judge.softuni.bg/Contests/Practice/Index/300. It should look like this:

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.



The judge system should **accept your solution** as correct:



4. Triangle of Dollars

Write a JS function that prints a triangle of **n** lines of "**$**" characters like shown in the examples.

The **input** comes as array of strings. Its first element holds the number **n** (**0** < **n** < **100**).

The **output** consists of **n** text lines like shown below.

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

Examples

Input	Output
3	$ $$ $$$

Input	Output
2	$ $$

Input	Output
4	$ $$ $$$ $$$$

Hints

Variant 1: use **nested loops** and for each row collect the '**$**' characters in a string and then print it:

```
let line = '';
for (let col = 1; col <= row; col++)
    line += '$';
console.log(line)
```

Variant 2: use a simple loop **row** = **1** … **n** and print **row** dollars this way:

```
console.log(new Array(row+1).join('$'))
```

Variant 3: you can print **row** dollars by **repeating** the '**$**' string **row** times:

```
console.log('$'.repeat(row));
```

5. Movie Prices

Movie ticket **prices** in a big retro-cinema depend on the **movie title** and on the **day of week** as shown below:

Movie Title	*Monday*	*Tuesday*	*Wednesday*	*Thursday*	*Friday*	*Saturday*	*Sunday*
The Godfather	12	10	15	12.50	15	25	30
Schindler's List	8.50	8.50	8.50	8.50	8.50	15	15
Casablanca	8	8	8	8	8	10	10
The Wizard of Oz	10	10	10	10	10	15	15

Write a JS function that **calculate the ticket price** by movie title and day of week.

The **input** comes as array of 2 strings. The first string holds the **movie title**. The second string holds the **day of week**.

The **output** should hold the **ticket price** or "**error**" if the title or day of week is invalid.

Examples

Input	Output
The Godfather Friday	15

Input	Output
casablanca sunday	10

Input	Output
Schindler's LIST monday	8.50

Input	Output
SoftUni Nined	error

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

								ay	

Hints

- Turn all input values to **lowercase** to avoid character casing mistakes.
- **Check the movie title** in **if-else-if** statement.
- For each movie title check the price in a **switch-case**.
- Beware of **invalid** movie titles and invalid days of week. Show "**error**" in such cases.

You may follow the **code example** below:

```
function moviePrices([title, day]) {
    // TODO: make title and day lowercase
    if (title == "the godfather") {
        switch (day) {
            case "monday": return 12;
            // TODO: check the other days ...
            default: return "error";
        }
    } else if (title == "schindler's list") {
        switch (day) {
            // TODO: return the price by day
        }
    // TODO: check the other movie titles ...
    } else {
        return "error";
    }
}
```

You may **test your code** locally like this:

```
console.log(moviePrices(["The Godfather", "Friday"]));
console.log(moviePrices(["casablanca", "sunday"]));
console.log(moviePrices(["Schindler's LIST", "monday"]));
console.log(moviePrices(["SoftUni", "Nineday"]));
```

6. Quadratic Equation

Write a JS function to **solve a quadratic equation** by given in standard form as its coefficients: **a**, **b**, **c**. You may learn more about quadratic equations here: https://www.mathsisfun.com/algebra/quadratic-equation.html.

The **input** comes as array of strings. Its holds the string representations of 3 real numbers: **a**, **b** and **c**. The value of **a** will be non-zero.

The **output** depends on the equation:

- It holds two numbers $\mathbf{x_1}$ and $\mathbf{x_2}$ if the equation has two different solutions (roots): $\mathbf{x_1}$ and $\mathbf{x_2}$.
    - First print the smaller root, then the greater.
- It holds a single number **x** if the equation has only one solution (root): **x**.
- It holds the text "**No**" if the equation has no solutions (no real roots).

SOFTWARE UNIVERSITY FOUNDATION © Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

Examples

Input	Output	Comments
6 11 -35	-3.5 1.66667	The equation is: $6x^2 + 11x - 35 = 0$ □ $a = 6$; $b = 11$; $c = -35$ The discriminant is: $d = b^2 - 4*a*c = 11*11 - 4*6*(-35) = 121 + 840 = 961$ We have positive discriminant ($d > 0$), so the equation has two real roots: • $x_1 = (-b + \sqrt{d}) / (2*a) = (-11 + 31) / 12 = 20/12 = 5/3 \approx 1.66667$ • $x_2 = (-b - \sqrt{d}) / (2*a) = (-11 - 31) / 12 = -42/12 = -7/2 = -3.5$ The output consists of two lines: • -3.5 (the smaller root at the first line) • 1.66667 (the bigger root at the second line)
1 -12 36	6	The equation is: $x^2 - 12x + 36 = 0$ □ $a = 1$; $b = -12$; $c = 36$ The discriminant is: $d = b^2 - 4*a*c = -12*-12 - 4*1*36 = 144 - 144 = 0$ We have zero discriminant ($d = 0$), so the equation has only one real root: • $x = -b / (2*a) = 12 / 2 = 6$ The output is only one line, holding the number 6.
5 2 1	No	The equation is: $5x^2 - 2x + 1 = 0$ □ $a = 5$; $b = -2$; $c = 1$ The discriminant is: $d = b^2 - 4*a*c = -2*-2 - 4*5*1 = 4 - 20 = -16$ We have negative discriminant ($d < 0$), so the equation has no real roots. The output is only one line, holding the text "No".

Hints

- Search for "***solving quadratic equation***" in Internet.
- Use **if**-conditions and expressions to calculate the **discriminant** and the equation **roots** (if any).

7. Multiplication Table

Write a JS function to print a **math multiplication table** of size **n**, formatted as **HTML table**.

The **input** comes as array of strings. Its first element holds the number **n** (**0** < **n** < **100**).

The **output** consists of **n+3** text lines like shown below.

Examples

Input	5
Output	<table border="1"> <tr><th>**x**</th><th>**1**</th><th>**2**</th><th>**3**</th><th>**4**</th> <th>**5**</th></**tr**> <tr><th>**1**</th><td>**1**</td><td>**2**</td><td>**3**</td><td>**4**</td> <td>**5**</td></**tr**> <tr><th>**2**</th><td>**2**</td><td>**4**</td><td>**6**</td><td>**8**</td> <td>**10**</td></**tr**> <tr><th>**3**</th><td>**3**</td><td>**6**</td><td>**9**</td><td>**12**</td> <td>**15**</td></**tr**> <tr><th>**4**</th><td>**4**</td><td>**8**</td><td>**12**</td><td>**16**</t

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

	d><td>**20**</td></**tr**> <**tr**><th>**5**</th><td>**5**</td><td>**10**</td><td>**15**</td><td>**20**</td><td>**25**</td></**tr**> </table>
Preview	**x** | **1** | **2** | **3** | **4** | **5** **1** | 1 | 2 | 3 | 4 | 5 **2** | 2 | 4 | 6 | 8 | 10 **3** | 3 | 6 | 9 | 12 | 15 **4** | 4 | 8 | 12 | 16 | 20 **5** | 5 | 10 | 15 | 20 | 25

Hints

- Create a function **multiplicationTable([n]) { … }**. It will return a table of size **n**.
- First, print the "**<table border='1'>**" opening tag.
- Using a simple loop, print the **heading row**.
  - It should hold **<tr>** + "**x**" + the numbers **1**…**n** (all surrounded in **<td></td>**) + **</tr>**.
- Print the **next n lines** using nested loops.
  - For each line start with **<tr>**, then append its elements in a loop (all surrounded in **<td></td>**) and finally append **</tr>**.
- After the loops, at the end, print the "**</table>**" closing tag.
- You may visualize your code's output in the browser like this:

```
document.body.innerHTML = mutiplicationTable(['5']);
```

8. Figure of 4 Squares

Write a JS function to print a **figure of 4 squares** of size **n** like shown in the examples below.

The **input** is an integer number **n** in the range [2 … 200].

The **output** consists of **n** lines for **odd n** and **n-1** lines for **even n**. Each line holds **2*n-1** characters (**+** or **|** or ***space***) as shown in the examples. The figure is fully symmetric (horizontally and vertically).

Examples

Input	Output
4	+--+--+ +--+--+ +--+--+

Input	Output
5	+---+---+ | | | +---+---+ | | | +---+---+

Input	Output
6	+----+----+ | | | +----+----+ | | | +----+----+

Input	Output
7	+-----+-----+ | | | | | | +-----+-----+ | | | | | | +-----+-----+

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

Hints

- Use **nested loops** and **if**-statements. Try to figure out the **logic of construction** of the above figures.

9. ** Monthly Calendar

Note: this problem is for champions only!

Write a JS function **calendar([day, month, year])** that returns a **monthly calendar as HTML table** (like in the examples below) by given **day**, **month** and **year**. Weeks start by "**Sun**" (Sunday) and end by "**Sat**" (Saturday).

The **input** comes as array of 3 strings, holding the following numbers:

- **day** (1 ≤ **day** ≤ 31)
- **month** (1 ≤ **month** ≤ 12)
- **year** (1900 ≤ **year** ≤ 2100)

The **output** should be an **HTML table**, holding the calendar rows and columns, like in the examples below. Display the **weeks** as table rows: **<tr>…</tr>**. Display the **days** as table cells: **<td>…</td>**. Display the days of the previous month with CSS class "**prev-month**", the days of the next month with CSS class "**next-month**" and the current day with CSS class "**today**". See the examples below.

Examples

Input	24 12 2012
Output	<table> <tr><th>**Sun**</th><th>**Mon**</th><th>**Tue**</th><th>**Wed**</th><th>**Thu**</th><th>**Fri**</th><th>**Sat**</th></tr> <tr><td class="**prev-month**">**25**</td><td class="**prev-month**">**26**</td><td class="**prev-month**">**27**</td><td class="**prev-month**">**28**</td><td class="**prev-month**">**29**</td><td class="**prev-month**">**30**</td><td>**1**</td></tr> <tr><td>**2**</td><td>**3**</td><td>**4**</td><td>**5**</td><td>**6**</td><td>**7**</td><td>**8**</td></tr> <tr><td>**9**</td><td>**10**</td><td>**11**</td><td>**12**</td><td>**13**</td><td>**14**</td><td>**15**</td></tr> <tr><td>**16**</td><td>**17**</td><td>**18**</td><td>**19**</td><td>**20**</td><td>**21**</td><td>**22**</td></tr> <tr><td>**23**</td><td class="**today**">**24**</td><td>**25**</td><td>**26**</td><td>**27**</td><td>**28**</td><td>**29**</td></tr> <tr><td>**30**</td><td>**31**</td><td class="**next-month**">1</td><td class="**next-month**">**2**</td><td class="**next-month**">**3**</td><td class="**next-month**">**4**</td><td class="**next-month**">**5**</td></tr> </table>

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

Preview	
	Sun Mon Tue Wed Thu Fri Sat 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 **24** 25 26 27 28 29 30 31 1 2 3 4 5

Input	4 9 2016
Output	<table> <tr><th>**Sun**</th><th>**Mon**</th><th>**Tue**</th><th>**Wed**</th><th>**Thu**</th><th>**Fri**</th><th>**Sat**</th></tr> <tr><td class="**prev-month**">**28**</td><td class="**prev-month**">**29**</td><td class="**prev-month**">**30**</td><td class="**prev-month**">**31**</td><td>**1**</td><td>**2**</td><td>**3**</td></tr> <tr><td class="**today**">**4**</td><td>**5**</td><td>**6**</td><td>**7**</td><td>**8**</td><td>**9**</td><td>**10**</td></tr> <tr><td>**11**</td><td>**12**</td><td>**13**</td><td>**14**</td><td>**15**</td><td>**16**</td><td>**17**</td></tr> <tr><td>**18**</td><td>**19**</td><td>**20**</td><td>**21**</td><td>**22**</td><td>**23**</td><td>**24**</td></tr> <tr><td>**25**</td><td>**26**</td><td>**27**</td><td>**28**</td><td>**29**</td><td>**30**</td><td class="**next-month**">**1**</td></tr> </table>
Preview	Sun Mon Tue Wed Thu Fri Sat 28 29 30 31 1 2 3 **4** 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1

HTML Skeleton

To simplify your work, use the below HTML code and write the missing code in the **calendar()** function:

calendar.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Monthly Calendar</title>
    <style>
        .prev-month, .next-month { color: #CCC }
        .today { font-weight: bold; background: #DDD; }
        .title { background: #AAAAFF; margin: 10px 0; padding:5px }
```

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

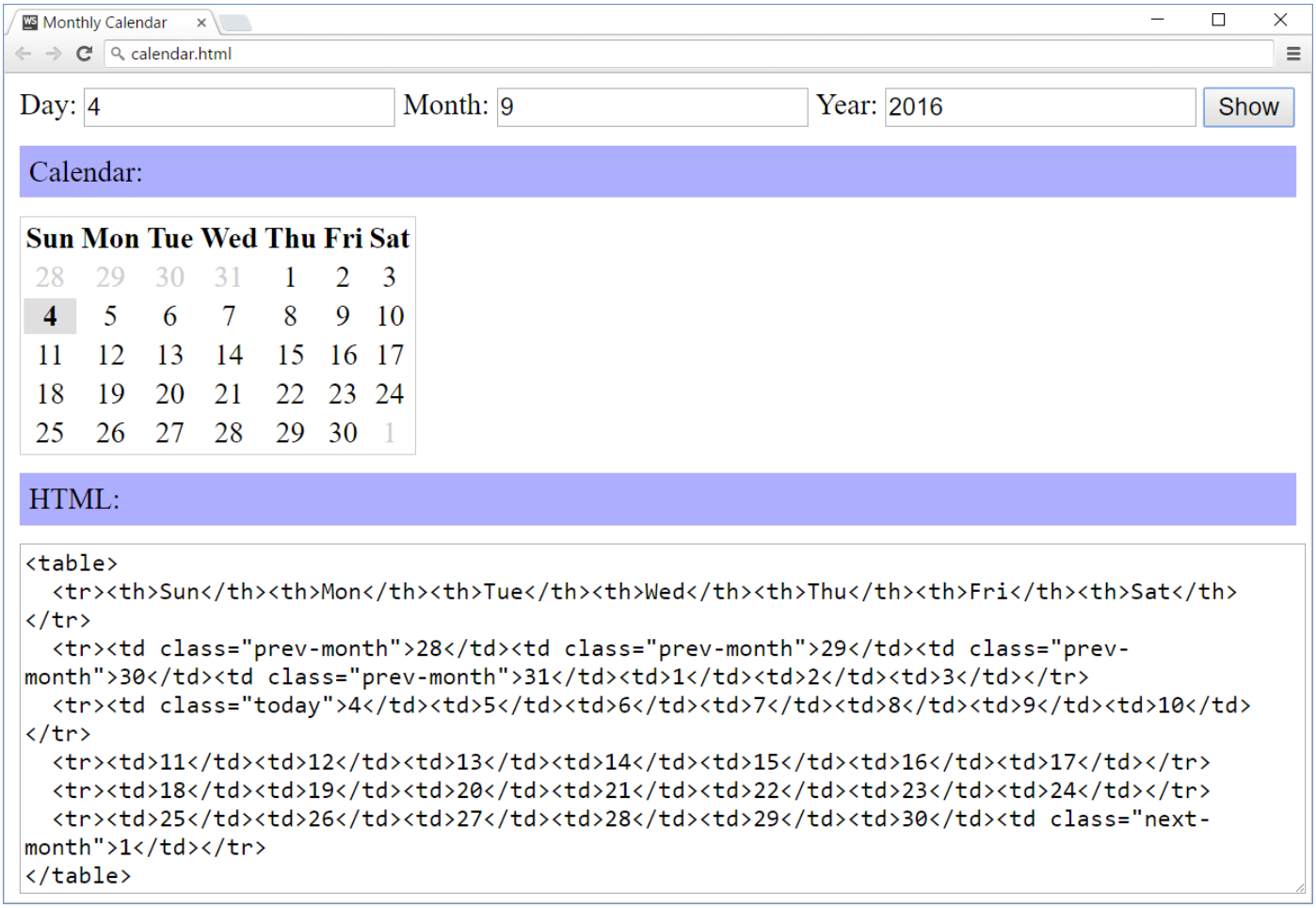
```
        table { border: 1px solid #CCC;}
        td { text-align: center; }
        #calendarCode { width: 100%; }
    </style>
    <script>
        function calendar([day, month, year])
        {
            // TODO: return the HTML text holding the calendar table
        }
    </script>
</head>

<body>
    Day: <input id="day" type="number" value="4" />
    Month: <input id="month" type="number" value="9" />
    Year: <input id="year" type="number" value="2016" />
    <input type="button" value="Show"
        onclick="let calendarHTML =
            calendar([document.getElementById('day').value,
                document.getElementById('month').value,
                document.getElementById('year').value]);
            document.getElementById('calendar').innerHTML = calendarHTML;
            document.getElementById('calendarCode').innerText = calendarHTML" />
    <div class="title">Calendar:</div>
    <div id="calendar">Calendar will be shown here</div>
    <div class="title">HTML:</div>
    <textarea rows="12" id="calendarCode"></textarea>
</body>

</html>
```

Submit in the **judge system** the JS code of your **calendar()** function only, without the above HTML code that visualizes the calendar in the Web browser.

Screenshot

This is how your calendar should look in Web browser, when the **calendar()** function is implemented correctly:

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.



Hints

- Printing a calendar in JS without using an external library is not as simple as the previous problems in this exercise! It may take a few hours or even more to implement it correctly.
- Play with the class **Date** in JavaScript and make some calculations.
- You may also search in Google, e.g. try "**JavaScript calendar code example**".
- Print the calendar table **headings** + days.
- Print the days from the **previous month** (if any).
  - Find the day of week for the first day of the input date.
  - If it is not Sunday, days from the previous month exist.
  - Go back a few days in the previous month to find the closest Sunday (first week day).
  - Start from it and print the days until the end of the previous month.
- Print the days from the **current month**.
  - Print the days, one after another.
  - Create a new table row after the last week day (Saturday).
- Print the days from the **next month** (if any).
  - Stop when you reach Saturday (the last week day).
- You may start from this code template:

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

```
function calendar([day, month, year])
{
    // Print the calendar table header
    let html = '<table>\n';
    // TODO: finish this code

    // Print the days of the previous month
    // TODO: finish this code

    // Print the days of the current month
    // TODO: finish this code

    // Print the days of the next month
    // TODO: finish this code

    html += '</table>';
    return html;
}
```

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.