

Exercises: Data Types, Expressions, Statements

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/308>.

1. Hello, JavaScript!

Write a JS function that can receive a name as input and print a greeting to the console.

The **input** comes as a single string that is the name of the person.

The **output** should be printed to the console.

Examples

Input	Output
Pesho	Hello, Pesho, I am JavaScript!
Bill Gates	Hello, Bill Gates, I am JavaScript!

Hints

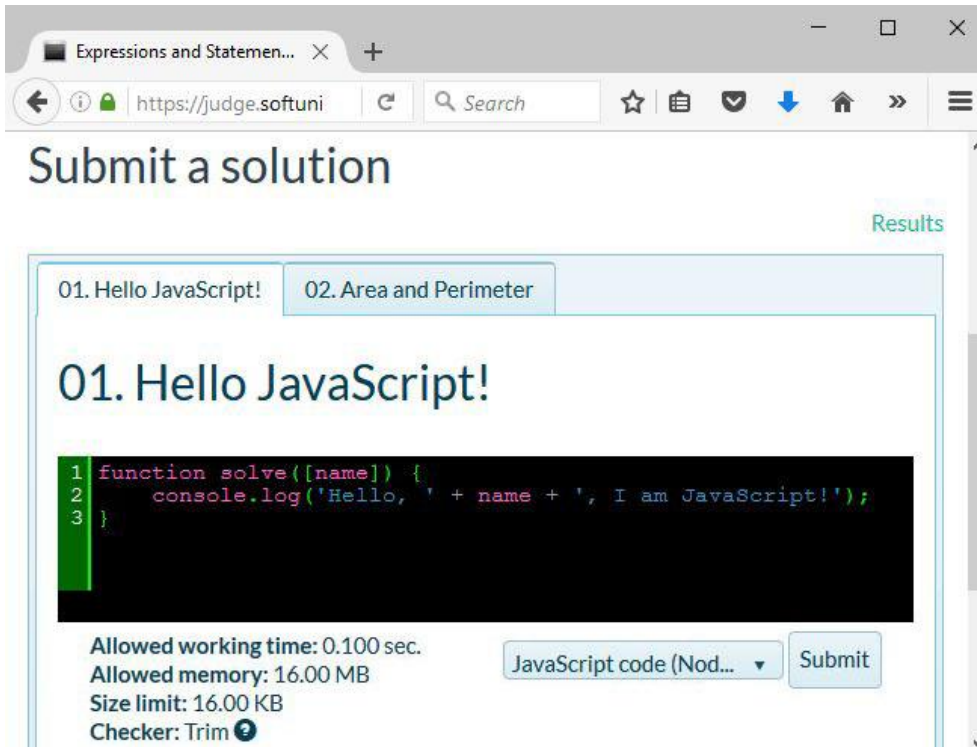
We would get the name from a single string variable that would be passed to our method.

```
function solve(name) {  
  
}
```

We need to concatenate three strings – the two static parts of our greeting and the name of the person in the middle. We can do this by simply adding the three strings with the addition operator. Since this is an operation which returns the concatenated string, we can directly perform this expression in a call to **console.log()**. Note the space at the end of the first string:

```
console.log('Hello, ' + name + ', I am JavaScript!');
```

You should be ready to submit your solution to the **judge system**. Open the judge contest for this homework and submit your code: <https://judge.softuni.bg/Contests/308>. It should look like this:



Submit a solution

Results

01. Hello JavaScript! 02. Area and Perimeter

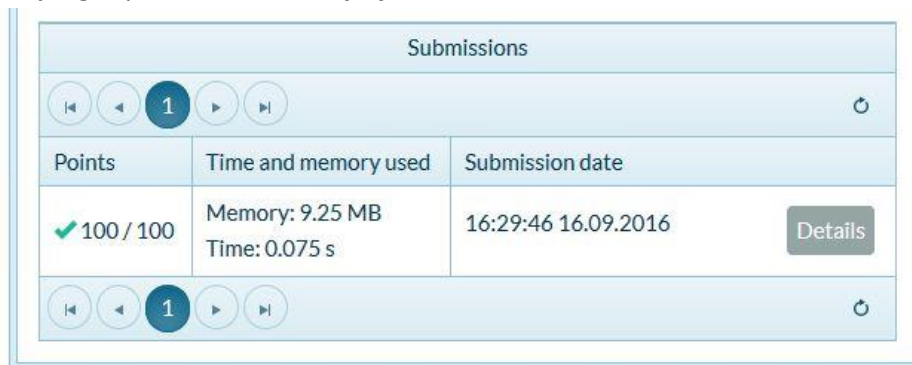
01. Hello JavaScript!

```
1 function solve([name]) {  
2   console.log('Hello, ' + name + ', I am JavaScript!');  
3 }
```

Allowed working time: 0.100 sec.
Allowed memory: 16.00 MB
Size limit: 16.00 KB
Checker: Trim

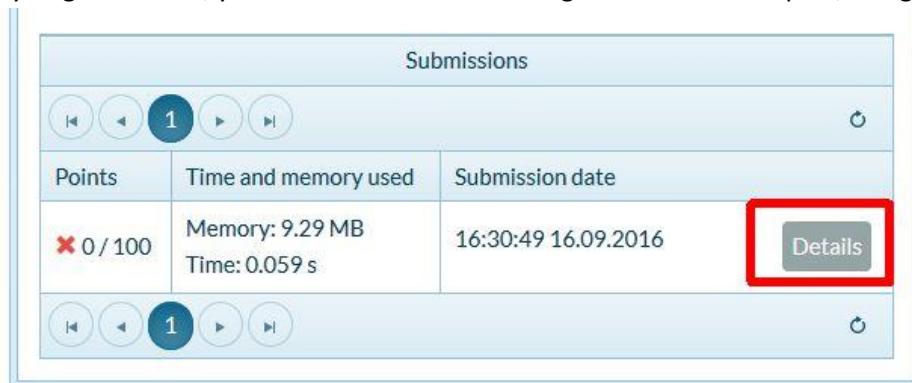
JavaScript code (Nod... Submit

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



Submissions		
Points	Time and memory used	Submission date
✓ 100 / 100	Memory: 9.25 MB Time: 0.075 s	16:29:46 16.09.2016

If you get an error, you can see what went wrong in the detailed report, using the highlighted button:



Submissions		
Points	Time and memory used	Submission date
✗ 0 / 100	Memory: 9.29 MB Time: 0.059 s	16:30:49 16.09.2016

Here you can see what the system expected and what your program's result was. You can also see what the input for the test was:

Zero test #1 (Incorrect answer) Run #19070086 Test #22483

The zero tests are not included in the final result.

Show input

Expected output:

1 Hello, Pesho, I am JavaScript!
2

Your output:

1 Hello, Pesho, I am JavaScript!
2

Time used: 0.075 s

Memory used: 9.22 MB

Note you can only view detailed information for zero tests – they do not give you points, but are handy for finding typos and debugging some errors. Chances are, if you manage to get all zero tests to pass, you'll also have some competitive tests passing too.

2. Area and Perimeter

Write a JS function that calculates the area and perimeter of a rectangle by given two sides.

The **input** comes as 2 numbers that are the lengths of the 2 sides of the rectangle (sideA and sideB)

The **output** should be printed to the console on two lines.

Examples

Input	Output
2 2	4 8
1 3	3 8
2.5 3.14	7.85 11.28

Hints

The multiplication operator will automatically coerce the input variables to numbers, so we can directly find the area of the rectangle by multiplying the two input elements.

```
function solve(a, b) {  
  
}
```

The remaining operations are straightforward arithmetic and finally printing the two results (area and perimeter) to the console.

3. Distance over Time

Two objects start from point **A** and travel in the same direction at constant speeds V_1 and V_2 for a period T . Write a JS function that calculates the distance between the two object at the end of the period.

The **input** comes as array of numbers. The first two elements are the speeds to the two objects in km/h and the third element is the time in seconds.

The **output** should be printed to the console. Calculate the distance in meters.

Examples

Input	Output
[0, 60, 3600]	60000
[11, 10, 120]	33.33333333333337
[5, -5, 40]	111.11111111111111

Hints

Speed, time and distance are related to each other with the following formula:

$$S = V * T$$

However, the incoming units need to be equalized first and after the calculation, a final conversion needs to be done to match the required output units. There are 3600 seconds in an hour and 1000 meters in a kilometer. We don't know which object covered a greater distance, so simply subtracting them from one another may result in a **negative number**. Distance however is absolute (always positive), so we need to get the absolute value of the operation, using the built-in **Math.abs(number)** functions:

```
let delta = Math.abs(dist1 - dist2);
```

4. Distance in 3D

Write a JS function that calculates the distance between the two points in 3D by given coordinates.

The **input** comes as an array of numbers. The first three elements are the x, y and z coordinates for the first point and the second set of arguments are the coordinates of the other point.

The **output** should be printed to the console.

Examples

Input	Output
[1, 1, 0, 5, 4, 0]	5
[3.5, 0, 1, 0, 2, -1]	4.5

Hints

You can find the horizontal and vertical offset between two points by calculating the difference between their respective coordinates. Use Pythagoras' theorem to find the distance.

5. Grads to Degrees

Land surveyors use grads (also known as gon, 400 grads in a full turn) in their documents. Grads are rather unwieldy though, so you need to write a JS function that converts from grads to degrees. In addition, your program needs to constrain the results within the range $0^\circ \leq x < 360^\circ$, so if you arrive at a value like -15° , it needs to be converted to 345° and 420° becomes 60° .

The **input** comes as single number.

The **output** should be printed to the console.

Examples

Input	Output
100	90

Input	Output
400	0

Input	Output
850	45

Input	Output
-50	315

Hints

You can use the remainder (modulo) operator to get a value that is cyclic – it returns the same result for all input values with offset equal to the second parameter. For instance, `n % 10` will return 3 with values for `n` 3, 13, 23, 243, 1003 and so on.

6. Compound Interest

Write a JS function that calculates the total accumulated value for a monetary deposit by given principal sum, interest rate, compounding frequency and overall length.

The **input** comes as an array of numbers. The first value is the principal sum, the second is the interest rate in percent, the third is the compounding period in months and the final value is the total timespan, given in years.

The **output** should be printed to the console, with two decimal places.

Examples

Input	Output
[1500, 4.3, 3, 6]	1938.84

Input	Output
[100000, 5, 12, 25]	338635.49

Hints

The formula for calculating compound interest is as follows:

$$F = P \left(1 + \frac{i}{n} \right)^{nt}$$

where:

- P is the principal sum
- i is the nominal interest rate
- n is the compounding frequency
- t is the overall length of time the interest is applied

Note that at the beginning of the task you are given the compounding period, which is inversely related to the frequency. You need to express the frequency as how many times in a year the interest is compounded. For

instance, a 3-month period means the interest will be updated 4 times in a year. Any percentages need to be expressed as a fraction.

7. *Rounding

Write a JS function that rounds numbers to specific precision.

The **input** comes as an array of two numbers. The first value is the number to be rounded and the second is the precision (significant decimal places). If a precision is passed, that is more than **15** it should automatically be reduced to **15**.

The **output** should be printed to the console. Do not print insignificant decimals.

Examples

Input	Output
[3.1415926535897932384626433832795, 2]	3.14

Input	Output
[10.5, 3]	10.5

8. Imperial Units

Imperial units are confusing, but still in use in some backwards countries (Myanmar, Liberia and the United States are the only countries still using them). They are so confusing in fact, that native users struggle to convert between them. Write a JS function that converts from inches to feet and inches. There are 12 inches in a foot. See the example for formatting details.

The **input** comes as a single number.

The **output** should be printed to the console.

Examples

Input	Output
36	3' - 0"

Input	Output
55	4' - 7"

Input	Output
11	0' - 11"

9. Now Playing

Write a JS function that displays information about the currently playing musical track.

The **input** comes as an array of string elements. The first element is the name of the track, the second is the name of the artist performing and the third is the duration in minutes and seconds.

The **output** should be printed to the console in the following format:

Now Playing: {artist name} - {track name} [{duration}]

Examples

Input	Output
['Number One', 'Nelly', '4:09']	Now Playing: Nelly - Number One [4:09]

10. Compose Tag

Write a JS function that composes an HTML image tag.

The **input** comes as an array of string elements. The first element is the location of the file and the second is the alternate text.

The **output** should be printed to the console in the following format:

```

```

Examples

Input	Output
['smiley.gif', 'Smiley Face']	

11. Binary to Decimal

Write a JS function that reads an 8-bit binary number and converts it to a decimal.

The **input** comes as one string element, representing a binary number.

The **output** should be printed to the console.

Examples

Input	Output	Input	Output
00001001	9	11110000	240

12. Assign Properties

Write a JS function that composes an object by given properties. There will be 3 sets of property-value pairs (a total of 6 elements). Assign each value to its respective property of an object and return the object as a result of the function.

The **input** comes as an array of string elements.

The **output** should be returned as a value.

Examples

Input
['name', 'Pesho', 'age', '23', 'gender', 'male']
Output
{ name: 'Pesho', age: '23', gender: 'male' }

Input
['ssid', '90127461', 'status', 'admin', 'expires', '600']
Output
{ ssid: '90127461', status: 'admin', expires: '600' }

13. *Last Month

Write a JS function that receives a date as array of strings containing **day**, **month** and **year** in that order. Your task is to print the last day of previous month (the month **BEFORE** the given date). Check the examples to better understand the problem.

The **input** comes as an array of numbers.

The **output** should be a single number representing the **last day** of the previous month.

Examples

Input	Output	Input	Output
[17, 3, 2002]	28	[13, 12, 2004]	30

Conditional Statements and Loops

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.

14. Biggest of 3 Numbers

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

The **input** comes as array of 3 numbers.

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

Examples

Input	Output	Input	Output	Input	Output	Input	Output	Input	Output
5 -2 7	7	130 5 99	130	43 43.2 43.1	43.2	5 5 5	5	-10 -20 -30	-10

Hints

Read the input and assign each number to a variable:


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

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 = 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:

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

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

The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following code and its output:

```

> function biggestNumber(input){
    let num1 = input[0];
    let num2 = input[1];
    let num3 = input[2];

    console.log(Math.max(num1, num2, num3));
}

biggestNumber([5, -2, 7]);
biggestNumber([130, 5, 99]);
biggestNumber([43, 43.2, 43.1]);
biggestNumber([5, 5, 5]);
biggestNumber([-10, -20, -30]);

```

The output shows the results of the function calls:

7	VM74:6
130	VM74:6
43.2	VM74:6
5	VM74:6
-10	VM74:6

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])

```

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

Enjoy!

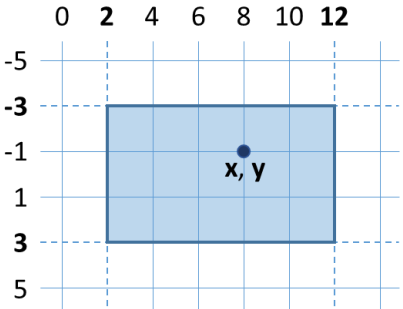
15. Point in Rectangle

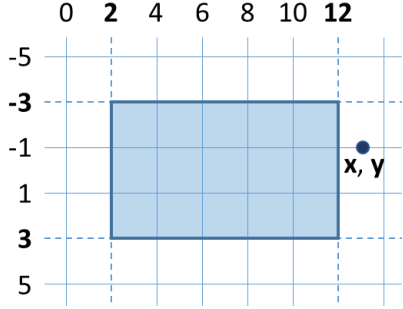
Write a JS function that takes as input 6 numbers **x**, **y**, **xMin**, **xMax**, **yMin**, **yMax** and prints whether the point {**x**, **y**} is **inside** the rectangle or **outside** of it. If the point is at the rectangle **border**, it is considered **inside**.

The **input** comes as array of numbers. Its holds the 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 numbers, passed as input parameter **input**. The numbers should be taken from the array elements **input[0]**, **input[1]**, ... as follows:

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

The above code could be significantly shortened by using a JS language feature called “**destructuring assignment**” like this:

```
function pointInRect(input) {  
    let [x, y, xMin, xMax, yMin, yMax] = input;  
    // 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:

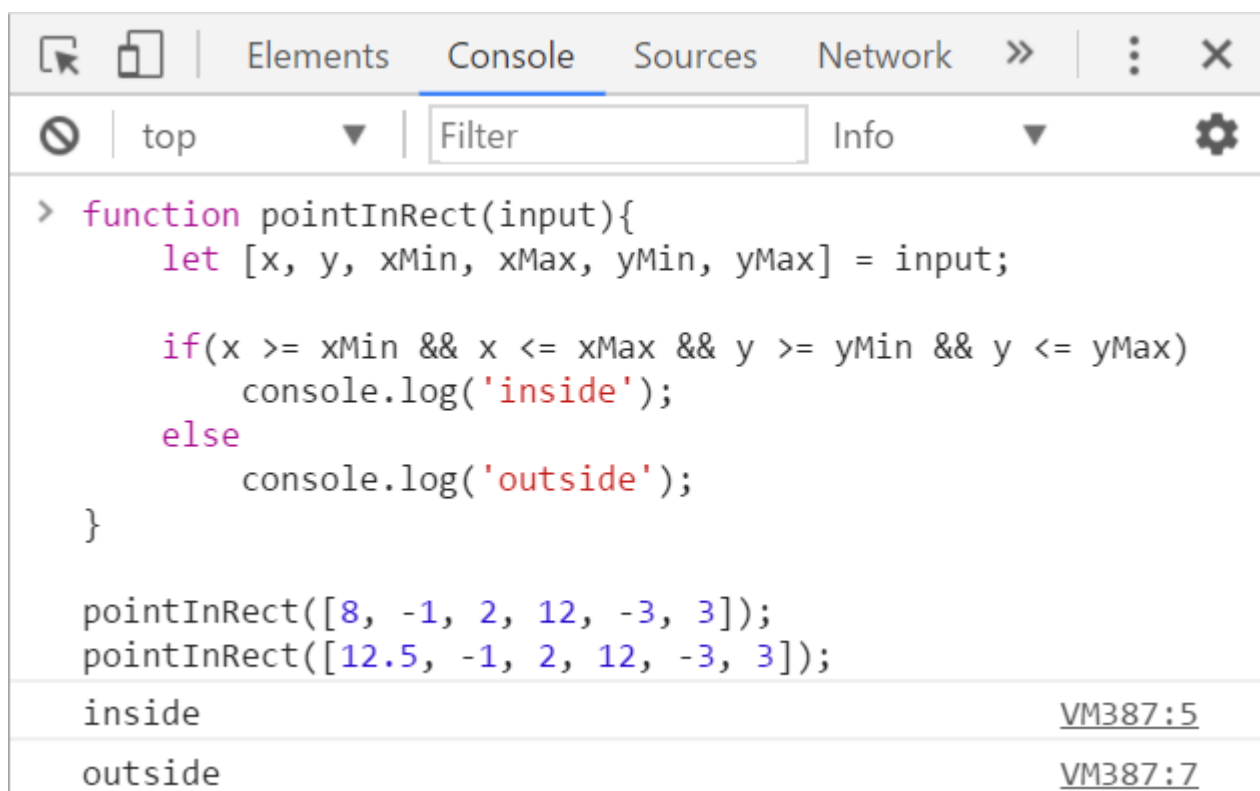
```
if (x >= xMin && x <= xMax && y >= yMin && y <= yMax)
    console.log('inside');
else
    console.log('outside');
```

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 **6 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**:

```
Command Prompt - node

> function pointInRect(input){
...   let [x, y, xMin, xMax, yMin, yMax] = input;
...   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 screenshot shows the 'Submit a solution' interface for the 'Point in Rectangle' contest. The page has a header with the contest name and a 'Results' link. Below the header, there are tabs for 'Point in Rectangle', 'Odd Numbers 1 to N', and 'Add task'. The 'Point in Rectangle' tab is active, showing a code editor with the following JavaScript code:

```
1 function pointInRect(input) {
2   let [x, y, xMin, xMax, yMin, yMax] = input.map(Number);
3   if (x >= xMin && x <= xMax && y >= yMin && y <= yMax)
4     console.log('inside');
5   else
6     console.log('outside');
7 }
```

Below the code editor, the allowed working time is 0.200 sec and the allowed memory is 16.00 MB. There is a dropdown menu for the programming language set to 'JavaScript code (Node.js)' and a 'Submit' button.

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

The screenshot shows the 'Submissions' page for the 'Point in Rectangle' contest. The page has a header with the contest name and a 'Results' link. Below the header, there are tabs for 'Submissions', 'Odd Numbers 1 to N', and 'Add task'. The 'Submissions' tab is active, showing a table of submissions. The table has three columns: 'Points', 'Time and memory used', and 'Submission date'. The first submission is highlighted with a blue circle and a 'Details' button.

Points	Time and memory used	Submission date
✓✓✓✓✓✓✓✓ 100 / 100	Memory: 9.32 MB Time: 0.246 s	14:39:19 03.09.2016

16. 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 a single number **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	Input	Output	Input	Output
5	1 3 5	4	1 3	7	1 3 5 7

Hints

First write a JS function to read the input number **n**.

```
function oddNumsInRange(n) {  
    // 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:

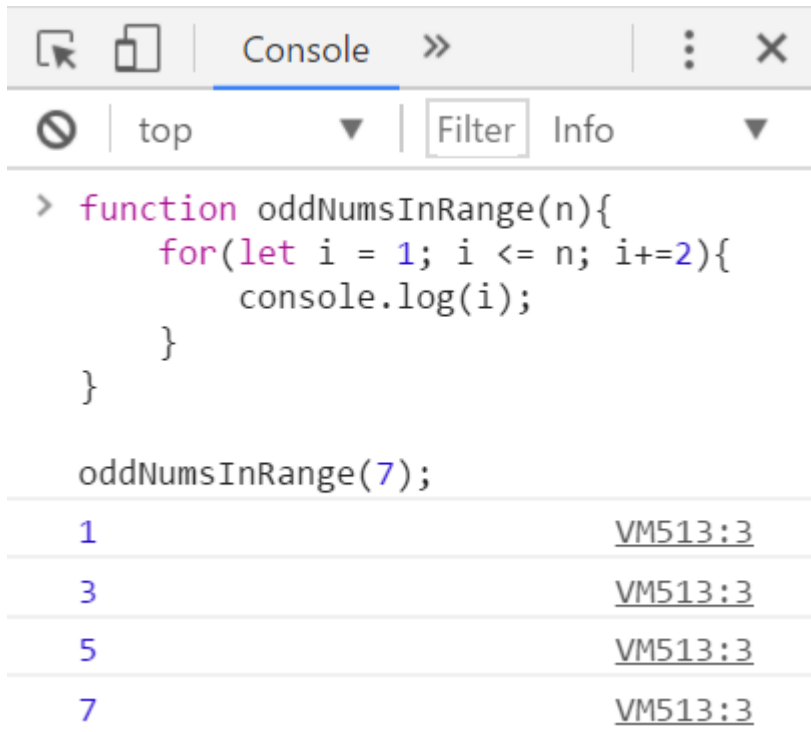
```
function oddNumsInRange(n) {  
    for(let i = 1; i <= n; i+=2) {  
        console.log(i);  
    }  
}
```

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

```
oddNumsInRange(7);
```

You may test your code in the console (with Node.js) or in your Web browser's JS console.

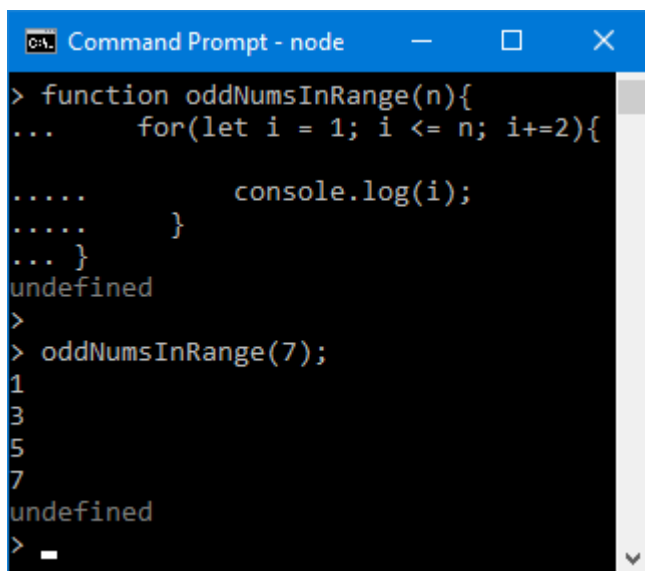
The screenshot below shows the above function, invoked in **Chrome JavaScript Console**:



```
> function oddNumsInRange(n){  
    for(let i = 1; i <= n; i+=2){  
        console.log(i);  
    }  
}  
  
oddNumsInRange(7);
```

1	VM513:3
3	VM513:3
5	VM513:3
7	VM513:3

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



```
> function oddNumsInRange(n){  
...   for(let i = 1; i <= n; i+=2){  
.....       console.log(i);  
.....   }  
... }  
undefined  
>  
> oddNumsInRange(7);  
1  
3  
5  
7  
undefined  
>
```

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:

Exercises: Conditiona x

https://judge.softuni.bg/Contests/Practice/Index/300#0

Submit a solution

Results

1. Odd Numbers 1 to N Add task

1. Odd Numbers 1 to N

Participants Tests Change Delete

Administration

```

1 function nums(input) {
2   let n = Number(input[0]);
3   for (let i=1; i<=n; i+=2) {
4     console.log(i);
5   }
6 }

```

Allowed working time: 0.200 sec.
Allowed memory: 16.00 MB

JavaScript code (Nod... Submit

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

Exercises: Conditiona x

https://judge.softuni.bg/Contests/Practice/Index/300#0

Submissions

1

Points	Time and memory used	Submission date
✓✓✓✓✓ 100 / 100	Memory: 9.39 MB Time: 0.074 s	18:49:11 02.09.2016 Details

1

17. 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 a single number **n** ($0 < n < 100$).

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

Examples

Input	Output	Input	Output	Input	Output
3	\$ \$\$ \$\$\$	2	\$ \$\$	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));
```

18. 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	Input	Output	Input	Output	Input	Output
The Godfather Friday	15	casablanca sunday	10	Schindler's LIST monday	8.50	SoftUni Nineday	error

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"]));
```

19. 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 3 numbers **a**, **b** and **c**. The value of **a** will be non-zero.

The **output** depends on the equation:

- It holds two numbers **x₁** and **x₂** if the equation has two different solutions (roots): **x₁** and **x₂**.
 - 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).

Examples

Input	Output	Comments
6 11 -35	-3.5 1.66667	<p>The equation is: $6x^2 + 11x - 35 = 0 \rightarrow a = 6; b = 11; c = -35$</p> <p>The discriminant is: $d = b^2 - 4*a*c = 11*11 - 4*6*(-35) = 121 + 840 = 961$</p> <p>We have positive discriminant ($d > 0$), so the equation has two real roots:</p> <ul style="list-style-type: none"> $x_1 = (-b + \sqrt{d}) / (2*a) = (-11 + 31) / 12 = 20/12 = 5/3 \approx 1.66667$ $x_2 = (-b - \sqrt{d}) / (2*a) = (-11 - 31) / 12 = -42/12 = -7/2 = -3.5$ <p>The output consists of two lines:</p> <ul style="list-style-type: none"> -3.5 (the smaller root at the first line) 1.66667 (the bigger root at the second line)

1 -12 36	6	<p>The equation is: $x^2 - 12x + 36 = 0 \rightarrow a = 1; b = -12; c = 36$</p> <p>The discriminant is: $d = b^2 - 4*a*c = -12*-12 - 4*1*36 = 144 - 144 = 0$</p> <p>We have zero discriminant ($d = 0$), so the equation has only one real root:</p> <ul style="list-style-type: none"> $x = -b / (2*a) = 12 / 2 = 6$ <p>The output is only one line, holding the number 6.</p>
5 2 1	No	<p>The equation is: $5x^2 - 2x + 1 = 0 \rightarrow a = 5; b = -2; c = 1$</p> <p>The discriminant is: $d = b^2 - 4*a*c = -2*-2 - 4*5*1 = 4 - 20 = -16$</p> <p>We have negative discriminant ($d < 0$), so the equation has no real roots.</p> <p>The output is only one line, holding the text "No".</p>

Hints

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

20. Multiplication Table

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

The **input** comes as a single number **n** ($0 < n < 100$).

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

Examples

Input	5																																				
Output	<pre><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</td><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></pre>																																				
Preview	<table><tr><td>x</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>2</td><td>2</td><td>4</td><td>6</td><td>8</td><td>10</td></tr><tr><td>3</td><td>3</td><td>6</td><td>9</td><td>12</td><td>15</td></tr><tr><td>4</td><td>4</td><td>8</td><td>12</td><td>16</td><td>20</td></tr><tr><td>5</td><td>5</td><td>10</td><td>15</td><td>20</td><td>25</td></tr></table>	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
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 = multiplicationTable(5);
```

21. 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	Input	Output	Input	Output	Input	Output
4	+ - - + - - + + - - + - - + + - - + - - +	5	+ - - + - - + + - - + - - + + - - + - - +	6	+ - - - + - - - + + - - - + - - - + + - - - + - - - +	7	+ - - - - + - - - - + + - - - - + - - - - + + - - - - + - - - - + + - - - - + - - - - +

Hints

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

22. ** 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 numbers:

- **day** ($1 \leq \text{day} \leq 31$)
- **month** ($1 \leq \text{month} \leq 12$)
- **year** ($1900 \leq \text{year} \leq 2100$)

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>

	<pre><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></pre>																																																	
Preview	<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>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>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>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>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr></table>	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
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	<pre><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></pre>																																										
Preview	<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>28</td><td>29</td><td>30</td><td>31</td><td>1</td><td>2</td><td>3</td></tr><tr><td>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>1</td></tr></table>	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
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
<pre> <!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 } </pre>

```

    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:

Monthly Calendar
calendar.html

Day:
Month:
Year:

Calendar:

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:

```

<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>

```

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:

```
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;
}
```