# Exercises: Data Types, Expressions, Statements

Problems for exercises and homework for the [“JavaScript Fundamentals” course @ SoftUni](https://softuni.bg/courses/javascript-fundamentals). Submit your solutions in the SoftUni Judge system at <https://judge.softuni.bg/Contests/308>.

## Hello, JavaScript! - JavaScript task

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 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:

## Area and Perimeter – JavaScript task

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.

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

## Distance over Time

Two objects start from point ***A*** and travel in the same direction at constant speeds ***V1*** and ***V2*** 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:



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:  


## 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.

## Grads to Degrees - JavaScript task

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 , 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** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 100 | 90 |  | 400 | 0 | 850 | 45 | -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.

## Compound Interest – JavaScript task

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 time span, given in years.

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

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| [1500, 4.3, 3, 6] | 1938.84 |  | [100000, 5, 12, 25] | 338635.49 |

### Hints

The formula for calculating compound interest is as follows:



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.

## \*Rounding in JavaScript

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** |  | **Input** | **Output** |
| [3.1415926535897932384626433832795, 2] | 3.14 |  | [10.5, 3] | 10.5 |

## Imperial Units - convert from inches to feet and inches in JavaScript

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** |  | **Input** | **Output** |  | **Input** | **Output** |
| 36 | 3'-0" | 55 | 4'-7" | 11 | 0'-11" |

## 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] |

## Compose Tag - compose an HTML image tag (JavaScript task)

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:

<img src="{file location}" alt="{alternate text}">

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['smiley.gif', 'Smiley Face'] | <img src="smiley.gif" alt="Smiley Face"> |

## Binary to Decimal – conversion in JavaScript

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 |

## 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' } |

## \*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.

## 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 |

## 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** |  | **Input** | **Output** | **Figure** |
| 8  -1  2  12  -3  3 | inside |  | 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:



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



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:



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

C:\Users\Valio\AppData\Local\Microsoft\Windows\INetCache\Content.Word\f8.png

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**:



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:



## 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 |

## 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 | $  $$ | 4 | $  $$  $$$  $$$$ |

## 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.

## 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 | The equation is: 6*x*2 + 11*x* - 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:   * x1 = (-*b* + √*d*) / (2\**a*) = (-11 + 31) / 12 = 20/12 = 5/3 ≈ 1.66667 * x2 = (-*b* - √*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 - 12*x* + 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: 5*x*2 - 2*x* + 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).

## 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** | <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> |
| **Preview** |  |

### 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:

C:\Users\Valio\AppData\Local\Microsoft\Windows\INetCache\Content.Word\fn5.png

## 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.

## \*\* 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 ≤ 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> |
| **Preview** |  |

|  |  |
| --- | --- |
| **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** |  |

### 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: 10**px 0**; padding:5**px** }  **table** { border: 1**px 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:



### 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**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/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).