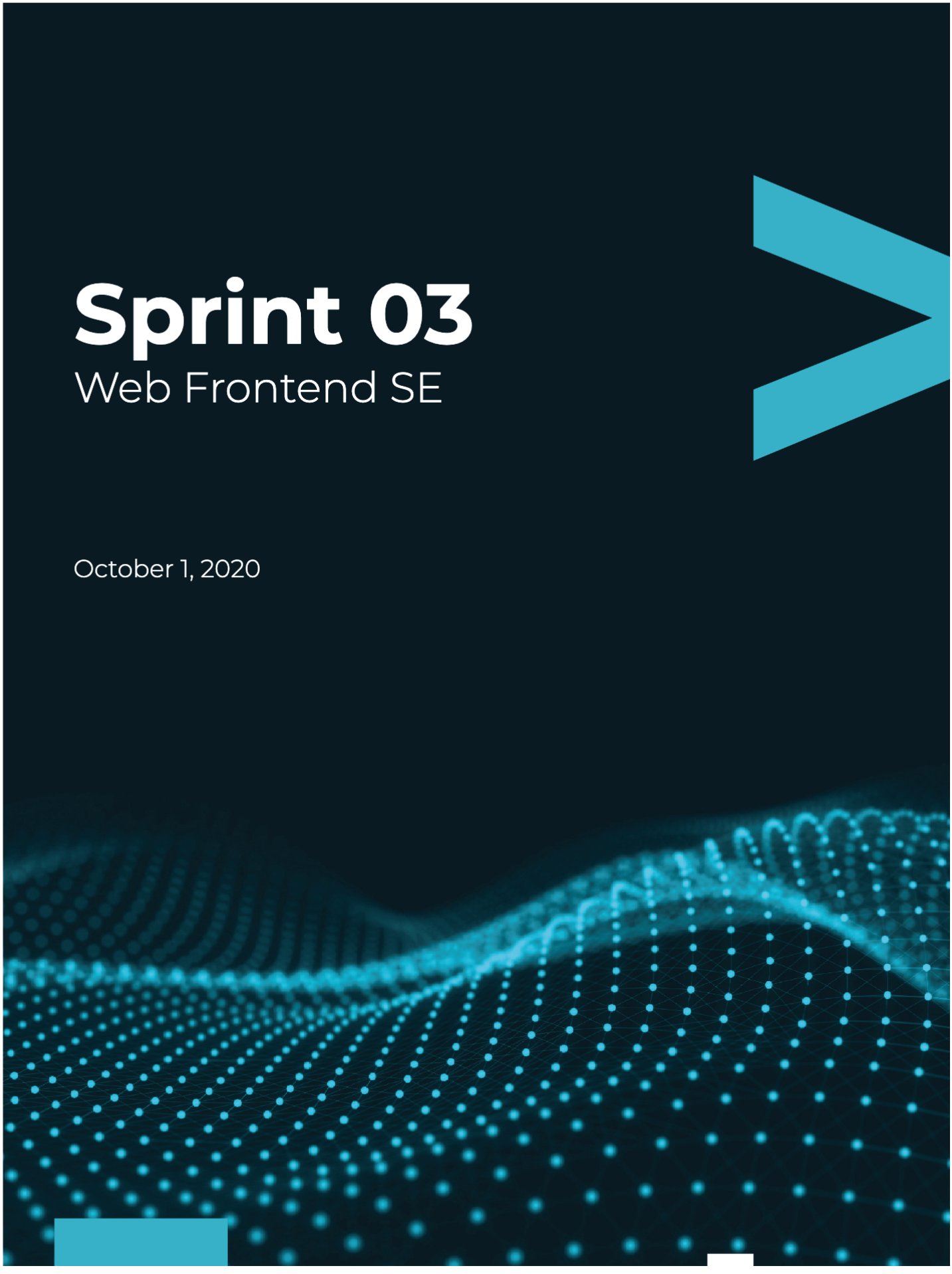

Sprint 03
Web Frontend SE
October 1, 2020

Contents


ucode

Engage

DESCRIPTION

Hello, future frontend superhero! What are your first impressions of JavaScript?

In the previous Sprint, you have familiarized yourself with the basic notions in JS and practiced implementing simple logic with loops and conditional statements. This week, you will get acquainted with some new concepts in JS that will allow you to interact with the HTML and CSS content of a web page.

Among other programming paradigms, JavaScript allows for object-oriented programming. Even though it's possible to code in JS without OOP, its features and mechanisms are used for communicating with the content of a web page, and needed for many native libraries and methods. In this Sprint you will practice creating JS objects and manipulating them with different functions.

You have learned how to add JS scripts to an HTML file, and now it’s time to progress to actually manipulating an HTML page and its elements using JavaScript. This is where the Document Object Model (DOM) steps in: it represents all elements of an HTML page as objects, allowing you to change and manipulate them with JavaScript.

Ready? Let's go!

BIG IDEA

Deepening in JavaScript.

ESSENTIAL QUESTION

What is JavaScript good for?

CHALLENGE

Continue learning JS and get acquainted with DOM.


ucode
Sprint 03 | Web Frontend SE > 2

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What are `objects` in programming?
- How to create an object in JavaScript?
- What is an object `property`?
- What loops can be used to traverse the contents of an object?
- How to create a `Date` object?
- How to turn text content into a particular format?
- How to interact with objects?
- What is `Closure`?
- What are the benefits of using Closure?
- What is chaining in JavaScript?
- What is `DOM`?
- How to use DOM in code?
- What are the different ways to get an element from DOM?
- How to create an HTML element using JS?
- How to detect various events happening on a web page?
- What are `cookies`? What are they used for?
- What are the advantages and disadvantages for the user when a web page uses cookies?
- What is `localStorage` in JS?
- What is the difference between `localStorage` and `sessionStorage`?
- What is a `storage` event? How is it created?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Read the story tasks.
- If you haven't understood something from Sprint02 - come back to it and do it once again.
- Read about Objects in JavaScript.
- In a JS file, create an object `student`, add some properties, and then display them.
- Read about DOM.


ucode
Sprint 03 | Web Frontend SE > 3

- Open any web page. In the console, write `window.document` and press return. Analyze the result.
- Create a web page that displays "Hello World!" using `getElementById` method and `innerHTML` property.
- Revise recommendations on good code style for JS.
- Clone your git repository that is issued on the challenge page in the LMS.
- Start to develop the solution. Offer improvements. Test your code.
- Employ the full power of P2P by brainstorming with other students.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Be attentive to all statements of the story. Examine the given examples carefully. They may contain details that are not mentioned in the task.
- Analyze all information you have collected during the preparation stages.
- Complete the `Document` section while developing a challenge. It is described after the `Act` section.
- Perform only those tasks that are given in this document.
- Submit only the specified files in the required directory and nothing else. Garbage shall not pass.
- Pay attention to what is allowed. Use of forbidden stuff is considered a cheat and your challenge will be failed.
- The web page in the browser must open through `index.html`.
- The scripts must be written outside the HTML file - in a separate JS file (script.js).
- You can always use the `Console` panel to test and catching errors.
- Complete tasks according to the rules specified in the following style guides:
  - HTML and CSS: Google HTML/CSS Style Guide. As per section 3.1.7 Optional Tags, it doesn't apply. Do not omit optional tags, such as `<head>` or `<body>`
  - JavaScript:
    * JavaScript Style Guide and Coding Conventions
    * JavaScript Best Practices
- The solution will be checked and graded by students like you. Peer-to-Peer learning.
- If you have any questions or don't understand something, ask other students or just Google it.


ucode
Sprint 03 | Web Frontend SE > 4

Act: Task 00

NAME

Team up!

DIRECTORY

`t00/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`alert(), prompt(), String.*, Object.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.

No matter if you're a superhero, or a super villain, you're stronger in a team! In your script, create an object `superTeam` with the following properties:

- `title` - string with the name of the team
- `leader` - string with the name of the team's leader
- `members` - array of strings with the names of the members of the team
- `memberCount` - total number of members (regular plus the leader)
- `agenda` - string describing what are the goals and ideas of the team
- `isEvil` - boolean that explains whether the team is evil or not

In order to get the values for these properties, prompt the user to enter them one after another. The only value that should not be entered by the user is the `memberCount`. Instead, calculate that value in your code.

For the `members` value, ask the user to enter the names separated by a comma.

After creating the object, display its contents using `alert()`. There is an example of what the structure of the display message must look like in the EXAMPLE section. The contents of the values are up to you.


ucode
Sprint 03 | Web Frontend SE > 5

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Team up!</title>
  <meta name="description" content="t00. Team up!">
</head>

<body>
  <h1>Team up!</h1>

  <script src="js/script.js"></script>
</body>

</html>
```

EXAMPLE

Here's the team:
name - The Brotherhood of Evil Mutants
leader - Magneto
members - Mystique, Toad, The Blob, Avalanche, Pyro, Mastermind, Multiple Man
memberCount - 8
agenda - They seek to establish mutants as the dominant race on the planet by any means necessary.
isEvil - true

SEE ALSO

JavaScript Objects
JavaScript object basics


ucode
Sprint 03 | Web Frontend SE > 6

Act: Task 01

NAME

Date

DIRECTORY

`t01/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`String.*, Array.*, Date.*, Object.*`

DESCRIPTION

Create a function `getFormattedDate()` that takes a date and formats it into a particular way as shown in the EXAMPLE.

Your JS file with the function should be included in the HTML file written in the SYNOPSIS. The test file in the EXAMPLE section is an example of how to test your function. Also, see the SYNOPSIS for the function prototype.

SYNOPSIS

```
  getFormattedDate(dateObject) : string
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Date</title>
  <meta name="description" content="t01. Date">
</head>

<body>
  <h1>Date</h1>

  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>

</html>
```


ucode
Sprint 03 | Web Frontend SE > 7

EXAMPLE

```
const date0 = new Date(1993, 11, 1);
const date1 = new Date(1998, 0, -33);
const date2 = new Date('42 03:24:00');

console.log(getFormattedDate(date0)); // 01.12.1993 00:00 Wednesday
console.log(getFormattedDate(date1)); // 28.11.1997 00:00 Friday
console.log(getFormattedDate(date2)); // 01.01.2042 03:24 Wednesday
```

SEE ALSO

Date and time


ucode
Sprint 03 | Web Frontend SE > 8

Act: Task 02

NAME

Clone of Steve Rogers

DIRECTORY

`t02/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`Object.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
The script must contain a function that makes a copy of the `user` object and its properties.

You can test your function using the test.js file written in the EXAMPLE. Add more test cases of your own.
Note: a copy of an object is not the same thing as a link to an object.

SYNOPSIS

```
function copyObj(obj)
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Clone of Steve Rogers</title>
  <meta name="description" content="t02. Clone of Steve Rogers">
</head>

<body>
  <h1>Clone of Steve Rogers</h1>

  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>

</html>
```


ucode
Sprint 03 | Web Frontend SE > 9

EXAMPLE

```
const user = {
  name: 'Steve',
  surname: 'Rogers',
  age: 101,
  city: 'New York'
};

console.log(user);
// {name: "Steve", surname: "Rogers", age: 101, city: "New York"}
let cpy = copyObj(user);
console.log(cpy);
// {name: "Steve", surname: "Rogers", age: 101, city: "New York"}


user.name = 'John';
console.log(user);
// {name: "John", surname: "Rogers", age: 101, city: "New York"}
console.log(cpy);
// {name: "Steve", surname: "Rogers", age: 101, city: "New York"}

cpy.age = 59;
console.log(user);
//{name: "John", surname: "Rogers", age: 101, city: "New York"}
console.log(cpy);
// {name: "Steve", surname: "Rogers", age: 59, city: "New York"}
```


ucode
Sprint 03 | Web Frontend SE > 10

Act: Task 03

NAME

Word for word

DIRECTORY

`t03/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

String.*, Array.*, Object.*

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS and tested using a test JS file, an example of which is available in the EXAMPLE section.

Implement several functions that manipulate the object `obj`. This object has the property `words`. A word is a string separated by a single space. Whitespace is not a word.

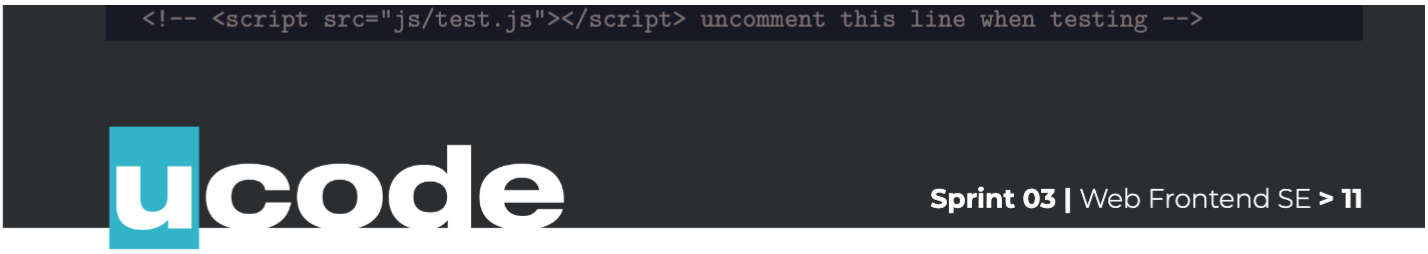
Create the following functions:

- `addWords(obj, wrds)` - add a string with words to the object's property
- `removeWords(obj, wrds)` - remove specified words from the object's property
- `changeWords(obj, oldWrds, newWrds)` - change one or more words in the object's property

Clear duplicates and spaces in the returned object.

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Word for word</title>
  <meta name="description" content="t03. Word for word">
</head>

<body>
  <h1>Word for word</h1>

  <script src="js/script.js"></script>
```


<!-- <script src="js/test.js"></script> uncomment this line when testing -->
ucode
Sprint 03 | Web Frontend SE > 11

```
</body>

</html>
```

EXAMPLE

```
const obj = {
  words: 'newspapers newspapers  books magazines'
};

console.log(obj); // {words: "newspapers newspapers  books magazines"}

addWords(obj, 'radio newspapers   ');
console.log(obj); // {words: "newspapers books magazines radio"}

removeWords(obj, 'newspapers   radio');
console.log(obj); // {words: "books magazines"}

changeWords(obj, 'books radio  magazines', 'tv internet');
console.log(obj); // {words: "tv internet"}
```

SEE ALSO

Array.prototype.splice()
Array.prototype.indexOf()


ucode
Sprint 03 | Web Frontend SE > 12

Act: Task 04

NAME

Hulk Closure

DIRECTORY

`t04/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`prompt(), String.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
The script must contain a function that concatenates two strings in two different ways:

- if a function is called with two strings, the result is a concatenated string
- if a function is called with one string, the result is a function that prompts for a second string

Also, a function must have the property `count` that counts the sub function call.

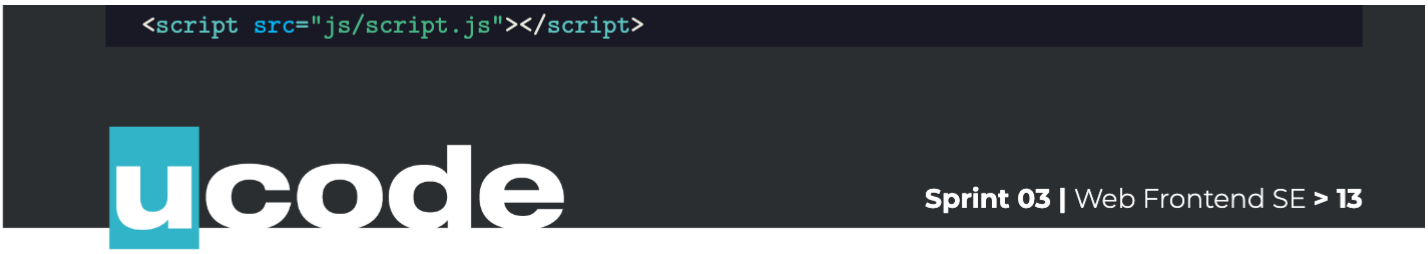
You can test your function using the test.js file written in the EXAMPLE. Add more test cases of your own.

In this task, you must use the `Closure` concept of JavaScript.

SYNOPSIS

```
concat(string1, string2) : string
concat(string1) : func1
func1.count : number
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Hulk Closure</title>
  <meta name="description" content="t04. Hulk Closure">
</head>

<body>
  <h1>Hulk Closure</h1>
```


<script src="js/script.js"></script>
ucode
Sprint 03 | Web Frontend SE > 13

```
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>

</html>
```

EXAMPLE

```
let phrase1 = concat("Hulk", "smash!");
let output = phrase1;
console.log(output); // Hulk smash!

let phrase2 = concat("Leave");
output = phrase2();
// a prompt appears. Enter "Hulk alone!" into the prompt

console.log(output); // Leave Hulk alone!
console.log(phrase2.count); // 1

output = phrase2();
// a prompt appears. Enter "me alone, please!" into the prompt

console.log(output); // Leave me alone, please!

output = phrase2();
// a prompt appears. Enter "HULK ALONE!" into the prompt

console.log(output); // Leave HULK ALONE!
console.log(phrase2.count); // 3

let phrase3 = concat("Go");
output = phrase3();
// a prompt appears. Enter "away!" into the prompt

console.log(output); // Go away!
console.log(phrase3.count); // 1
console.log(phrase2.count); // 3

/* Result in Console panel:
Hulk smash!
Leave Hulk alone!
1
Leave me alone, please!
Leave HULK ALONE!
3
Go away!
1
3
*/
```


ucode
Sprint 03 | Web Frontend SE > 14

SEE ALSO

JavaScript Closures
Variable scope


ucode
Sprint 03 | Web Frontend SE > 15

Act: Task 05

NAME

Calculator

DIRECTORY

`t05/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`alert(), setTimeout()`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS and tested using a test JS file, an example of which is available in the EXAMPLE section.

In your JS file, create a function-constructor that creates a calculator object with the following methods:

- `init(num)` - set value for calculating
- `add(num)` - addition
- `sub(num)` - subtraction
- `mul(num)` - multiplication
- `div(num)` - division
- `alert()` - alert-message with the current result after a 5 seconds delay

The calculator has the property: `result`.

Alert from the `Calculator.alert()` method must show with a 5 seconds delay.

In this task, you can use the `Closure` and `Chaining` concepts of JavaScript.

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Calculator</title>
  <meta name="description" content="t05. Calculator">
```


</head>
ucode
Sprint 03 | Web Frontend SE > 16

```
<body>
  <h1>Calculator</h1>

  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>

</html>
```

EXAMPLE

```
const calc = new Calculator();

console.log(
    calc
        .init(2)
        .add(2)
        .mul(3)
        .div(4)
        .sub(2).result // 1
);

calc.alert();
```


ucode
Sprint 03 | Web Frontend SE > 17

Act: Task 06

NAME

Secret lab

DIRECTORY

`t06/`

SUBMIT

`index.html, css/style.css, js/script.js`

ALLOWED FUNCTIONS

DOM

DESCRIPTION

Create a function that changes the content and style of a web page. Use the HTML and CSS files available in the SYNOPSIS.

As you can see, on click of the button, the function `transformation()` is being called. Create this function inside your `script.js` file.

The function switches between two states of the web page.
State 1:

- displays `Bruce Banner` with `60px` font size and `2px` letter spacing
- has a `#ffb300` background color

State 2:

- displays `Hulk` with `130px` font size and `6px` letter spacing
- has a `#70964b` background color

It must be possible to keep endlessly switching between the two states.


ucode
Sprint 03 | Web Frontend SE > 18

SYNOPSIS

HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Secret lab</title>
  <meta name="description" content="t06. Secret lab<">
  <link rel="stylesheet" href="css/style.css">
  <link href="https://fonts.googleapis.com/css2?family=Bowlby+One&display=swap"
        rel="stylesheet">
</head>

<body>
<h1>Secret lab</h1>

<p>
  Click <strong>Transform</strong>
  to transform from a man into a monster, or from a monster into a man.
</p>

<div id="lab">
  <h2 id="hero">Bruce Banner</h2>
  <input id="btn" type="button" onclick="transformation()" value="Transform">
</div>

<script src="js/script.js"></script>
</body>

</html>
```


ucode
Sprint 03 | Web Frontend SE > 19

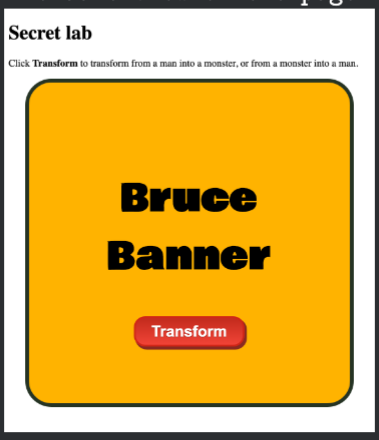
CSS

```css
#lab {
  display: flex;
  align-items: center;
  justify-content: center;
  padding:10px;
  border-radius: 8%;
  width: 500px;
  height: 500px;
  margin: auto;
  flex-direction: column;
  background: #ffb300;
  border: 6px solid #253324;
  text-align: center;
}

#btn{
  display: inline-block;
  padding: 7px 25px;
  cursor: pointer;
  box-shadow: 3px 4px 0 1px #8a2a21;
  background: #c62d1f linear-gradient(to bottom, #c62d1f 5%, #f24437 100%);
  border-radius: 18px;
  border: 3px solid #d02718;
  color: #ffffff;
  text-align: center;
  font-size: 25px;
  font-weight: bold;
  text-decoration: none;
  text-shadow: 0 1px 0 #810e05;
}

#btn:hover {
  background: #f24437 linear-gradient(to bottom, #f24437 5%, #c62d1f 100%);
}

#btn:active {
  position: relative;
  top: 1px;
}

#hero{
  font-size: 60px;
  font-family: 'Bowlby One', cursive;
  letter-spacing: 2px;
}
```


ucode
Sprint 03 | Web Frontend SE > 20

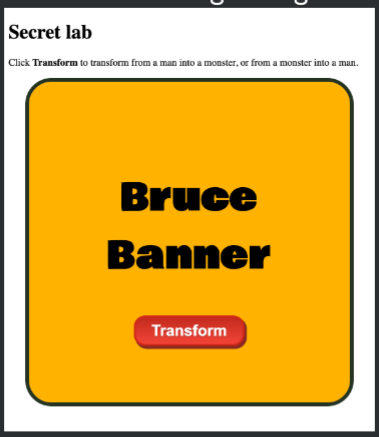
EXAMPLE

Default view of the page



After clicking "Transform"

After clicking it again



SEE ALSO

Introduction to the DOM
HTML DOM getElementById() Method

HTML DOM Style Object

Act: Task 07

NAME

Elements

DIRECTORY

`t07/`

SUBMIT

`index.html, css/style.css, js/script.js`

ALLOWED FUNCTIONS

DOM, String.*, Array.*

DESCRIPTION

Create a JS script that changes the content of a web page. In this task you will need to work with HTML attributes. Use the HTML and CSS files available in the SYNOPSIS.

The web page contains a list of characters. Each character has some attributes. The `class` attribute specifies whether the character is good or evil. And the `data-element` attribute lists the elements that the character can control.

Compare the appearance of the web page with the screenshot in the EXAMPLE. Without editing the HTML and CSS files, use JS only to achieve the same result.

For each `li` element, your script must do the following:

- correct errors in attributes
  - if the `class` attribute is missing, or doesn't equal to `good`, `evil` or `unknown`, make the `class` equal to `unknown`
  - if the `data-element` attribute is missing, make the `data-element` equal to `none`
- append circle `div` elements depending on the attributes
  - for each `data-element` attribute, append a circle
  - each circle must have two class names: `elem`, and the name of the respective `data-element` attribute
  - to each `none` circle, append a `div` element with the class name `line`

Do not hardcode the solution. Your script must work, even if the contents of the list changes (items are added, removed, attributes are changed, etc.).


ucode
Sprint 03 | Web Frontend SE > 22

SYNOPSIS

HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Elements</title>
  <meta name="description" content="t07. Elements<">
  <link rel="stylesheet" href="css/style.css">
  <link href="https://fonts.googleapis.com/css2?family=Bowlby+One&display=swap"
        rel="stylesheet">
</head>

<body>
  <h1>Elements</h1>
  <ul id="characters">
    <li class="good" data-element="air water earth fire">Aang</li>
    <li class="evil" data-element="fire">Zuko</li>
    <li class="good" data-element="water">Katara</li>
    <li class="good">Sokka</li>
    <li class="good" data-element="fire">Iroh</li>
    <li class="evil" data-element="fire">Azula</li>
    <li class="good" data-element="air water earth fire">Korra</li>
    <li>Suki</li>
    <li class="good" data-element="earth">Toph Beifong</li>
  </ul>
  <script src="js/script.js"></script>
</body>

</html>
```


ucode
Sprint 03 | Web Frontend SE > 23

CSS

```css
body {
  background-color: #333;
  text-align: center;
  max-width: 800px;
  margin: 0 auto;
}

h1 {color: white;}

li {
  list-style: none;
  display: inline-block;
  background-color: lightgrey;
  padding: 15px;
  margin: 5px;
  border-radius: 10%;
  text-align: center;
  border: none;
  width: 120px;
}

.good {border: 3px solid #59a440;}
.evil {border: 3px solid #ba2d29;}
.unknown {border: 3px solid #764cae;}

.elem {
  display: inline-block;
  width: 20px;
  height: 20px;
  border-radius: 50%;
  border: 2px solid white;
  margin: 3px;
}

.air {background-color: #5a8de1;}
.water {background-color: #0f1b8b;}
.earth {background-color: #496b2e;}
.fire {background-color: #9f000e;}
.none {background-color: lightgrey;}

.line {
  position: relative;
  width: 20px;
  height: 2px;
  background-color: white;
  border-radius: 0;
  top: 9px;
  transform: rotate(-225deg);
```


}
ucode
Sprint 03 | Web Frontend SE > 24

EXAMPLE

SEE ALSO

`Element.attributes`
`HTML | DOM appendChild() Method`


ucode
Sprint 03 | Web Frontend SE > 25

Act: Task 08

NAME

Sort table

DIRECTORY

`t08/`

SUBMIT

`index.html, css/style.css, js/script.js`

ALLOWED FUNCTIONS

String.*, Array.*, Object.*, DOM

DESCRIPTION

Create a web page with a table using JS. Follow these requirements:

- a table must have three columns and ten rows
- the first column must be filled with Marvel superheroes
- the second must be filled with the hero's strength
- the second must be filled with the hero's age
- the web page must have click events for the headers of the table:
  - sort the table according to the selected column by rearranging the rows. Sorting can be performed in ascending and descending order
  - show a service message Sorting by [parameter], order: __. Instead of "__" must be "ASC" or "DESC"

The design of your web page must look like the image in the EXAMPLE. Use CSS to achieve the result.

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1" />
  <title>Sort table</title>
  <meta name="description" content="t08. Sort table">

  <link rel="stylesheet" href="css/style.css">
  <script src="js/script.js" defer></script>
```


ucode
Sprint 03 | Web Frontend SE > 26

```
<body>
  <main>
    <h1>Sort table</h1>

    <div id="placeholder">Placeholder with <b>HTML</b></div>
    <div id="notification">Click on cell to sort the column</div>
  </main>
</body>

</html>
```

EXAMPLE

Sort table

Sorting by Name, order: ASC

Name	Strength	Age
Black Panther	66	53
Captain America	79	137
Captain Marvel	97	26
Hulk	80	49
Iron Man	88	48
Spider-Man	78	16
Thanos	99	1000
Thor	95	1000
Yon-Rogg	73	52


ucode
Sprint 03 | Web Frontend SE > 27

Act: Task 09

NAME

Tic tac toe

DIRECTORY

`t09/`

SUBMIT

`index.html, css/style.css, js/script.js`

ALLOWED FUNCTIONS

`String.*, Array.*, Object.*, DOM`

DESCRIPTION

Create a classic game of Tic tac toe using HTML, CSS, and JavaScript.

In terms of the layout, recreate the layout in the EXAMPLE section. There is a grid of nine elements, and a sidebar containing information about the game. The `Player 1` and `Player 2` boxes indicate whose turn it is. Below, there is a turn counter, and, in case the game is over, a message about the winner. In the bottom of the sidebar, there is a button to reset the board.

Behavior requirements:

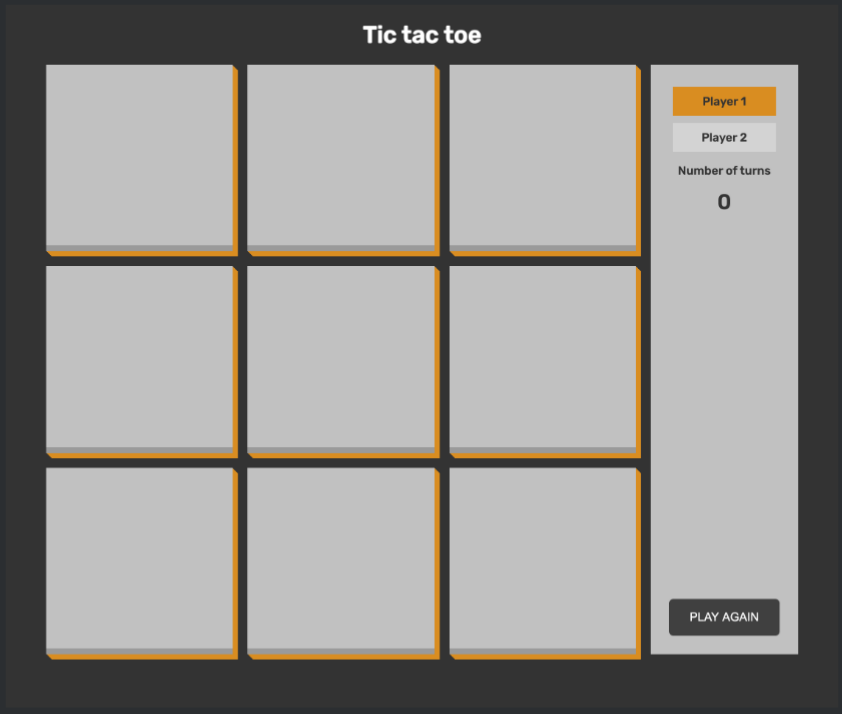
- player 1 plays with `x`, player 2 - with `o`
- player 1 always has the first turn, even when the game is restarted
- players can place their mark by clicking an empty space
- it's not possible to place a mark over an existing one
- it's not possible to skip a turn
- once a mark is placed, the turn switches automatically to the other player
- if there are three marks of the same type in a horizontal, vertical, or diagonal line, the game ends and the sidebar displays a message telling which player won
- if there are no empty spaces available, the game also ends, and the sidebar displays a message telling that it's a draw
- if the game is won, and there are still empty spaces available, it's not possible to place a mark in them
- if the game is won, the winning three-mark line is highlighted in some way

The style of the game is up to you. However, make sure to make it appealing and user-friendly.

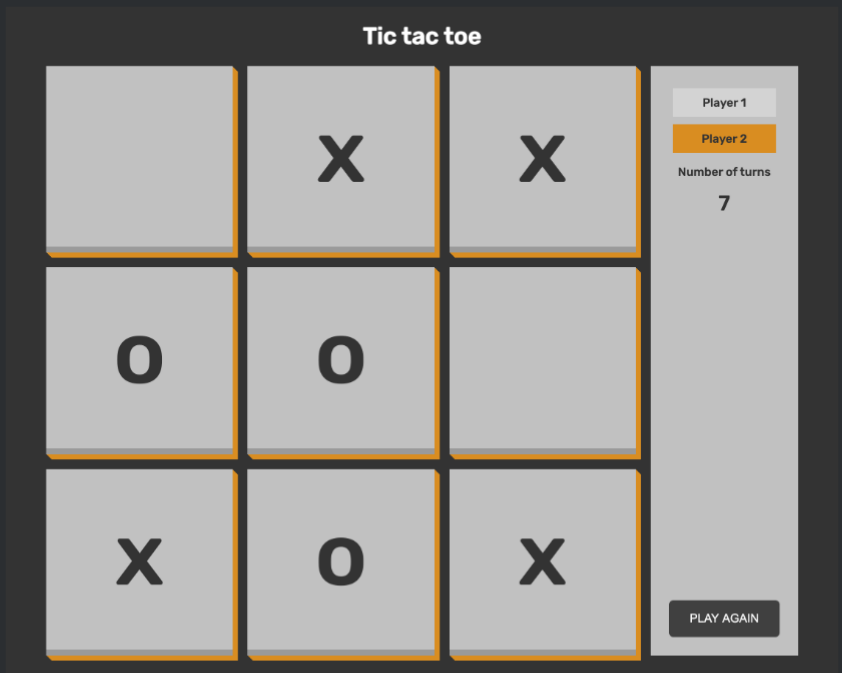

ucode
Sprint 03 | Web Frontend SE > 28

EXAMPLE

At the start of the game, empty board.

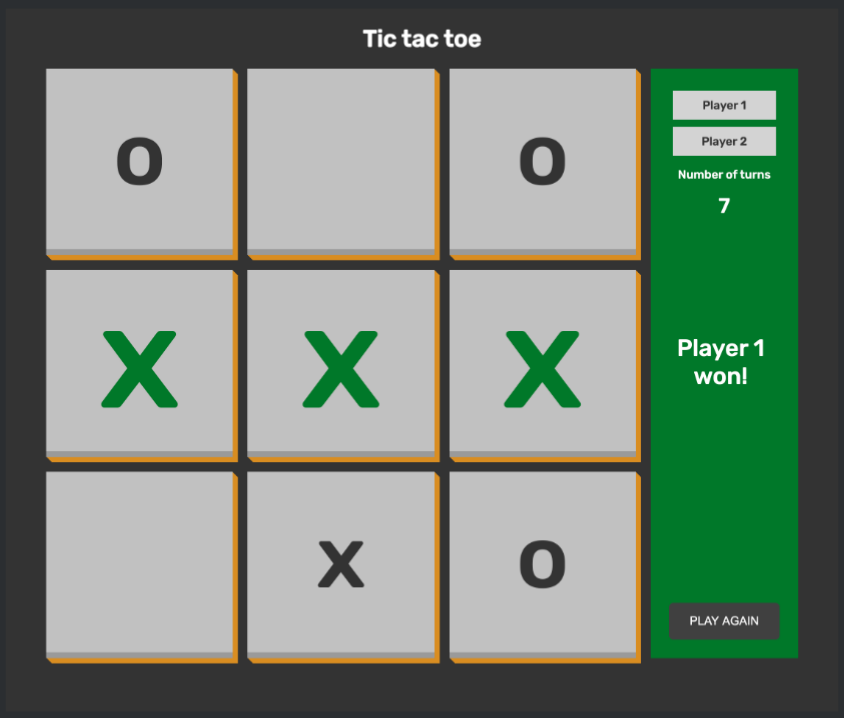


Game in progress.

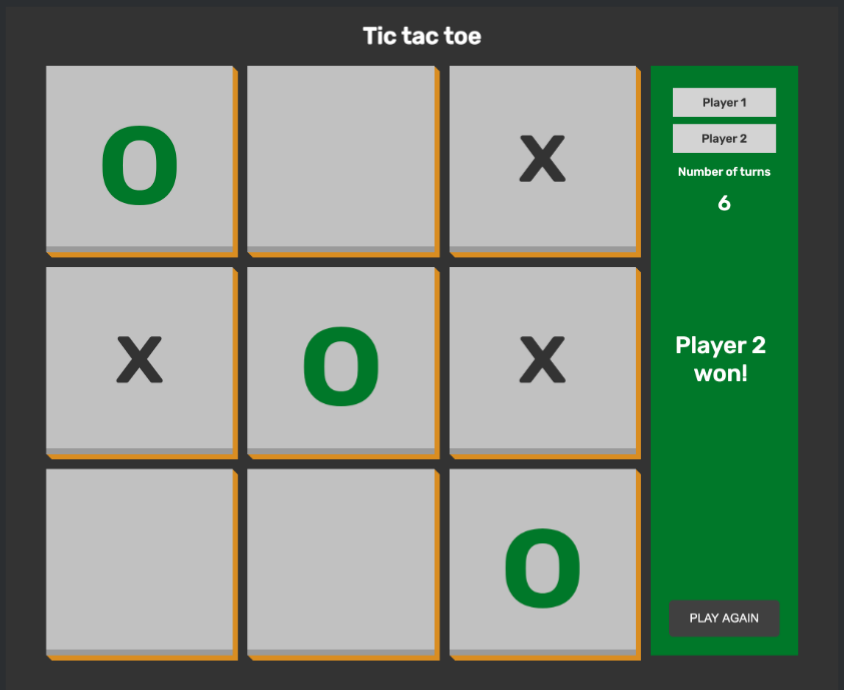



ucode
Sprint 03 | Web Frontend SE > 29

```
Player 1 won.
```

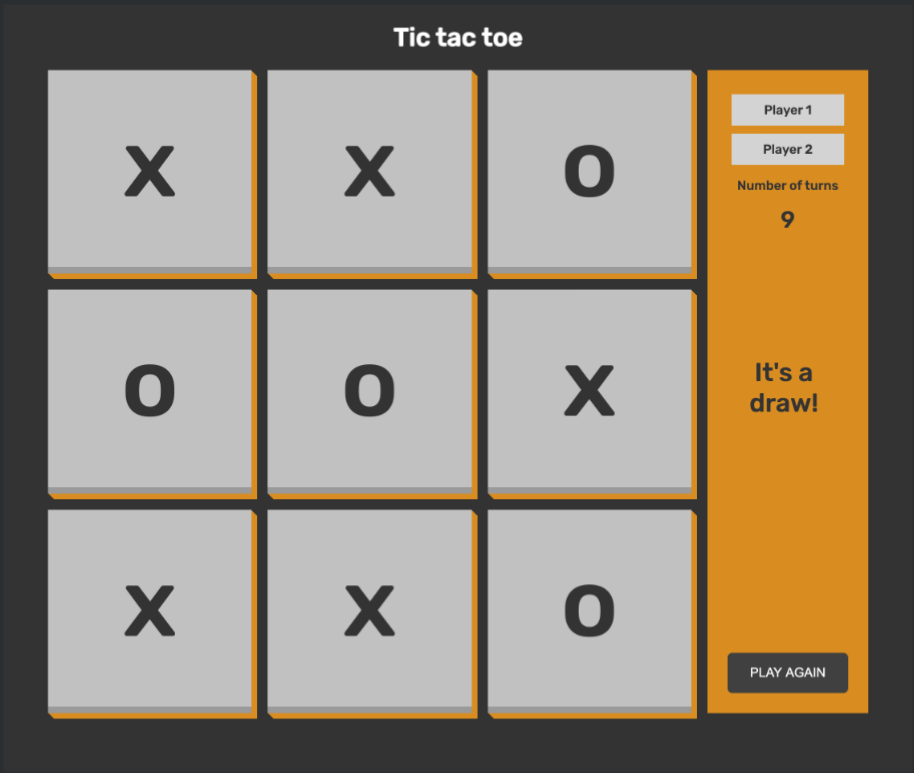



```
Player 2 won.
```




ucode
Sprint 03 | Web Frontend SE > 30

End condition with no winner.



SEE ALSO

HTML DOM Events


ucode
Sprint 03 | Web Frontend SE > 31

Act: Task 10

NAME

Image slider

DIRECTORY

`t10/`

SUBMIT

`index.html, css/style.css, js/script.js`

ALLOWED FUNCTIONS

`String.*, Array.*, Object.*, DOM, setInterval(), clearInterval()`

DESCRIPTION

Create a web page with an image slider. Follow these requirements:

- one image is shown at a time
- the "left" and "right" buttons let the user to see the previous/next image
- the images slide every 3 seconds by default until user click any button to slide the image

The design of your web page must look like the image in the EXAMPLE. Use CSS to achieve the result.

EXAMPLE


ucode
Sprint 03 | Web Frontend SE > 32

Act: Task 11

NAME

Notes with cookies

DIRECTORY

`t11/`

SUBMIT

`index.html, css/style.css, js/script.js`

ALLOWED FUNCTIONS

`alert(), confirm(), String.*, Array.*, Date.*, Object.*, DOM, JSON.*`

DESCRIPTION

Create a note-tracking web page that contains:

- text area field `Text input`
- button `Add to cookies`
- button `Clear cookies`
- output field `Notes archive`

Look at the EXAMPLE. The design is up to you.

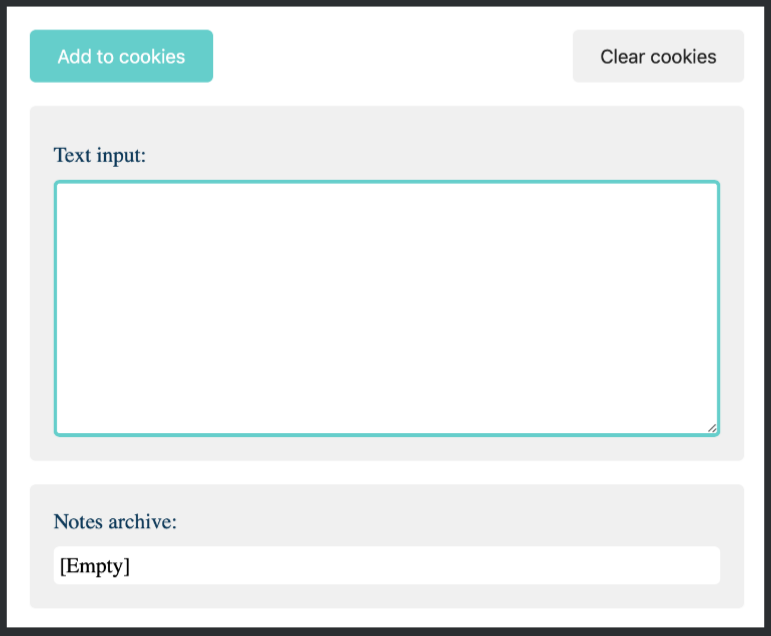
Behavior of the web page:

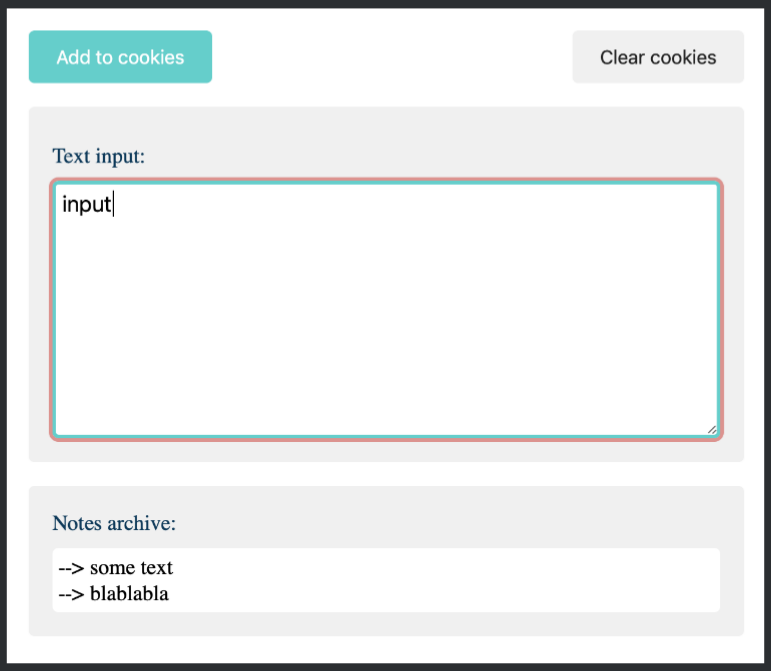
- if the button `Add to cookies` is pressed - the text is added to the `Notes archive`
- if the text input area is empty - an alert box appears with a message `It's empty. Try to input something in "Text input".`
- if the button `Clear cookies` is pressed - the confirm box appears with a message `Are you sure?`, and if the answer is positive - the output field is cleared
- `[Empty]` is displayed in the output field if the web page is launched for the first time, or the date storage has expired

In this task, you must use cookies for implementation. Set the expiry date to 30 days. After refreshing the page, the `Notes archive` stays the same (doesn't clear).


ucode
Sprint 03 | Web Frontend SE > 33

EXAMPLE





SEE ALSO

JavaScript Cookies
Cookies, document.cookie


ucode
Sprint 03 | Web Frontend SE > 34

Act: Task 12

NAME

Notes with local storage

DIRECTORY

`t12/`

SUBMIT

`index.html, css/style.css, js/script.js`

ALLOWED FUNCTIONS

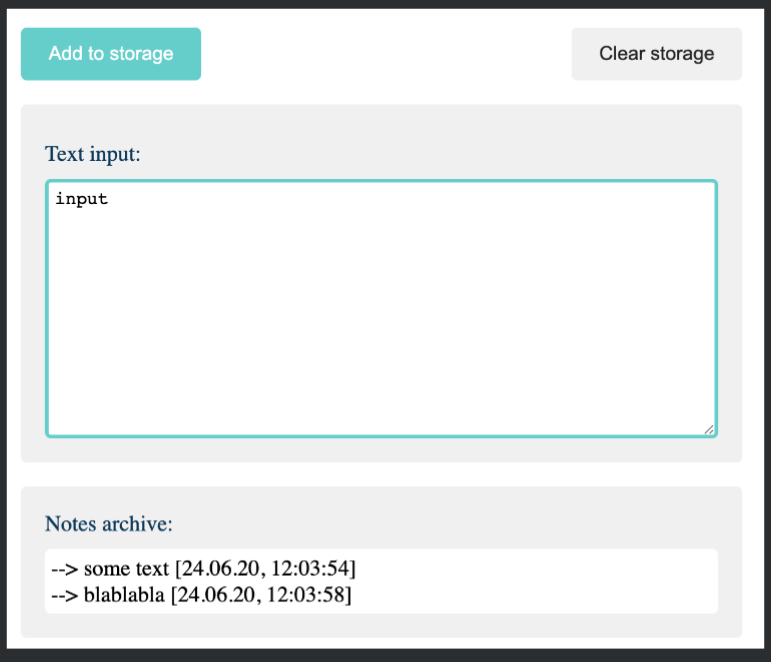
`alert(), confirm(), String.*, Array.*, Date.*, Object.*, RegExp.*, DOM, localStorage.*, JSON.*`

DESCRIPTION

This task is very similar to the previous, but with one key difference, your web page will "remember" things using localStorage instead of cookies.
In this task, you must:

- change data storage from cookies to localStorage
- rename the buttons `Add to storage` and `Clear storage`. Of course, their behavior must be the same as the previous task
- add date display as in the EXAMPLE

EXAMPLE




ucode
Sprint 03 | Web Frontend SE > 35

SEE ALSO

LocalStorage
The complete guide to using localStorage


ucode
Sprint 03 | Web Frontend SE > 36

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences.

During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.


ucode
Sprint 03 | Web Frontend SE > 37