Test Automation Training

Part 4

Prepared By
Yana Yaroshevich

Homework check

Results

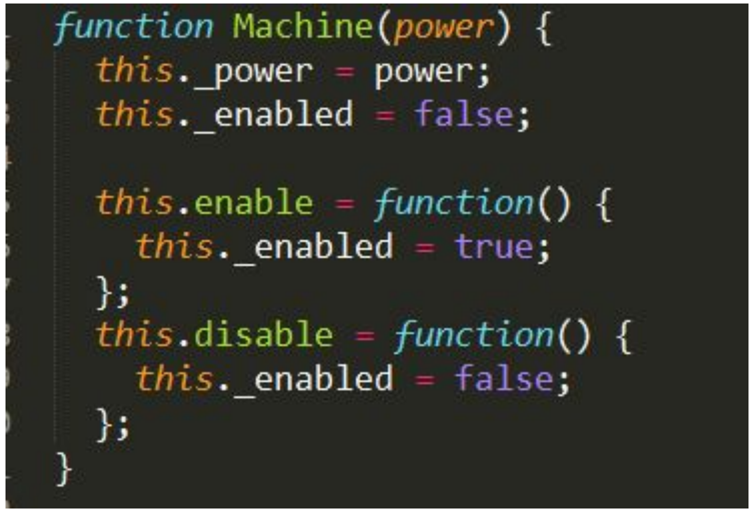
Task 1

Create a class Fridge, which inherits from Machine. Fridge has a private property food and public methods addFood(..) and getFood(..).

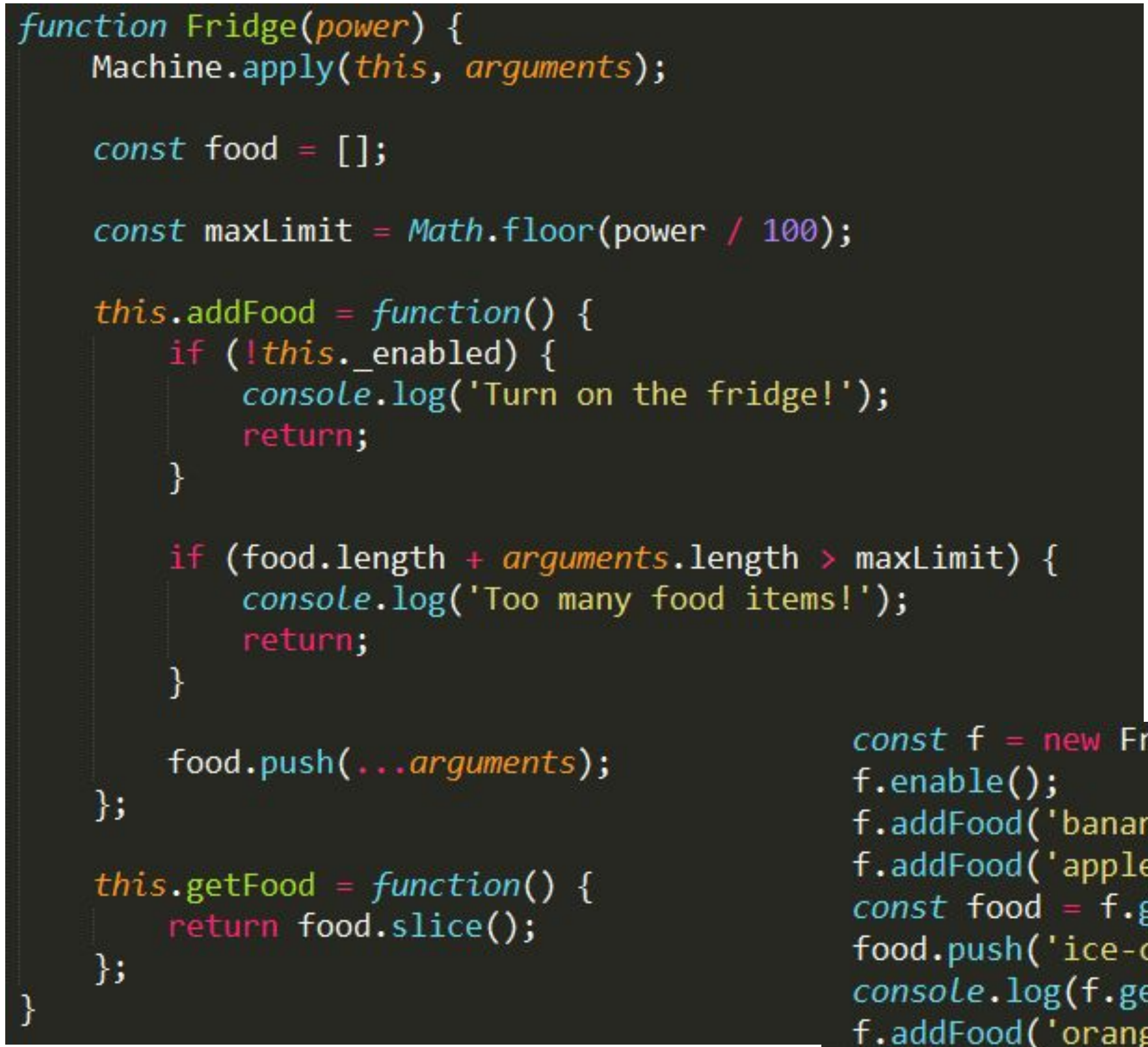
- private property food is an array of food items.
- method addFood(item) adds to food array new food item, you may call it with several arguments: addFood(item1, item2, ...) for adding several food items at once.
- if the fridge is off, it is impossible to add food items (error)
- max quantity of food items is limited and equals to power / 100 - (where power is the power of our fridge (from constructor)). tries to add more food items cause errors.
- public method getFood() returns food array from the fridge. work with this array shouldn't affect the property of the class.

Task 1

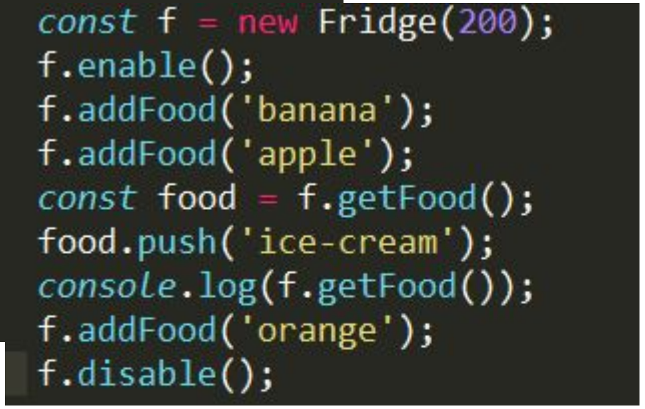
Possible solution:



```javascript
function Machine(power) {
  this._power = power;
  this._enabled = false;

  this.enable = function() {
    this._enabled = true;
  };
  this.disable = function() {
    this._enabled = false;
  };
}
```



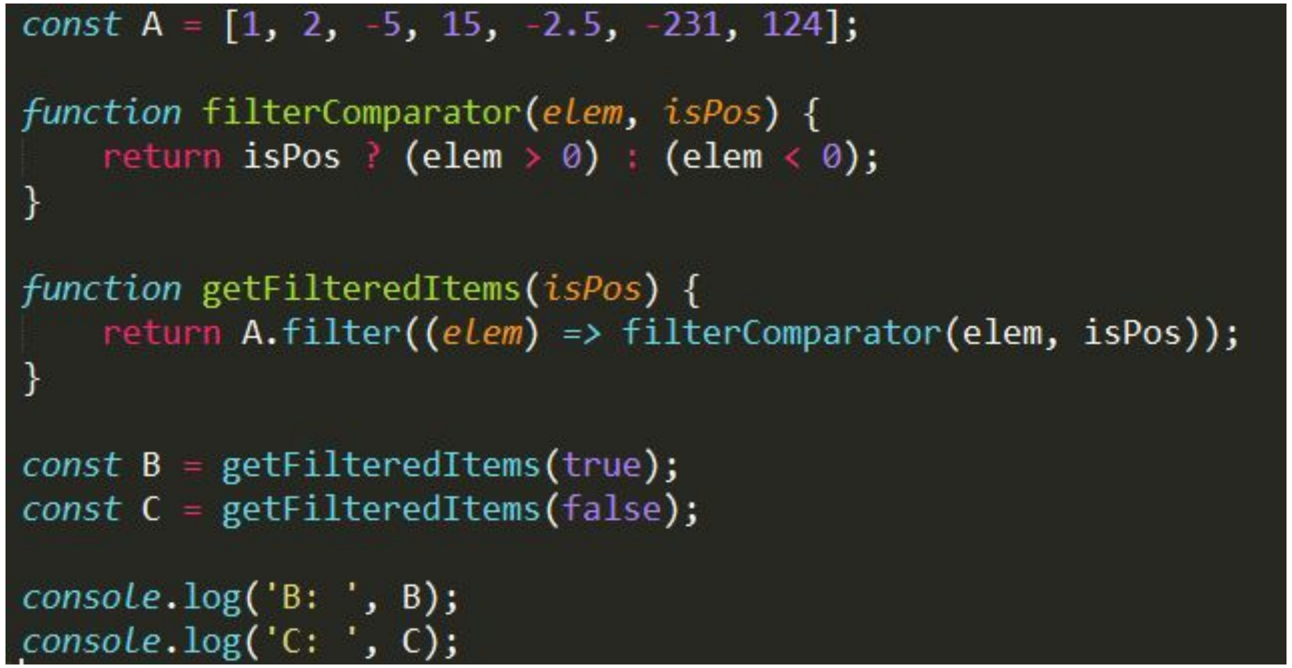
```javascript
function Fridge(power) {
    Machine.apply(this, arguments);

    const food = [];

    const maxLimit = Math.floor(power / 100);

    this.addFood = function() {
        if (!this._enabled) {
            console.log('Turn on the fridge!');
            return;
        }

        if (food.length + arguments.length > maxLimit) {
            console.log('Too many food items!');
            return;
        }

        food.push(...arguments);
    };

    this.getFood = function() {
        return food.slice();
    };
}
```



```javascript
const f = new Fridge(200);
f.enable();
f.addFood('banana');
f.addFood('apple');
const food = f.getFood();
food.push('ice-cream');
console.log(f.getFood());
f.addFood('orange');
f.disable();
```

Task 2.1

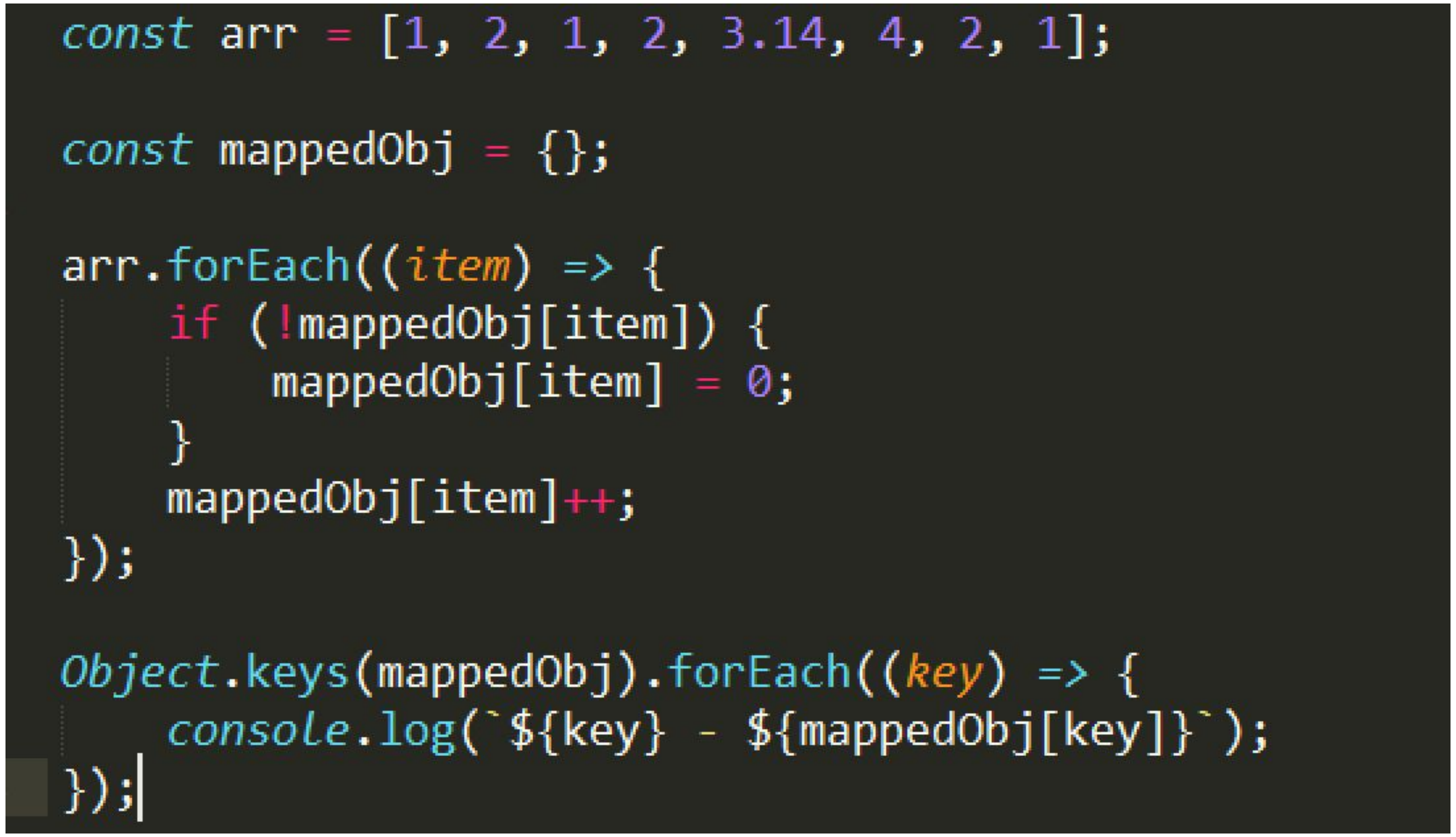
The numeric array A is given. You should move elements > 0 to the array B, elements < 0 to the array C. Log on console both arrays.



```
const A = [1, 2, -5, 15, -2.5, -231, 124];

function filterComparator(elem, isPos) {
    return isPos ? (elem > 0) : (elem < 0);
}

function getFilteredItems(isPos) {
    return A.filter((elem) => filterComparator(elem, isPos));
}

const B = getFilteredItems(true);
const C = getFilteredItems(false);

console.log('B: ', B);
console.log('C: ', C);
```

Task 2.2

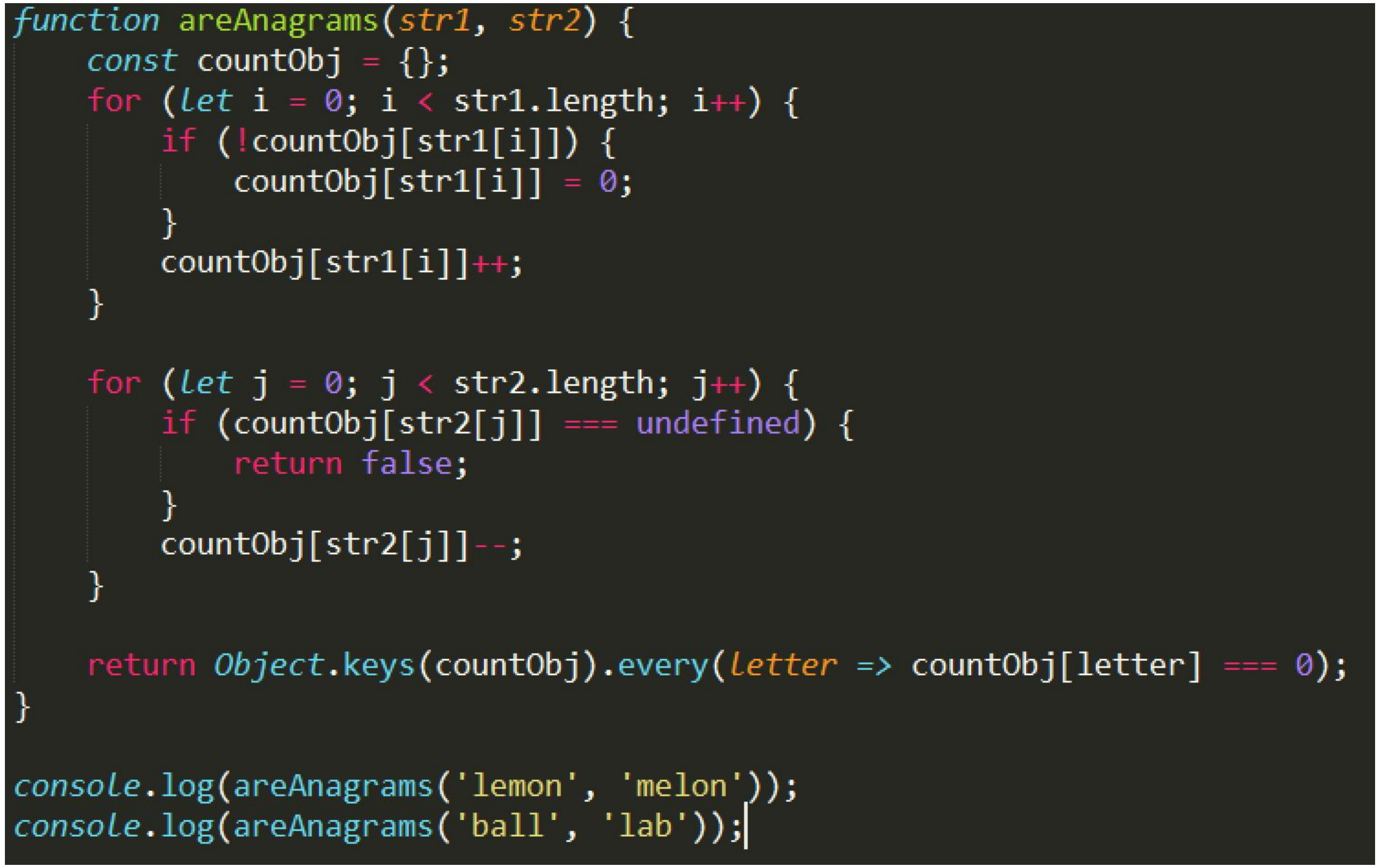
The numeric array is given. You should find amount of reps for each element in the array and log it on console. For example: for the array [1, 2, 1, 2, 3.14, 4, 2, 1] you should log:
1 - 3, 2 - 3, 3.14 - 1, 4 - 1 (not necessary in this order)



```
const arr = [1, 2, 1, 2, 3.14, 4, 2, 1];

const mappedObj = {};

arr.forEach((item) => {
    if (!mappedObj[item]) {
        mappedObj[item] = 0;
    }
    mappedObj[item]++;
});

Object.keys(mappedObj).forEach((key) => {
    console.log(`${key} - ${mappedObj[key]}`);
});
```

Task 2.4

Implement a function with two string arguments, which returns true if the given strings are anagrams, and false otherwise. Anagrams are the strings, which consist of the same symbols, but perhaps in a different order. For example ‘melon’ and ‘lemon’ are anagrams, but ‘ball’ and ‘lab’ - no, because they have different amounts of letter ‘l’.



```javascript
function areAnagrams(str1, str2) {
    const countObj = {};
    for (let i = 0; i < str1.length; i++) {
        if (!countObj[str1[i]]) {
            countObj[str1[i]] = 0;
        }
        countObj[str1[i]]++;
    }

    for (let j = 0; j < str2.length; j++) {
        if (countObj[str2[j]] === undefined) {
            return false;
        }
        countObj[str2[j]]--;
    }

    return Object.keys(countObj).every(letter => countObj[letter] === 0);
}

console.log(areAnagrams('lemon', 'melon'));
console.log(areAnagrams('ball', 'lab'));
```

HTML / CSS

HTML (tags)

Simple syntax:

```
<tag> content goes here ... </tag>
```

Nested elements:

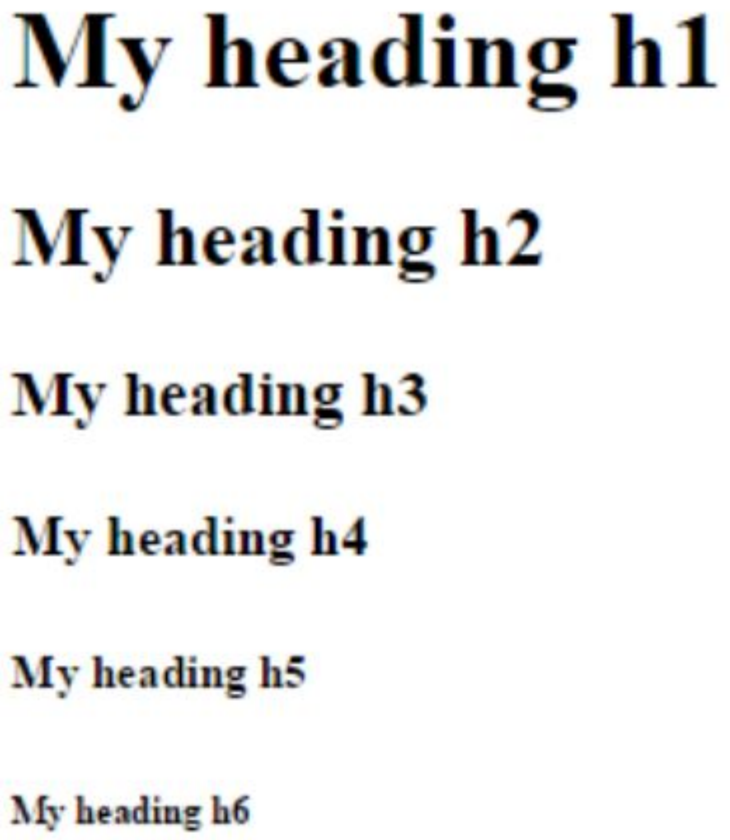
```
<tag>
    <tag2>
        <tag3>
            some content
        </tag3>
    </tag2>
</tag>
```

HTML (structure)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Title</title>
        ...
    </head>
    <body>
        <h1>heading</h1>
        ...
    </body>
    <script src="...”></script>
</html>
```

HTML (headings)

```
<h1>My heading h1</h1>

<h2>My heading h2</h2>

<h3>My heading h3</h3>

<h4>My heading h4</h4>

<h5>My heading h5</h5>

<h6>My heading h6</h6>
```



My heading h1

My heading h2

My heading h3

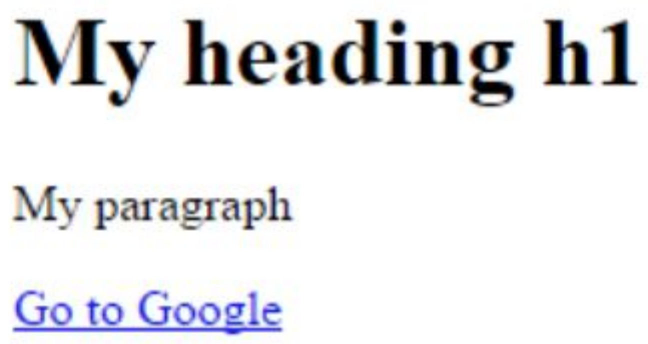
My heading h4

My heading h5

My heading h6

HTML (<p> and <a>)

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <h1>My heading h1</h1>
        <p> My paragraph </p>
        <a href="https://google.by"
           title="Go to Google">Go to Google</a>
    </body>
</html>
```



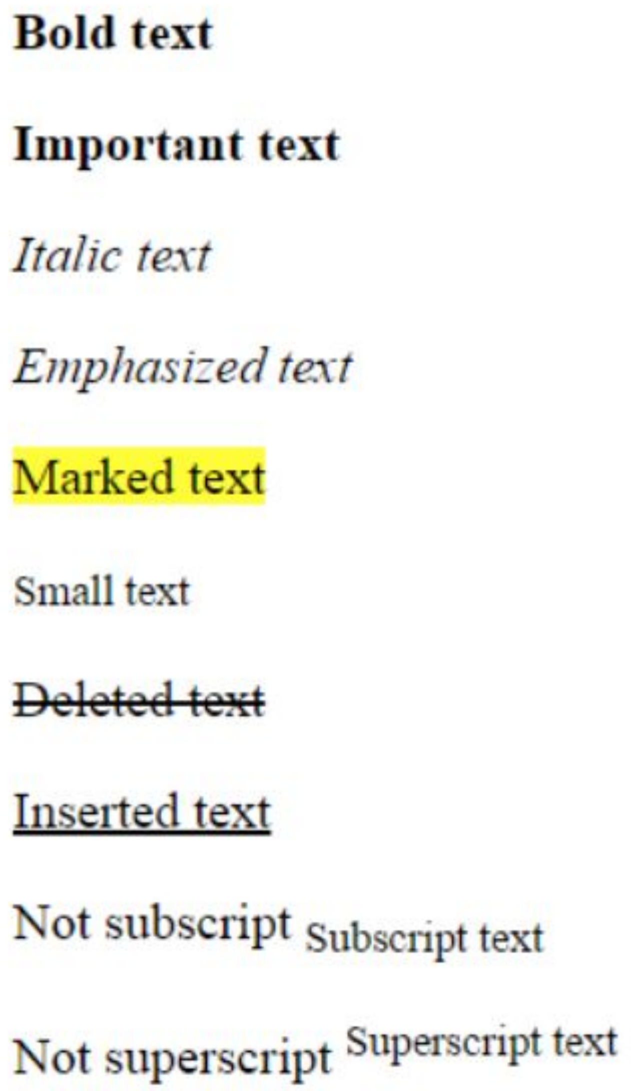
My heading h1

My paragraph

Go to Google

HTML (formatting)

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <b> Bold text </b>
        <strong> Important text </strong>
        <i> Italic text </i>
        <em> Emphasized text</em>
        <mark> Marked text</mark>
        <small> Small text</small>
        <del> Deleted text</del>
        <ins> Inserted text</ins>
        <sub> Subscript text</sub>
        <sup> Superscript text </sup>
    </body>
</html>
```



Bold text

Important text

Italic text

Emphasized text

Marked text

Small text

~~Deleted text~~

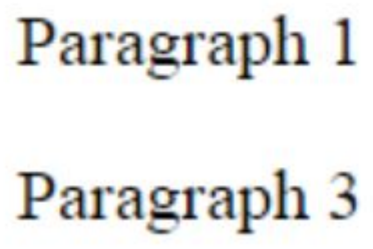
Inserted text

Not subscript $_{\text{Subscript text}}$

Not superscript $^{\text{Superscript text}}$

HTML (comments)

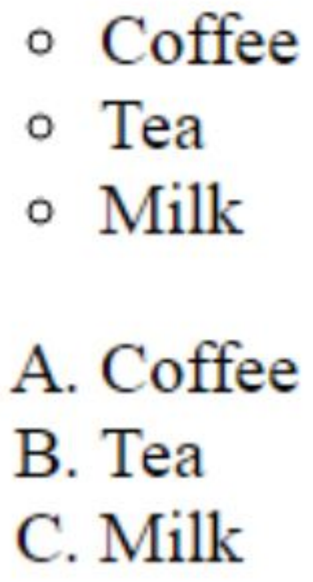
```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <p>Paragraph 1</p>
        <!-- <p>Paragraph 2</p> →
        <p>Paragraph 3</p>
    </body>
</html>
```


Paragraph 1

Paragraph 3

HTML (lists)

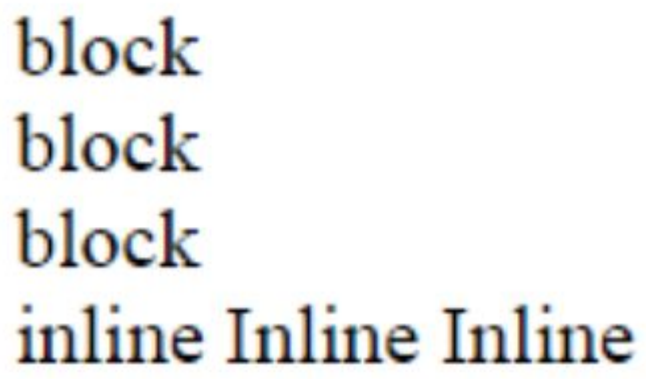
```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <ul style="list-style-type: circle">
            <li>Coffee</li>
            <li>Tea</li>
            <li>Milk</li>
        </ul>
        <ol type="A">
            <li>Coffee</li>
            <li>Tea</li>
            <li>Milk</li>
        </ol>
    </body>
</html>
```



```
○ Coffee
○ Tea
○ Milk

A. Coffee
B. Tea
C. Milk
```

HTML (blocks)

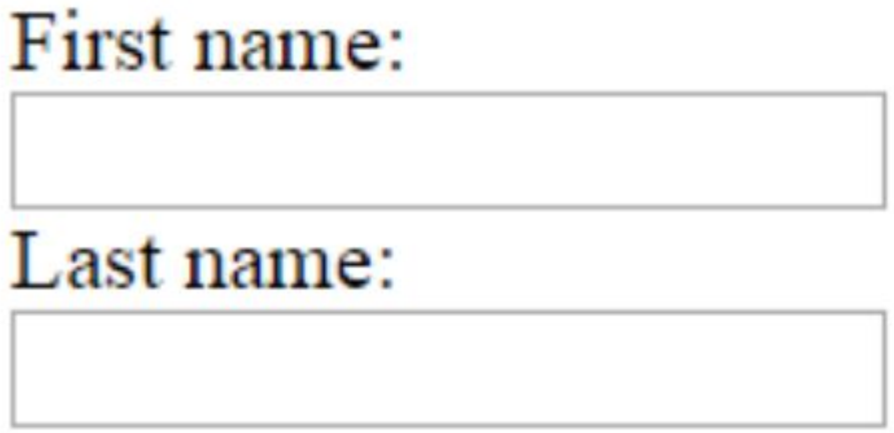
```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <div>block</div>
        <div>block</div>
        <div>block</div>
        <span>inline</span>
        <span>Inline</span>
        <span>Inline</spa>
    </body>
</html>
```



block
block
block
inline Inline Inline

HTML (forms)

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <form>
            First name:
            <br>
            <input type="text" name=
            "firstname">
            <br>
            Last name:
            <br>
            <input type="text" name=
            "lastname">
        </form>
    </body>
</html>
```



Useful links

Html: https://www.w3schools.com/html/default.asp

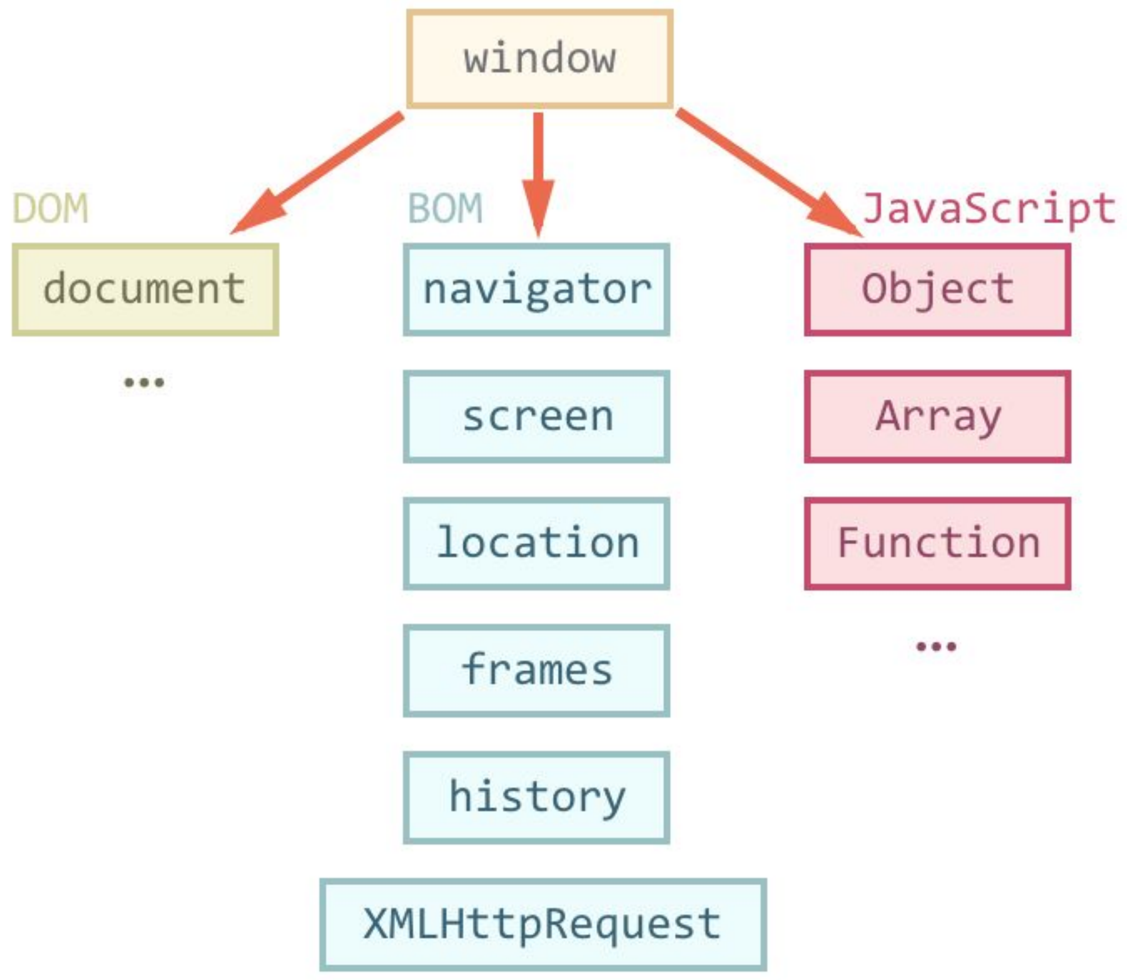
Css: https://www.w3schools.com/css/default.asp

DOM, BOM and JS

Window

By default, the JavaScript language does not have methods for work with the browser.
It does not know anything about HTML at all.
But it is easy to expand this language with new functions and objects.

The picture schematically shows the structure of the set of browser objects.



DOM (document object model)

The global object **document** allows you to interact with the contents of the page.

Example of the usage:

```
document.body.style.background = 'red';
document.body.style.background = '';
```

The **document** and lots of its methods and properties are described in the standard W3C DOM.

BOM (browser object model)

BOMs are objects for working with anything but the document.

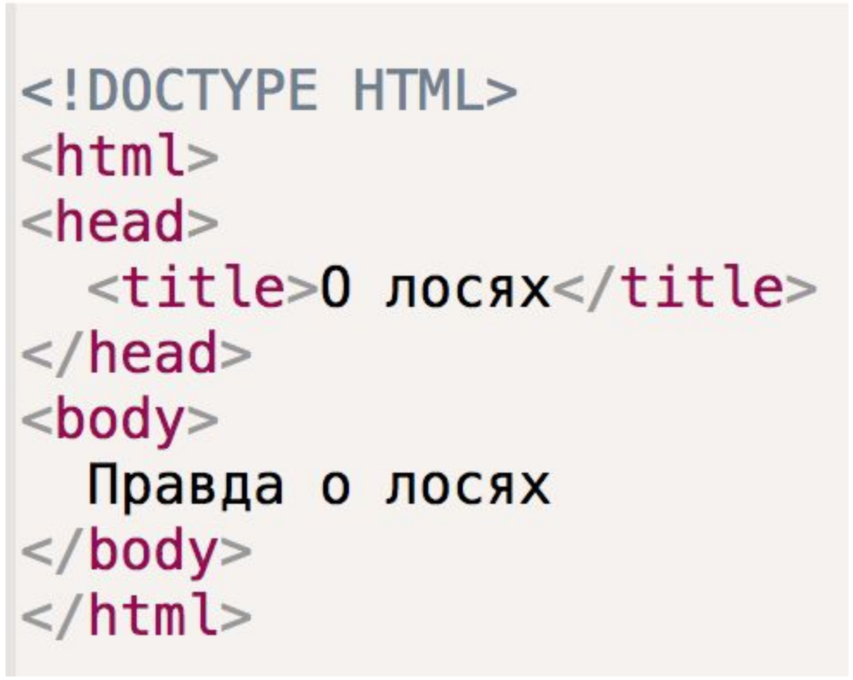
For example:

- The **navigator** object contains general information about the browser and the operating system. Especially two properties are useful: **navigator.userAgent** - contains information about the browser, **navigator.platform** - contains information about the platform, allows you to differ Windows / Linux / Mac, etc.
- The **location** object contains information about the current page URL and allows you to redirect the visitor to a new URL.
- The **alert / confirm / prompt** functions are also included to the BOM.

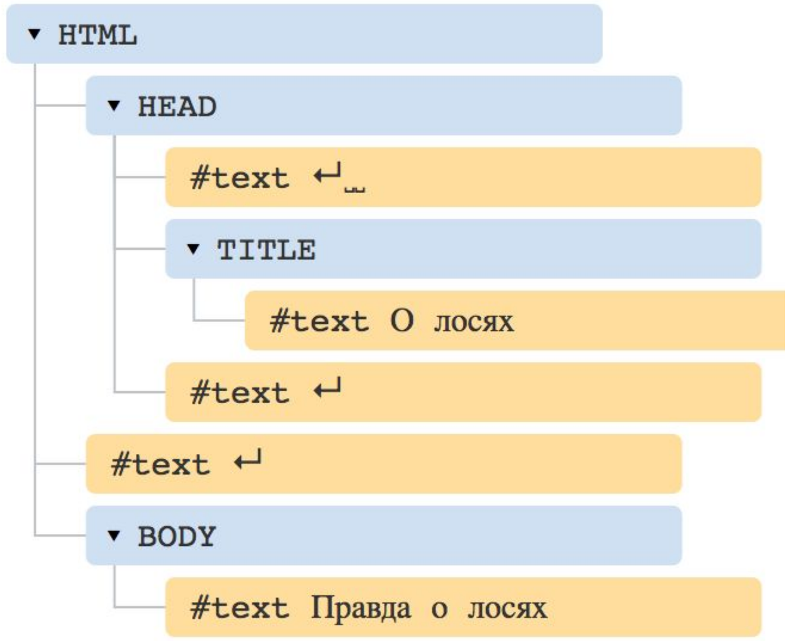
DOM tree

DOM (document object model)

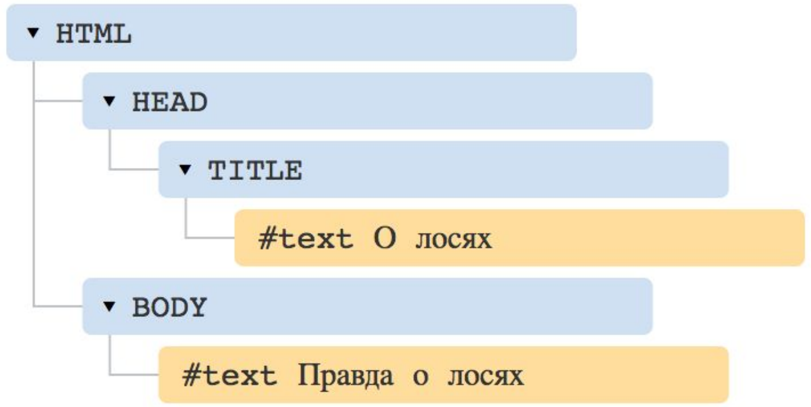
According to the DOM-model, the document is a hierarchy, a tree.
Each HTML tag forms a tree node with an "element" type.
Nested tags become child nodes. To represent the text, nodes with the type "text" are created.
The **DOM** is a document view in the form of an object tree, which is available for modification through JavaScript.



```
<!DOCTYPE HTML>
<html>
<head>
  <title>О лосях</title>
</head>
<body>
  Правда о лосях
</body>
</html>
```



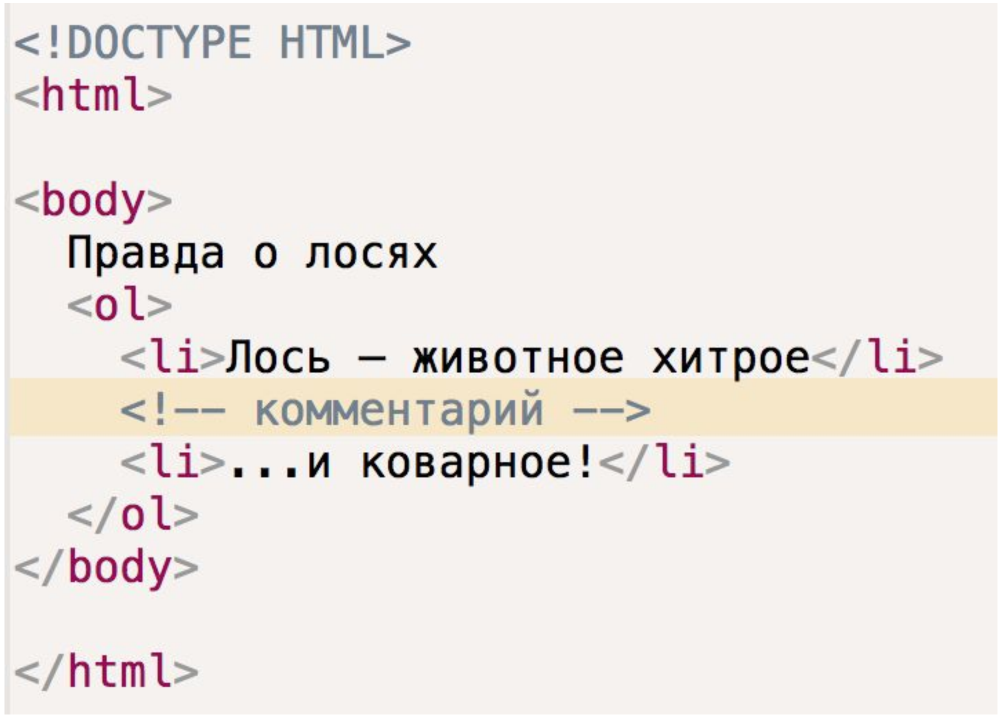
```
▾ HTML
  ▾ HEAD
    #text ↵␣
    ▾ TITLE
      #text О лосях
    #text ↵
  #text ↵
  ▾ BODY
    #text Правда о лосях
```



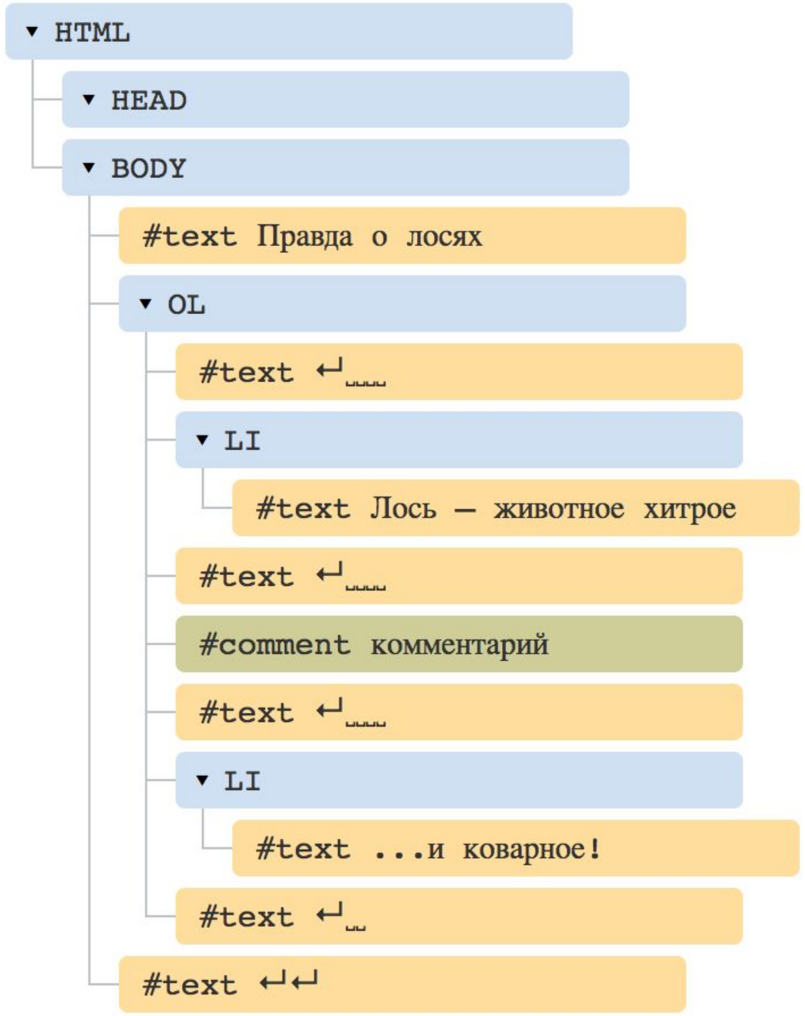
```
▾ HTML
  ▾ HEAD
    ▾ TITLE
      #text О лосях
  ▾ BODY
    #text Правда о лосях
```

DOM (elements)

Everything from HTML is presented in DOM.
Even the **document** object itself, formally, is the DOM node, but the very root one. There are 12 types of elements, but we work with 4 of them: document, simple nodes, text nodes, comments.



```
<!DOCTYPE HTML>
<html>

<body>
  Правда о лосях
  <ol>
    <li>Лось — животное хитрое</li>
    <!-- комментарий -->
    <li>...и коварное!</li>
  </ol>
</body>

</html>
```



DOM (Developer Tools)

You can investigate and modify the DOM using developer tools built into the browser.
We will review such tools built in Google Chrome.

- **Elements** tab - helps to review all elements and interesting information about them (styles, computed styles, metrics ...). Spacing symbols are not shown in Elements view.

- Access to **previously chosen** elements in **console** (using $0, $1, ...).
  `$0.style.backgroundColor = 'red'`

- Access to **specific** elements in **console**
  `$$("div.my")`- all elements, which satisfy the CSS-selector
  `$("div.my")` - first element, which satisfies the CSS-selector

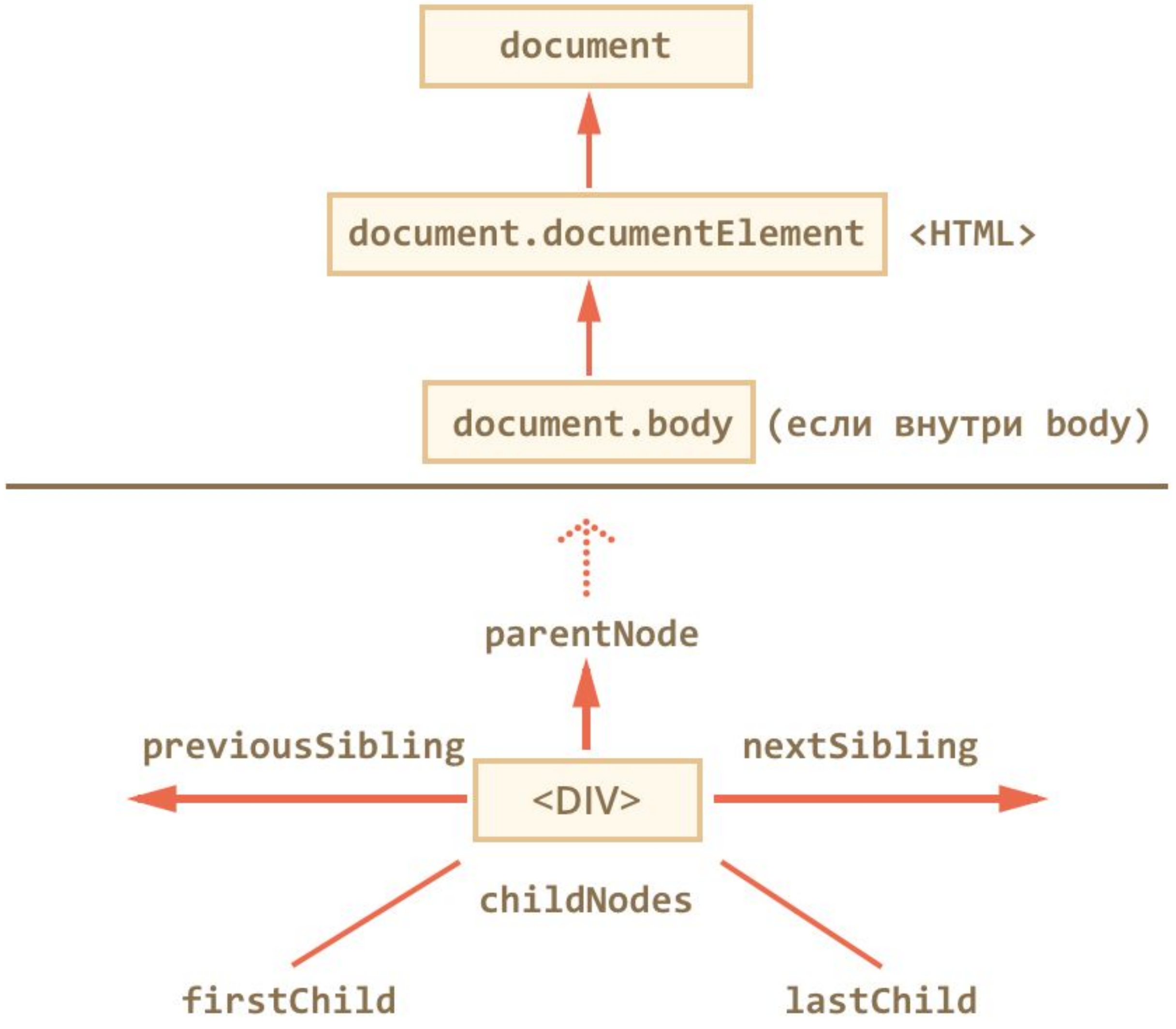
More you can find there.

DOM (navigation)

Top elements can be accessed directly from document (`document.doctype`).

- <**HTML**>: `document.documentElement`
- <**BODY**>: `document.body`
- <**HEAD**>: `document.head`
- **child nodes**: `childNodes` (pseudo-array with child nodes, only for reading!), `firstChild`, `lastChild` (fact access to first and last child)
- **neighbours**: `previousSibling`, `nextSibling`
- **parent**: `parentNode`

!!! All arrays are pseudo (no inner methods).
!!! If want to access only elements (without text nodes and other - use **children, firstElementChild, lastElementChild, previousElementSibling, nextElementSibling, parentElement).**



DOM (query selectors)

- `document.getElementById('identificator')` (only for document)

```
<div id="content">Nice element</div>
<script>
  var elem = document.getElementById('content');
  elem.style.background = 'red';
</script>
```

- `elem.getElementsByTagName('tagName')`

```
var elements = document.getElementsByTagName('div');
elem.getElementsByTagName('*');
```
(get all children, grandchildren, ...)

- `elem.getElementsByName('someName')`

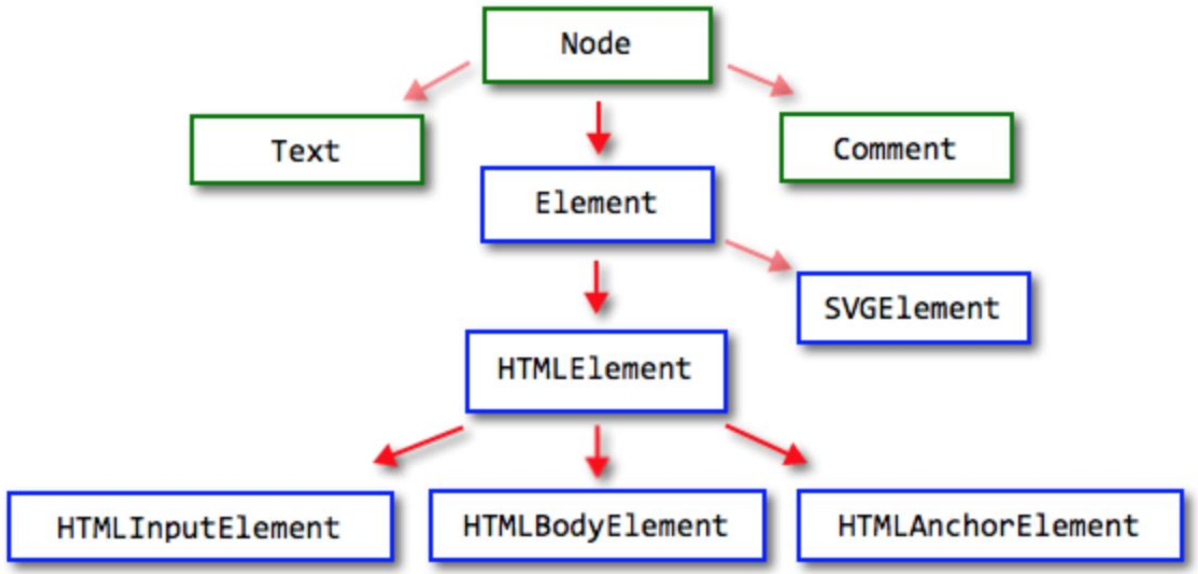
  `elem.getElementsByClassName('className')`

- `elem.querySelectorAll('query')` (all elements, which satisfy the css selector)
- `elem.querySelector('query')` (first element, which satisfies the css selector)
- `elem.closest('query')` (the closest element, including current, in the tree above)

DOM (types)

To check node’s class, convert it to string:

```
alert( document.body ); // [object
HTMLBodyElement]

element.nodeType (number)
(ELEMENT_NODE = 1, ATTRIBUTE_NODE = 2,
TEXT_NODE = 3, CDATA_SECTION_NODE = 4,
ENTITY_REFERENCE_NODE = 5, ENTITY_NODE = 6,
PROCESSING_INSTRUCTION_NODE = 7, COMMENT_NODE
= 8, DOCUMENT_NODE = 9, DOCUMENT_TYPE_NODE =
10, DOCUMENT_FRAGMENT_NODE = 11,
NOTATION_NODE = 12)
```



DOM (types)

- `elem.tagName, elem.nodeName`
  (node name in uppercase, tagName is applicable only for element type, nodeName for all nodes)

- `elem.innerHtml (read and write)`

```
<body>
  <p>Text</p>
  <div>Div</div>
  <script>
    alert( document.body.innerHTML ); // read the content
    document.body.innerHTML = 'New BODY!'; // change the content
  </script>
</body>
```

- `elem.outerHtml (read)`

```
<div>Hello, World!</div>
<script>
    alert( div.outerHTML ); // <div>Hello, World!</div>
</script>
```

DOM (types)

- `elem.data`

  `elem.innerHtml` is applicable only for elements, elem.data for other nodes:

```
<body>
  Hello
  <!-- Comment -->
  <script>
    for (var i = 0; i < document.body.childNodes.length; i++) {
      alert( document.body.childNodes[i].data );
    }
  </script>
  Yo
</body>
```

  Hello – the content of the first node (text).
  Comment – the content of the second node (comment).
  Spaces – the content of space node between comment and script.
  undefined – the content of script node, it doesn’t have data.

DOM (attributes)

`elem.hasAttribute(name)` – checks, if attribute exists
`elem.getAttribute(name)` – gets the attribute value
`elem.setAttribute(name, value)` – sets the attribute
`elem.removeAttribute(name)` – deletes the attribute
`elem.attributes` – gets all element attributes

```
<body>
  <div id="elem" about="Elephant"></div>
  <script>
    alert( elem.getAttribute('About') ); // 'Elephant'
    elem.setAttribute('Test', 123); // attr Test is set
    var attrs = elem.attributes; // get attributes collection
    for (var i = 0; i < attrs.length; i++) {
      alert( attrs[i].name + " = " + attrs[i].value );
    }
  </script>
</body>
```

DOM (elements manipulations)

Creating elements:

- **`document.createElement(tag)`**: `var div = document.createElement('div');`
- **`document.createTextNode(text)`**: `var textElem = document.createTextNode('Hello');`

```
var div = document.createElement('div');
div.className = "alert alert-success";
div.innerHTML = "<strong>Yahoo!</strong>";
```

Adding elements:

- **`elem.appendChild(childElem)`** (adds childElem to the end of children list)
- **`elem.insertBefore(childElem, nextSibling)`** (adds childElem before nextSibling to the children list)

```
list.insertBefore(newLi, list.firstChild);
```

DOM (elements manipulations)

Cloning elements:

- `elem.cloneNode(deep)` – creates copy of the element. if deep == true, then with all child nodes, if no - without.

Removing elements:

- `elem.removeChild(childElem)` - deletes childElem from the children list of elem
- `elem.replaceChild(newElem, childElem)` - replaces childElem with newElem in the children list of elem

All methods, mentioned above, return an element.

Live coding

Homework

ToDo Paper (deadline 09.05 19:59)

Add new file “dom.js” to the same directory (with index.html, script.js and functions.js) and add it to index.html. Implement in dom.js file functions, which work with the dom. Call these functions in an appropriate function from functions.js file. Add <div id=”todo-items”></div> to index.html (store your todoItems there).

- `addTodoItemDom(todoItem: object)`
  Add todoItem to the DOM. It should include the text and id as text fields and a checkbox (for the completed field visualization).
- `viewTodoListDom(itemsType: string)`
  Function takes itemsType argument (‘completed’, ‘not_completed’, ‘all’). It should delete all current items from the DOM and render items of an appropriate itemsType.
- `editTodoItemDom(todoItemId: number, newText: string)`
  It should replace the text of the item with the given todoItemId.
- `deleteTodoItemDom(todoItemId: number)`
  It should delete todoItem with the given todoItemId

Please, don’t forget to review all places, where `viewTodoListDom` should be called.

Thanks for your attention!

Questions?