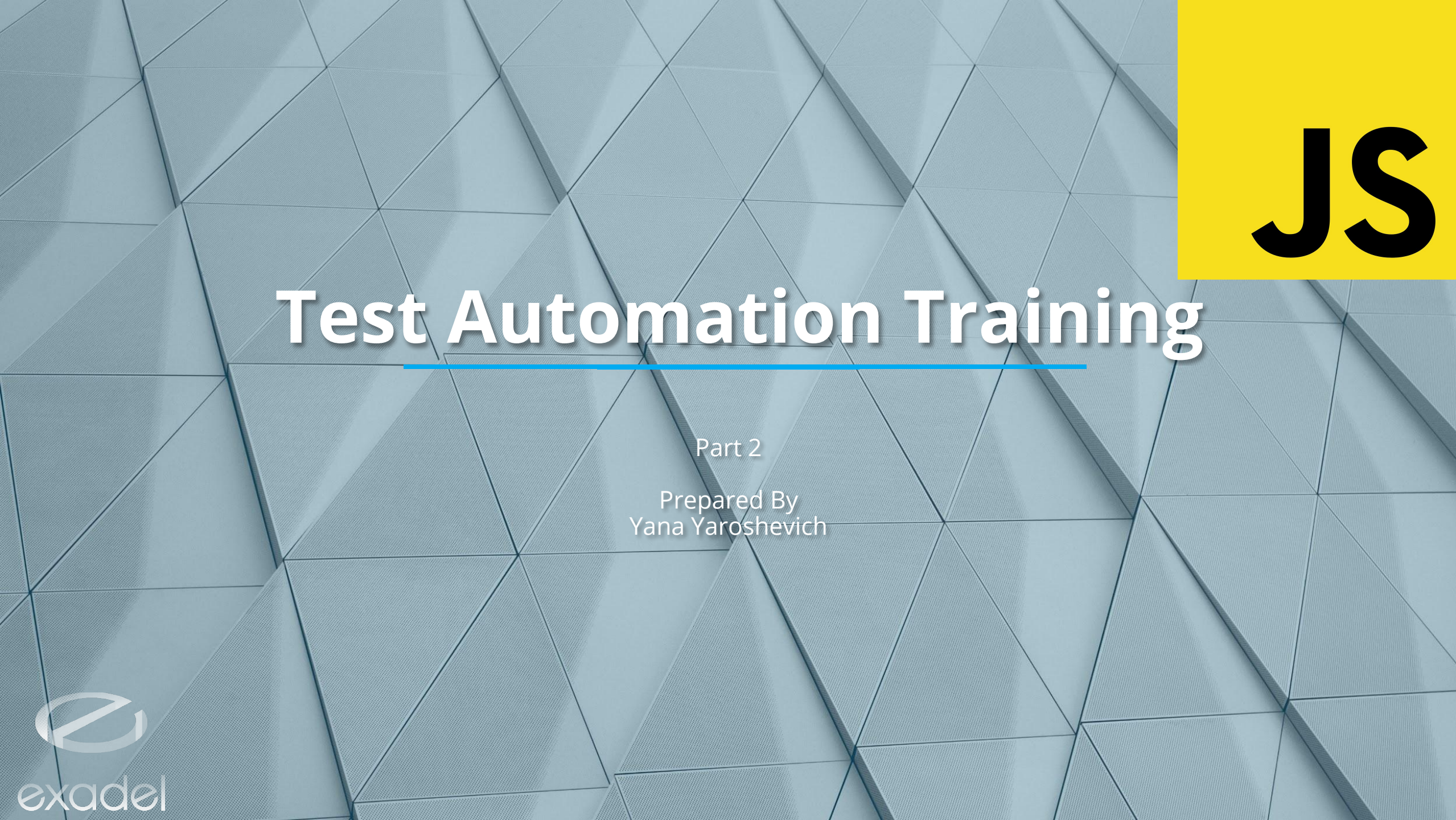


Test Automation Training

Part 2

Prepared By
Yana Yaroshevich

exadel

Homework check Results

Task 1

Find roots of quadratic equation with the given coefficients *a, b, c*. Log on console roots or some message, if there are no roots.

```
// D > 0

/*var a = 2;
var b = 3;
var c = 1;*/

// D = 0
/* var a = 4; var b = 4; var c = 1; */

// D < 0
var a = 3; var b = 3; var c = 3;

var D = b * b - 4 * a * c;
console.log(D);

if (D > 0) {
    var x1 = (-b + Math.sqrt(D)) / (2 * a);
    var x2 = (-b - Math.sqrt(D)) / (2 * a);
    console.log('First root:', x1, 'Second root:', x2);
} else if (D === 0) {
    var x = -b / (2 * a);
    console.log('Root', x);
} else if (D < 0) {
    console.log('No roots!');
}

```

Task 2

Check whether a given number is prime.
Log on console true or false.

```
1  // var n = 7;
2
3  // var n = 29;
4
5  var n = 42;
6
7  var isPrime = true;
8
9  var nSqrt = Math.sqrt(n);
10 var nSqrtInt = Math.floor(nSqrt);
11
12 if (n === 1) {
13     isPrime = false;
14 } else {
15     for (var i = 2; i <= nSqrtInt; i++) {
16         if (n % i === 0) {
17             isPrime = false;
18             break;
19         }
20     }
21 }
22
23 console.log(isPrime);
24
```

Task 3

Find the following sum: S = 1 + 1/2 + 1/3 + 1/4 + ... + 1/n for a given *n*. Log the result on console.

```
1  var n = 5;
2
3  // var n = 100;
4
5  // var n = 200;
6
7  var sum = 1;
8
9  for (var i = 2; i <= n; i++) {
10     sum += 1 / i;
11 }
12
13 console.log(sum);
```

Task 4

Reverse a given number *n* (123 -> 321).
Log the result on console.

```
1  var n = 4858;
2
3
4  var reversedN = 0;
5
6  while (n) {
7      reversedN *= 10;
8      reversedN += n % 10;
9      n /= 10;
10     n = Math.floor(n);
11 }
12
13
14 console.log(reversedN);
```

Objects and functions

Objects

Object literal is a very convenient way to create a new object:

```
var emptyObject = {};
var person = {
    firstName: 'John',
    lastName: 'Snow'
};
```

An object field name may be an any string. If this string is not a valid literal for a variable name in JavaScript, then the field name should be quoted:

```
var obj = {
    'first-name': 'John'
};
```

So, quotes are necessary for the field name ‘first-name’, but not necessary for the ‘firstName’ or ‘first_name’.

Objects, nesting

Objects can be nested, for example:

```
var flight = {
    airline: "Oceanic",
    number: 815,
    departure: {
        IATA: "SYD",
        time: "2004-09-22 14:55",
        city: "Sydney"
    }
};
```

Objects, getting values

To get a field value you may use “**[]**” .
If a field name is known and it is a valid literal for a variable name in JavaScript, then you may use “**.**” for getting a value.

```
person['firstName'] // 'John'
person.firstName // 'John'

obj.first-name // Impossible
obj['first-name'] // 'John'

flight.airline.number // 815
flight['airline']['number'] // 815

// If a field name is dynamic (stored in variable):
var key = 'firstName';
person[key] // 'John'
person.key // looks for the field with a name ‘key’
```

Objects, changing values

The value of the field can be changed by an **assignment**.
If the object does not have a field with this name, it will be added to the object, if there is, then the field value will be changed.

```
var person = {
    firstName: 'John'
};

person.age = 18;

person.education = {
    school: 'Some School'
};
```

To delete a field use **delete** operator:

```
delete person.firstName;
```

Functions

In JavaScript function is an **object.**
Functions may be **stored** in variables, objects and arrays.
Functions may be passed **as arguments to other functions.**
Functions may **return** other functions.

```
function showMessage() {
  alert( 'Hello everybody!' );
}
```

Function **without return** returns **undefined** .
If **return is without value,** then function also returns **undefined.**

```
showMessage(); // undefined
```

Functions, arguments

If arguments are not passed, then they are equal to **undefined** .
The quantity of arguments may be **any**.

Missing arguments will be equal to **undefined** :

```
function showMessage(from, text) {
  text = text || 'text is not passed';

  ...
}

showMessage('Yana');
```

Objects are passed to functions as links, other values are passed as copies.

Functions, expression & declaration

```
var sum = function(a, b) {
    var result = a + b;
    return result;
}
sum(1, 2) // 3
```

Such syntax, when a function is created in some context (in this case in assignment), is called a **Function Expression**.
Syntax, when functions are declared in a usual way, is called a **Function Declaration.**

Functions, created via Function Declaration, are created by interpreter when entering the scope (at the beginning of a script execution), so they can be executed before declaration. That is the main difference with a Function Expression creation.

Usually it is convenient, but it could be a problem, when you need to create function dependent on some condition. In this case and in many other cases Function Expression is used.

Functions, anonymous functions

```
function ask(isConfirm, yes, no) {
    if (isConfirm) {
        yes();
    }
    else {
        no();
    }
}
function showOk() {
  alert( "You confirmed." );
}
function showCancel() {
  alert( "You canceled." );
}

ask(true, showOk, showCancel);
```

Functions, scope

When accessing an undeclared variable, the function will look for an external variable with that name.

```
var userName = 'Vasya';

function showMessage() {
  var message = 'Hello, I am ' + userName;
  console.log(message);
}

showMessage(); // Hello, I am Vasya
```

Arrays

Arrays

In JavaScript **arrays** are actually **special objects**, which are similar to arrays in other C-like programming languages. Getting and setting values in array happens like in object, but the difference is that the field name is a number.

Arrays have their literal **[]** and a lot of useful built-in methods.

```
var arr = [];
var fruits = ["Apple", "Orange", "Banana"];

alert( fruits[0] ); // Apple
alert( fruits.length ); // 3
```

In array we can store elements of any type:

```
var arr = [ 1, 'Имя', { name: 'Петя' }, true ];
```

Arrays, main methods

- Add to the end: **push**

```
var fruits = ['Apple', 'Orange']; fruits.push('Pineapple');
console.log(fruits); // Apple, Orange, Pineapple
```

- Delete last element: **pop**

```
var fruits = ['Apple', 'Orange', 'Pineapple']; fruits.pop(); // deleted 'Pineapple'
console.log(fruits); // Apple, Orange
```

- Add to the beginning: **unshift**

```
var fruits = ['Orange', 'Pineapple']; fruits.unshift('Apple');
console.log(fruits); // Apple, Orange, Pineapple
```

- Delete from the beginning: **shift**

```
var fruits = ['Apple', 'Orange', 'Pineapple']; fruits.shift();
console.log(fruits); // Orange, Pineapple
```

Arrays, splice

Method **splice** is a universal folding knife for a work with arrays.
It can do **almost everything:** delete elements, insert elements, replace elements.

Its syntax: **arr.splice(index[, deleteCount, elem1, ..., elemN])**

Delete deleteCount elements, starting from number index, and then insert elem1, ..., elemN on their position.
Returns an array of deleted elements.

```
var arr = ["I", "am", "learning", "JavaScript"];

// delete 2 first elements and add other elements on their place
arr.splice(0, 2, "We", "are");

alert( arr ); // теперь ["We", "are", "learning" ,"JavaScript"]
```

Arrays, slice

Method **slice(begin, end)** copies part of an array from begin to end, not including end.
The original array is not changed.

```
var arr = ["Why", "do", "we", "learn",  "JavaScript"];
var arr2 = arr.slice(1, 3); // elements 1, 2 (not including 3)
alert( arr2 ); // "do", "we"
```

Without end parameter an array is copied till its end:

```
var arr = ["Why", "do", "we", "learn",  "JavaScript"];
alert( arr.slice(1) ); // take all elements, starting from 1 ("do", "we", "learn",  "JavaScript")
```

We may use negative indexes, then we will start from the end of array:

```
var arr2 = arr.slice(-2); // copy from 2-nd element from the end and farther ("learn", "JavaScript")
```

Without parameters full array is copied:

```
var fullCopy = arr.slice(); // "Why", "do", "we", "learn",  "JavaScript"
```

Arrays, split, join

There is a **split(s)** method, which transforms string to array, splitting it by a delimiter s.

```
var names = 'Masha, Petya, Marina, Vasiliy';
var arr = names.split(', ');
for (var i = 0; i < arr.length; i++) {
        alert( 'Message for ' + arr[i] );
}
```

Method split has **an optional second argument – limit on the number of elements in the array**.

```
alert( "a,b,c,d".split(',', 2) ); // a,b
```

Calling split with an empty string as an argument will split a string on letters:

```
var str = "test";
alert( str.split('') ); // t,e,s,t
```

Opposite method - method **join(s).** It makes a string from the array.

Arrays, sort

Method **sort()** sorts an array. For example:

```
var arr = [ 1, 2, 15 ];
arr.sort();
alert( arr );  // 1, 15, 2
```

It happened because by default method does conversion of elements to strings.
If you want to define your sort order, then pass your **comparator:** function, which can compare two elements.

It **should return**:

- Positive value, if a > b,
- Negative value, if a < b,
- If values are equal, it may return 0, but actually it is not important, what to return, if their mutual order doesn't matter.

```
function compareNumeric(a, b) {
  return a - b;
}
```

Arrays, loops

- forEach

```
var arr = ["Apple", "Orange", "Pineapple"];

arr.forEach(function(item, i, arr) {

    console.log( i + ": " + item + " (array:" + arr + ")" );

});

// 0: Apple (array:Apple,Orange,Pineapple)

// 1: Orange (array:Apple,Orange,Pineapple)

// 2: Pineapple (array:Apple,Orange,Pineapple)
```

Method forEach doesn't return anything, it is used only as a loop, it is more **«elegant»** option, than a usual **for** loop.

Arrays, filter

Method **filter** is used for filtering an array using callback function.
It creates a new array, with the elements for which function returned true.

```
var arr = [1, -1, 2, -2, 3];

var positiveArr = arr.filter(function(number) {
  return number > 0;
});

alert( positiveArr ); // 1,2,3
```

Arrays, map

Method **arr.map(callback)** is used for array **transformations**.

It creates a **new array**, which consists of **the results of callback(item, i, arr) execution** for all elements of the arr.

```
var names = ['HTML', 'CSS', 'JavaScript'];

var nameLengths = names.map(function(name) {
  return name.length;
});

// got the array with lengths
alert( nameLengths ); // 4,3,10
```

Arrays, other methods

- **some** - check if any element in the array satisfies the given condition
- **every** - check if every element in the array satisfies the given condition
- **reduce / reduceRight** - is used for each element processing with saving the result on each step.

And other....

Dates

Dates

For work with date and time in JavaScript we use **Date** object.

For creating Date object instance we may use:

```
var now = new Date();

// now - object Date with current date and time

var createdAt = new Date('2012-04-04T24:00:00');
```

For sorting array of objects by date property:

```
function compareDates(a, b) {
    return a.date – b.date;
}
```

The fact is that when date objects are used in a numerical context, they are converted to milliseconds.

Live coding

exadel

Homework

exadel

Homework (deadline - 27.04.2018 - 19.59)

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.
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)
3. Algorithms for tasks 1 and 2 implement as functions with one argument. Run both function with the same array in one script.js file.
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’.

All given tasks should be pushed to your remote repository before deadline!
Please follow the naming policy:

- One task - one folder (name folders according to a task number in the list and lesson number, e.g. l2t1)
- Each folder contains one index.html file and one script.js file

Good luck and God help us all! :)

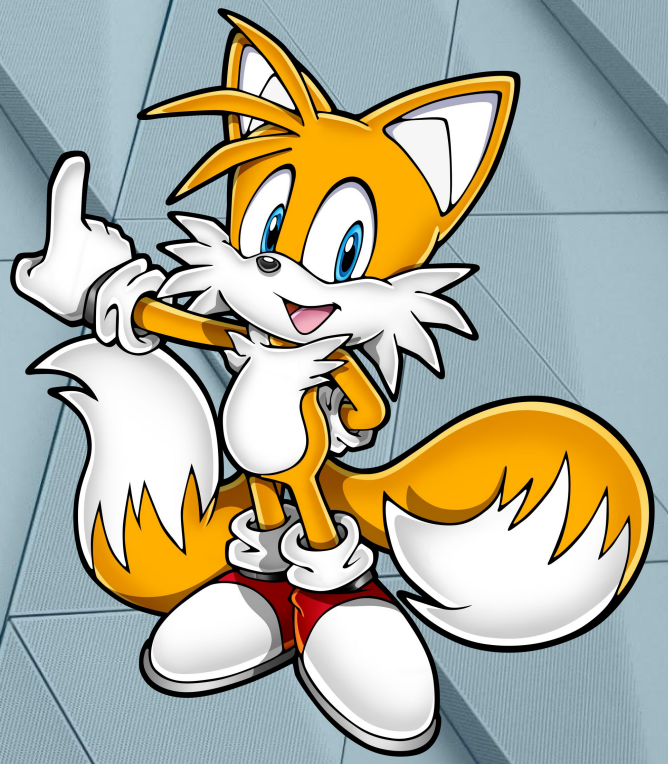
Project notes (ToDo Paper)

Add new file “functions.js” to the same directory (with index.html and script.js) and add it to index.html before script.js. In script.js create an empty array for ToDo items (`var todoItems = [];`)
Implement in functions.js file several functions, which work with the created array (via function declaration):

- `addTodoItem(todoItem: object)`
  Check if todoItem is valid and if yes, add it to todoItems array. Function should return the result of validity check (text field is not empty, all fields are present and id is unique).
  `todoItem : { text: string, completed: boolean, id: number }.`
- `viewTodoList(itemsType: string)`
  Function takes itemsType argument (‘completed’, ‘not_completed’, ‘all’) and returns all items of this type.
- `editTodoItem(todoItemId: number, newText: string)`
  If newText is not empty, function changes text of todoItem by todoItemId on the new text. It should return flag, whether edit was successful.
- `deleteTodoItem(todoItemId: number)`
  Delete todoItem by todoItemId, return flag, whether delete was successful.
- `completeTodoItem(todoItemId: number)`
  Change completed field of todoItem (get it by todoItemId) on true.

Thanks for your attention!

Questions?



exadel