INTRODUCTION TO JAVASCRIPT LANGUAGE

Beograd, 17.10.2019.

AGENDA SLIDE

01 Values, Types and Operators

02 Program Structure

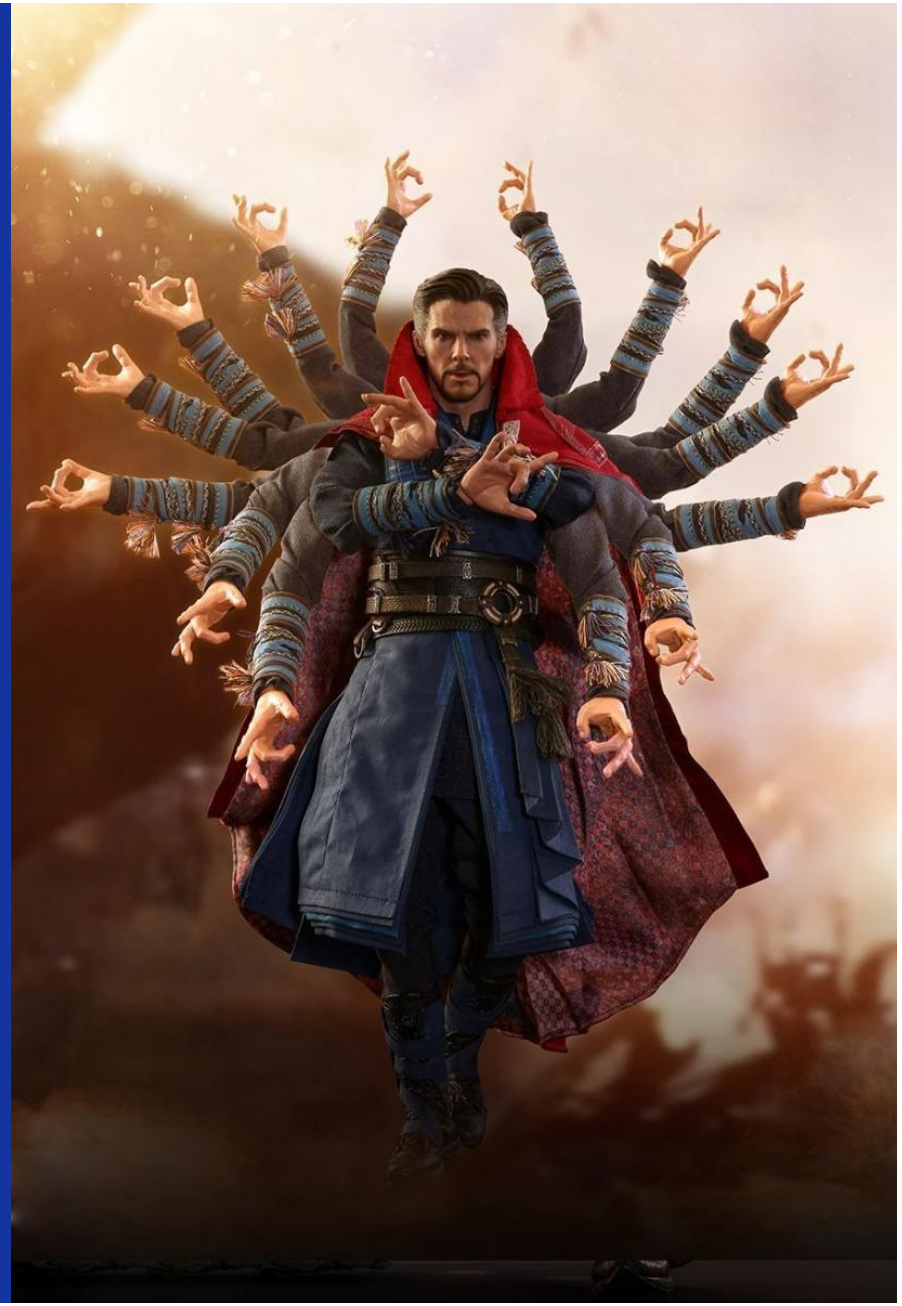
03 Functions

04 Project requirements

01

VALUES, TYPES AND OPERATORS

The very basics of JavaScript.



IS JAVASCRIPT TYPED LANGUAGE?

- JavaScript is **loosely** typed programming language.
- In JavaScript, variables don't have types - **values have types**.
- Variables can hold any value, at any time.

VARIABLES IN JAVASCRIPT

- In JavaScript, variables are declared by using **var** keyword.
- JavaScript is **case-sensitive** language.
- The general rules for constructing names for variables:
  - Names can contain letters, digits, underscores, and dollar signs.
  - Names must begin with a letter.
  - Names can also begin with $ and _.
  - Names are case sensitive (y and Y are different variables).
  - Reserved words (like JavaScript keywords) cannot be used as names.

USING VARIABLES

```
// Declaring a variable
var name;

// Assigning a value to the variable
name = 'Will';

// Or, we can do both at the same time
var FACTOR_007 = 18;
var activeUser = true;
var price$ = null;
var _person = { age: 19 };
```

BUILT-IN TYPES IN JAVASCRIPT

- There are **seven** built-in types in JavaScript:
  - null
  - undefined
  - boolean
  - number
  - string
  - object
  - symbol

PRIMITIVE VS NON-PRIMITIVE TYPES

PRIMITIVE TYPES	NON-PRIMITIVE TYPES
null undefined boolean number string symbol	objects

BY VALUE

BY REFERENCE

NON-VALUE TYPES

- **null** and **undefined** types represent absence of value.
- **null** type represents an **empty** value.
- **undefined** type represents a **missing** value.

LOGICAL TYPE

- **boolean** type represents logical values.
- Possible values are **true** and **false**
- There's an object wrapper around this value type called **Boolean**.

LOGICAL TYPE

```
var a = true;
var b = false;

var c = Boolean('something truthy');
console.log(c);     // true

var d = Boolean('');
console.log(d);     // false
```

NUMERICAL TYPE

- **number** type represents numberical values.
- There's an object wrapper around this value type called **Number.**

NUMERICAL TYPE

```
var a = 42;
var b = 0.42;
var c = .42;
var d = -42.67
var e = 10e5;
var f = NaN;        // For example, 10/'a' produces NaN
var g = Infinity;   // For example, 10/0 produces Infinity

// Object wrapper `Number`
console.log(Number.isInteger(b));           // false
console.log(Number.isInfinite(g));          // true
console.log(Number.parseInt('42'));         // 42
console.log(Number.parseFloat('42.82'));    // 42.82
console.log(Number.MAX_SAFE_INTEGER);       // 9007199254740991
console.log(a.toExponential());             // 4.2e+1
console.log(b.toString());                  // "0.42"
console.log(c.toFixed(1));                  // 0.4
```

STRING TYPE

- **string** type represents an array of characters values.
- **string** values, as primitives, don't have any methods,

  but object wrapper **String** provide methods for values manipulation.

CHARACTERS TYPE

```
var a = "hello world!";                    // With double quotes
var b = 'hello world!';                    // With single quotes
var c = `hello world!`;                    // With backtick quotes

console.log(a[4]);                         // 'o'
console.log(a.includes('worl'));           // true
console.log(a.toUpperCase());              // 'HELLO WORLD';
console.log(a.split(' '));                 // ['hello', 'world']
```

OBJECT TYPE

- **object** type represents only non-primitive values.
- object type is the only **mutable** type in JavaScript.
- There are several built-in subtypes of object type in JavaScript, such as **Array, Date** or **Function** used for more-specific values.

OBJECT TYPE

```javascript
var a = { name: 'John' };       // In a single line

var b = {                       // In multiple lines
  name: 'John'
};

var c = {                       // With double quoted property name
  "name": "John"
};

var d = {                       // With multiple properties
 name: 'John',
  age: 19,
  active: true
};

var e = {                       // Nested
  name: 'John',
  age: 19,
  active: true,
  parent: {
    name: 'Bill',
    age: 43
  }
};
```

```javascript
// Accessing object's properties
console.log(e.name);                // 'John'
console.log(e.parent.name);         // 'Bill'
console.log(e['parent']['age']);    // 45

// Modifying object
a.name = 'William';
console.log(a);                     // { name: 'William' }

// Adding new property in object
a.phone = '+123456789';
console.log(a);                     // { name: 'William', phone: '+123456789' }
```

ARRAYS

- **Array** is special subtype of object used for storing multiple values in a single variable.
- It provides handy methods for manipulation with arrays.
- Arrays in JavaScript can contain multiple types.
- Indexing starts from 0.

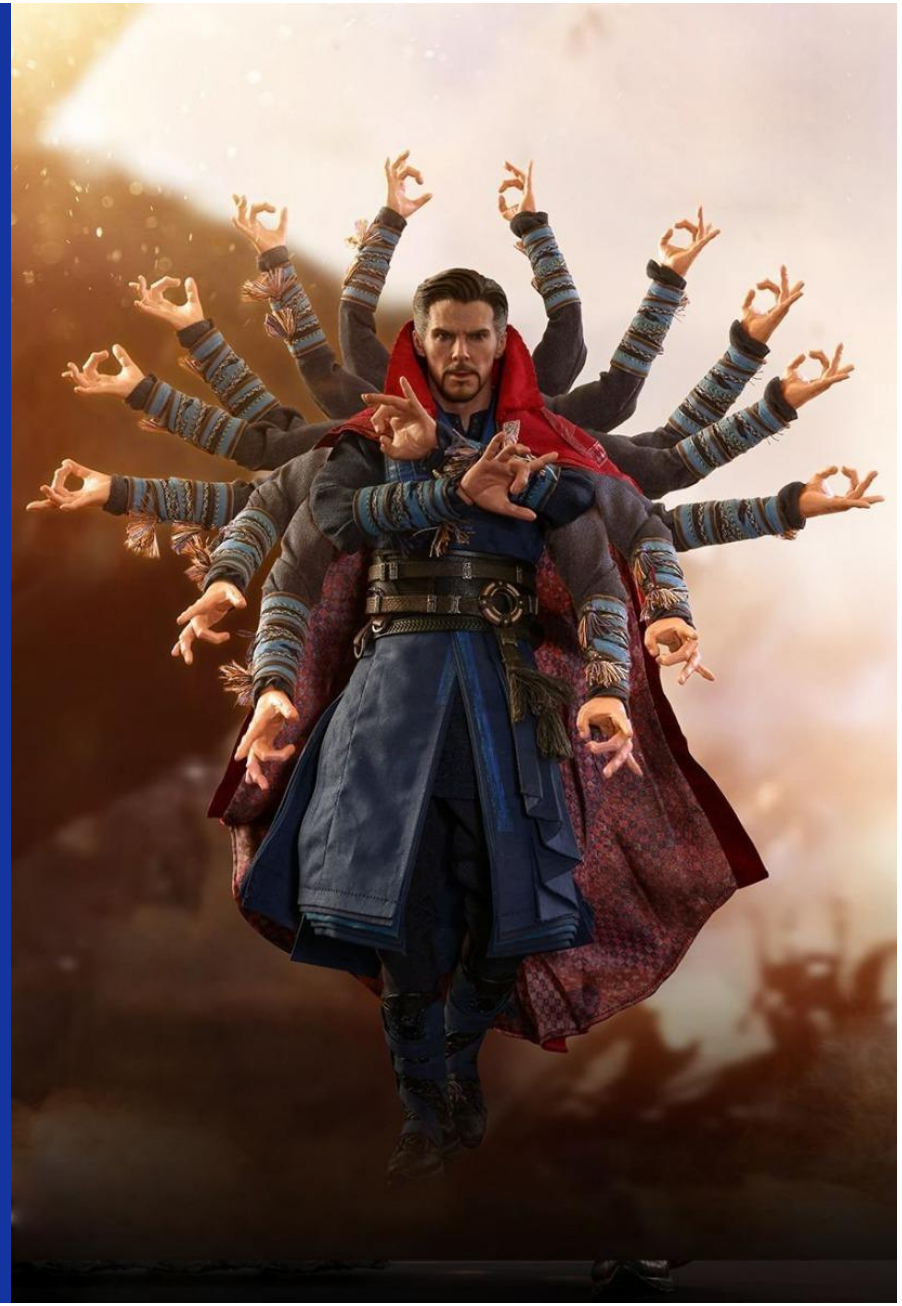
ARRAYS

```javascript
var a = [1, 2, 3];                          // Defining an array

var b = [true, null, Infinity, "boo"];      // Array with multiple value types

var c = [];                                 // Empty array

// Accessing arrays' properties
console.log(a[0]);      // 1
console.log(b[2]);      // Infinity
console.log(c[5]);      // undefined

// Modifying arrays
a[2] = 999;
console.log(a);             // [1, 2, 999];

var d = [1, 2, 3, 4, 5];
d.push(6);
console.log(d);             // [1, 2, 3, 4, 5, 6]

var e = [1, 2, 3, 4, 5];
e.pop();
console.log(e);             // [1, 2, 3, 4]

var f = [1, 2, 3];
f.reverse();
console.log(f);             // [3, 2, 1]

var g = [1, 2, 3, 4, 5, 6];
g = g.filter(function filterEven(item) {
    return item % 2;
});
console.log(g);             // [1, 3, 5];

var h = [1, 2, 3, 4, 5, 6];
h = h.map(function doubleItems(item) {
    return item * 2;
});
console.log(h);             // [2, 4, 6, 8, 10, 12];
```

02

CONTROL FLOW

JavaScript syntax in a nutshell.



SEMICOLON & USE-STRICT

- In JavaScript, semicolon is optional.
- 'use strict' directive is used in order to write more secure code and avoid unwanted errors.

OPERATORS & ORDERS OF PRECENDENCE

Operator	Operation	OoP	OoE
++	Increment	1	Right to Left
--	Decrement	1	Right to Left
-	Negation	1	Right to Left
!	NOT	1	Right to Left
*, /, %	Multiplication, division, modulus	2	Left to Right
+, -	Addition, subtraction	3	Left to Right
+	Concatenation	3	Left to Right
<, <=	Less than, less that or equal	4	Left to Right
>, >=	Greater than, greater than or equal	4	Left to Right

Operator	Operation	OoP	OoE
==	Equal	5	Left to Right
!=	Not equal	5	Left to Right
===	Identity	5	Left to Right
!==	Non-identity	5	Left to Right
&&	AND	6	Left to Right
||	OR	6	Left to Right
?:	Ternary	7	Right to Left
=	Assignment	8	Right to Left
+=, -=, *=, /=, %=	Arithmic assignment	8	Right to Left

CONDITIONAL EXECUTION – IF-ELSE

- For conditional execution of code, **if-else** blocks are used.

```
var active = true;

if (active) {
  console.log('User is active.');
} else {
  console.log('User is inactive.');
}

// Output:
// 'User is active'
```

CONDITIONAL EXECUTION – SWITCH-CASE

- For multiple branches, **switch-case** syntax is preferred.

```javascript
var status = 'active';

switch (status) {
  case 'active':
    console.log('User is active.');
    break;
  case 'inactive':
    console.log('User is inactive.');
    break;
  case 'blocked':
    console.log('User is blocked.');
    break;
  default:
    console.log('User status is unknown.');
}
```

LOOPS - FOR

- Handy if we want to execute block of code repeatedly, **knowing** how many times we want to execute it.

```
var arr = [1, 2, 3, 4, 5];

for (var i = 0; i < arr.length; i++) {      // 1, 2, 3, 4, 5
    console.log(arr[i]);
}

arr.forEach(function logItem(item){         // 1, 2, 3, 4, 5
    console.log(item);
});

for (var item of arr) {                     // 1, 2, 3, 4, 5
    cosole.log(item);
}
```

LOOPS - WHILE

- Handy if we want to execute block of code repeatedly, **not knowing** how many times we want to execute it.

```
var arr = ['Max', 'Viktor', 'Todd', 'Kyle'];
var i = 0;

while (arr[i] !== 'Todd') {          // 'Max', 'Viktor'
    console.log(arr[i++]);
}
```

LOOPS – BREAK & CONTINUE

- Useful if we want stop execution of loop or to skip some iteration if given condition is matched.

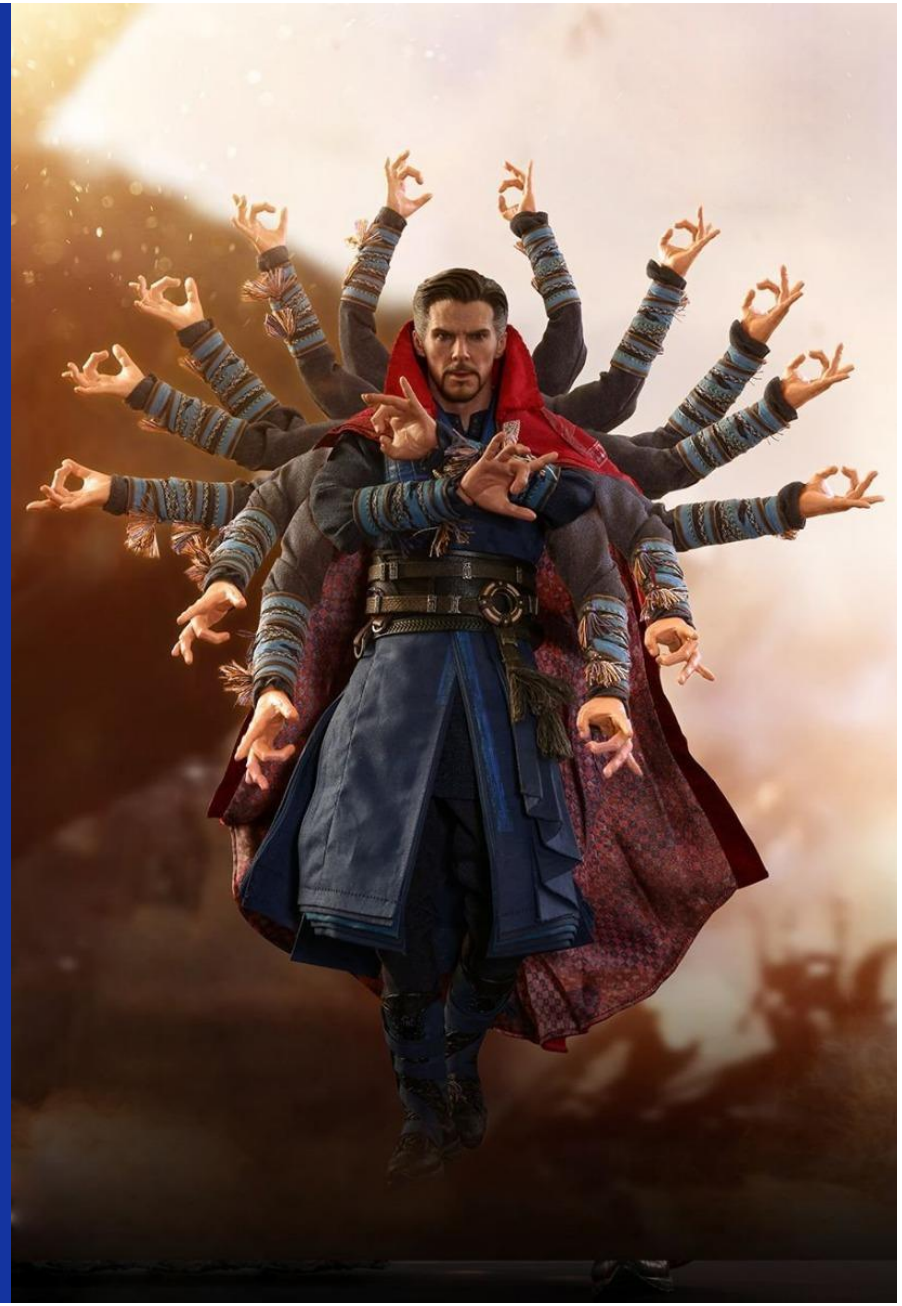
```
var fruit = ['Apple', 'Orange', 'Lemon', 'Watermelon'];

for (var i = 0; i < fruit.length; i++) {
    if (fruit[i] === 'Lemon') {
        break;
    }

    console.log(fruit[i]);
}

// Output:
// Apple, Orange
```

```
var fruit = ['Apple', 'Orange', 'Lemon', 'Watermelon'];

for (var i = 0; i < fruit.length; i++) {
    if (fruit[i] === 'Lemon') {
        continue;
    }

    console.log(fruit[i]);
}

// Output:
// Apple, Orange, Watermelon
```

03

FUNCTIONS

Coding – the better way.



THE MAIN PURPOSE

- Split the code into smaller, reusable chunks.
- Cleaner and more testable code.

PARAMETERS & RETURN VALUE

- As in any other languages, functions can receive arguments when invoked.
- Parameters can be optional, if default value is provided.
- Values are returned by using **return** keyword.
- If nothing's explicitly returned, function will return **undefined** implicitly.

DEFINING THE FUNCTIONS

- In JavaScript functions are defined by using **function** keyword.

```
function min(a, b) {
    return a < b ? a : b;
}

function writeMyNameInConsole(name) {
    console.log('My name is ' + name);
}

function sumNumbersFromTo(from, to) {
    var sum = 0;

    for (var n = from; n <= to; n++) {
        sum += n;
    }

    return sum;
}

// Invoking functions
console.log(min(4, 5));                 // 4
writeMyNameInConsole('Kyle');           // 'My name is Kyle'
console.log(sumNumbersFromTo(5, 10))    // 45
```

MULTIPLE WAYS OF DEFINING THE FUNCTIONS

- You can either define the function by using **function** keyword (function declaration), or by **assigning a value** to the variable (function expression).

```
function min(a, b) {
    return a < b ? a : b;
}

const minimum = function(a, b) {
    return a < b ? a : b;
}

// Invoking functions
console.log(min(4, 5));      // 4
console.log(minimum(4, 5));  // 4
```

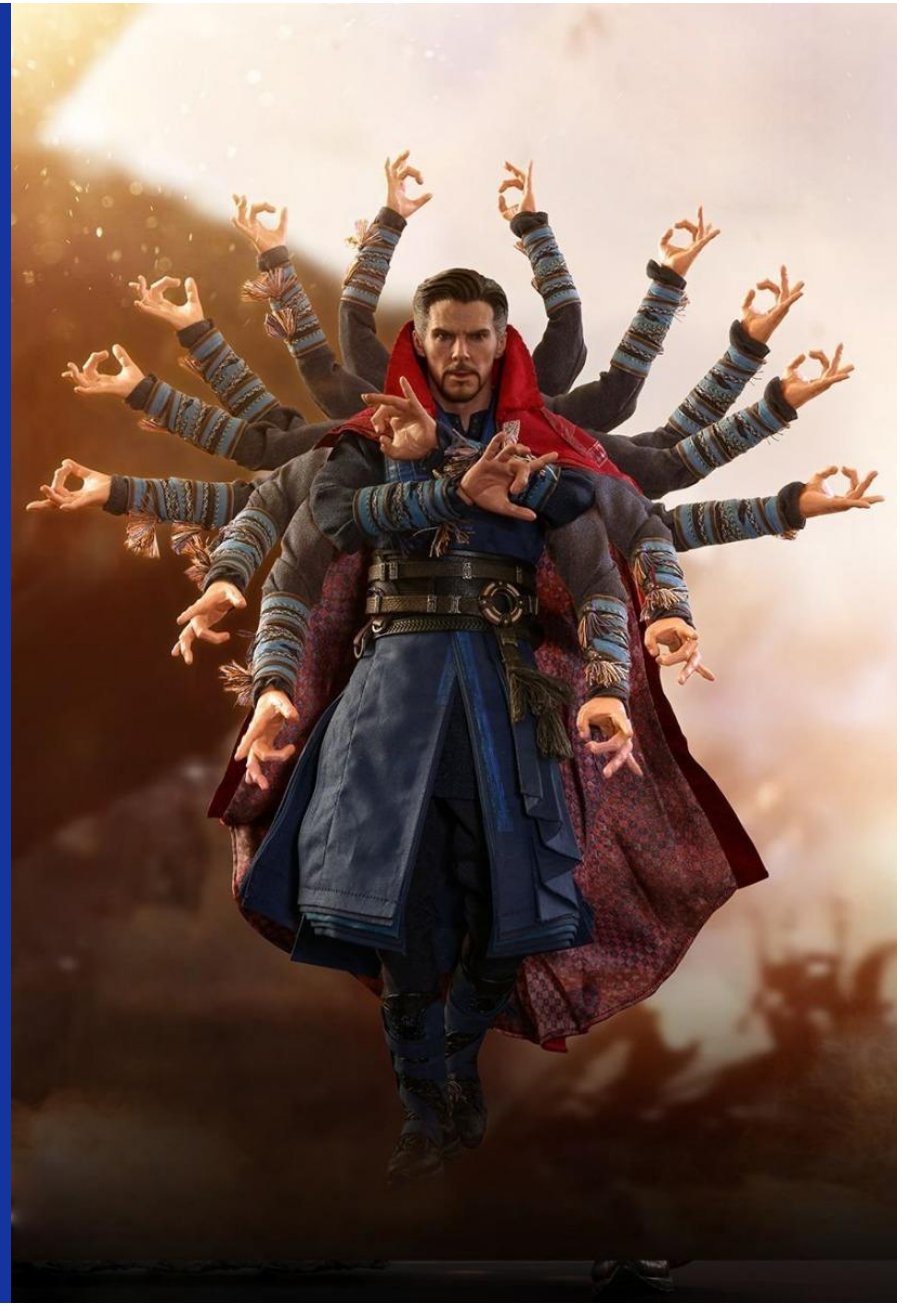
PARAMETERS & RETURN VALUE

```
function writeMyNameInConsole(name = 'Bill') {
    console.log('My name is ' + name);
}

writeMyNameInConsole();                    // 'My name is Bill'
console.log(writeMyNameInConsole());       // 'My name is Bill', undefined
```

04

PROJECT

Fun part.



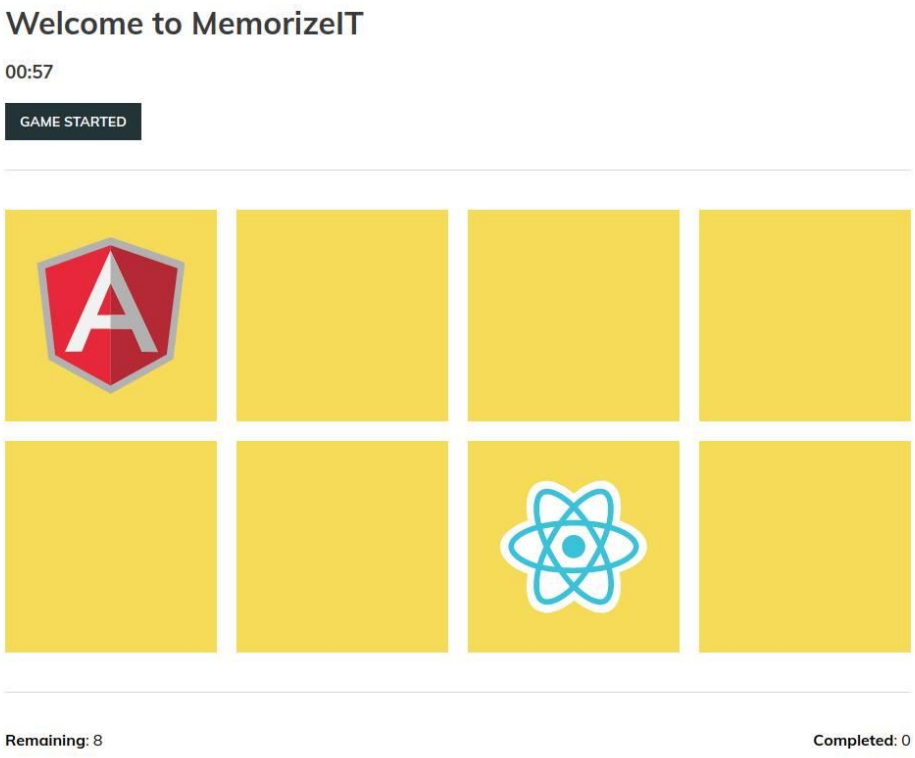
BEFORE THE CODING

- Make **3** GitHub repositories:
  - JavaScript Project - Playable game in pure JavaScript
  - Node.js Project – Node.js API for receiving scores
  - Leaderboard Project - ReactJS built leaderboard page

PROJECT

- For JavaScript Project:
  - Create layout for the game by using HTML and CSS technologies.
  - Implement main function that initializes the game state.
  - Implement function for starting the game.
  - Implement functions for game events such as click, keypress, etc.
  - Implement function for checking if the game has ended.
  - Implement function for sending the results to the server.

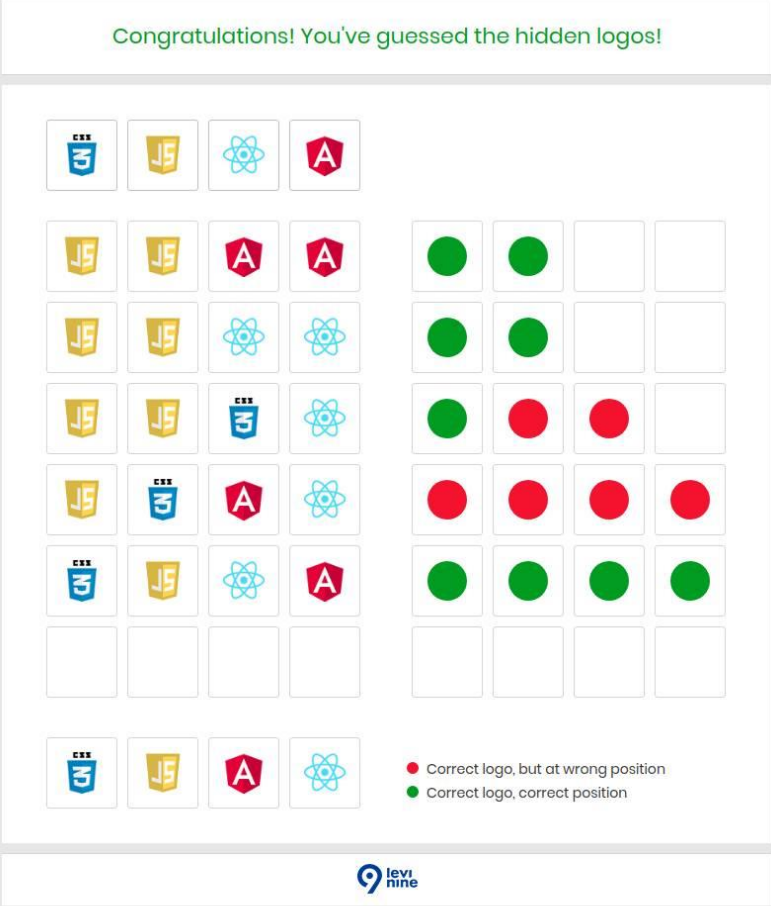
MEMORY GAME


Welcome to MemorizeIT
00:57
GAME STARTED
Remaining: 8
Completed: 0

CATCH IT

Your score: 50

MASTERMIND


Congratulations! You've guessed the hidden logos!
Correct logo, but at wrong position
Correct logo, correct position
levi nine

LITERATURE

- Eloquent JavaScript: https://eloquentjavascript.net
- You Don't Know JS: https://github.com/getify/You-Dont-Know-JS/tree/1st-ed

THANK YOU

levi nine
Technology Services