Code Academy

Gytis Juozenas

JavaScript arrays, functions and events

2022

JavaScript programavimo kalba

Šiandien išmoksite

01 Arrays

02 Functions

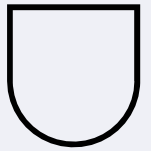
Masyvai (Array) ir jų metodai (teorija)

Gana dažnai pastebime, kad mums reikia numeruotos kolekcijos, kur turime 1, 2, 3 elementus ir t.t.

Pavyzdžiui, mums reikia, kad būtų išsaugotas kažkoks sąrašas:

- Vartotojai;
- Prekės;
- HTML elementai ir kt.

Yra speciali duomenų struktūra, pavadinimu Masyvas (Array), skirta saugoti numeruojamus sąrašus (kolekcijas).



Masyvai (Array) ir jų metodai (teorija)

Tuščio masyvo (Array) kūrimo sintaksės (yra dvi):

let arr = new Array() //Geras, bet stengitės nenaudoti

let arr = [] //Geras;
*Beveik visada naudojama antroji sintaksė.

Pvz.: let fruits = ["Apple", "Orange", "Plum"];

Masyvai (Array) ir jų metodai (teorija)

Masyvo (Array) elementai yra sunumeruoti, pradedant nuo nulio.

Elementą iš masyvo galime gauti laužtiniuose skliaustuose nurodę jo numerį:

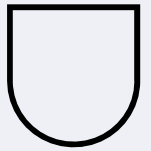
```
let fruits = ["Apple", "Orange", "Plum"];

console.log( fruits[0] ); // Apple
console.log( fruits[1] ); // Orange
console.log( fruits[2] ); // Plum
```

Užduotis nr. 1

Susikurkite array ir output‘inkite arba alert‘inkite kiekvieną array elementą.

Pvz.:

```
let fruits = ["Apple", "Orange", "Plum"];

console.log( fruits[0] ); // Apple
console.log( fruits[1] ); // Orange
console.log( fruits[2] ); // Plum
```



Masyvai (Array) ir jų metodai (teorija)

Galime masyvo (Array) elementą pakeisti:

let fruits = ["Apple", "Orange", "Plum"];

fruits[2] = 'Pear'; // dabar["Apple", "Orange", "Pear"]

... arba pridėkite naują elementą prie masyvo (Array):

fruits[3] = 'Lemon';
// dabar ["Apple", "Orange", "Pear", "Lemon"]

Masyvai (Array) ir jų metodai (teorija)

Bendras elementų skaičius masyve (Array) randamas su length:

```
let fruits = ["Apple", "Orange", "Plum"];

console.log( fruits.length ); // 3
```

Užduotis nr. 2

Prieš tai sukurtame array pakeiskite ir pridėkite po elementą ir po visko patikrinkite naudodamiesi console.log koks ilgis yra jūsų sukurto ir update‘into array.

Pvz.:

fruits[2] = 'Pear‘; //pakeisti naudojate skaičių laužtiniuose skliaustuose to elemento kurį norite pakeisti
fruits[3] = 'Lemon‘; //pridėti naudojate skaičių vienu didesniu nei didžiausias esančio elemento skaičius masyve

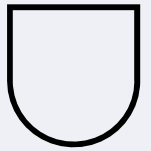
console.log(fruits.length); // ilgi surandate

Masyvai (Array) ir jų metodai (teorija)

Ciklai (Loops) ir Masyvai (Arrays)

Vienas seniausių būdų manipuliuoti masyvo (Array) elementais yra ciklas (Loops), kurio pagalba pasiekiami masyvo elementai:

```
let arr = ["Apple", "Orange", "Pear"];

for (let i = 0; i < arr.length; i++) {
  console.log( arr[i] );
}
```



Masyvai (Array) ir jų metodai (teorija)

Ciklai (Loops) ir Masyvai (Arrays)

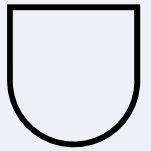
Tačiau masyvams (Arrays) yra dar viena ciklų forma, for..of:

```
let fruits = ["Apple", "Orange", "Plum"];

// iterates over array elements
for (let fruit of fruits)
{ console.log( fruit );
}
```

Užduotis nr. 3

Susikurkite bet kokį array ir su dviem loop būdais jį outputinkite.

```
let arr = ["Apple", "Orange", "Pear"];
for (let i = 0; i < arr.length; i++)
{ console.log( arr[i] );
}

let fruits = ["Apple", "Orange", "Plum"];
// iterates over array elements
for (let fruit of fruits)
{ console.log( fruit );
}
```

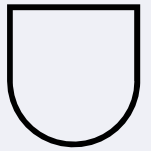


JavaScript funkcijos (teorija)

Gana dažnai norime atlikti panašų veiksmą daugelyje savo programos vietų.

Pavyzdžiui, mums reikia parodyti gražiai atrodantį pranešimą, kai lankytojas prisijungia, atsijungia ir galbūt kur nors kitur.

Funkcijos yra pagrindiniai programos elementai (“building blocks”). Jie leidžia kodą iškviesti daugybę kartų be pasikartojimo jo rašymo.



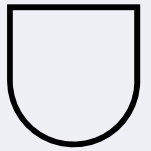
JavaScript funkcijos (teorija)

Jau matėme naršyklėje integruotų funkcijų, tokių kaip *alert(message) ir prompt(message, default).*

Tačiau mes taip pat galime sukurti savo funkcijas!

Beje, funkcijos yra objektai!

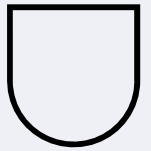
Funkcijos gali būti priskirtos kintamiesiems, saugomoms objektuose ar masyvuose, perduodamos kaip argumentas kitoms funkcijoms ir grąžinamos iš funkcijų.



JavaScript funkcijos (teorija)

Funkciją galima apibrėžti trimis būdais:

- **Function Declaration (Function Statement), plačiau čia;**
- Function Expression (Function Literal), plačiau čia;
- Arrow Function, plačiau čia.



JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Norėdami sukurti funkciją, galime naudoti funkcijos deklaraciją (Function Declaration):

```
function showMessage()
  { console.log( 'Hello
  everyone!' );
}
```

JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Raktinis žodis *function* eina pirmas, tada eina funkcijos pavadinimas, tada parametrų sąrašas tarp skliaustelių (jeigu funkcija juos turi) ir galiausiai funkcijos kodas, taip pat vadinamas „funkcijos kūnu“ (“the function body”), viduje {...} skliaustų.

Sintaksė:

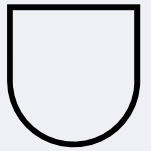
```
function name(parameters) {
  ...body...
}
```

JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Funkcijos iškvietimas:

```
function showMessage() {
  console.log( 'Hello everyone!' );
}

showMessage(); // nurodydami funkcijos pavadinimą mes ją iškviesime.
```



JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Vietiniai kintamieji (local variables) – funkcijos viduje deklaruojamas kintamasis matomas tik tos funkcijos viduje.

```
function showMessage() {
  let message = "Hello, I'm JavaScript!"; // local variable
  console.log( message );
}

showMessage(); // Hello, I'm JavaScript!

console.log( message ); // <-- Error! The variable is local to the function
```

JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Išoriniai kintamieji (outer variables) – funkcija gali pasiekti ir išorinį kintamąjį.

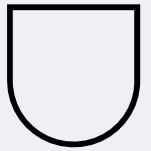
```
let userName = 'John';

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

showMessage(); // Hello, John
```

JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Išoriniai kintamieji (outer variables) – funkcija turi visišką prieigą prie išorinio kintamojo (gali jį ir modifikuoti).

```
let userName = 'John';

function showMessage() {
  userName = "Bob"; // changed the outer variable

  let message = 'Hello, ' + userName;

  console.log(message);

}
console.log( userName ); // John before the function call

showMessage();

console.log( userName ); // Bob, the value was modified by the function
```



JavaScript funkcijos (teorija)

SVARBU: Visuotiniai kintamieji (Global variables)

Kintamieji, deklaruojami už bet kurios funkcijos ribų, pavyzdžiui, ankščiau esančiame kode nurodytas userName, vadinami globaliais (global).

Visuotiniai kintamieji matomi visoms funkcijoms.

Gera praktika yra sumažinti visuotinių kintamųjų (global variables) naudojimą. Šiuolaikiniame kode jų yra nedaug arba visai nėra. Dauguma kintamųjų priklauso jų funkcijoms.

Užduotis nr. 4

Pasirašykite bet kokią funkciją ir pabandykite ją iškviesti.

```
let userName = 'John';
function showMessage() {
  userName = "Bob"; // (1) changed the outer variable
  let message = 'Hello, ' + userName;
  console.log(message);
}
console.log( userName ); // John before the function call
showMessage();
console.log( userName ); // Bob, the value was modified by the function
```

JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Parametrai (Parameters) - galime perduoti savavališkus duomenis į funkcijas naudodami parametrus (dar vadinamus funkcijų argumentais).

```
function showMessage(from, text) { // parametrai: from, text
  console.log(from + ': ' + text);
}

showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
```

JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Numatytosios vertės (Default values) - Jei parametras nepateikiamas, jo vertė undefined, tokiu atveju reikia nurodyti numatytąsias vertes (default values).

Pvz., pirmiau minėtą funkciją showMessage(from, text) galima iškviesti vienu argumentu:

showMessage("Ann");

Tai nėra klaida. Toks iškvietimas pateiks „Ann: neapibrėžtas“. Nėra teksto, todėl daroma prielaida, kad tekstas === neapibrėžtas.

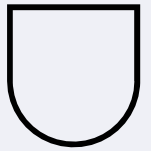
JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Numatytosios vertės (Default values) - Jei parametras nepateikiamas, jo vertė undefined, tokiu atveju reikia nurodyti numatytąsias vertes.

Jei šiuo atveju norime naudoti „numatytąjį“ tekstą, mes galime jį nurodyti po =:

```
function showMessage(from, text = "no text given")
  { console.log( from + ": " + text );
}

showMessage("Ann"); // Ann: no text given
```



JavaScript funkcijos (teorija)

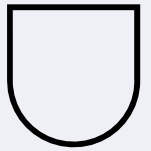
Function Declaration (Function Statement)

Funkcija turi grąžinti vertę (Returning a value).

return gali būti, bet kurioje funkcijos vietoje. Kai funkcija ją pasiekia, funkcija sustoja ir vertė grąžinama į iškvietimo kodą (priskirtą rezultatui aukščiau).

```
function sum(a, b) {
  return a + b;
}
let result = sum(1, 2);
console.log( result ); // 3
```

Daugiau apie return.



JavaScript funkcijos (teorija)

Function Declaration (Function Statement)

Funkcijos įvardijimas

Funkcijos yra veiksmai. Taigi jų pavadinimas dažniausiai yra veiksmažodis. Jis turėtų būti trumpas, kiek įmanoma tikslesnis ir aprašyti, ką atlieka funkcija, kad kas nors, perskaitęs kodą, gautų nuorodą, ką funkcija atlieka.

Pvz., Funkcijos, kurios prasideda žodžiu “show“, paprastai ką nors rodo.

"get…" – return a value,

"calc…" – calculate something,

"create…" – create something,

"check…" – check something and return a boolean, etc.

Užduotis nr. 5

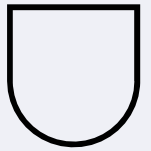
Pasirašykite funkciją kuri pagal amžių, kurį įrašysime po funkcijos iškviesdami funkciją dalyje nuspręs ar žmogui galima gerti energetinį. Jei 18 ir daugiau, funkcija returnins alerta, jei mažiau returnins pasiklausą ar tėvai davė leidimą (confirm, pasigooglinti).

```
function sum(a, b) {
  return a + b;
}

let result = sum(1, 2);
console.log( result ); // 3
```

Užduotis nr. 6

Persirašykite praeitą funkciją kad joje nebūtų if else, bet vietoj jo conditional ternary operatorius atsirastų.

```
function sum(a, b) {
  return a + b;
}

let result = sum(1, 2);
console.log( result ); // 3
```



Įvykių registravimas (Events) (teorija)

Įvykiai yra veiksmai, kurie vyksta jūsų programuojamoje sistemoje, apie kuriuos sistema jums praneša, kad galėtumėte į juos tam tikru būdu reaguoti (jei norite).

Pvz., Jei vartotojas paspaudžia pelės mygtuką tinklalapyje, galbūt norėsite reaguoti į tą veiksmą pateikdami informacijos langelį.

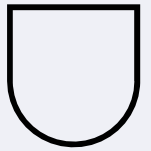
document.body.addEventListener("click", () => alert("Ačiū, kad paspaudėte mygtuką!"));

Įvykių registravimas (Events) (teorija)

Naršyklės atveju įvykiai suaktyvinami naršyklės lange ir yra paprastai pridedami prie konkretaus jame esančio elemento – tai gali būti atskiras (1) elementas, (2) elementų rinkinys, (3) HTML dokumentas, įkeltas į dabartinį skirtuką, arba (4) visas naršyklės langas.

Naršyklėje gali įvykti daugybė įvairių tipų įvykių, pavyzdžiui (įvykių sąrašo nuoroda):

- Vartotojas spustelėjęs pelės mygtuką virš tam tikro elemento arba užvedęs žymeklį ant tam tikro elemento.
- Vartotojas paspaudžia klaviatūros mygtuką.
- Vartotojas keičia dydį arba uždaro naršyklės langą.
- Puslapis baigiamas įkelti.
- Pateikiama forma.
- Vaizdo įrašas atkuriamas, pristabdomas arba baigiamas.
- Įvyko klaida ir t.t.



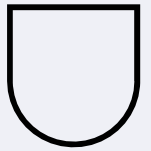
Įvykių registravimas (Events) (teorija)

Kiekviename įvykyje (*event*) yra įvykių tvarkyklė (*event handler* arba dar vadinami *event listeners*, tačiau griežtai kalbant jie veikia kartu, bet nėra tas pats), tai yra kodo blokas (paprastai JavaScript funkcija, kurią sukuriate jūs kaip programuotojas), kuris bus vykdomas įvykus įvykiui.

SVARBU:

- *event listeners* “išklauso” įvykį, o *event handler* yra kodas, kuris paleidžiamas reaguojant į įvykį.

Daugiau apie event listeners ir event handlers čia.



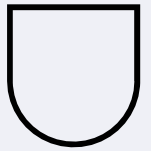
Įvykių registravimas (Events) (teorija)

Event handlers

Norėdami reaguoti į (*events)* įvykius, galime paskirti *handler* – funkciją, kuri vykdoma įvykus įvykiui. Taigi, *handlers* yra būdas paleisti JavaScript kodą priklausomai nuo vartotojo veiksmų.

Yra 3 būdai, kaip priskirti įvykių tvarkytojus (*event handlers*):

- HTML atributas: *onclick = "..."*.
- DOM savybė: *elem.onclick = function*.
- **Metodai:**
  - ***elem.addEventListener(event, handler[, phase]) – pridėti įvykiui;***
  - *removeEventListener – pašalinti įvykiui.*



Įvykių registravimas (Events) (teorija)

Event handlers

Yra keletas būdų, kaip priskirti handler. Pažiūrėkime kelis

HTML atributas:

<input value="Click me" onclick="alert('Click!')" type="button">

Įvykių registravimas (Events) (teorija)

Event handlers

Yra keletas būdų, kaip priskirti handler. Pažiūrėkime kelis.

DOM nuosavybė:

```
<input id="elem" type="button" value="Click me">
<script>
  elem.onclick = function() {
    alert('Thank you');
  };
</script>
```

Įvykių registravimas (Events) (teorija)

Event handlers

Yra keletas būdų, kaip priskirti handler. REKOMENDUOJAMAS:

addEventListener (įvykio pridėjimas):

element.addEventListener(event, handler[, options]);

removeEventListener (įvykio panaikinimas):

element.removeEventListener(event, handler[, options]);

Event – įvykio pavadinimas (pvz. “click”);
Handler – funkcija, kuri bus kviečiama įvykus įvykiui (event);

Options (neprivaloma) – true arba false (plačiau)

Užduotis nr. 7

Susikurkite bet kokį event listenerį ir išbandykite ar jis veikia, panaudokite visus tris būdus.

```
<input value="Click me" onclick="alert('Click!')" type="button">

<input id="elem" type="button" value="Click me">
<script>
  elem.onclick = function() {
    alert('Thank you');
  };
</script>

element.addEventListener(event, handler[, options]);
```

JavaScript funkcijos (teorija)

Function Expression (Function Literal);

- *function* nėra pirmasis raktinis žodis eilutėje
- funkcijos vardas yra neprivalomas. Gali būti anoniminė funkcijos išraiška arba pavadinta funkcijos išraiška.
- funkciją reikia apibrėžti, tada ją galima vykdyti;
- funkciją galima automatiškai vykdyti po apibrėžimo (vadinamą “IIFE” Immediately Invoked Function Expression)

Sintaksė:

let doSomething = function() {}

Užduotis nr. 8

Paverskite mano prieš tai pavyzdinę rodytą funkcijos deklaraciją į funkcijos expression (anonymous function). Tik jums čia nebūtina bus jos priskirti kintamajam.

```
<p id="demo"></p>

document.addEventListener("click", myFunction);

function myFunction() {

  document.querySelector("#demo").textContent = "Hello World";

}
```

Pvz.:

let doSomething = function() {}

JavaScript funkcijos (teorija)

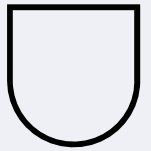
Arrow Function

Rodyklės funkcija (Arrow Function) yra vadinamas “sugar syntax”, skirta sukurti anoniminę funkcijos išraišką;

let func = (arg1, arg2, ...argN) => expression

Tai yra trumpesnė versija šio kodo:

```
let func = function(arg1, arg2, ...argN) {
  return expression;
};
```



JavaScript funkcijos (teorija)

Arrow Function

```
let sum = (a, b) => a + b;

/* This arrow function is a shorter form of:
function sum(a, b) {
  return a + b;
};
*/

console.log( sum(1, 2) ); // 3
```

JavaScript funkcijos (teorija)

Arrow Function

Vizualiai rodyklės funkcija (**arrow function**) leidžia jums rašyti funkcijas naudojant trumpesnę sintaksę:

iš function declaration:

```
function getData() {
  //...
}
```

į arrow function (atkreipkite dėmesį, kad mes čia neturime funkcijos vardo):

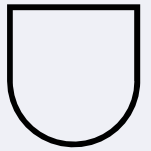
```
() => {
//…
}
```

JavaScript funkcijos (teorija)

Arrow Function

Rodyklės funkcijos (arrow functions) anonimiškos. Jas privalome priskirti kintamajam:

```
let getData = () => {
  //...
}
getData()
```



JavaScript funkcijos (teorija)

Arrow Function

Jei funkcijos kode yra tik vienas sakinys, galite praleisti skliaustus ir viską surašyti vienoje eilutėje:

const getData = () => console.log('hi!')

JavaScript funkcijos (teorija)

Arrow Function

Parametrai pateikiami skliaustuose:

const getData = (param1, param2) =>
console.log(param1, param2)

Jei turite vieną (ir tik vieną) parametrą, skliaustelius galite praleisti:

const getData = param => console.log(param)

JavaScript funkcijos (teorija)

Arrow Function

Rodyklių funkcijos (arrow functions) leidžia gauti numanomą grąžą - vertės grąžinamos nenaudojant *return* raktažodžio. Tai veikia, kai funkcijos kūne (function body) yra vienos eilutės kodas:
const getData = () => 'test'

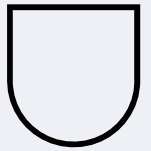
getData () // 'test‘

Jei arrow function kodas yra išsiplėtęs per kelias eilutes, return gali ir būna naudojamas:

```
const magic =
function() {
  return new Date();
};
```

```
const magic = ()
=> {
  return new
  Date();
};
```

```
const magic = () => new
Date();
```



JavaScript funkcijos (teorija)

Arrow Function

Kaip ir įprastos funkcijos, rodyklių funkcijos (arrow functions) gali turėti numatytąsias parametrų reikšmes, jei jos nebus perduotos:

```
const getData = (color = 'black', age = 2) => {
  //do something
}
```

JavaScript funkcijos (teorija)

Arrow Function

Pastabos:

- Kaip ir įprastos funkcijos (function declaration, function expression), rodyklių funkcijos (arrow functions) gali **grąžinti tik vieną vertę**.
- Rodyklių funkcijose (arrow functions) taip pat, gali būti kitų rodyklių funkcijų (arrow functions) ar net įprastų funkcijų (function declaration).
- Rodyklių funkcijos (arrow functions) nėra hoisted, t.y. funkcija turi būti aprašyta prieš jos iškvietimą.

Daugiau apie Arrow Function čia ir čia.

Užduotis nr. 9

Paverskite žemiau pavaizduotas funkcijas į arrow funkcijas:

```
let show = function () {
console.log('Anonymous
function‘);
};
let add = function (a, b)
{ return a + b;
};
```

Sintaksė:

const getData = (param1, param2) => console.log(param1, param2);

Užduotis nr. 10

Parašykite kodą kuris turės tris eventListener'ius, kuriuose bus iškviečiama po funkciją:

1. Užvedus pelytę ant ekrano išmeta iš naujos eilutės paragrafe tekstą Mouse over. Jums reikės event „mouseover";
2. Patraukus pelytę nuo ekrano išmeta iš naujos eilutės paragrafe tekstą Mouse out. Jums reikės event „mouseout";
3. Paspaudus pelytės mygtuką ant ekrano išmeta iš naujos eilutės paragrafe tekstą Clicked. Jums reikės event „click".

```
element.addEventListener(event, handler[, options]);
function name(parameters) {
  ...body...
}
```