Code Academy

# JavaScript funkcijos

2 LYGIS

2 DALIS

2021

JavaScript programavimo kalba

# Šiandien išmoksite

01 JavaScript funkcijos

# 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 pasikartojimojo rašymo.

# JavaScript funkcijos (teorija)

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

**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() {
  cosnole.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 {...} sklaistų.

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"; // (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)

**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.

# 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).

Direktyva *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
```

# JavaScript funkcijos (teorija)

## Function Declaration (Function Statement)

```
function checkAge(age) {
  if (age >= 18) {
    return true;
  } else {
    return confirm('Do you have permission from your parents?');
  }
}

let age = prompt('How old are you?', 18);
if ( checkAge(age) ) {
  alert( 'Access granted' );
} else {
  alert( 'Access denied' );
}
```

# 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. 1

**Function Declaration (Function Statement)**

Ar skirsis grąžinama vertė?

```
1  function checkAge(age) {
2    if (age > 18) {
3      return true;
4    } else {
5      // ...
6      return confirm('Did parents allow you?');
7    }
8  }
```

```
1  function checkAge(age) {
2    if (age > 18) {
3      return true;
4    }
5    // ...
6    return confirm('Did parents allow you?');
7  }
```

## Užduotis nr. 2

**Function Declaration (Function Statement)**

Funkcija grąžins true, jei parametro amžius yra didesnis nei 18 metų.

```
function checkAge(age) {
  if (age > 18) {
    return true; //console.log(true)
  } else {
    return confirm('Did parents allow you?');
  }
}
```

Perrašykite funkcijos kodą, kad jis būtų be if ir vienoje eilutėje:

- Naudokite: ternary operator ?

## Užduotis nr. 3

**Function Declaration (Function Statement)**

Parašykite funkciją min(a, b), kuri grąžina mažiausiai iš dviejų skaičių a ir b

Pvz.:

min(2, 5) == 2

min(3, -1) == -1

min(1, 1) == 1

# JavaScript funkcijos (teorija)

**Function Expression (Function Literal);**

- *funkction* 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() {}*

# 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'
```

# 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) galime **grąžinti tik vieną vertę;**
- Rodyklių funkcijose (arrow functions) taip pat, gali būti kitų rodyklių funkcijų (arrow functions) ar net įprastų funkcijų (function declaration);
- Rodyklių funkcijose (arrow functions) nėra hosited, t. y. funkcija turi būti aprašyta prieš jos iškvietimą.

| | |
|---|---|
| **Function Declaration (Function Statement)** | https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function |
| **Function Expression (Function Literal)** | https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function |
| **Arrow Function** | https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions |

# Naudinga informacija