# **Array**

Masyvai

## **Sukurti nauja masyva (create)**

Masyvas sukuriamas panaudojus laužtinius skliaustus []Masyvas savyje gali laikyti bet kokia Javascript reikšmę.

| const array\_name = [item1, item2, ...]; |
| --- |

| const arra\_name ["string", 4, true, undefined, null, {}, []]; |
| --- |

Aprašant masyvą, galima narius sudėti į skirtingas eilutes, tarpai ir naujos eilutės pradėjimas neturi reikšmės.

| const cars = [  "Saab",  "Volvo",  "BMW" ]; |
| --- |

Masyva galime sukurti priskyrę atitinkamam masyvo indeksui po reikšmę.

| const cars = []; cars[0]= "Saab"; cars[1]= "Volvo"; cars[2]= "BMW"; |
| --- |

Paprastai, pagal konvencija, kurdami masyvus naudojame const kintamojo tipą.

Masyva galime sukurti ir naudodami new bazinį žodį ir Array konstruktorių.:

| const array2 = new Array("eat", "sleep"); |
| --- |

Masyvų pavyzdžiai:

| *// tuščias array* const myList = [ ];  *// skaičių masyvas* const numberArray = [ 2, 4, 6, 8];  *// string masyvas* const stringArray = [ 'eat', 'work', 'sleep'];  *// maišytų reikšmių masyvas* const newData = ['work', 'exercise', 1, true]; |
| --- |

Taip pat masyvuose galime saugoti funkcijas arba objektus.

| const newData = [  {'task1': 'exercise'},  [1, 2 ,3],  function hello() { console.log('hello')} ]; |
| --- |

## **Pasiekti masyvo elementą (access)**

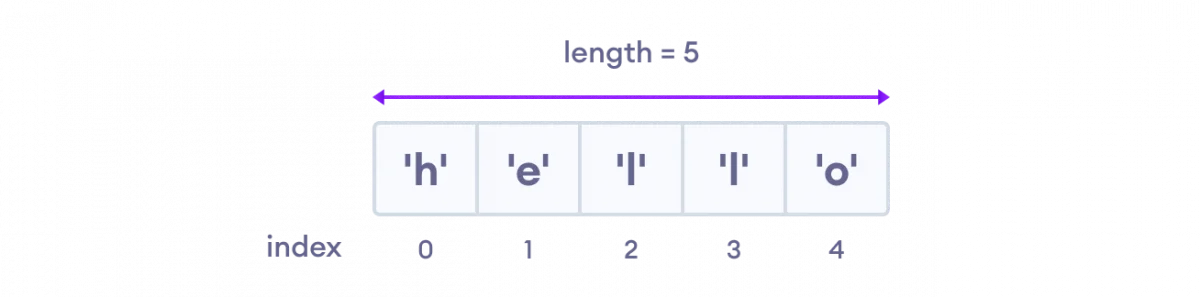
Masyvo elementai yra pasiekiami nurodant nario indeksą. Nuo 0 einant iki pat masyvo pabaigos.

| const myArray = ['h', 'e', 'l', 'l', 'o', [1, 2]];  *// first element* console.log(myArray[0]); *// "h"*  *// second element* console.log(myArray[1]); *// "e"*  *console.log(*myArray[6][0]*);* |
| --- |

Galima nurodyti ir neegzistuojantį masyvo nario indeksą, tačiau tuomet Javascript gražins undefined. Taip pat verta paminėti, kad masyvo pirmo nario indeksas prasideda nuo 0.

Masyvas paaiškinimo schema:

const array = ['h', 'e','l','l','o'];



## 

## **Redaguoti ir atnaujinti masyvo elementus (update)**

Masyvo elementus mes galime keisti ir priskirti jiems naujas reikšmes. Nurodant kuri masyvo elementa mes taikome ir su priskyrimo operatoriumi ‘=’, jam priskirti naują reikšmę.

| const cars = ["Saab", "Volvo", "BMW"]; cars[0] = "Opel"; console.log(cars); *// ["Opel", "Volvo", "BMW"]* |
| --- |

Nauja masyvo reikšmė, gali būti bet kokia kita Javascript palaikoma reikšmė. Labai dažnai, ciklo pagalba mes redaguojame esamas reikšmes į norimą formatą kurį vėliau galime panaudoti.

Kadangi masyvai yra objektai, mes jiems galime priskirti bet kokius atributus arba užrašyti ant viršaus jau egzistuojančius. Aišku tai daryti tikrai nerekomenduojama, nes dėl pakeistos masyvo elgsenos gali sugriūti kodas, tačiau reikia žinoti kad tai įmanoma, ypatingai vertinant kodo kibernetinį saugumą, nes piktybinis vartotojas, gali pakeisti mūsų kodo elgseną.

| const fruits = ['apple', 'orange', 'banana']; console.log(fruits.length) *// 3* fruits.length = 5; console.log(fruits.length); *// 5 Nors tai yra netiesa, bet Javascript leidžia ant viršaus užrašyti bazinius atributus. Ta patį galima padaryti ir su metodais.* |
| --- |

Taip pat galime sukurtam masyvui priskirti naujus metodus. Tačiau vėlgi to daryti nerekomenduojama, kadangi masyvo tikslas yra dirbti su serija narių ir standartinių bazinių metodų turi pakakti. Bet koks naujas sukurtas metodas neatitiks konvencijos ir klaidins kitus programuotojus.

| const fruits = ['apple', 'orange', 'banana'];  fruits.getFirstElement = function() {  return fruits[0]; }  console.log(fruits.getFirstElement()); *// 'apple'* |
| --- |

Galima sukurti nauja masyvo narį nurodant daug didesnį indeksą, negu yra narių masyve. Nors ir Javascript leidžia daryti tokius veiksmus, jie yra griežtai nerekomenduojame.

| const fruits = ['apple', 'orange', 'banana']; *// Sukuriame naują narį priskirdami jam didesnį indeksą negu yra masyve narių* fruits[10] = 'strawberry';  console.log(fruits.length); *// 11* console.log(fruits); *// [ 'apple', 'orange', 'banana', <7 empty items>, 'strawberry' ]* |
| --- |

## **Pridėti nauja elementa (add)**

Nors pridėti naujus elementus mes galime tiesiogiai nurodant reikšmę elementui kurio indeksas array.length + 1, tačiau teisingas būdas tai padaryti yra panaudojus du masyvo bazinius metodus. **push()** ir **unshift()**.

**push**() metodas prideda elementa į masyvo pabaigą.

| let dailyActivities = ['eat', 'sleep'];  *// add an element at the end* dailyActivities.push('exercise');  console.log(dailyActivities); *// ['eat', 'sleep', 'exercise']* |
| --- |

**unshift**() metodas prideda elementa į masyvo pradžią ir padaro tai pirmu elementu liste.

| let dailyActivities = ['eat', 'sleep'];  *//add an element at the start* dailyActivities.unshift('work');   console.log(dailyActivities); *// ['work', 'eat', 'sleep']* |
| --- |

Kaip ir minėta, elementus galima pridėti ir priskiriant reikšmę aukštesniam indeksui. **Šio metodo stenkitės ne naudoti.**

| let dailyActivities = [ 'eat', 'sleep'];  *// this will add the new element 'exercise' at the 2 index* dailyActivities[2] = 'exercise';  console.log(dailyActivities); *// ['eat', 'sleep', 'exercise']* |
| --- |

## 

## **Ištrinti elementa**

Norėdami ištrinti elementus iš masyvo galima naudoti bazinius masyvo metodus **pop()** ir **shift()**.

**pop**() metodas ištrina paskutinį masyvo elementa, modifikuoja masyva, ir kaip rezultatą (return value) gražina ištrinta elementa.

| let dailyActivities = ['work', 'eat', 'sleep', 'exercise'];  *// šitas veiksmas ištrina paskutnį masyvo narį* dailyActivities.pop(); console.log(dailyActivities); *// ['work', 'eat', 'sleep']*  *// dailyActivities.pop() ištrina paskytinį dailyActivities masyvo narį ir ištrinta narį grąžina kaip reikšmę, kurią priskiriama kintamajam.*  const removedElement = dailyActivities.pop();  console.log(removedElement); *// 'sleep'* console.log(dailyActivities); *// ['work', 'eat']* |
| --- |

**shift**() metodas ištrina pirma masyvo elementa, modifikuoja patį masyva, ir kaip rezultatą (return value) gražina ištrinta elementa.

| let dailyActivities = ['work', 'eat', 'sleep'];  *// ištrina pirma elementa* dailyActivities.shift();  console.log(dailyActivities); *// ['eat', 'sleep']* |
| --- |

## **Masyvo ilgis length**

| const dailyActivities = [ 'eat', 'sleep'];  console.log(dailyActivities.length); *// 2* |
| --- |

**Masyvo metodai**

## **Modifikavimo metodai**

## **Metodas 1: push()**

Prideda nauja elementa į masyvo pabaigą.

**Metodas 2: unshift()**

Prideda nauja elementa į masyvo pradžią.

## **Metodas 3: pop()**

Ištrina paskutinį masyvo elementa.

## **Metodas 4: shift()**

Ištrina pirma masyvo elementa.

## **Metodas 5: concat()**

Metodas skirtas sudėti du arba daugiau masyvų į viena masyvą.

**Sintaksė:**

| **array.concat**() **array.concat**(value0) **array.concat**(value0, value1) **array.concat**(value0, value1, ... , valueN) |
| --- |

Pvz:

| const array1 = ['a', 'b', 'c']; const array2 = ['d', 'e', 'f']; const array3 = array1.concat(array2);  console.log(array3); *// expected output: Array ["a", "b", "c", "d", "e", "f"]* |
| --- |

## **Metodas 6: reverse()**

Metodas kuris masyvo narių eiliškumą apsuka atvirkščiai. Modifikuoja originalų masyvą.

| const array1 = ['one', 'two', 'three']; console.log('array1:', array1); *// expected output: "array1:" Array ["one", "two", "three"]*  const reversed = array1.reverse(); console.log('reversed:', reversed); *// expected output: "reversed:" Array ["three", "two", "one"]* |
| --- |

## **Metodas 7: splice()**

Metodas kuris keičia turinį egzistuojančio masyvo, ištrindamas ar pekeičiantis esančius elementus arba pridėdamas naujus. Jis gali pasiekti dalį masyvo ir jį modifikuoti.

Grąžinama reikšmė yra masyvas, su ištrintais elementais. Jeigu buvo ištrintas tik vienas narys, tai masyvas su šiuo vienu elementu ir bus gra=intas. Jei nei vienas elementas buvo ištrintas, tai grąžinamas tuščias array.

**Sintaksė:**

| **array.splice**(start) **array.splice**(start, deleteCount) **array.splice**(start, deleteCount, newItem1) **array.splice**(start, deleteCount, newItem1, newItem2 valueN) |
| --- |

Pvz:

| const months = ['Jan', 'March', 'April', 'June']; months.splice(1, 0, 'Feb'); *// inserts at index 1* console.log(months); *// expected output: Array ["Jan", "Feb", "March", "April", "June"]*  months.splice(4, 1, 'May'); *// replaces 1 element at index 4* console.log(months); *// expected output: Array ["Jan", "Feb", "March", "April", "May"]* |
| --- |

| const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.splice(2, 0, "Lemon", "Kiwi");  console.log(fruits); *// [ 'Banana', 'Orange', 'Lemon', 'Kiwi', 'Apple', 'Mango' ]* |
| --- |

## **Metodas 8: slice()**

Metodas kuris grąžina paviršutiniška masyvo kopija, kaip naują masyvą, nemodifikuojamas senosios kopijos. Jei metodas kviečiamas be parametrų, tuomet gražina paviršutiniška kopija visų narių.

**Sintaksė:**

| **array.slice**() **array.slice**(start) **array.slice**(start, end) |
| --- |

Pvz:

| const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];  console.log(animals.slice(2)); *// expected output: Array ["camel", "duck", "elephant"]*  console.log(animals.slice(2, 4)); *// expected output: Array ["camel", "duck"]*  console.log(animals.slice(1, 5)); *// expected output: Array ["bison", "camel", "duck", "elephant"]*  console.log(animals.slice(-2)); *// expected output: Array ["duck", "elephant"]*  console.log(animals.slice(2, -1)); *// expected output: Array ["camel", "duck"]* |
| --- |

## **Metodas 9: join()**

Metodas kuris sukuria ir gražina string duomenų tipą, sujungdamas juos į viena, atskirdamas kableliais, arba atskyrimo simboliu, jei jis buvo nurodytas kaip argumentas.

**Sintaksė:**

| **array.join**() **array.join**(separator) |
| --- |

Pvz:

| const elements = ['Fire', 'Air', 'Water'];  console.log(elements.join()); *// expected output: "Fire,Air,Water"*  console.log(elements.join('')); *// expected output: "FireAirWater"*  console.log(elements.join('-')); *// expected output: "Fire-Air-Water"* |
| --- |

## **Rūšiavimo metodai**

## **Metodas 9: sort()**

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort>

Metodas kuris rūšiuoja masyvo elementus [pagal eiliškumą ir grąžina išrušiuotą masyvą.

**Sintaksė:**

| *// Functionless* **array.sort**() *// Arrow function* **array.sort**((firstEl, secondEl) => { */\* ... \*/* } ) *// Compare function* **array.sort**(compareFn) *// Inline compare function* **array.sort**(function compareFn(firstEl, secondEl) { /\* ... \*/ }) |
| --- |

Pvz:

| const months = ['March', 'Jan', 'Feb', 'Dec']; months.sort(); console.log(months); *// expected output: Array ["Dec", "Feb", "Jan", "March"]*  const array1 = [1, 30, 4, 21, 100000]; array1.sort(); console.log(array1); *// expected output: Array [1, 100000, 21, 30, 4]* |
| --- |

| function compare(a, b) {  if (a is less than b by some ordering criterion) {  return -1;  }  if (a is greater than b by the ordering criterion) {  return 1;  }  *// a must be equal to b*  return 0; } |
| --- |

| function compareNumbers(a, b) {  return a - b; } |
| --- |

| var numbers = [4, 2, 5, 1, 3]; numbers.sort(function(a, b) {  return a - b; }); console.log(numbers);  *// [1, 2, 3, 4, 5]* |
| --- |

| let numbers = [4, 2, 5, 1, 3]; numbers.sort((a, b) => a - b); console.log(numbers);  *// [1, 2, 3, 4, 5]* |
| --- |

| var items = [  { name: 'Edward', value: 21 },  { name: 'Sharpe', value: 37 },  { name: 'And', value: 45 },  { name: 'The', value: -12 },  { name: 'Magnetic', value: 13 },  { name: 'Zeros', value: 37 } ];  *// sort by value* items.sort(function (a, b) {  return a.value - b.value; });  *// sort by name* items.sort(function(a, b) {  var nameA = a.name.toUpperCase(); *// ignore upper and lowercase*  var nameB = b.name.toUpperCase(); *// ignore upper and lowercase*  if (nameA < nameB) {  return -1;  }  if (nameA > nameB) {  return 1;  }   *// names must be equal*  return 0; }); |
| --- |

Pvz:

| let stringArray = ['Blue', 'Humpback', 'Beluga']; let numericStringArray = ['80', '9', '700']; let numberArray = [40, 1, 5, 200]; let mixedNumericArray = ['80', '9', '700', 40, 1, 5, 200];  function compareNumbers(a, b) {  return a - b; }  stringArray.join(); *// 'Blue,Humpback,Beluga'* stringArray.sort(); *// ['Beluga', 'Blue', 'Humpback']*  numberArray.join(); *// '40,1,5,200'* numberArray.sort(); *// [1, 200, 40, 5]* numberArray.sort(compareNumbers); *// [1, 5, 40, 200]*  numericStringArray.join(); *// '80,9,700'* numericStringArray.sort(); *// [700, 80, 9]* numericStringArray.sort(compareNumbers); *// [9, 80, 700]*  mixedNumericArray.join(); *// '80,9,700,40,1,5,200'* mixedNumericArray.sort(); *// [1, 200, 40, 5, 700, 80, 9]* mixedNumericArray.sort(compareNumbers); *// [1, 5, 9, 40, 80, 200, 700]* |
| --- |

## **Ciklų metodai**

## **Metodas 10: forEach()**

Metodas kuris šaukia callback funkcija ant kiekvieno elemento, ir tos funkcijos pirmas parametras yra pats elementas. Antras šaukiamo elemento indeksas ir trečias pats masyvas.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach>

**Sintaksė:**

| ***// Arrow function* array.forEach((element) => { */\* ... \*/* } ) array.forEach((element, index) => { */\* ... \*/* } ) array.forEach((element, index, array) => { */\* ... \*/* } )  *// Callback function* array.forEach(callbackFn) array.forEach(callbackFn, thisArg)  *// Inline callback function* array.forEach(function(element) { */\* ... \*/* }) array.forEach(function(element, index) { */\* ... \*/* }) array.forEach(function(element, index, array){ */\* ... \*/* }) array.forEach(function(element, index, array) { */\* ... \*/* }, thisArg)** |
| --- |

Pvz:

| const numbers = [45, 4, 9, 16, 25]; let txt = ""; numbers.forEach(myFunction);  function myFunction(value, index, array) {  txt += value + "<br>"; } |
| --- |

## **Metodas 10: map()**

Metodas kuris iteruoja per kiekviena masyvo elementa ir pakeičia egzistuojantį elementa callback funkcijos grąžinama reikšmė.

**Sintaksė:**

| *// Arrow function* **array.map**((element) => { */\* ... \*/* }) **array.map**((element, index) => { */\* ... \*/* }) **array.map**((element, index, array) => { */\* ... \*/* })  *// Callback function* **array.map**(callbackFn) **array.map**(callbackFn, thisArg)  *// Inline callback function* **array.map**(function(element) { */\* ... \*/* }) **array.map**(function(element, index) { */\* ... \*/* }) **array.map**(function(element, index, array){ */\* ... \*/* }) **array.map**(function(element, index, array) { */\* ... \*/* }, thisArg) |
| --- |

Pvz:

| const numbers1 = [45, 4, 9, 16, 25]; const numbers2 = numbers1.map(myFunction);  function myFunction(value, index, array) {  return value \* 2; } |
| --- |

## **Metodas 11: filter()**

Metodas kuris grąžina nauja masyva, kuris atitinka callback funkcijos testa ar sąlygą.

**Sintaksė:**

| *// Arrow function* **array.filter**((element) => { */\* ... \*/* }) **array.filter**((element, index) => { */\* ... \*/* }) **array.filter**((element, index, array) => { */\* ... \*/* })  *// Callback function* **array.filter**(callbackFn) **array.filter**(callbackFn, thisArg)  *// Inline callback function* **array.filter**(function(element) { */\* ... \*/* }) **array.filter**(function(element, index) { */\* ... \*/* }) **array.filter**(function(element, index, array){ */\* ... \*/* }) **array.filter**(function(element, index, array) { */\* ... \*/* }, thisArg) |
| --- |

Pvz:

| const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];  const result = words.filter(word => word.length > 6);  console.log(result); *// expected output: Array ["exuberant", "destruction", "present"]* |
| --- |

| const numbers = [45, 4, 9, 16, 25]; const over18 = numbers.filter(myFunction);  function myFunction(value) {  return value > 18; } |
| --- |

## **Metodas 12: reduce()**

Metodas kuris šaukia callback funkcija ant kiekvieno masyvo elemento ir sugeneruoja viena reikšmę, kurią gražina.

| *// Arrow function* **array.reduce**((previousValue, currentValue) => { */\* ... \*/* } ) **array.reduce**((previousValue, currentValue, currentIndex) => { */\* ... \*/* } ) **array.reduce**((previousValue, currentValue, currentIndex, array) => { */\* ... \*/* } ) **array.reduce**((previousValue, currentValue, currentIndex, array) => { */\* ... \*/* }, initialValue)  *// Callback function* **array.reduce**(callbackFn) **array.reduce**(callbackFn, initialValue)  *// Inline callback function* **array.reduce**(function(previousValue, currentValue) { */\* ... \*/* }) **array.reduce**(function(previousValue, currentValue, currentIndex) { */\* ... \*/* }) **array.reduce**(function(previousValue, currentValue, currentIndex, array) { */\* ... \*/* }) **array.reduce**(function(previousValue, currentValue, currentIndex, array) { */\* ... \*/* }, initialValue) |
| --- |

Pvz:

| const numbers = [45, 4, 9, 16, 25]; let sum = numbers.reduce(myFunction);  function myFunction(total, value, index, array) {  return total + value; } |
| --- |

## **Metodas 13: every()**

Metodas kuris patikrina ar kiekvienas masyvo elementas atitinka sąlygą.

| *// Arrow function* **array.every**((element) => { */\* ... \*/* } ) **array.every**((element, index) => { */\* ... \*/* } ) **array.every**((element, index, array) => { */\* ... \*/* } )  *// Callback function* **array.every**(callbackFn) **array.every**(callbackFn, thisArg)  *// Inline callback function* **array.every**(function(element) { */\* ... \*/* }) **array.every**(function(element, index) { */\* ... \*/* }) **array.every**(function(element, index, array){ */\* ... \*/* }) **array.every**(function(element, index, array) { */\* ... \*/* }, thisArg) |
| --- |

Pvz:

| const numbers = [45, 4, 9, 16, 25]; let allOver18 = numbers.every(myFunction);  function myFunction(value, index, array) {  return value > 18; } |
| --- |

## **Metodas 14: some()**

Metodas kuris patikrina ar bent vienas masyvo elementas atitinka sąlygą.

| *// Arrow function* **array.some**((element) => { */\* ... \*/* } ) **array.some**((element, index) => { */\* ... \*/* } ) **array.some**((element, index, array) => { */\* ... \*/* } )  *// Callback function* **array.some**(callbackFn) **array.some**(callbackFn, thisArg)  *// Inline callback function* **array.some**(function(element) { */\* ... \*/* }) **array.some**(function(element, index) { */\* ... \*/* }) **array.some**(function(element, index, array){ */\* ... \*/* }) **array.some**(function(element, index, array) { */\* ... \*/* }, thisArg) |
| --- |

Pvz:

| const numbers = [45, 4, 9, 16, 25]; let someOver18 = numbers.some(myFunction);  function myFunction(value, index, array) {  return value > 18; } |
| --- |

## **Paieškos metodai**

## **Metodas 15: indexOf()**

Metodas kuris grąžina pirmo rasto elemento indeksa. Indeksas elementus lygina su strict lygybę (===). Jei elemento nerado, tuomet gražina -1.

**Sintaksė:**

| **array.indexOf**(searchElement) **array.indexOf**(searchElement, fromIndex) |
| --- |

Pvz:

| const beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];  console.log(beasts.indexOf('bison')); *// expected output: 1*  *// start from index 2* console.log(beasts.indexOf('bison', 2)); *// expected output: 4*  console.log(beasts.indexOf('giraffe')); *// expected output: -1* |
| --- |

## **Metodas 16: find()**

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find>

Metodas kuris grąžina pirma masyvo narį kuris atitinka callback funkcijos sąlygą.

**Sintaksė:**

| *// Arrow function* **array.find**((element) => { */\* ... \*/* } ) **array.find**((element, index) => { */\* ... \*/* } ) **array.find**((element, index, array) => { */\* ... \*/* } )  *// Callback function* **array.find**(callbackFn) **array.find**(callbackFn, thisArg)  *// Inline callback function* **array.find**(function(element) { */\* ... \*/* }) **array.find**(function(element, index) { */\* ... \*/* }) **array.find**(function(element, index, array){ */\* ... \*/* }) find(function(element, index, array) { */\* ... \*/* }, thisArg) |
| --- |

Pvz:

| const numbers = [4, 9, 16, 25, 29]; let first = numbers.find(myFunction);  function myFunction(value, index, array) {  return value > 18; } |
| --- |

## **Metodas 17: includes()**

Metodas kuris pasako ar masyvas turi konkrečią reikšmę ar ne.

**Sintaksė:**

| **array.includes**(searchElement) **array.includes**(searchElement, fromIndex) |
| --- |

| const array1 = [1, 2, 3];  console.log(array1.includes(2)); *// expected output: true*  const pets = ['cat', 'dog', 'bat'];  console.log(pets.includes('cat')); *// expected output: true*  console.log(pets.includes('at')); *// expected output: false* |
| --- |