# **Funkcijos**

Functions

## Kas yra funkcija?

Javascript funkcijos yra kodo blokai kurie atlieka jūsų aprašytas komandas. Funkcija yra paleidžiama, kada kažkas ją pašaukia.

## Funkcijos apibrėžimas (function declaration)

Funkcija yra apibrėžiama su baziniu žodžiu function, po kurio seka funkcijos pavadinimas, ir riestiniai skliaustai.

Pvz: Javascript funkcija

| function manoFunkcija() {  *// kodo blokas kurį paleis, kada funkcija bus pašaukta* } |
| --- |

Į skliaustelius esančius po funkcijos pavadinimu galime apibrėžti funkcijos parametrus. Parametrai yra atskiriami kableliais. Funkcijos argumentai yra parametrų reikšmės kurias įgaus funkcija pašaukus. Viduje funkcijos kodo bloko, argumentai (parametrai) bus laikomi lokalūs kintamieji.

Pvz: Javascript funkcija su parametrais

| function manoFunkcija(p1, p2) {  return p1 \* p2; *// p1 ir p2 parametrai tampa lokalūs kintamieji. Kurių reikšmės perduodamos funkcija pašaukus* } |
| --- |

Funkcijų vardams galioja tokios pačios taisyklės kaip ir aprašant kintamuosius. Pavadinimuose gali prasidėti tik A-Z raidėmis arba  $ ir \_ simboliai. Po to gali sekti skaitmenys nuo 0 iki 9. Javascript kalboje paprastai yra naudojami camelCase vardai jei funkcijos pavadinimas susideda iš sudurtinių žodžių.

## Funkcijos pašaukimas (function call, invocation)

Funkcijos pašaukimas vykdomas kada kode rašomas funkcijos vardas ir skliausteliai. manoFunkcija(); Kuomet funkcija yra pašaukiama ji iškarto pradeda vykdyti kodo komandas kurias nurodėte kodo bloke aprašydami funkcija. Funkcija galima pašaukti kartu su argumentais, kurie naudojami kaip lokalūs kintamieji jūsų vykdomuose komandose. manoFunkcija(10, 20);

Pvz: Funkcijos pašaukimas perduodant reikšmių argumentus

| manoFunkcija(4, 2); manoFunkcija(10, 20); manoFunkcija(140, 2); |
| --- |

Jeigu aprašant funkcija nenurodėte jokių parametrų, tuomet funkcija šaukiama be parametrų. Tai dažniausiai daroma, jei funkcija naudoja globalius kintamuosius, arba kintamųjų nereikia visai.

Pvz: Funkcijos pašaukimas ne argumentų

| manoFunkcija(); |
| --- |

() yra vadinamas funkcijos pašaukimo operatorius. Į šio operatoriaus vidų yra rašomi parametrai jei juos norite naudoti.

Kada daromas funkcijų pašaukimas? Funkcijos pašaukimas gali būti vykdomas, kai internetinio puslapio vartotojas atlieka atitinkamus veiksmus. Galima pašaukti tiesiog kode. Arba funkcija gali pasišaukti automatiškai (su atitinkama sintakse). Funkcija gali pašaukti save iš funkcijos vidaus, tai vadinama rekursija.

## Funkcijos return reikšmė - (tai ką gražina funkcija)

Kada funkcija daro kodo komandų vykdyma, pačioje pabaigoje gali grąžinti priskirta reikšmę. Kuomet return sakinys yra pasiekiamas, kodo vykdymas funkcijos viduje yra stabdomas ir generuojama return reikšmė. Jei funkcijos reikšmėje yra matematinė operacija, ar kitos funkcijos šaukimas, tai prieš grąžinant, funkcija apskaičiuoja visų vykdomų operacijų reikšmes ir gražina galutinę reikšmę.

Pvz: Funkcijos grąžinama reikšmė

| function manoFunkcija(p1, p2) {  return p1 \* p2;  }  manoFunkcija(2,3); *// Funkcijos grąžinama reikšmė lygi 6* |
| --- |

Pvz: Funkcijos reikšmę galime priskirti kintamajam.

| function manoFunkcija(p1, p2) {  return p1 \* p2;  }  const x = manoFunkcija(10, 2);  *// x reikšmė bus tai ką grąžins manoFunkcija(10, 2), pagal funkcijos aprašymą galim sakyti, kad tai bus 20.* |
| --- |

| function toCelsius(fahrenheit) {  return (5/9) \* (fahrenheit-32); }  let x = toCelsius(200); *// 93.33333333333334* let text = "The temperature is " + x + " Celsius"; *// "The temperature 93.33333333333334 Celsius"* |
| --- |

Funkcija gali grąžinti naujai sukurta funkciją, kurią galima iššaukti vėliau.

Pvz: Funkcija kaip reikšmę gražina kita funkcija

| function sumFactory() {  return function(a, b) {  return a \* b;  } }  const sum = sumFactory();  sum(2, 3); *// 6* sum(3, 5); *// 15* |
| --- |

## Kintamieji funkcijoje

Funkcijoje sukurti kintamieji yra vadinami lokalūs kintamieji (local) ir iš išorės nėra pasiekiami.

| *// šioje vietoje, kodas NEGALI naudoti carName*  function myFunction() {  let carName = "Volvo";  *// šioje vietoje, kodas GALI naudoti carName* }  *// šioje vietoje, kodas NEGALI naudoti carName* |
| --- |

Kadangi kintamieji funkcijos viduje yra pasiekiami tiktai iš funkcijos vidaus, skirtingose funkcijos aprašymuose mes galime kurti kintamuosius su tais pačiais pavadinimais. Lokalūs kintamieji yra sukuriami kada funkcija pradės vykdyti koda ir iš karto bus ištrinti, kada kodo vykdymas bus įvykdytas.

## Pirmos klasės funkcijos

Programavimo kalboje, pirmos klasės funkcijos yra laikomos funkcijos kurios atitinka tris kriterijus.

1. Funkcija galima priskirti kintamajam.
2. Funkcija galima perduoti kaip parametrą kitoje funkcijoje.
3. Funkcija galima grąžinti kaip reikšmę kitoje funkcijoje.

#### Funkcija kaip kintamasis, (function expression)

Pvz: Apibrėžta funkcija priskiriam kintamajam, ir kintamąjį galime šaukti kaip nauja funkcija

| function summation(a, b) {  return a + b; }  const sum = summation;  sum(5, 4); *// 9* sum(2, 2); *// 4* |
| --- |

Pvz: Kintamajam galima priskirti anonimine funkcija, ir naujai sukurta kintamąjį šaukti kaip nauja funkcija.

| const sum = function() {  return a + b; }  sum(6, 9); *// 15* sum(4, 5); *// 9* |
| --- |

#### Funkcija kaip parametras kitai funkcijai

Pvz: Funkcija galima perduoti kaip parametra kitai funkcijai. Toks funkcijos parametras vadinamas callback funkcija.

| function createPriceWidgetHtml(priceValue) {  const html = `  <div class="price-block">  <h2 class="price-title">Product Price</h2>  <p class="price-value">${priceValue} EUR</p>  </div>  `;   return html; }  function totalPriceCalculator(price, taxRate, callbackFunction) {  let totalPrice = price + (price \* taxRate);  return callbackFunction(totalPrice); }  const priceHTML = totalPriceCalculator(100, 0.21, createPriceWidgetHtml);  document.getElementById("demo").innerHTML = priceHTML; |
| --- |

#### Funkcijos grąžinama reikšmė kita funkcija

Pvz: Funkcija gražina kita funkcija

| function sumFactory(a) {  return function(b) {  return a + b;  } }  const sumWithTwo = sumFactory(2);   sumWithTwo(3); *// 5* sumWithTwo(5); *// 7* |
| --- |

## 

## IIFE Funkcijos - (Immediately Invoked Function Expression)

Pvz: Iškarto išsišaukianti funkcija

| *// IIFE* (function selfInvokingFunction() {  console.log(  "Function buvo iskviesta"  ); })(); |
| --- |

Tokia funkcija pradės vykdyti savo koda, iškarto, nuo pat jos aprašymo.

## Globalios Funkcijos

Kadangi Javascript kalba leidžia naudotis globaliai aprašytomis funkcijomis, kurias galime pasiekti iš bet kurios kodo vietos. Šios funkcijos sukurtus patogumo tikslams, kurios gali atlikti sudėtingas operacijas arba sąveikauti su sudėtingomis ir žemesnio lygio API’s.

Globalios funkcijos

| *// Atidės funkcijos paleidimą*  setTimeout(callBack, 2000); *// callback yra jūsų aprašyta funkcija kurią norite kad*  *// Funkcija kuri šaukia jūsų funkcija nurodytu intervalu, norint sustabdyti, reikia pašaukti kita funkcija.* setInterval(callBack, 1000);   *// Funkcija kuri paverčia string i number su float point* parseFloat();   *// Funkcija kuri paverčia string i integer* parseInteger();   *// Funkcija kuri patikrina ar reikšmė yra NaN* isNaN();  *// Funkcija kuri pavers objektą į skaičių* Number();  *// Funkcija kuri pavers objekto reikšmę į string* String()  *// Funkcija kuri interpretuos string kaip Javascript koda.*  eval(); |
| --- |

### Funkcijos naudojamos tam, kad aprašomos viena kartą ir gali būti šaukiamos daug, tiek kiek nori ir kada nori. Dėl galimybės nurodyti parametrus, funkcija galime šaukti su skirtingomis argumentų reikšmėmis ir gauti skirtingas return reikšmes.

## Function hoisting (Funkciju deklaracijos perkėlimas į viršų)

Funkciju kėlimas yra bazinė javascript elgsena kai funkcijos deklaracija yra pakeliama į viršų esame scope kontekste.

Pvz: Apačioje esantis pavyzdys yra teisingas dėl - Hoisting, funkcijos deklaracija perkeliama į viršų, kodo interpretavimo momente, nors aprašant koda, funkcijos deklaracija parašyta žemiau, negu pats funkcijos šaukimas.

| myFunction(5);  function myFunction(y) {  return y \* y; } |
| --- |

## Funkcijos yra Objektai

Nors typeof operatorius gražina 'object' tačiau funkcija yra laikoma objektu ir turi labai daug objekto savybių. Funkcijos turi tiek atributus tiek metodos, kaip ir visi kiti objektai Javascript kalboje.

| function myFunction(a, b) {  return arguments.length; } |
| --- |

| function myFunction(a, b) {  return a \* b; }  let text = myFunction.toString(); |
| --- |

## Funkcijos .call(), .apply(), .bind() metodai.

// informacija bus pridėta