# **String**

Tekstas

## **Sukurti string**

Javascript string yra viskas kas parašyta tarp dviejų kabučių, įskaitant tuščia tarpą, simbolius, arba tuščias kabutes.

Sukurti string reikšmę galima panaudoti trijų tipų kabutes. Viengubas kabutes - '', dvigubas kabutes - "", ir “nugarines kabutes” - ``.

| const string1 = "string primitive"; const string2 = 'Irgi string primitive'; const string3 = `Dar vienas string primitive`; |
| --- |

Visi būdai yra teisingi, tačiau patys populiariausi yra viengubos ar dvigubos kabutės. Pasirinkti kurias naudoti, reikia nuo jūsų kompanijos arba komandos vidinių susitarimų.

Tokiu būdu sukurtas string yra vadinamas string primitive, t.y. primityvus duomenu tipas. String galima kurti ir kaip objektą.

| const string4 = new String("A String object"); |
| --- |

Toks string kūrimas sugrąžins string objektą, Šitas konceptas Javascript paprastai nėra naudojamas, tiesiog žinokite, kad toks yra.

Kadangi string duomenų tipas techniškai yra objektas, todėl sukūrus naują string jis iš karto paveldi bazinius atributus ir metodus, kuriuos panaudojus ant šito duomenų tipo, galima atlikti skirtingas operacijas.

## **Pasiekti teksto simbolius (character)**

Javascript yra du būdai pasiekti teksto simbolius:

Pvz. charAt funkcija gražina

| return 'cat'.charAt(0) *// returns "c"*  return 'cat'.charAt(1) *// returns "a"* |
| --- |

Kitas būdas nurodyti charakterio indeksa.

| return 'cat'[1] *// returns "a"* |
| --- |

Naudojant laužtinių skliaustų notaciją galima tik pasiekti teksto simbolį, tačiau užrašyti naują reikšmę, ištrinti negalima.

**Lyginti string tarpusavio sąlygose**

| let a = 'a' let b = 'b' if (a < b) { *// if true*  console.log(a + ' yra mažiau negu ' + b); } |
| --- |

**String length (ilgis)**

| let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; let length = text.length; *// 26* |
| --- |

**Escape simboliai**

| let text = 'I\'m person'; |
| --- |

| Code Result Description \' ' Single quote \" " Double quote \\ \ Backslash |
| --- |

| let text = "We are the so-called \"Vikings\" from the north."; |
| --- |

**Breaking Long Code lines**

| let text = "Hello \ Dolly!"; |
| --- |

| let text = "Hello " + "Dolly!"; |
| --- |

| let text = \ // negali buti tarpo po sito simbolio "Hello Dolly!"; |
| --- |

**String Templates**

| let firstName = "John"; let lastName = "Doe";  let text = `Welcome ${firstName}, ${lastName}!`; |
| --- |

**Multiline String**

| let text = `The quick brown fox jumps over the lazy dog`; |
| --- |

| let price = 10; let VAT = 0.25;  let total = `Total: ${(price \* (1 + VAT)).toFixed(2)}`; |
| --- |

**HTML**

| let header = "Templates Literals"; let tags = ["template literals", "javascript", "es6"];  let html = `<h2>${header}</h2><ul>`; for (const x of tags) {  html += `<li>${x}</li>`; }  html += `</ul>`; |
| --- |

**String built-in funkcijos**

| **Metodas** | **Aprašymas** |
| --- | --- |
| charAt() | Grąžins simbolį nurodytam indekse |
| concat() | Sujungia du arba daugiau string ir gražina nauja sujungta |
| endsWith() | Patikrina ar string baigiasi nurodytu string/simboliu |
| indexOf() | Grąžina pirma pasitaikiusį atitikmens indeksa |
| lastIndexOf() | Grąžina paskutinį pasitaikiusį atitikmens indeksa |
| match() | Ieško string atitikimo pagal paduotą reguliariaja ekspresija ir gražina atitikmenį. |
| repeat() | Grąžina ta patį string tiek kartų kiek buvo nurodyta argumente. |
| replace() | Ieško nurodyto string arba reguliarios ekspresijos ir jei randa atitikmenį pakeičia į nurodyta. |
| search() | Ieško nurodyto string arba reguliarios ekspresijos ir jei randa gražina pozisijos skaičių |
| slice() | Iškerpa dalį string ir gražina nauja stringa |
| split() | Išskaldo string į masyva nurodyto substrings |
| startsWith() | Patikrina ar string prasideda su nurodytu string |
| substring() | Grąžina sub string tarp nurodytos indekso pradžios ir pabaigos |
| toLowerCase() | Konvertuoja string simbolius į mažąsias raides |
| toString() | Konvertuoja reikšmę į string |
| toUpperCase() | Konvertuoja visus string simbolius į didžiąsias raides |
| trim() | Ištrina tuščias vietas string priekyje ir pabaigoje. |

**String metodu pavyzdžiai**

## **Metodas 1: charAt()**

Pvz:

| return 'cat'.charAt(1) *// returns "a"* |
| --- |

## 

## **Metodas 2**: **concat()**

Pvz:

| concat(str1) concat(str1, str2) concat(str1, str2, ... , strN) |
| --- |

| console.log(str2.concat(', ', str1)); *// "World, Hello"* |
| --- |

**Metodas 3: indexOf()**

Pvz:

| const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was it really lazy?';  const searchTerm = 'dog'; const indexOfFirst = paragraph.indexOf(searchTerm); |
| --- |

| const url = '/news/articles/2021-12-15?colorTheme=dark' *// Ieškome frazės sakinyje, šiuo atveju ieškome ar url turi atitinkama parametra* if (url3.indexOf('colorTheme=dark') !== -1) {  console.log('Page theme is dark') } |
| --- |

**Metodas 4: search()**

Šitas metodas yra labai panašus kaip indexOf(), skirtumas tik tas, kad jis ima regular regex kaip parametra, kai tuo tarpu indexOf() priima kita string. Jeigu mes įdėsime tiesiog string šaukdami šita funkcija, string reikšmė bus paversta į Regex automatiškai.

Pvz:

| const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was it really lazy?';  *// any character that is not a word character or whitespace* const regex = /[^\w\s]/g;  console.log(paragraph.search(regex)); *// expected output: 43*  console.log(paragraph[paragraph.search(regex)]); *// expected output: "."* |
| --- |

**Metodas 5: lastIndexOf()**

**Sintaksė:**

| **lastIndexOf(searchValue)**  **lastIndexOf(searchValue, fromIndex)** |
| --- |

Pvz:

| **'canal'.lastIndexOf('a'); *// returns 3*** |
| --- |

**Metodas 6: repeat()**

Pvz:

| const chorus = 'Because I\'m happy. ';  console.log(`Chorus lyrics for "Happy": ${**chorus**.**repeat**(27)}`);  *// expected output: "Chorus lyrics for "Happy": Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. Because I'm happy. "* |
| --- |

**Metodas 7: match()**

Pvz:

| **const paragraph = 'The quick brown fox jumps over the lazy dog. It barked.'; const regex = /[A-Z]/g; const found = paragraph.match(regex);**  **console.log(found); *// expected output: Array ["T", "I"]*** |
| --- |

| const str = 'For more information, see Chapter 3.4.5.1'; const re = /see (chapter \d+(\.\d)\*)/i; const found = str.match(re);  console.log(found);  *// logs [ 'see Chapter 3.4.5.1',* *// 'Chapter 3.4.5.1',* *// '.1',* *// index: 22,* *// input: 'For more information, see Chapter 3.4.5.1' ]* |
| --- |

Priklausomai nuo regex -g flag parametro ši funkcija grąžina skirtingus rezultatus. Jeigu yra nurodytas -g parametras, tuomet gražina masyvą su rastais atitikmenimis. Jei nurodytas nėra tuomet gražina masyva su vienu elementu, ir papildomais atributais. Groups, index, input. Tačiau tai vis tiek yra masyvas ir rasta elementą galima pasiekti per [0] atributą.

**Metodas 8: replace()**

**Sintaksė:**

| **replace(regexp, newSubstr) replace(regexp, replacerFunction) replace(substr, newSubstr) replace(substr, replacerFunction)** |
| --- |

Pvz:

| const p = 'The quick brown fox jumps over the lazy dog. If the dog reacted, was it really lazy?';  console.log(p.replace('dog', 'monkey')); *// expected output: "The quick brown fox jumps over the lazy monkey. If the dog reacted, was it really lazy?"* |
| --- |

| const regex = /Dog/i; console.log(p.replace(regex, 'ferret')); *// expected output: "The quick brown fox jumps over the lazy ferret. If the dog reacted, was it really lazy?"* |
| --- |

**Metodas 9: slice()**

**Sintaksė:**

| **slice(beginIndex) slice(beginIndex, endIndex)** |
| --- |

Pvz:

| **const str = 'The quick brown fox jumps over the lazy dog.';  console.log(str.slice(31)); *// expected output: "the lazy dog."*  console.log(str.slice(4, 19)); *// expected output: "quick brown fox"*  console.log(str.slice(-4)); *// expected output: "dog."*  console.log(str.slice(-9, -5)); *// expected output: "lazy"*** |
| --- |

**Metodas 10: substring()**

### [Differences between substring() and slice()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/substring#differences_between_substring_and_slice)

The substring() and slice() methods are almost identical, but there are a couple of subtle differences between the two, especially in the way negative arguments are dealt with.

The substring() method swaps its two arguments if indexStart is greater than indexEnd, meaning that a string is still returned. The slice() method returns an empty string if this is the case.

Pvz:

| const str = 'Mozilla';  console.log(str.substring(1, 3)); *// expected output: "oz"*  console.log(str.substring(2)); *// expected output: "zilla"* |
| --- |

**Metodas 11: split()**

Pvz:

| **split() split(separator) split(separator, limit)** |
| --- |

| const str = 'The quick brown fox jumps over the lazy dog.';  const words = str.split(' '); console.log(words); [ 'The',  'quick',  'brown',  'fox',  'jumps',  'over',  'the',  'lazy',  'dog.' ]   const chars = str.split(''); console.log(chars[8]); *// expected output: "k"*  const strCopy = str.split(); console.log(strCopy); *// expected output: Array ["The quick brown fox jumps over the lazy dog."]* |
| --- |

**Metodas 12: startsWith()**

Pvz:

| const str1 = 'Saturday night plans';  console.log(str1.startsWith('Sat')); *// expected output: true*  console.log(str1.startsWith('Sat', 3)); *// expected output: false* |
| --- |

**Metodas 13: endsWith()**

Pvz:

| const str1 = 'Cats are the best!';  console.log(str1.endsWith('best', 17)); *// expected output: true*  const str2 = 'Is this a question';  console.log(str2.endsWith('?')); *// expected output: false* |
| --- |

**Metodas 14: toLowerCase()**

Pvz:

| const sentence = 'The quick brown fox jumps over the lazy dog.';  console.log(sentence.**toLowerCase**()); *// expected output: "the quick brown fox jumps over the lazy dog."* |
| --- |

**Metodas 15: toUpperCase()**

Pvz:

| const sentence = 'The quick brown fox jumps over the lazy dog.';  console.log(sentence.toUpperCase()); *// expected output: "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG."* |
| --- |

**Metodas 16: trim()**

Pvz:

const greeting = ' Hello world! ';  
  
console.log(greeting);  
*// expected output: " Hello world! ";*  
  
console.log(greeting.**trim**());  
*// expected output: "Hello world!"*;

**Metodas 17: toString()**

Pvz:

| const stringObj = new String('foo');  console.log(stringObj); *// expected output: String { "foo" }*  console.log(stringObj.toString()); *// expected output: "foo"* |
| --- |