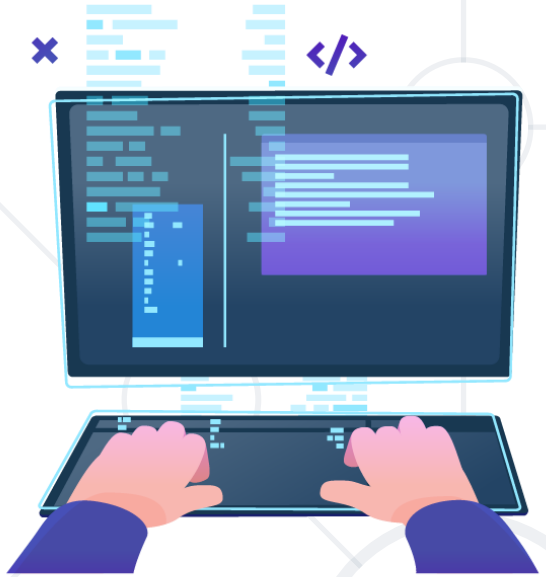
JS Arrays and Strings

Arrays and Strings



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#js-front-end

Table of Contents

Working with Arrays of Elements

Arrays in JavaScript

What is an Array?

- Arrays are **list-like objects**
- Arrays are a **reference type**, the variable points to an address in memory

- Elements are **numbered** from **0** to **length - 1**
- Creating an array using **an array literal**

```
let numbers = [10, 20, 30, 40, 50];
```

What is an Array?

- Neither the **length** of a JavaScript array **nor** the **types** of its elements are **fixed**
- An array's **length can be changed** at any time
- Data can be stored at non-contiguous locations in the array
- JavaScript arrays are not guaranteed to be dense

Arrays of Different Types

```
// Array holding numbers
let numbers = [10, 20, 30, 40, 50];
```

```
// Array holding strings
let weekDays = ['Monday', 'Tuesday', 'Wednesday',
   'Thursday', 'Friday', 'Saturday', 'Sunday'];
```

```
// Array holding mixed data (not a good practice)
let mixedArr = [20, new Date(), 'hello', {x:5, y:8}];
```

Accessing Elements

- Array elements are accessed using their **index**

```
let cars = ['BMW', 'Audi', 'Opel'];
let firstCar = cars[0];   // BMW
let lastCar = cars[cars.length - 1];  // Opel
```

- Accessing indexes that do not exist in the array returns **undefined**

```
console.log(cars[3]);   // undefined
console.log(cars[-1]);  // undefined
```

Destructuring Syntax

- Expression that **unpacks values** from **arrays** or **objects**, into distinct **variables**

```
let numbers = [10, 20, 30, 40, 50];
let [a, b, ...elems] = numbers;

console.log(a) // 10
console.log(b) // 20
console.log(elems) // [30, 40, 50]
```

Rest operator

- The **rest operator** can also be used to collect function parameters into an array

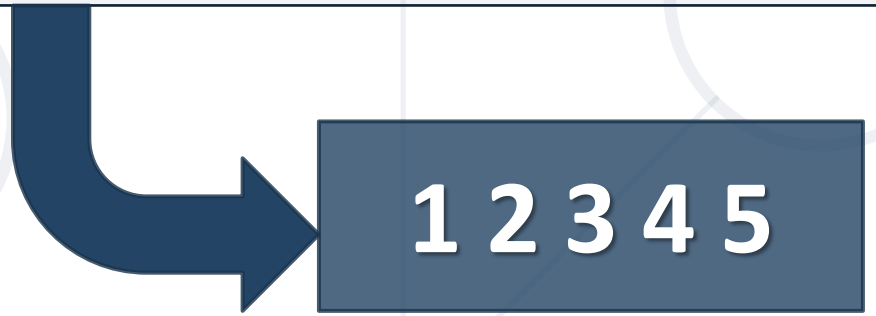
For-of Loop

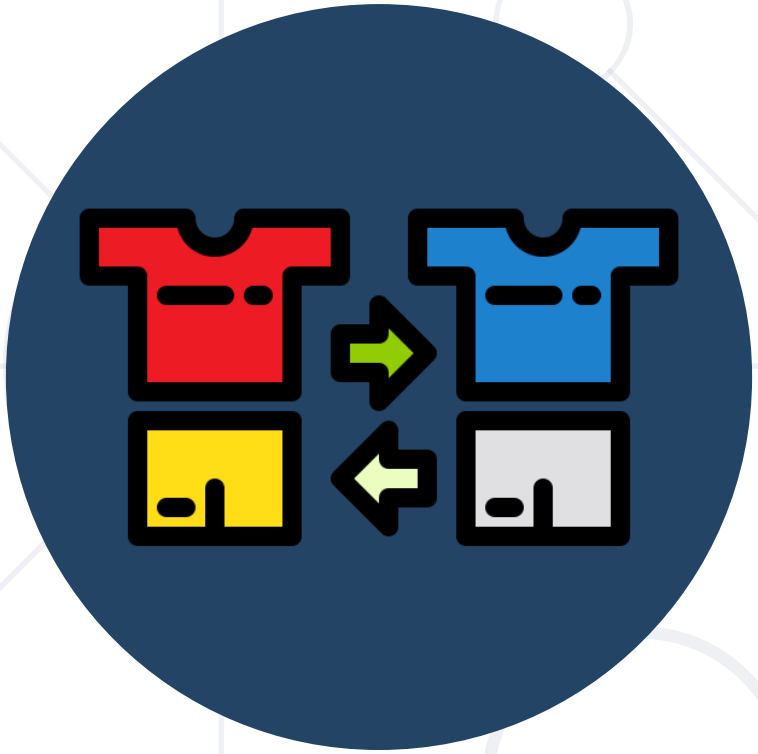
- Iterates through all **elements** in a collection
- Cannot access the current index

```
for (let el of collection) {
    // Process the value here
}
```

Print an Array with For-of

```
let numbers = [ 1, 2, 3, 4, 5 ];
let output = '';
for (let number of numbers)
    output += `${number} `;
console.log(output);
```





Methods

Modify the Array

Pop

- Removes the **last element** from an array and returns that element
- This method **changes** the **length** of the array

```
let nums = [10, 20, 30, 40, 50, 60, 70];
console.log(nums.length); // 7
console.log(nums.pop());  // 70
console.log(nums.length); // 6
console.log(nums);        // [ 10, 20, 30, 40, 50, 60 ]
```

Push

- The **push()** method **adds one or more** elements to the **end** of an array and **returns** the new **length** of the array

```
let nums = [10, 20, 30, 40, 50, 60, 70];
console.log(nums.length);   // 7
console.log(nums.push(80)); // 8 (nums.length)
console.log(nums); // [ 10, 20, 30, 40, 50, 60, 70, 80 ]
```

Shift

- The `shift()` method **removes** the **first element** from an array and **returns** that **removed element**
- This method **changes** the **length** of the array

```
let nums = [10, 20, 30, 40, 50, 60, 70];
console.log(nums.length); // 7
console.log(nums.shift()); // 10 (removed element)
console.log(nums);  // [ 20, 30, 40, 50, 60, 70 ]
```

Unshift

- The `unshift()` method **adds one or more** elements to the **beginning** of an array and **returns** the new **length** of the array

```
let nums = [40, 50, 60];
console.log(nums.length);          // 3
console.log(nums.unshift(30));  // 4 (nums.length)
console.log(nums.unshift(10,20)); // 6 (nums.length)
console.log(nums);  // [ 10, 20, 30, 40, 50, 60 ]
```

Splice

- Changes the contents of an array by **removing** or **replacing** existing **elements** and / or **adding new** elements

```
let nums = [1, 3, 4, 5, 6];
nums.splice(1, 0, 2); // inserts at index 1
console.log(nums); // [ 1, 2, 3, 4, 5, 6 ]
nums.splice(4, 1, 19); // replaces 1 element at index 4
console.log(nums); // [ 1, 2, 3, 4, 19, 6 ]
let el = nums.splice(2, 1); // removes 1 element at index 2
console.log(nums); // [ 1, 2, 4, 19, 6 ]
console.log(el); // [ 3 ]
```

Reverse

- Reverses the array
  - The **first** array **element becomes** the **last**, and the last array element becomes the first

```
let arr = [1, 2, 3, 4];
arr.reverse();
console.log(arr); // [ 4, 3, 2, 1 ]
```

Join

- Creates and returns a **new string** by **concatenating** all of the elements in an array (or an array-like object), **separated** by commas or a **specified separator** string

```
let elements = ['Fire', 'Air', 'Water'];
console.log(elements.join()); // "Fire,Air,Water"
console.log(elements.join('')); // "FireAirWater"
console.log(elements.join('-')); // "Fire-Air-Water"
console.log(['Fire'].join("."); // Fire
```

Slice

- The `slice()` method **returns** a shallow **copy** of a **portion** of an array into a **new array** object selected from begin to end (end not included)
- The **original array** will **not** be **modified**

```
let fruits = ['Banana', 'Orange', 'Lemon', 'Apple'];
let citrus = fruits.slice(1, 3);
let fruitsCopy = fruits.slice();
// fruits contains ['Banana', 'Orange', 'Lemon', 'Apple']
// citrus contains ['Orange', 'Lemon']
```

Includes

- Determines whether an array contains a certain element, returning **true** or **false** as appropriate

```
// array length is 3
// fromIndex is -100
// computed index is 3 + (-100) = -97
let arr = ['a', 'b', 'c'];
arr.includes('a', -100); // true
arr.includes('b', -100); // true
arr.includes('c', -100); // true
arr.includes('a', -2); // false
```

IndexOf

- The `indexOf()` method **returns** the **first index** at which a given **element** can be **found** in the array
  - Output is **-1** if element is **not present**

```
const beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];

console.log(beasts.indexOf('bison')); // 1
// start from index 2
console.log(beasts.indexOf('bison', 2)); // 4
console.log(beasts.indexOf('giraffe')); // -1
```

ForEach

- The `forEach()` method **executes a provided function** once for each array element
- Converting a for loop to forEach

```
const items = ['item1', 'item2', 'item3'];
const copy = [];

// For Loop
for (let i = 0; i < items.length; i++) {
  copy.push(items[i]);
}

// ForEach
items.forEach(item => { copy.push(item); });
```

Map

- **Creates a new array** with the results of calling a **provided function** on every element in the calling array

```
let numbers = [1, 4, 9];
let roots = numbers.map(function(num, i, arr) {
  return Math.sqrt(num)
});
// roots is now [1, 2, 3]
// numbers is still [1, 4, 9]
```

Find

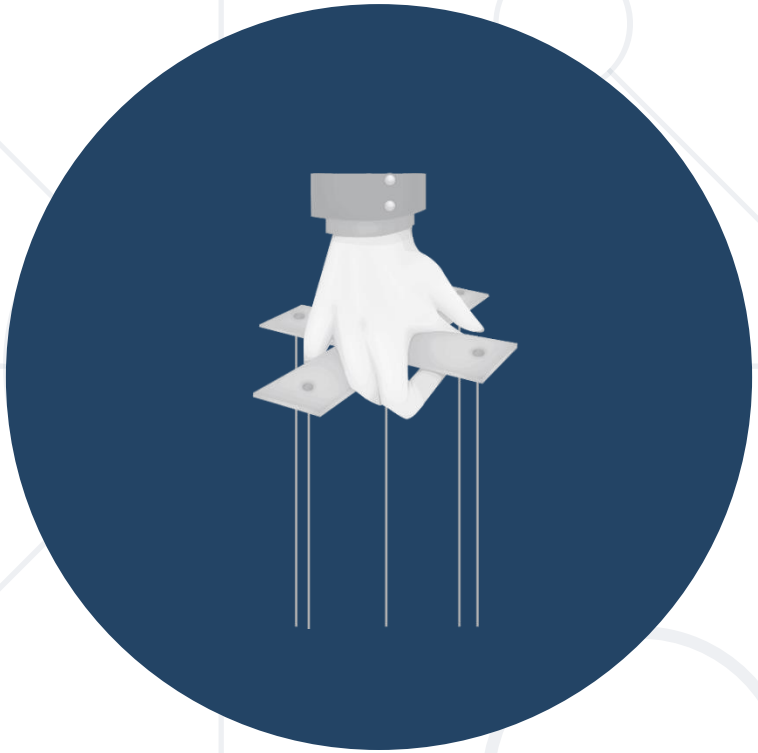
- Returns the **first found value** in the array, if an **element** in the array **satisfies** the **provided** testing **function** or **undefined** if not found

```
let array1 = [5, 12, 8, 130, 44];
let found = array1.find(function(element) {
  return element > 10;
});
console.log(found); // 12
```

Filter

- Creates a **new array** with **filtered elements only**
- Calls a **provided** callback **function** once for each element in an array
- **Does not mutate** the **array** on which it is called

```
let fruits = ['apple', 'banana', 'grapes', 'mango', 'orange'];
 // Filter array items based on search criteria (query)
function filterItems(arr, query) {
  return arr.filter(function(el) {
      return el.toLowerCase().indexOf(query.toLowerCase()) !== -1;
  });
};
console.log(filterItems(fruits, 'ap')); // ['apple', 'grapes']
```



Manipulating Strings

Concatenating

- Use the "+" or the "+=" operators

```
let text = "Hello" + ", ";
// Expected output: "Hello, "
text += "JS!"; // "Hello, JS!"
```

- Use the concat() method

```
let greet = "Hello, ";
let name = "John";
let result = greet.concat(name);
console.log(result); // Expected output: "Hello, John"
```

Searching for Substrings

- `indexOf(substr)`

```
let str = "I am JavaScript developer";
console.log(str.indexOf("Java")); // Expected output: 5
console.log(str.indexOf("java")); // Expected output: -1
```

- `lastIndexOf(substr)`

```
let str = "Intro to programming";
let last = str.lastIndexOf("o");
console.log(last); // Expected output: 11
```

Extracting Substrings

- `substring(startIndex, endIndex)`

```
let str = "I am JavaScript developer";
let sub = str.substring(5, 10);
console.log(sub); // Expected output: JavaS
```

String Operations

- `replace(search, replacement)`

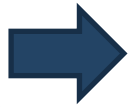
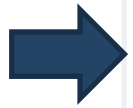
```
let text = "Hello, john@softuni.bg, you have been
using john@softuni.bg in your registration.";

let replacedText = text.replace(".bg", ".com");

console.log(replacedText);
// Hello, john@softuni.com, you have been using
john@softuni.bg in your registration.
```

Problem: Substring

- Receives a **string**, a **start index**, and **count** characters
- Print the **substring** of the received string

`"ASentence", 1, 8`		`Sentence`
`"JavaScript", 4, 6`		`Script`

Solution: Substring

```
function solve(text, startIndex, count) {
 let substring = text
     .substring(startIndex, startIndex + count);

 console.log(substring);
}
```

Splitting and Finding

- split(separator)

```
let text = "I love fruits";
let words = text.split(' ');
console.log(words); // Expected output: ['I', 'love', 'fruits']
```

- includes(substr)

```
let text = "I love fruits.";
console.log(text.includes("fruits")); // Expected output: True
console.log(text.includes("banana")); // Expected output: False
```

Repeating Strings

- `repeat(count)` - Creates a new string repeated count times

```
let n = 3;
for(let i = 1; i <= n; i++) {
  console.log('*'.repeat(i));
}
```

```
// *
// **
// ***
```

Problem: Censored Words

- Receives a **text** and a **single word**
- Find all **occurrences** of that word in the text and **replace** them with the corresponding amount of **'*'**

```
A small sentence with some words,
small
```

```
A ***** sentence with some words
```

Solution: Censored Words

```
function solve(text, word) {
 while (text.includes(word)) {
    text = text.replace(word, '*'.repeat(word.length));
 }
 console.log(text);
}
```

Trimming Strings

- Use `trim()` method to remove **whitespaces** (spaces, tabs, no-break space, etc.) from **both ends** of a string

```
let text = "   Annoying spaces      ";
console.log(text.trim()); // Expected output: "Annoying spaces"
```

- Use `trimStart()` or `trimEnd()` to remove whitespaces **only** at the beginning or at the end

```
let text = "   Annoying spaces      ";
text = text.trimStart(); text = text.trimEnd();
console.log(text); // Expected output: "Annoying spaces"
```

Starts With/Ends with

- Use **`startsWith()`** to determine whether a string **begins** with the characters of a specified substring

```
let text = "My name is John";
console.log(text.startsWith('My')); // Expected output: true
```

- Use **`endsWith()`** to determine whether a string **ends** with the characters of a specified substring

```
let text = "My name is John";
console.log(text.endsWith('John')); // Expected output: true
```

Padding at the Start and End

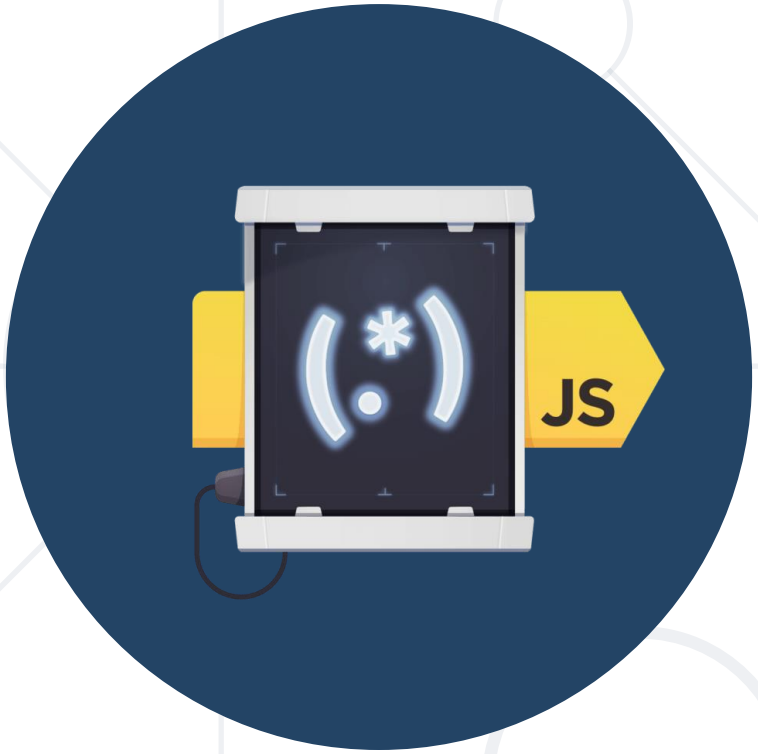
- Use **padStart()** to add to the current string **another substring** at the **start** until a **length** is reached

Receives **length** and **substring**

```
let text = "010";
console.log(text.padStart(8, '0')); // Expected output: 00000010
```

- Use **padEnd()** to add to the current string **another substring** at the **end** until a **length** is reached

```
let sentence = "He passed away";
console.log(sentence.padEnd(20, '.'));
// Expected output: He passed away......
```



Regular Expressions in JS

RegExp in JS

- In JS you construct a regular expression in one of two ways:
  - Regular Expression Literal
  - The constructor function **RegExp**

```
// Provides compilation when the script is loaded
let regLiteral = /[A-Za-z]+/g

// Provides runtime compilation
// Used when the pattern is from another source
let regExp = new RegExp('[A-Za-z]+', 'g');
```

Validating String by Pattern

- The method `test(string)`
  - Determines whether there is a match

```
let text = 'Today is 2015-05-11';
let regexp = /\d{4}-\d{2}-\d{2}/g;

let containsValidDate = regexp.test(text);
console.log(containsValidDate); // true
```

Checking for Matches

- The method **match(regexp)**
  - Returns an **array** of all matches (strings)

```
let text = 'Peter: 123 Mark: 456';
let regexp = /([A-Z][a-z]+): (\d+)/g;
let matches = text.match(regexp);

console.log(matches.length); // 2
console.log(matches[0]); // Peter: 123
console.log(matches[1]); // Mark: 456
```

Using the exec() Method

- The method **exec(string, text)**
  - Works with a pointer & returns the **groups**

```
const text = "Peter: 123 Mark: 456";
const regexp = /([A-Z][a-z]+): (\d+)/g;
const firstMatch = regexp.exec(text);

console.log(firstMatch[0]); // Peter: 123
console.log(firstMatch[1]); // Peter
console.log(firstMatch[2]); // 123
```

Replacing with RegExp

- The method `replace(regexp, stringReplacement)`
  - Replaces all strings that **match the pattern** with the provided replacement

```
const text = 'Peter: 123 Mark: 456';
const replacement = '999';
const regexp = /\d{3}/g;
const result = text.replace(regexp, replacement);
// Peter: 999 Mark: 999
```

MatchAll

- The method **matchAll(regexp)**
  - returns an iterator of all results matching a string against a **regular expression**, including **capturing groups**

```
const regexp = /t(e)(st(\d?))/g;
const str = 'test1test2';
const array = [...str.matchAll(regexp)];
console.log(array[0]);
// ['test1', 'e', 'st1', '1', index: 0, input:'test1test2',
length: 4]
```

Splitting with RegExp

- The method **split(regexp)**
  - Splits the text by the pattern
  - Returns an array of strings

```
let text = '1   2 3      4';
let regexp = /\s+/g;
let result = text.split(regexp);
console.log(result) // ['1', '2', '3', '4'];
```

Problem: Count String Occurrences

- Receive a **text** and a **word** that you need to **search**
- Find the number of **all occurrences** of that word and print it

```
"This is a word and it also is a sentence",
"is"
```

```
2
```

Solution: Count String Occurrences

```
function solve(text, search) {
  let words = text.split(' ');
  let counter = 0;
  for (let w of words) {
    if (w === search) {
      counter++;
    }
  }
  console.log(counter);
}
```

Summary

- Array
  - Methods
- Associative Array
- Text Processing
  - **Regular expressions** describe **patterns** for searching through text

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

© SoftUni – https://about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners

Software University

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg