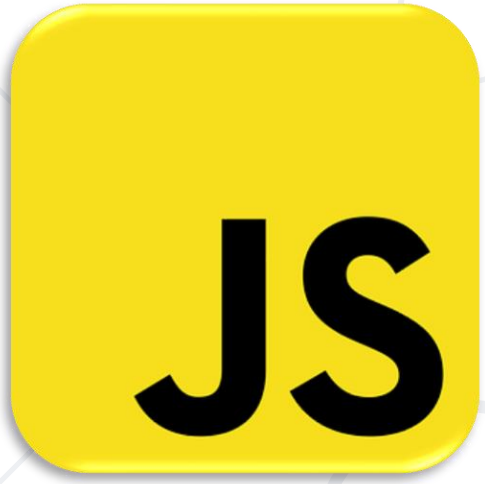
JavaScript Basics

Syntax, Data Type and Variables, Operators,
Conditional Statements, Loops, Debugging, Arrays



SoftUni Team

Technical Trainers

Software University

Software University

https://softuni.bg

You Have Questions?

Software University

sli.do

#QA-BackEnd

Table of Contents

JavaScript Overview

Definition, Execution, IDE Setup

What is JavaScript?

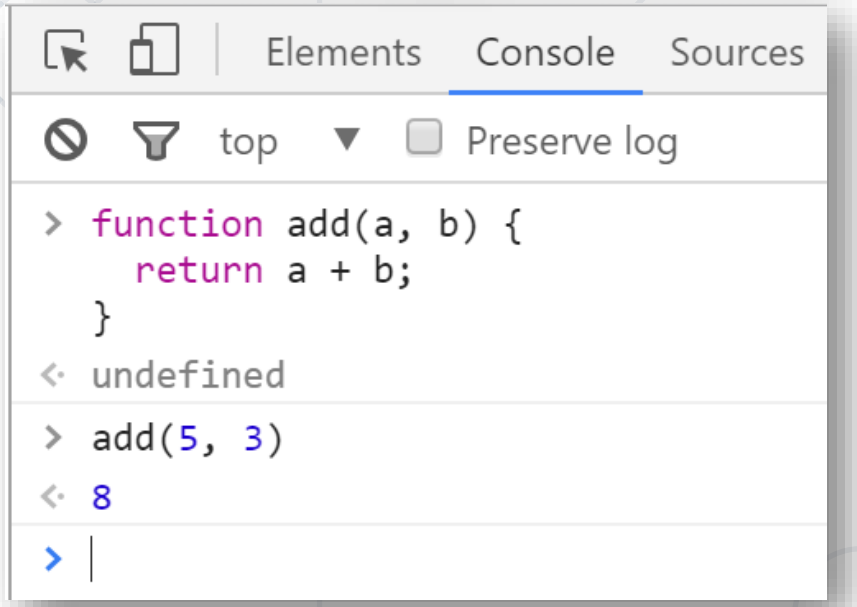
- JavaScript (**JS**) is a **high-level** programming language
  - One of the **core technologies** of the World Wide Web
  - Enables **interactive** web pages and applications
  - Can be **executed** on the **server** and on the **client**
- Features
  - C-like **syntax** (curly-brackets, identifiers, operator)
  - **Multi-paradigm** (imperative, functional, OOP)
  - Dynamic **typing**

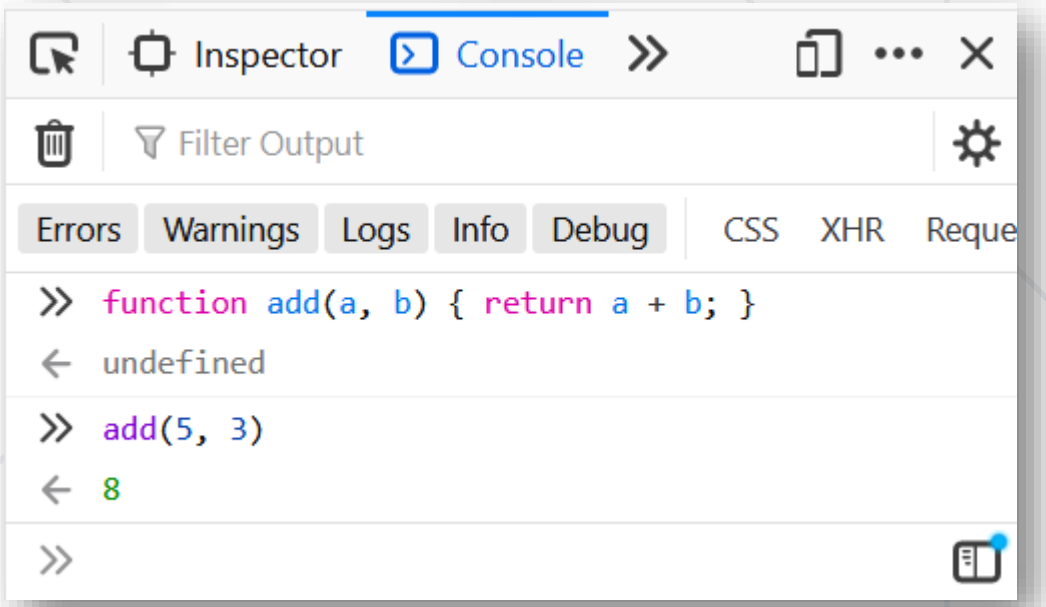
Dynamic Programming Language

- JavaScript is a **dynamic programming language**
  - Operations otherwise done at **compile-time** can be done at **run-time**
- It is **possible** to change the **type** of a variable or add new properties or methods to an object **while** the program is **running**
- In **static programming languages**, such changes are normally **not possible**

Web Browser Dev Console

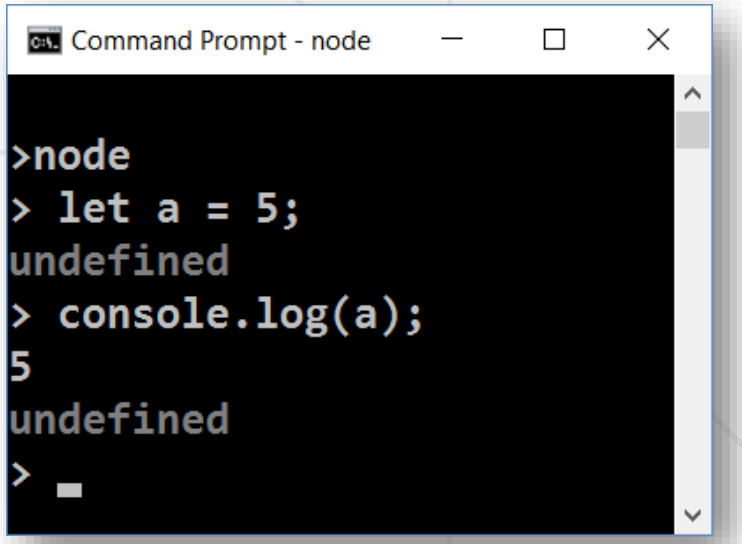
- Developer Console: **[F12]**





Node.js

- What is **Node.js**?
  - **Server-side** JavaScript runtime
  - Chrome V8 JavaScript engine
  - NPM **package manager**
  - Install node packages



JavaScript Syntax

Functions, Operators, Input and Output

JavaScript Syntax

- Defining and initializing variables

Variable name

Declare a variable with `let`

```
let a = 5;
let b = 10;
```

Variable value

- Conditional statement

Body of the conditional statement

```
if (b > a) {
  console.log(b);
}
```

Functions and Input Parameters

- In order to solve different problems, we are going to use **functions** and the **input** will come as **parameters**
- A function is similar to a **procedure**, which executes when called

declaration

parameters

```
function solve (num1, num2) {
    // some logic
}

solve(2, 3);
```

calling the function

Printing to the Console

- We use the **console.log()** method to print to console

```
function solve (name, grade) {
  console.log('The name is: ' + name + ', grade: ' + grade);
}

solve('Peter', 3.555);
// The name is: Peter, grade: 3.555
```

- Text can be composed easier using interpolated strings
  - Works only with the ` brackets

```
console.log(`The name is: ${name}, grade: ${grade}`);
```

Printing to the Console

- To format a number, use the `toFixed()` method
  - Converts a number to `string`
    - Rounds the string to a specified number of decimals
      - Default value is 0 (no decimals)

```
grade.toFixed(2);
// The name is: Peter, grade: 3.56
```

Number of digits after the decimal sign

    - If the number of decimals is higher than in the number, zeros are added

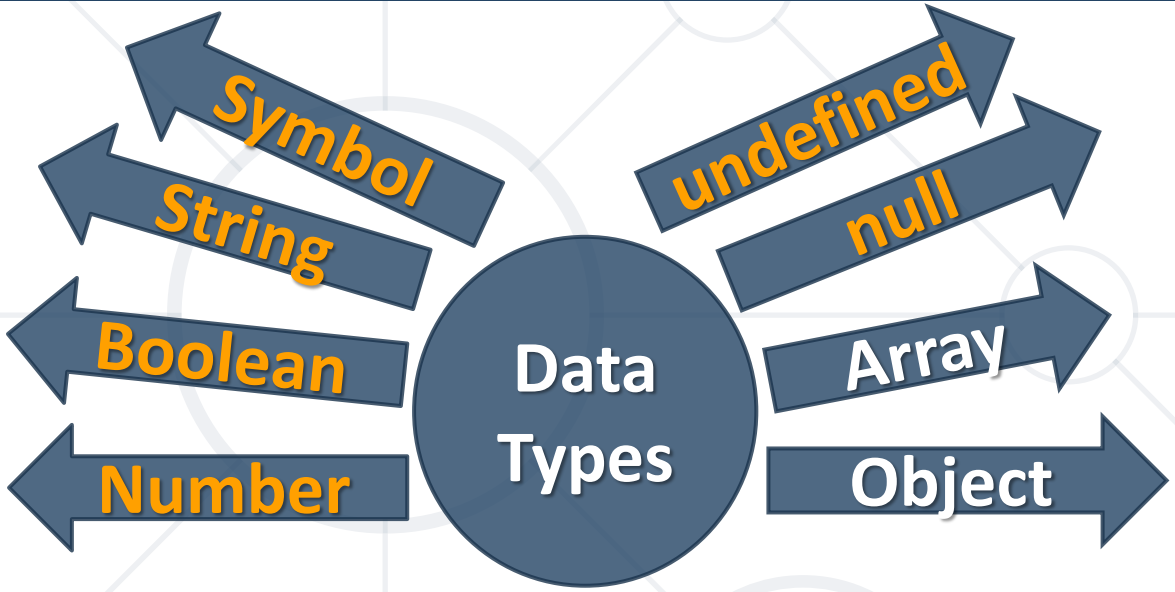
Data Types and Variables

Definitions and Examples

JavaScript Data Types

- Seven **primitive types**
  - Boolean
  - null
  - undefined
  - Number
  - String
  - Symbol
  - BigInt
- and **Objects** (including Functions and Arrays)

Data Types Examples




```
let number = 10;                                  // Number
let person = {name: 'George', age: 25};           // Object
let array = [1, 2, 3];                            // Array
let isTrue = true;                                // Boolean
let name = 'George';                              // String
let empty = null;                                 // null
let unknown = undefined;                          // undefined
```

Variable Scope

- **var**
  - Use **function scope**
  - Can be accessed anywhere in the function, including outside the initial block

```
{
  var x = 2;
}
console.log(x);
// 2
```

- **let** and **const**
  - Use **block scope**
  - Can **NOT** be accessed from outside the **{}** block where initially declared

```
{
  let x = 2;
}
console.log(x);
// Error
```

let vs const

- `let`
  - Can be reassigned after initial assignment
  - Variable's value can change
  - `let` is used when reassignment is necessary
- `const`
  - Cannot be reassigned after initial assignment, remains constant
  - Variable's value remains fixed
  - `const` is used when variable will not be reassingned

Undefined

- A variable without a value has the value **undefined**
  - The **typeof** is also **undefined**

```
let car;
// Value is undefined, type is undefined
```

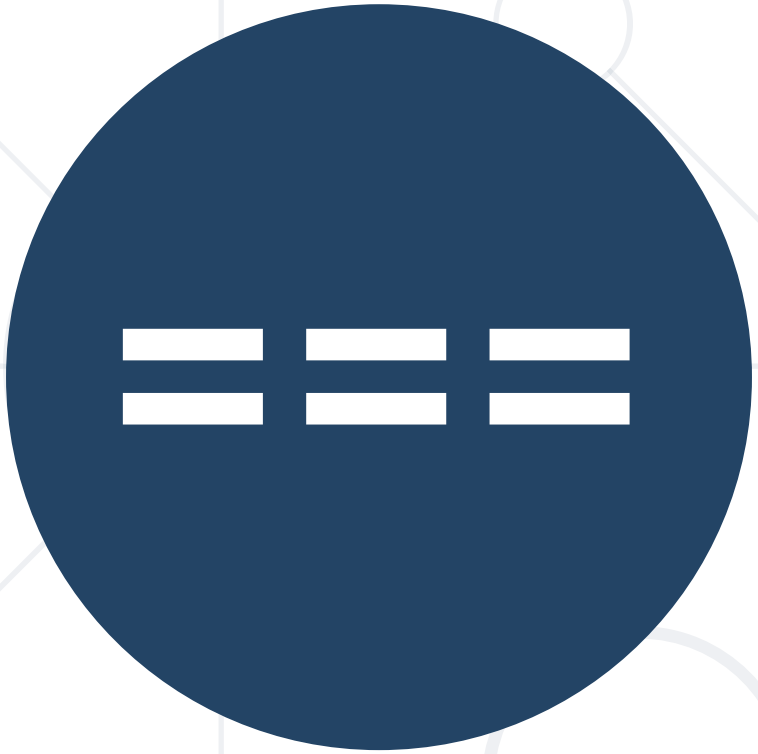
- A variable can be emptied, by setting the value to **undefined**
  - The type will also be **undefined**

```
let car = undefined;
// Value is undefined, type is undefined
```

Null

- **Null** is **"nothing"**
- It is supposed to be something that doesn't exist
- The **typeof** null is an **object**

```
let person = {
  firstName:"John",
  lastName:"Doe",
  age:50
};
person = null;
console.log(person);          // null
console.log(typeof(person)); // object
```



Operators

Overview of Different Types of Operators

Arithmetic Operators

- **Arithmetic operators**
  - Take numerical values (either literals or variables) as their operands
  - Return a single numerical value
    - Addition (+)
    - Subtraction (-)
    - Multiplication (*)
    - Division (/)
    - Remainder (%)
    - Exponentiation (**)

```
let a = 15;
let b = 5;
let c;
c = a + b;  // 20
c = a - b;  // 10
c = a * b;  // 75
c = a / b;  // 3
c = a % b;  // 0
c = a ** b; // 15^5 = 759375c
```

Comparison Operators

- Used in logical statements to determine equality or difference between various variables or values

Operator	Notation in JS
Equal value	==
Equal value and type	===
Not equal value	!=
Not equal value/type	!==
Greater than	>
Greater than or Equal	>=
Less than	<
Less than or Equal	<=

Comparison Operators – Examples

```
console.log(1 == '1');   // true
console.log(1 === '1'); // false
console.log(3 != '3');   // false
console.log(3 !== '3'); // true
console.log(5 < 5.5);    // true
console.log(5 <= 4);     // false
console.log(2 > 1.5);    // true
console.log(2 >= 2);     // true
console.log((5 > 7) ? 4 : 10); // 10
```

Ternary operator

Typeof Operator

- The **typeof** operator returns a string indicating the type of an operand

```
const val = 5;
console.log(typeof val);    // number
```

```
const str = 'hello';
console.log(typeof str);    // string
```

```
const obj = {name: 'Maria', age:18};
console.log(typeof obj);   // object
```

Conditional Statements

Implementing Control-Flow Logic

What is a Conditional Statement?

- The `if-else` statement
  - Do action depending on a specified condition

```
let a = 5;
if (a >= 5) {
  console.log(a);
}
```

If the condition is met, the code will execute

  - You can chain conditions

```
else {
 console.log('no');
}
```

Continue on the next condition, if the first is not met

Chained Conditional Statements

- The **`if-else if-else…`** construct is a series of checks

```
let a = 5;
if (a > 10)
  console.log("Bigger than 10");
else if (a < 10)
  console.log("Less than 10");
else
  console.log("Equal to 10");
```

Only "Less than 10" will be printed

- If one condition is true, it does not proceed to verify the next conditions

Logical Operators

- **Logical operators** give us the ability to write multiple conditions in one `if` statement
- They return a boolean result (`true` or `false`)

Operator	Description	Example
!	NOT	!false → true
&&	AND	true && false → false
||	OR	true || false → true

The Switch-Case Statement

- Works as a series of `if-else if-else if…`

```
switch (...) {
    case ...:
     // code
     break;
    case ...:
     // code
     break;
    default:
     // code
     break;
}
```

The condition in the switch case is a value

List of conditions (values) for the inspection

Code to be executed if there is no match with any case

Loops

Code Block Repetition

Loops in JavaScript

- Loops execute a block of code a number of times
- JavaScript supports 5 kinds of loops
  - `for`
  - `for-in`
  - `for-of`
  - `while`
  - `do-while`

Types of Loops

- The **for** loop
  - Loops through a block of code a specified number of times

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

- The **for-of** loop
  - Iterates through all **elements** in an iterable object
  - Cannot access the current index

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

Types of Loops

- The **while** loop
  - Executes a block of code as long as the specified condition is true

```
while (condition) {
  // code to be executed
}
```

- The **do-while** loop
  - Executes a block of code once, then checks the condition

```
do {
   // code to be executed
}
while (condition);
```

Debugging Techniques

Strict Mode, IDE Debugging Tools

Strict Mode

- **Strict mode** limits certain "sloppy" language features
  - Silent errors will **throw exception** instead

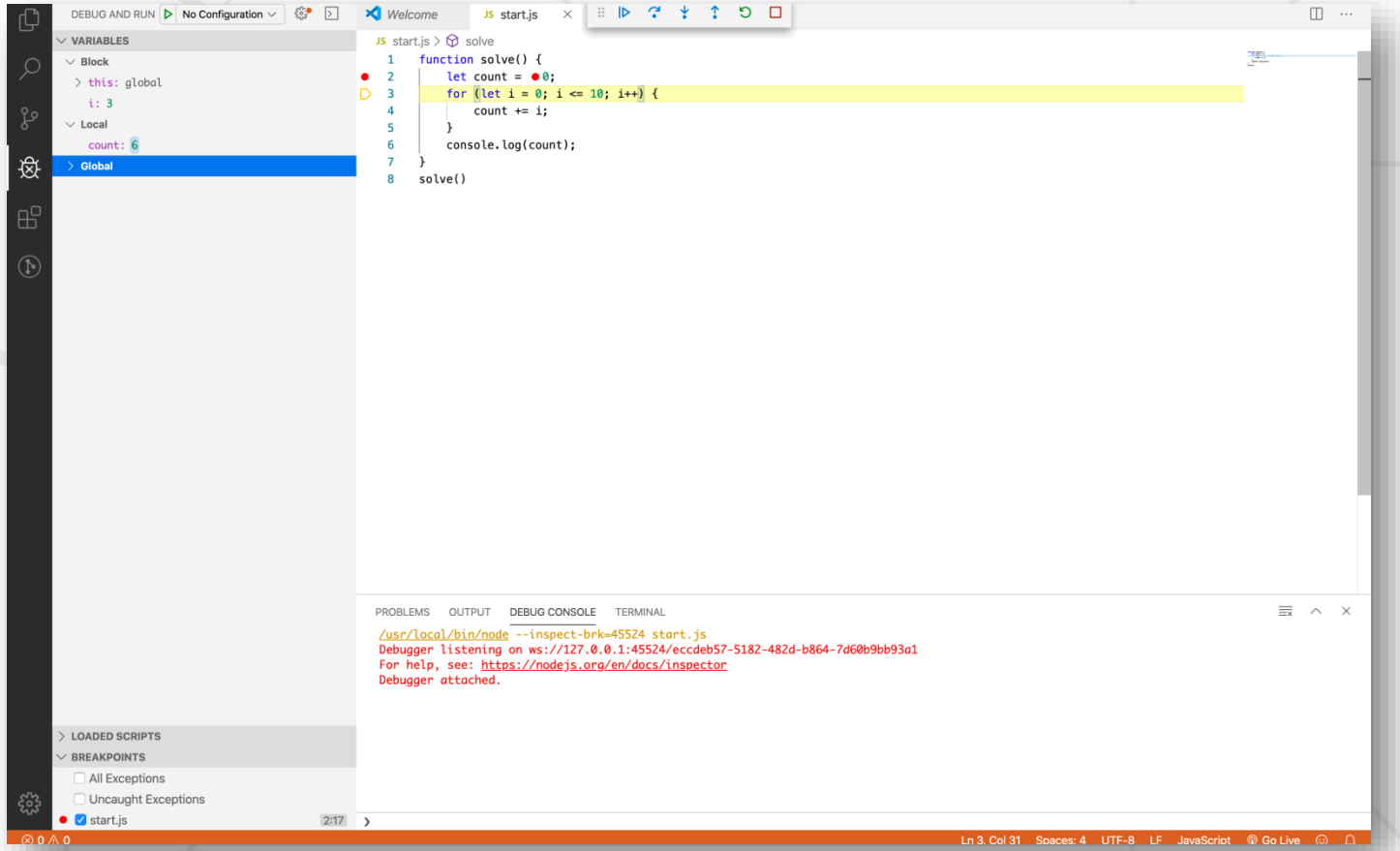
```
'use strict';                // File-level
mistypeVariable = 17; // ReferenceError
```

```
function strict() {
    'use strict';            // Function-level
    mistypeVariable = 17;
}
```

- Enabled by default in **modules**

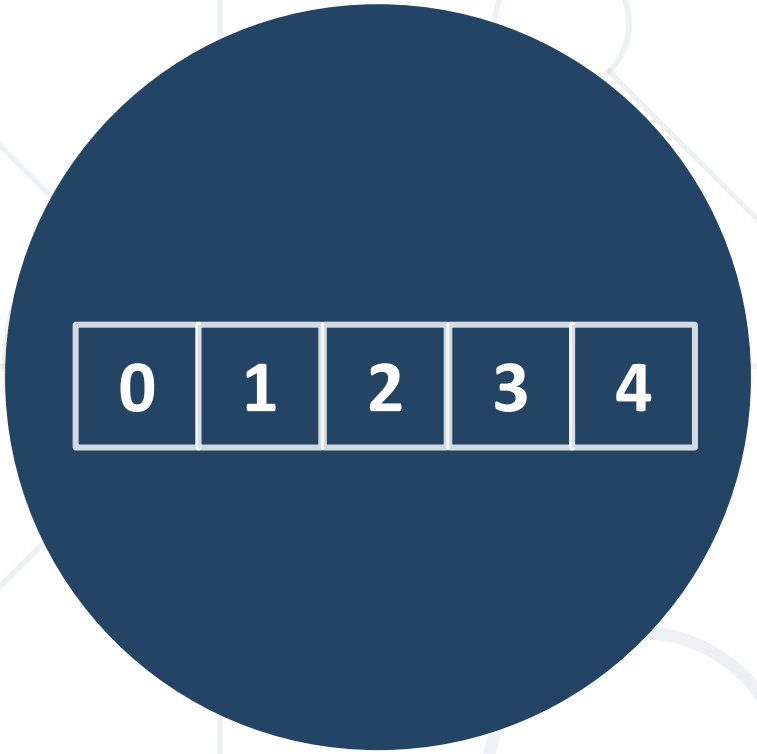
Debugging in Visual Studio Code

- Visual Studio Code has a built-in **debugger**
- It provides
  - **Breakpoints**
  - Ability to **trace** the code execution
  - Ability to **inspect** variables at runtime



Using the Debugger in Visual Studio Code

- Start without Debugger: **[Ctrl+F5]**
- Start with Debugger: **[F5]**
- Toggle a breakpoint: **[F9]**
- Trace step by step: **[F10]**
- Force step into: **[F11]**



Working with Arrays of Elements

Arrays in JavaScript

Arrays in JavaScript

- 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

Creating Arrays in JavaScript

- Array **literal**

```
let myArray = ["John Doe", 24, true];
```

```
let myArray = [];
myArray[0] = "John Doe";
myArray[1] = 24;
myArray[2] = true;
```

- Array **constructor**

```
let myArray = new Array("John Doe", 24, true);
```

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

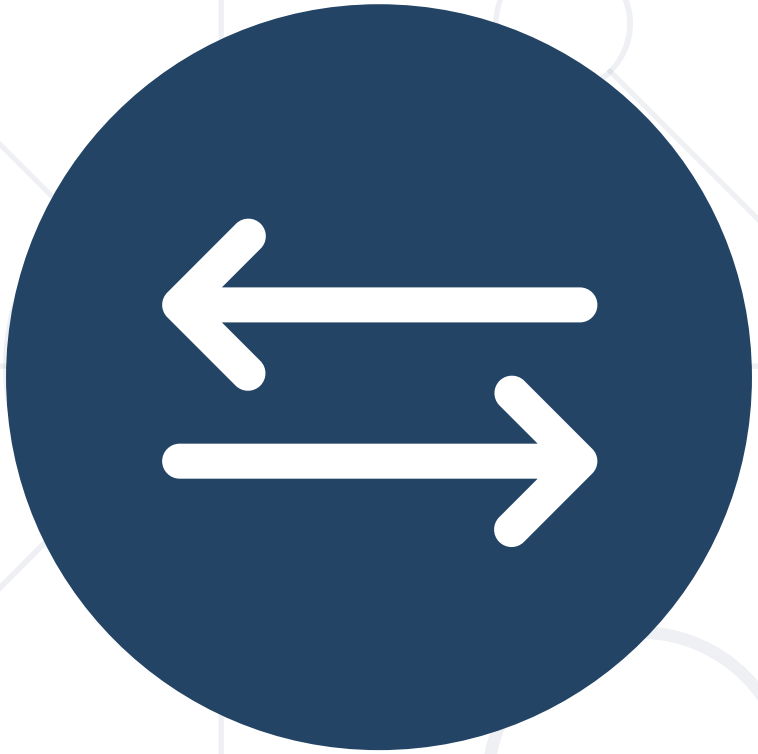
- Expression that **unpacks values** from **arrays** or **objects**, into distinct **variables**

```
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



Array's 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

```
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

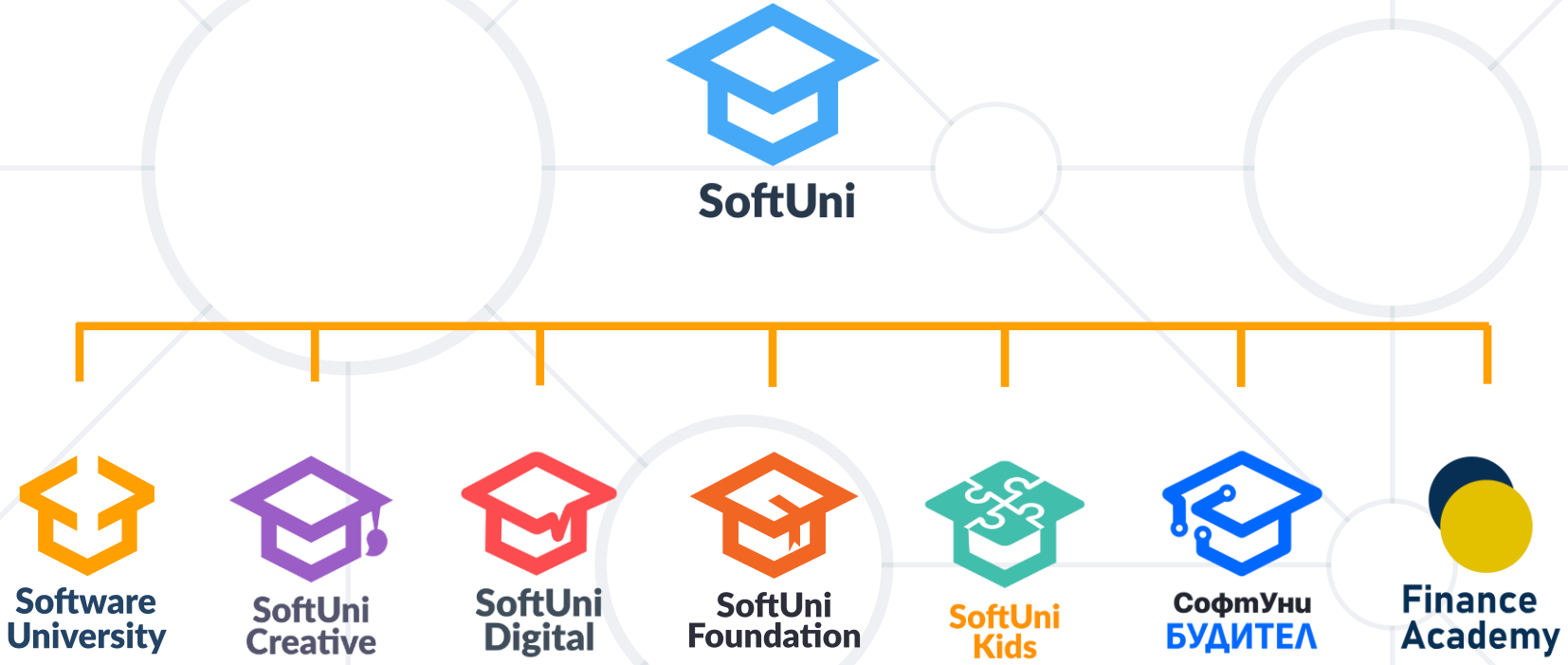
- Creates a **new array** with **filtered elements only**
- Calls a **provided** callback **function** once for each element in an array
- **Does not mutate** the **array** on which it is called

```
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']
```

Summary

- JS == a high-level programming language
- Node.js == server-side JS runtime
- There are objects and 7 primitive data types
- 3 variable types – let, const, var
- Conditional statement – if-else, switch-case
- Loops – for, for-in, for-of, while, do-while
- Different debugging techniques
- Arrays in JS can hold mixed data
- Various methods for working with them

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

PHAR VISION

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