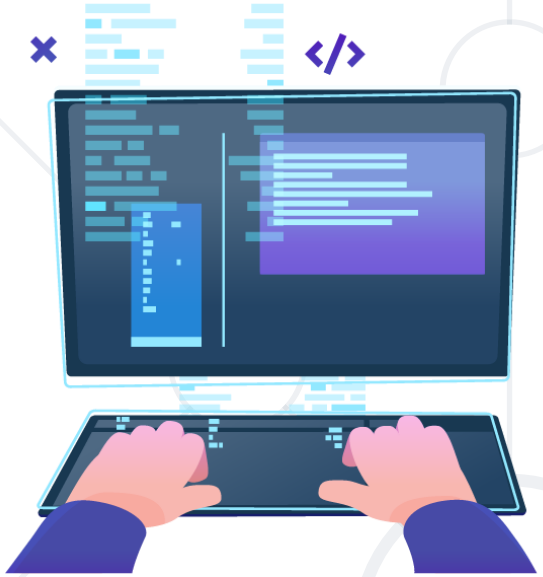
JS Syntax Fundamentals

Syntax, Conditional Statements, Loops, Data Type and Variables, Array



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#js-front-end

JavaScript Overview

Definition, Execution, IDE Setup

What is JavaScript?

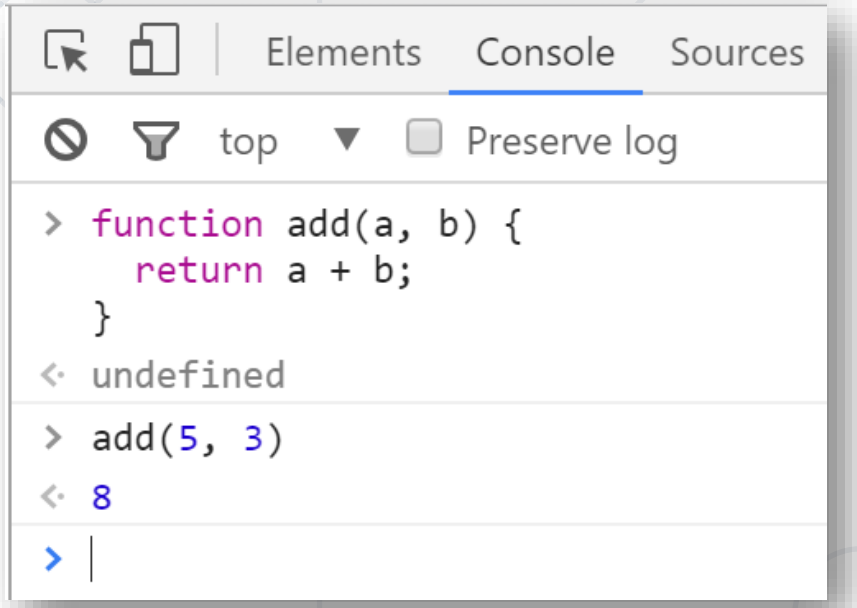
- JavaScript (**JS**) is a **high-level** programming language
  - One of the **core technologies** of the World Wide Web
  - Enables **interactive** web pages and applications
  - Can be **executed** on the **server** and on the **client**
- Features:
  - C-like **syntax** (curly-brackets, identifiers,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

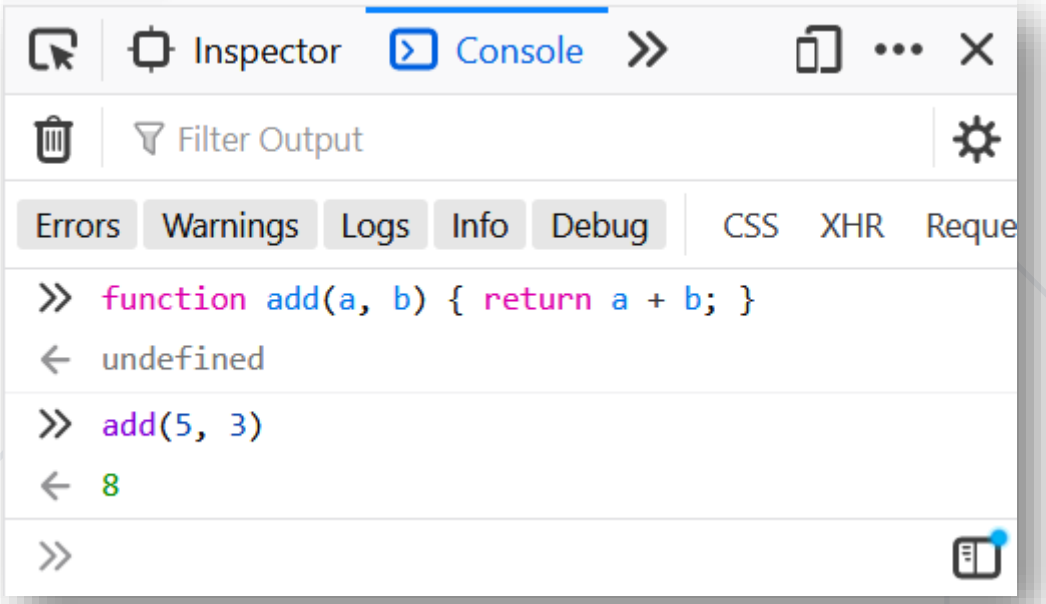
Dynamic Programming Language

- 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**

Chrome Web Browser

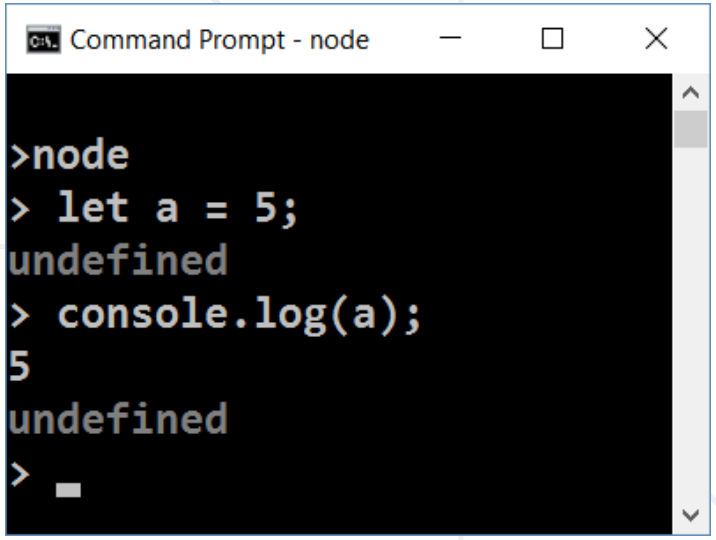
Developer Console: [F12]





Node.js

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



Install the Latest Node.js

Downloads

Latest LTS Version: **14.15.4** (includes npm 6.14.10)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

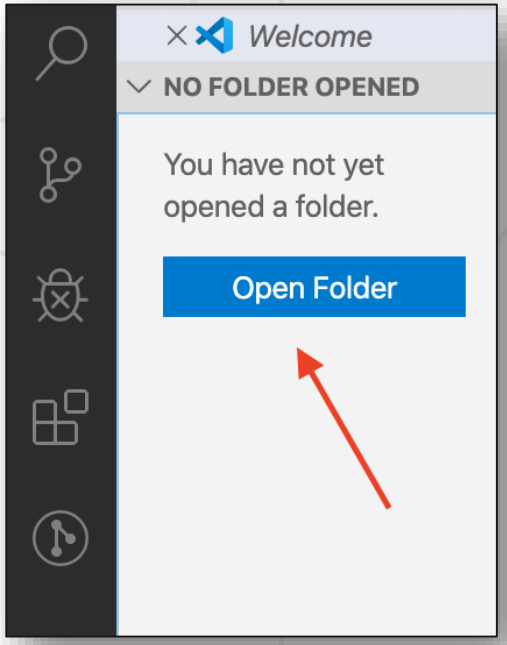
LTS Recommended For Most Users	Current Latest Features

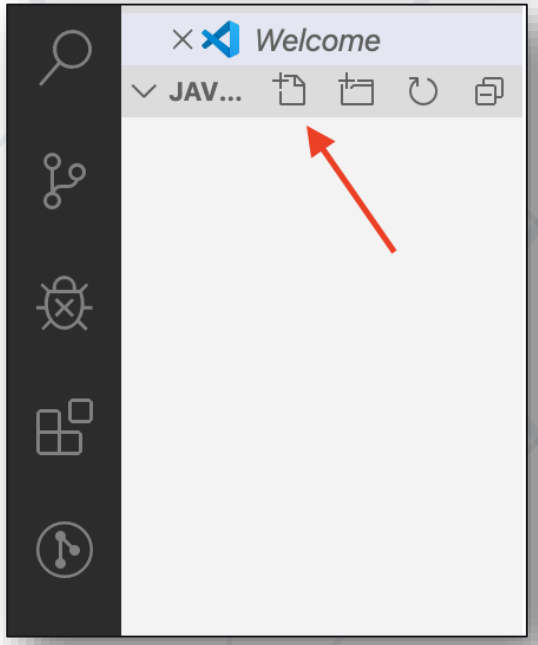
Windows Installer	macOS Installer	Source Code
node-v14.15.4-x64.msi	node-v14.15.4.pkg	node-v14.15.4.tar.gz

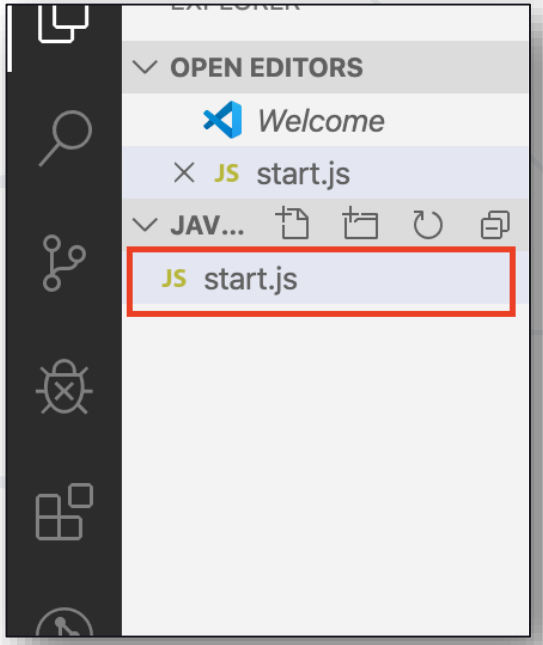
Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit	
macOS Binary (.tar.gz)	64-bit	
Linux Binaries (x64)	64-bit	
Linux Binaries (ARM)	ARMv7	ARMv8
Source Code	node-v14.15.4.tar.gz	

Using Visual Studio Code

- **Visual Studio Code** is powerful text editor for JavaScript and other projects
- In order to create your **first project**:







JavaScript Syntax

Functions, Operators, Input and Output

JavaScript Syntax

- C-like **syntax** (curly-brackets, identifiers, operator)
- Defining and Initializing variables:

Declare a variable with let

Variable name

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

Variable value

- Conditional statement:

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

Body of the conditional statement

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**, that 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:

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

- To format a number, use the **toFixed()** method (converts to **string**):

Number of decimal places

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

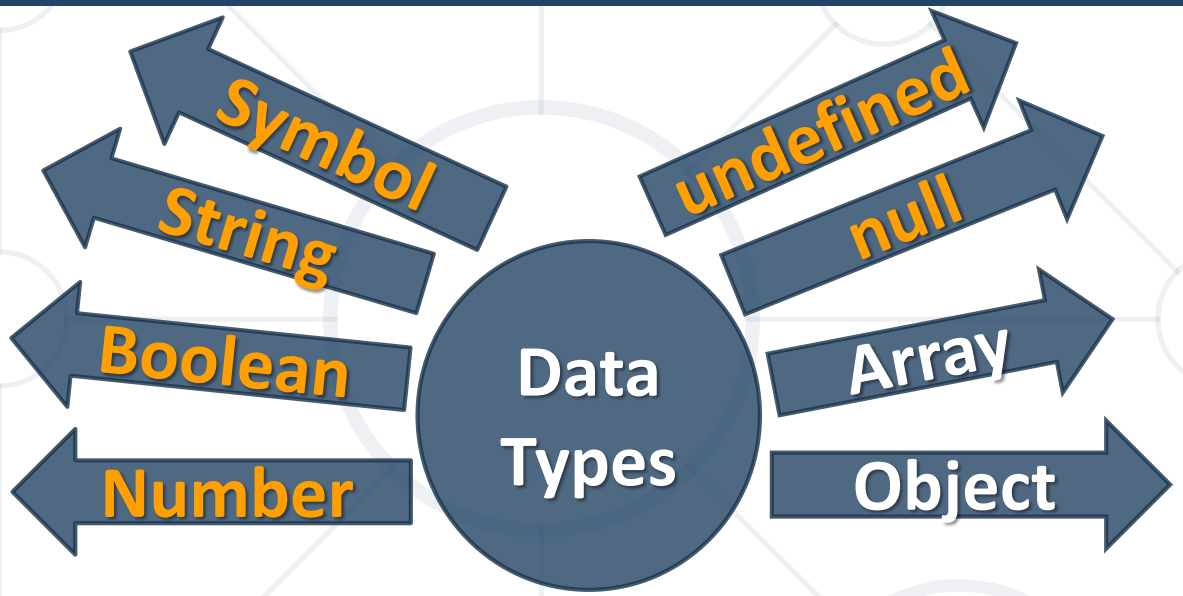
Data Types and Variables

Definition and Examples

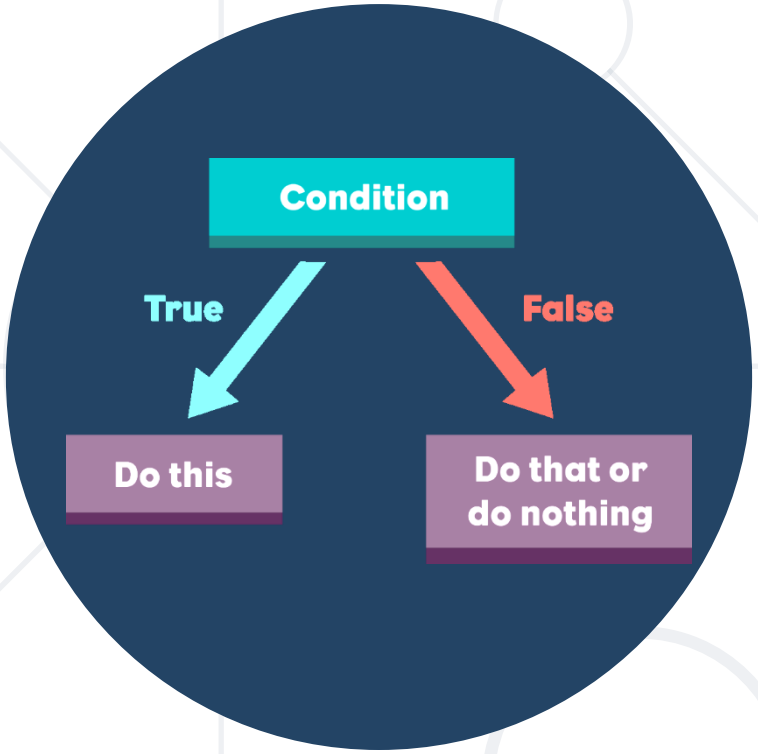
What is a Data Type?

- A **data type** is a classification that specifies what type of operations can be applied to it and the way values of that type are stored
- After **ECMAScript** 2015 there are **seven primitive** data types:
  - Seven **primitive**: Boolean, null, undefined, Number, String, Symbol, BigInt
  - and **Objects** (including Functions and Arrays)

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
```



Conditional Statements

Implementing Control-Flow Logic

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

```
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

What is a Conditional Statement?

The **if-else** statement:

- Do action depending on 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 following conditions

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

Logical Operators

- **Logical operators** are used to determine the logic between variables or values. They return the value of one of the operands based on certain rules, not always just (**true** or **false**).

Operator	Description	Example
!	NOT	!false -> true
&&	AND	true && false -> false
||	OR	true || false -> true

Logical Operators: Examples

- Logical "**AND**"
  - Checks the fulfillment of several conditions simultaneously

```
let a = 3;
let b = -2;
console.log(a > 0 && b > 0); // expected output: false
```

- Logical "**OR**"
  - Checks that at least one of several conditions is met

```
let a = 3;
let b = -2;
console.log(a > 0 || b > 0); // expected output: true
```

Logical Operators: Examples (2)

- Logical "**NOT**"
  - Checks if a condition is **not** met

```
let a = 3;
let b = -2;
console.log(!(a > 0 || b > 0));
// expected output: false
```

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
```

Loops

Code Block Repetition

What is a Loop?

The **for** loop:

- Repeats until the condition is evaluated

```
for (let i = 1; i <= 5; i++){
  console.log(i)
}
```

Incrementation in the condition

The **while** loop:

- Does the same, but has different structure

```
let i = 1
while (i <= 5) {
  console.log(i)
  i++
}
```

Incrementation outside the condition

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

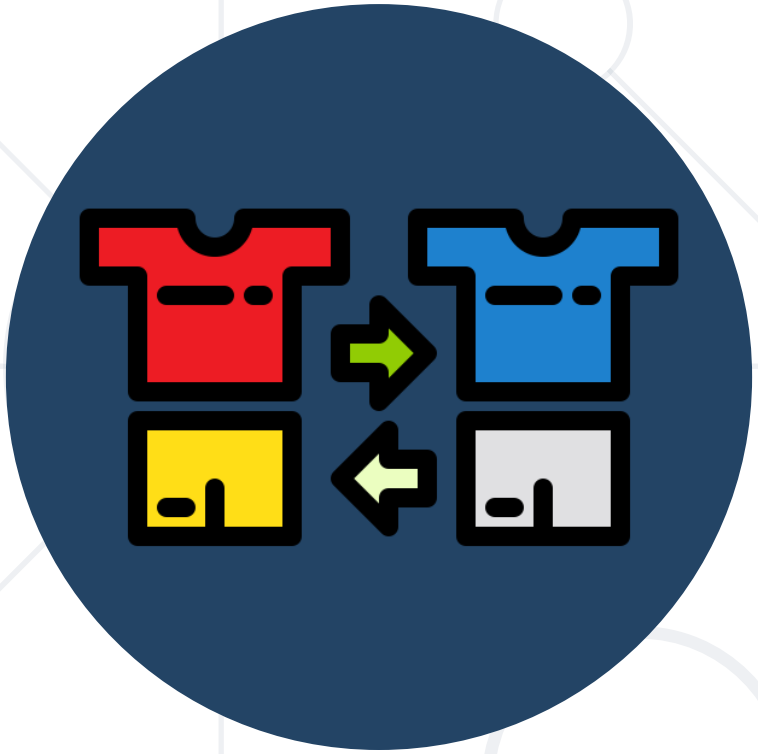
- Iterates through all **elements** in a collection
- Cannot access the current index

```
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);
```

```
1 2 3 4 5
```



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

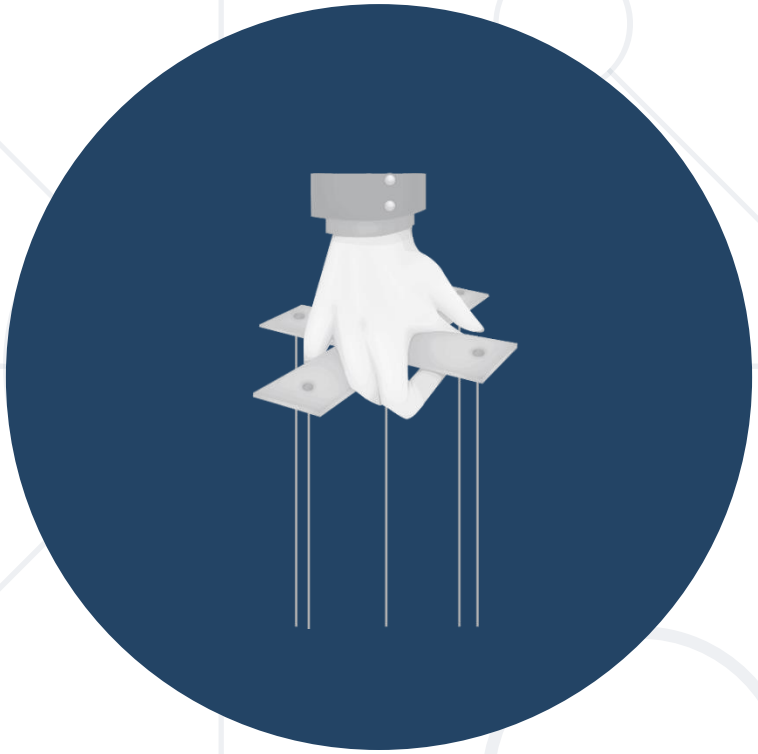
- Returns the **first found value** in the array, if an **element** in the array **satisfies** the **provided** testing **function** or **undefined** if not found

```
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.....
```

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);
}
```

Live Exercises

Debugging Techniques

Strict Mode, IDE Debugging Tools

Strict Mode

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

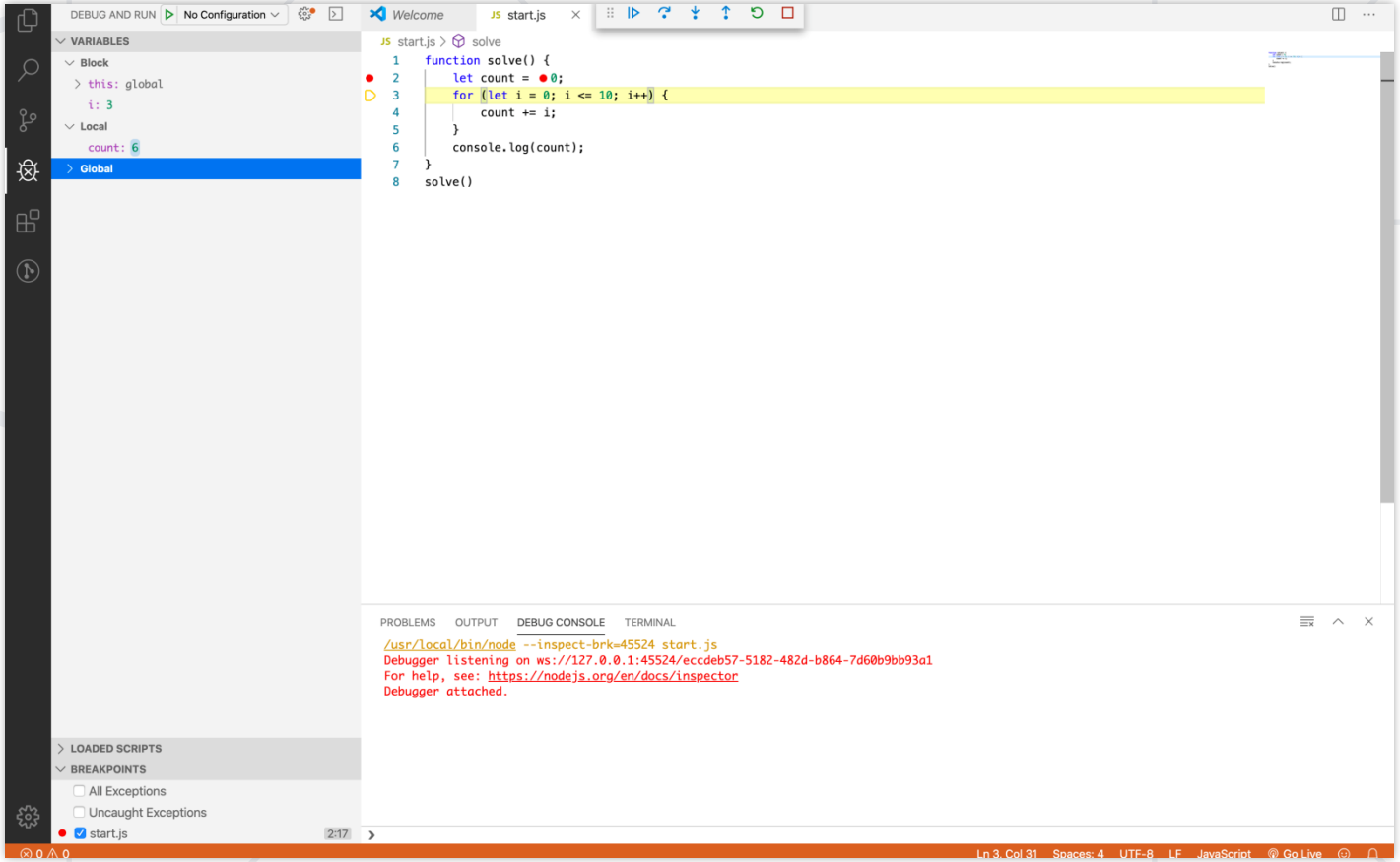
```
'use strict';                // File-level
mistypeVariable = 17; // ReferenceError
```

```
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



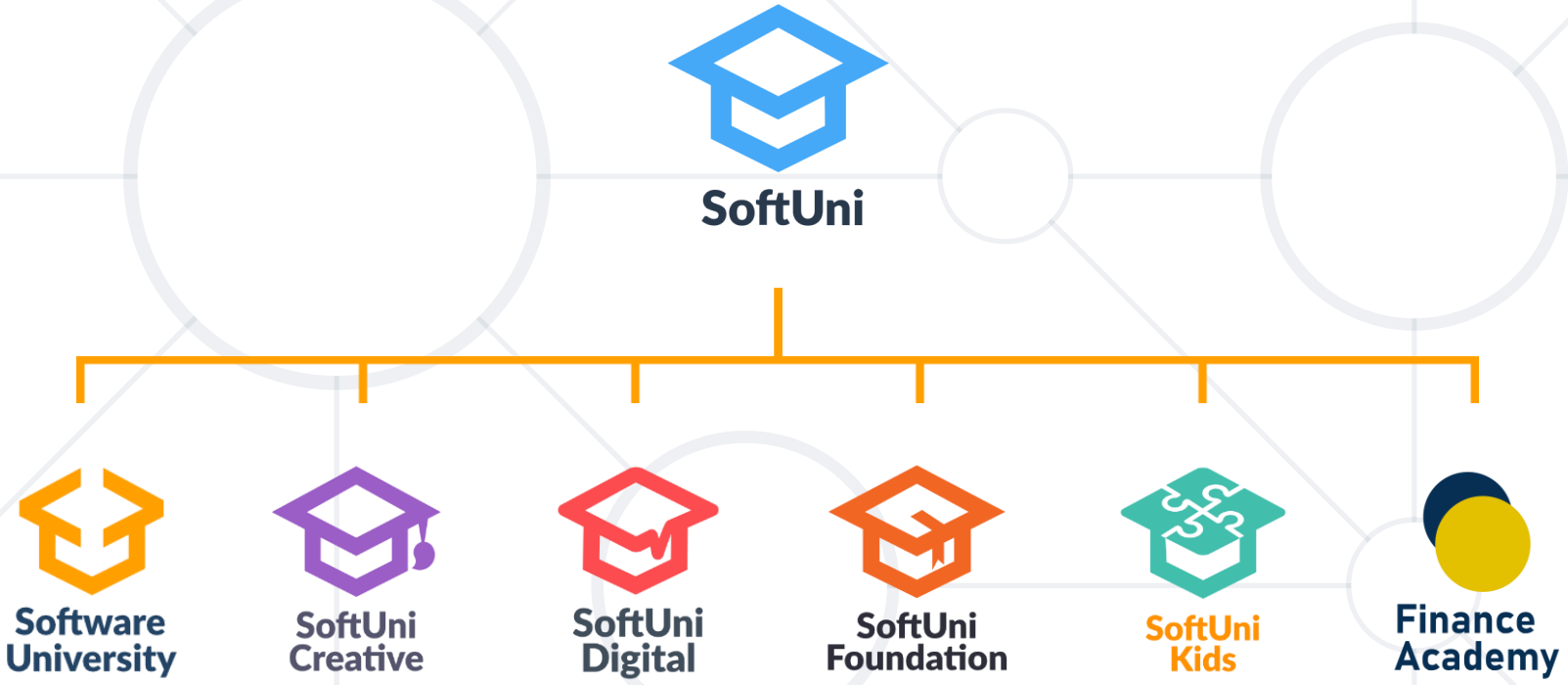
Using the Debugger in Visual Studio Code

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

Summary

- JS is a high-level programming language
- Conditional statement – If-else, Switch-case
- Loops – For-loop, While-loop
- Data Types
  - String, Number, Boolean, Null, Undefined
- Array
  - Methods
- Associative Array

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

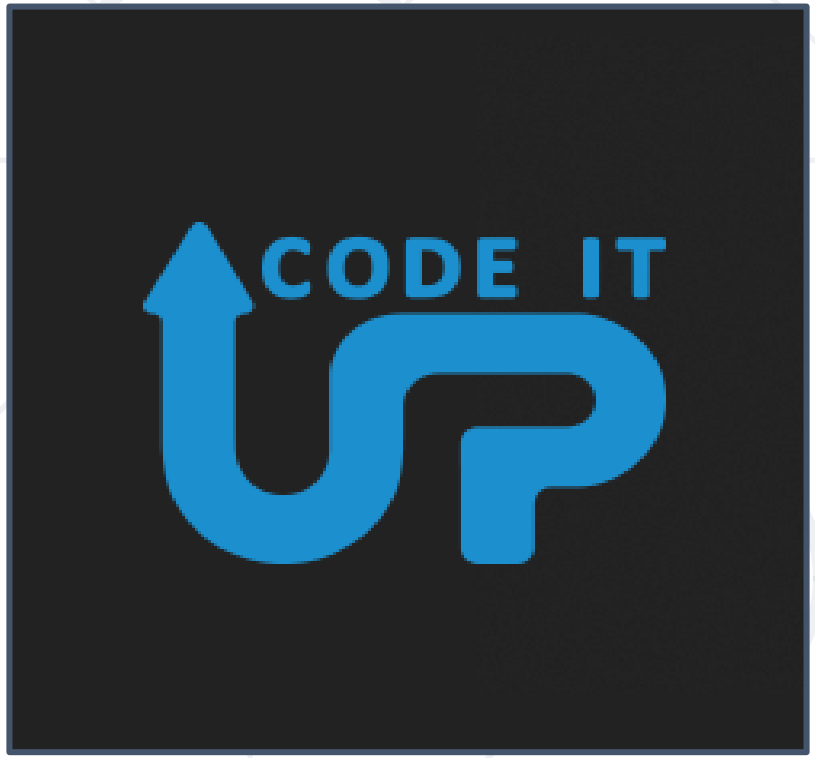
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



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

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
- Software University Forums
  - forum.softuni.bg