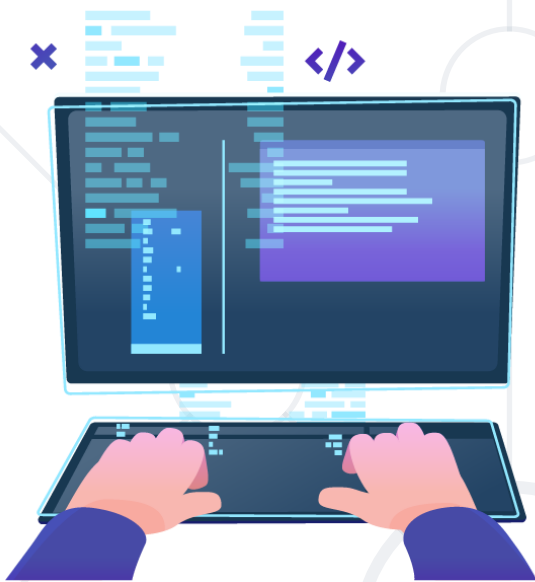# JS Syntax Fundamentals

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

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

Table of Contents

1. JavaScript Syntax

2. Data Types and Variables

3. Conditional Statements

4. Loops

5. Arrays

6. Text Processing

7. Debugging

sli.do

#Dev-Ops

JavaScript Overview

Definition, Execution, IDE Setup
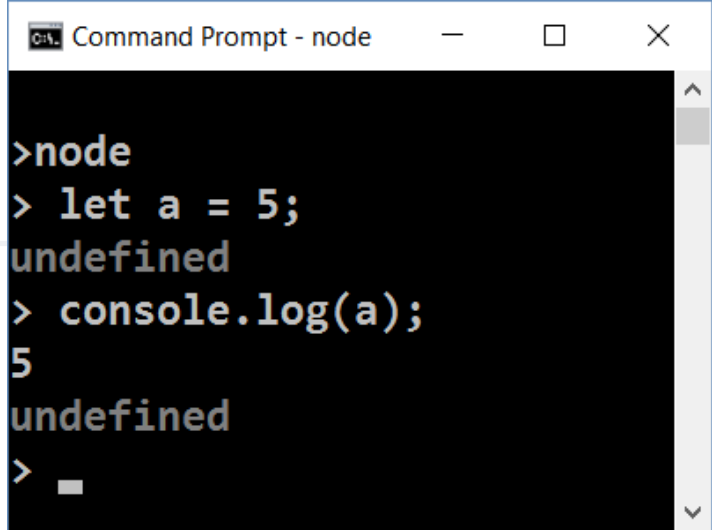
What is JavaScript?

- 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**

Node.js

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

```
Command Prompt - node

>node
> let a = 5;
undefined
> console.log(a);
5
undefined
>
```

JavaScript Syntax

Functions, Operators, Input and Output

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

```javascript
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

```javascript
function solve (name, grade) {
  console.log('The name is: ' + name + ', grade: ' + grade);
}
solve('Peter', 3.555);
//The name is: Peter, grade: 3.555
```
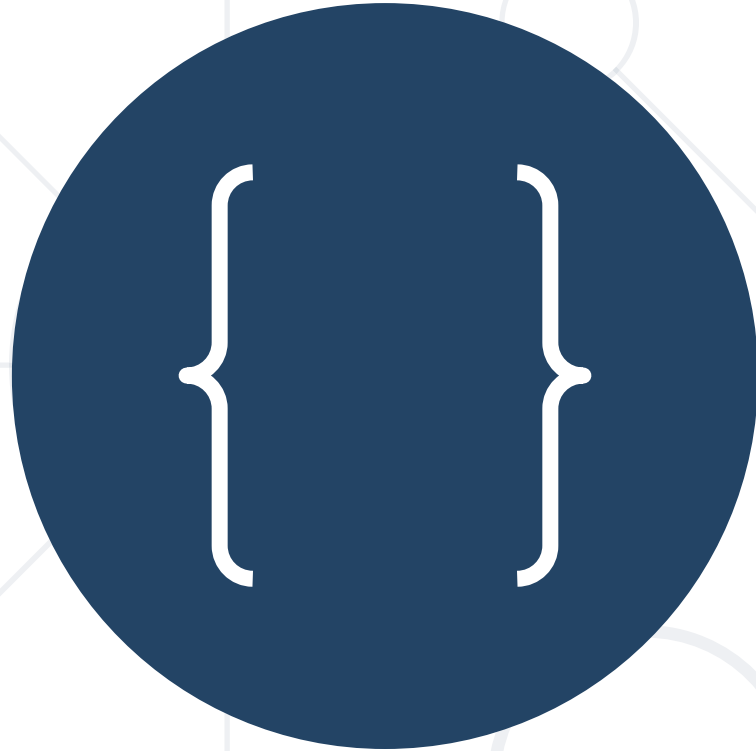
- Text can be composed easier using interpolated strings

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

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

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

Number of decimal places

Data Types and Variables

Definition and Examples

JavaScript Data Types

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

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;//15⁵
= 759375c
```

Comparison Operators

```javascript
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

Chained Conditional Statements

- The **if** – **else if** – **else**... construct is a series of checks

```javascript
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** 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

Typeof Operator

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

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

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

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

Loops

Code Block Repetition

Loops in JavaScript (1)

- The **for** loop

  - Incrementation in the condition

```javascript
for (let i = 1; i <= 5; i++){
    console.log(i)
}
```
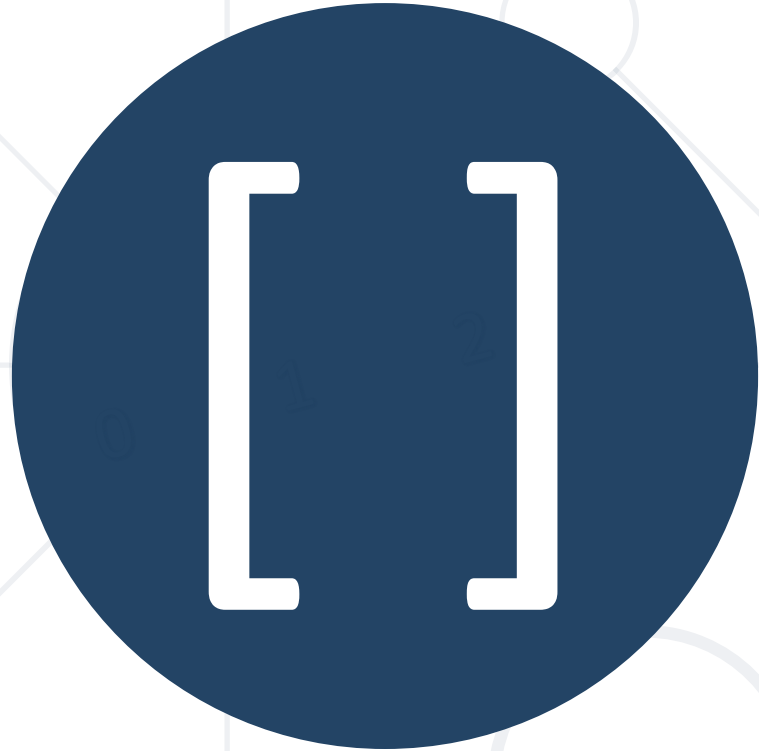
- The **while** loop

  - Incrementation outside of the condition

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

Loops in JavaScript (1)

- The **for-of** Loop

  - Iterates through all **elements** in a collection

  - Cannot access the current index

```javascript
for (let el of collection) {

    // Process the value here

}
```

Working with Arrays of Elements

Arrays in JavaScript

Arrays of Different Types

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

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

```javascript
// 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
```
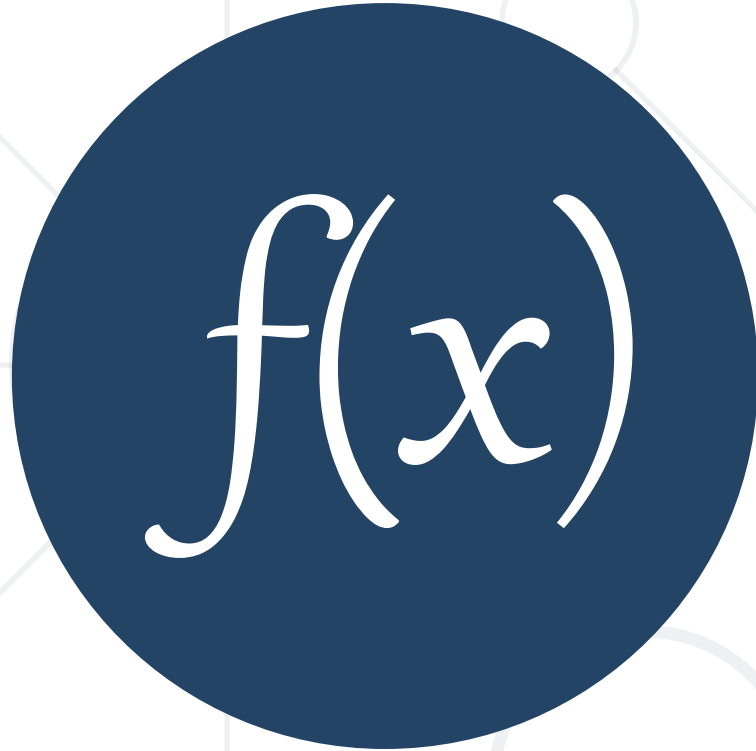
Destructuring Syntax

- 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

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

```javascript
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**

```javascript
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

```javascript
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

```javascript
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

```javascript
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']
```

Concatenating

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

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

- Use the **concat()** method

```javascript
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.
```

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));
}
```

➡

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

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......
```

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**

    - **Array's methods**

- **Working with strings**

Questions?
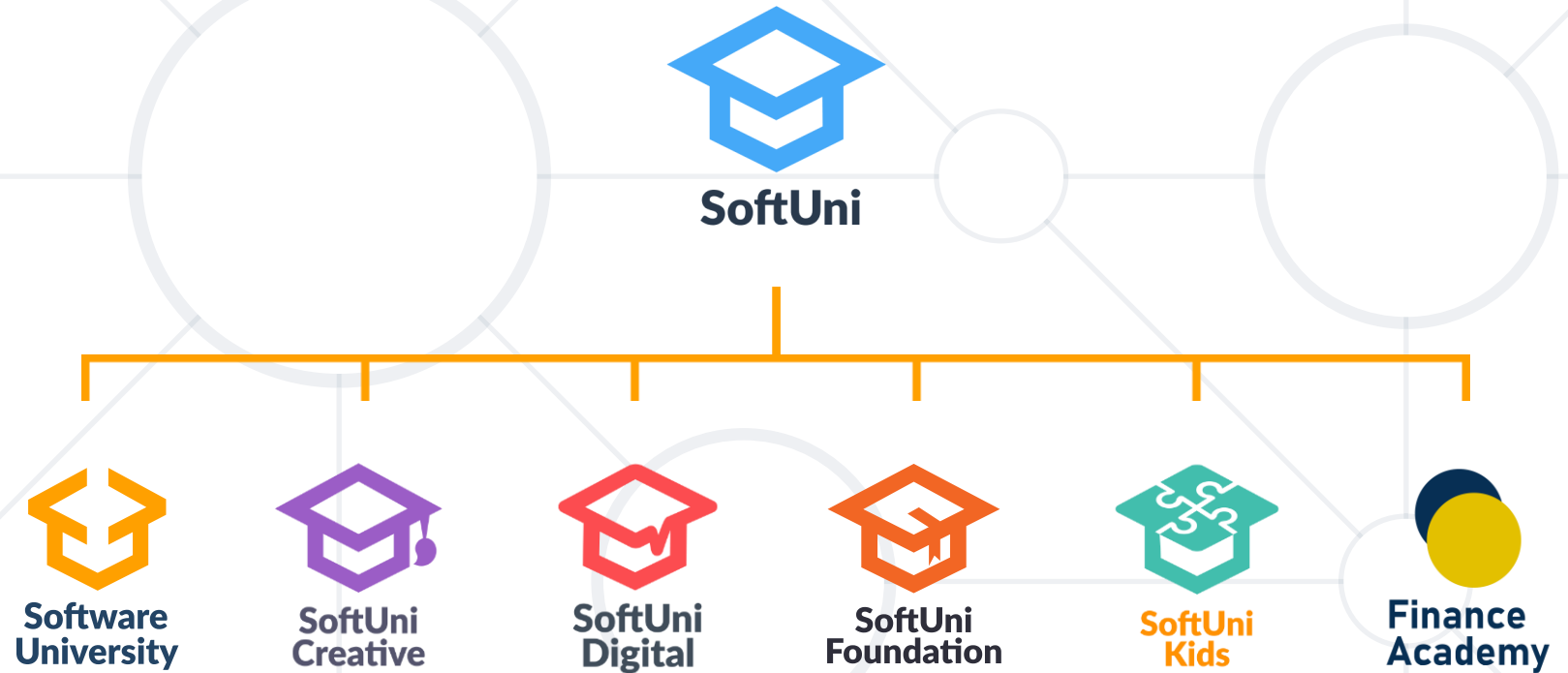
SoftUni Diamond Partners

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