

# Front End Basics

# JS



SoftUni Team  
Technical Trainers



**SoftUni**



Software University

<https://softuni.bg>

## 1. JavaScript

- What's JavaScript?
- Functions
- Objects

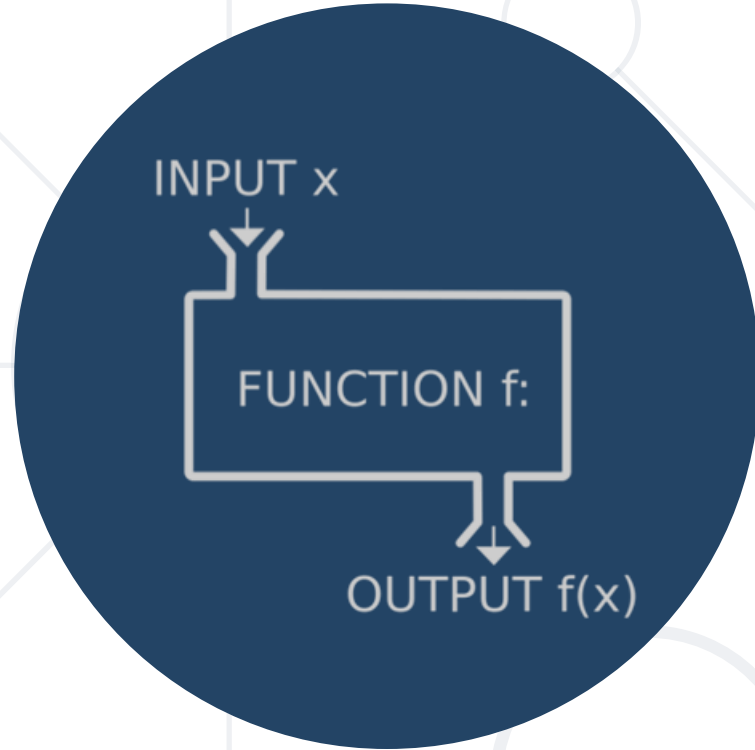
## 2. Bootstrap

## 3. Grid system

## 4. Bootstrap components

sli.do

**#java-web**



**JavaScript**

- 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**

- Seven **data types** that are **primitives**
  - **String** - used to represent textual data
  - **Number** - a numeric data type
  - **Boolean** - a logical data type
  - **Undefined** - automatically assigned to variables
  - **Null** - represents the **intentional absence** of any object value
  - **BigInt** - represent integers with **arbitrary precision**
  - **Symbol** - **unique** and **immutable** primitive value
- **Data structures**

# Variable Values

- **let**, **const** and **var** are used to declare variables
  - **let** - for **reassigning** a variable

```
let name = "George";  
name = "Maria";
```

- **const** - once assigned it **cannot** be modified

```
const name = "George";  
name = "Maria"; // TypeError
```

- **var** - defines a variable in the lexical scope **regardless** of block scope

```
var name = "George";  
name = "Maria";
```



- Variables in JavaScript are **not** directly **associated** with any particular **value type**
- Any variable **can** be assigned (and re-assigned) values of all types

```
let foo = 42;      // foo is now a number  
foo = 'bar';      // foo is now a string  
foo = true;       // foo is now a boolean
```



# Comparison Operators (1)

Operator	Notation in JS
EQUAL value	<code>==</code>
EQUAL value and type	<code>===</code>
NOT EQUAL value	<code>!=</code>
NOT EQUAL value/type	<code>!==</code>
Greater than	<code>&gt;</code>
Greater than OR EQUAL	<code>&gt;=</code>
LESS than	<code>&lt;</code>
LESS than OR EQUAL	<code>&lt;=</code>


# Comparison Operators (2)

```
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 ? 4 : 10);  // 4
```

The "?" is a ternary operator



# Functions

- 
- **Function** - named list of instructions (statements and expressions)
    - Can take **parameters** and return **result**
    - Function names and parameters use **camel case**
    - The ' **{** ' stays on the same line

```
function printStars(count) {  
    console.log("*".repeat(count));  
}
```

- **Invoke** the function

```
printStars(10);
```

# Declaring Functions

- Function declaration

```
function walk() {  
  console.log("walking");  
}
```

- Function expression

```
let walk = function () {  
  console.log("walking");  
}
```

- Arrow functions

```
let walk = () => {  
  console.log("walking");  
}
```



- You can instantiate parameters with no value

```
function foo(a,b,c){  
  console.log(a);  
  console.log(b);  
  console.log(c); //undefined  
}  
foo(1,2)
```

- The unused parameters are ignored

```
function foo(a,b,c){  
  console.log(a);  
  console.log(b);  
  console.log(c);  
}  
foo(1,2,3,6,7)
```

- Variable and function declarations are **put into memory** during the **compile** phase, but stay exactly where you **typed** them in your code
- **Only declarations are hoisted**

```
console.log(num); // Returns undefined  
var num;  
num = 6;
```

# Hoisting Variables

```
num = 6;  
console.log(num); // returns 6  
var num;
```

```
num = 6;  
console.log(num); // ReferenceError: num is not defined  
let num;
```

```
console.log(num); // ReferenceError: num is not defined  
num = 6;
```



# Hoisting Functions

```
run(); // running  
function run() {  
    console.log("running");  
};
```

```
walk(); // ReferenceError: walk is not defined  
let walk = function () {  
    console.log("walking");  
};
```

```
console.log(walk); //undefined  
walk(); // TypeError: walk is not a function  
var walk = function () {  
    console.log("walking");  
};
```





# What is an Object?

- An object is a **collection of fields**, and a field is an association between a name (or **key**) and a **value**
- Objects are a **reference data type**
- You define (and create) a JavaScript object with an **object literal**:

```
let person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50  
};
```



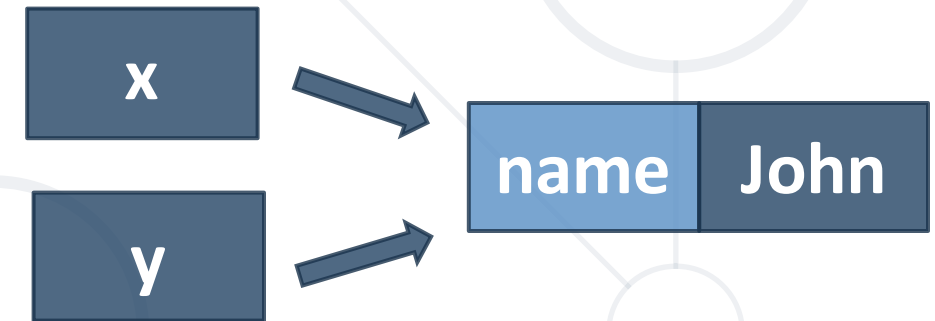
# Variables Holding References

- The **in-memory value** of a reference type is the **reference** itself (a memory address)

```
let x = {name: 'John'};
```


```
let y = x;
```

```
y.name = "John";  
console.log(x.name); // John
```



# Object Properties

- A **property** of an object can be explained as a **variable** that is **attached** to the object
- Object properties are basically the same as **ordinary** JavaScript **variables**, except for the **attachment** to objects



Property Name	Property Value
firstName	John
lastName	Doe
age	50


# Object Keys and Values

```
let course = { name: 'JS Core', hall: 'Open Source' };  
let keys = Object.keys(course);  
console.log(keys); // [ 'name', 'hall' ]  
if (course.hasOwnProperty('name')) {  
    console.log(course.name); // JS Core  
}
```

```
let values = Object.values(course);  
console.log(values); // [ 'JS Core', 'Open Source' ]  
if (values.includes('JS Core')) {  
    console.log("Found 'JS Core' value");  
}
```

# For... in Loop

- **for ... in** - iterates a specified variable over all the enumerable properties of an object



```
let obj = {a: 1, b: 2, c: 3};  
for (const key in obj) {  
  console.log(`obj.${key} = ${obj[key]}`);  
}
```

*// Output:*

*// "obj.a = 1"*

*// "obj.b = 2"*

*// "obj.c = 3"*

- The **for...of** statement creates a loop iterating over iterable objects

```
let obj = {a: 1, b: 2, c: 3};  
for (const key of Object.keys(obj)) {  
  console.log(`obj.${key} = ${obj[key]}`);  
}  
// "obj.a = 1"  
// "obj.b = 2"  
// "obj.c = 3"
```

```
for (const val of Object.values(obj)) {console.log(val);}  
// 1  
// 2  
// 3
```



**Bootstrap**

# What is a Responsive Design?

- **Presentation layers** that adjust according to the screen size of the different devices





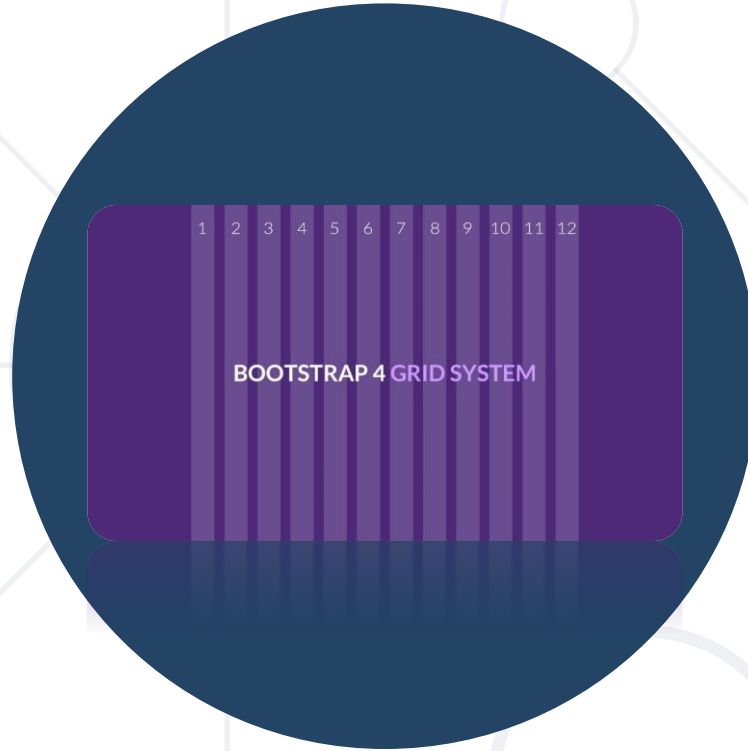
- World's most popular front-end **component library**
- Open source toolkit for developing with **HTML**, **CSS**, and **JS**
- Works with
  - Responsive **grid system**
  - Extensive prebuilt **components**
  - Powerful plugins built on jQuery



- Be sure to place **jQuery** and **Popper** first, as the Bootstrap code depends on them

JavaScript, Popper.js  
and jQuery

```
<script  
src="https://code.jquery.com/jquery3.3.1.slim.min.js"></script>  
<script  
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>  
<script  
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
```



# Bootstrap Grid System

Build Layouts with Grid – Twelve Column System

# Bootstrap Grid System Demo

index.html

```
...  
<div class="container">  
  <div class="row">  
    <div class="col-xs m-3">Column one</div>  
    <div class="col-xs m-3">Column two</div>  
    <div class="col-xs m-3">Column three</div>  
  </div>  
</div>  
...
```

Container

Row

Columns



← → ↻ 🏠 🔍 index.html

Column one

Column two

Column three

- Rows must be placed in **containers**
  - **.container** has one fixed width for each screen size in bootstrap (**xs, sm, md, lg**)
  - **.container-fluid** expands to fill the available width

Responsive  
pixel width

width:  
100%



- Determines **how many columns** to use on **different screen sizes**

index.html

```
<div class="col-sm-8 col-lg-4">Column one</div>  
<div class="col-sm-2 col-lg-4">Column two</div>  
<div class="col-sm-2 col-lg-4">Column three</div>
```

- **.col-xs**: width less than 768px
- **.col-sm**: width between 768px and 992px
- **.col-md**: width between 992px and 1200px
- **.col-lg**: width over 1200px



- Handful of **color utility classes**

index.html

```
<p class="text-primary">.text-primary</p>
<p class="text-secondary">.text-secondary</p>
<p class="text-success">.text-success</p>
<p class="text-danger">.text-danger</p>
<p class="text-warning">.text-warning</p>
<p class="text-info">.text-info</p>
<p class="text-light bg-dark">.text-light</p>
<p class="text-dark">.text-dark</p>
<p class="text-muted">.text-muted</p>
<p class="text-white bg-dark">.text-white</p>
```

.text-primary

.text-secondary

.text-success

.text-danger

.text-warning

.text-info

.text-light

.text-dark

.text-muted

.text-white

- Easily set the **background** of an element to any contextual **class**

index.html

```
<div class="bg-primary text-white">.bg-primary</div>
<div class="bg-secondary text-white">.bg-secondary</div>
<div class="bg-success text-white">.bg-success</div>
<div class="bg-danger text-white">.bg-danger</div>
<div class="bg-warning text-dark">.bg-warning</div>
<div class="bg-info text-white">.bg-info</div>
<div class="bg-light text-dark">.bg-light</div>
<div class="bg-dark text-white">.bg-dark</div>
<div class="bg-white text-dark">.bg-white</div>
```

.bg-primary

.bg-secondary

.bg-success

.bg-danger

.bg-warning

.bg-info

.bg-light

.bg-dark

.bg-white





# Bootstrap Components

- Custom **button styles** with support for multiple sizes, states, and more



index.html

```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
. . .
```

Documentation: <https://getbootstrap.com/docs/4.0/components/buttons/>

- Provide contextual feedback messages for typical user actions with the handful of flexible **alert messages**

```
<div class="alert alert-success alert-dismissible">  
  <a class="close" data-dismiss="alert" aria-label="close">x</a>  
  <strong>Success!</strong>  
  This alert box could indicate a successful or positive action.  
</div>
```

...

← → ↻ 🔍 index.html

**Success!** This alert box could indicate a successful or positive action.



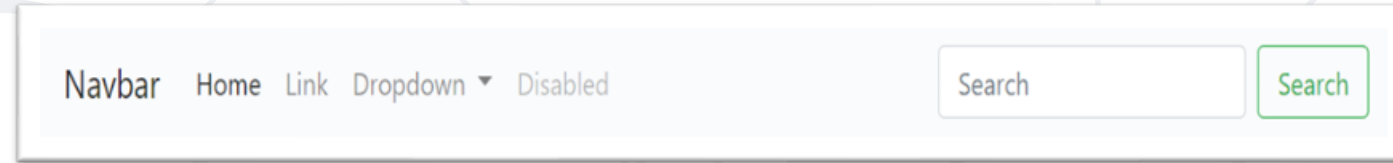
**Danger!** This is a danger alert! Be careful!



**Warn!** This is a warning alert!

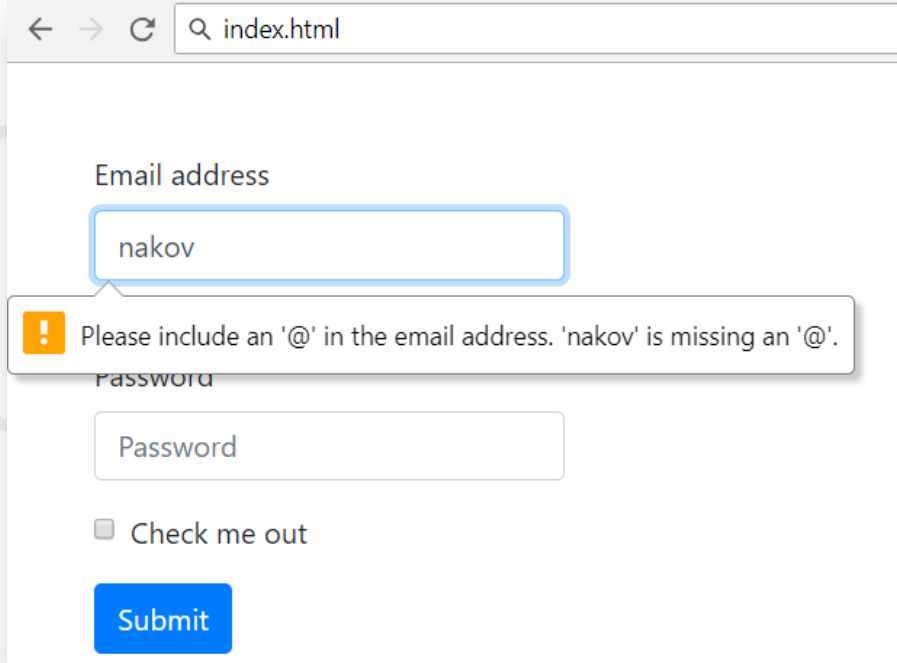


- Require a wrapping **.navbar**
- **Responsive** by default
- Come with built-in support for a handful of **sub-components**
  - **.navbar-brand** for your company, product, or project name
  - **.navbar-nav** for a full-height and lightweight navigation
  - **.nav-item** for every item in navigation



See more at: <https://getbootstrap.com/docs/4.0/components/navbar/>

- Form **control styles**, **layout options** and custom **components** for creating a wide variety of forms
- Use **type** attribute on all inputs to take advantage of newer input controls
  - Email **verification**
  - Number **selection**



index.html

Email address

nakov

! Please include an '@' in the email address. 'nakov' is missing an '@'.

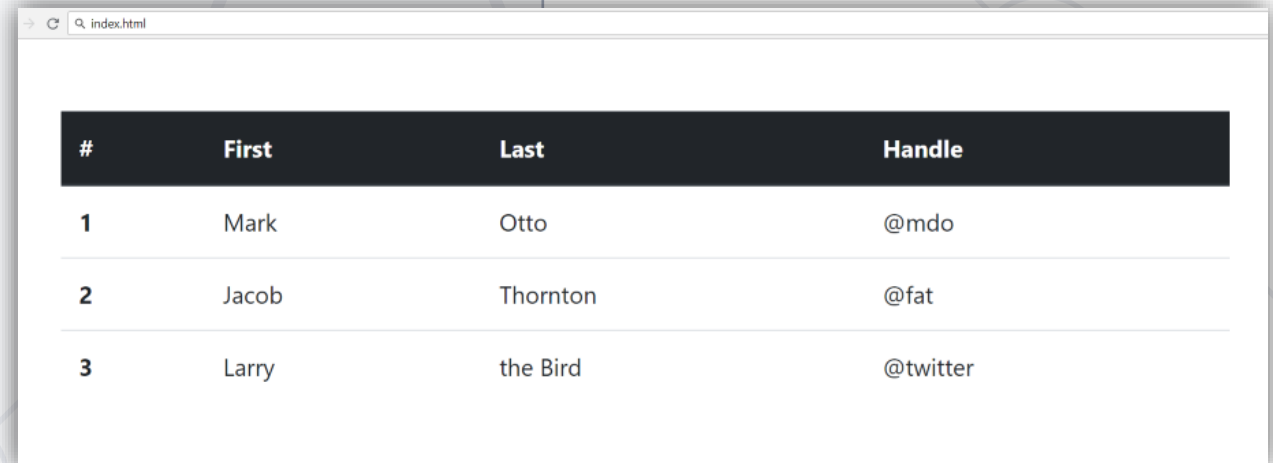
Password

Password

☐ Check me out

Submit

```
<table class="table">
  <thead class="thead-dark">
    <tr> <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th> </tr>
  </thead>
  <tbody>
    <tr> <th scope="row">1</th>
      <td>Mark</td>
      <td>Otto</td>
      <td>@mdo</td> </tr>
    <tr> . . . </tr>
    <tr> . . . </tr>
  </tbody>
</table>
```



#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

- **Lightweight, flexible** component for showcasing hero unit style content

```
<div class="jumbotron">
  <h1 class="display-4">Hello,
world!</h1>
  <p class="lead">This is a ...</p>
  <hr class="my-4"><p>It uses ...</p>
  <p class="lead">
  <a class="btn btn-primary btn-lg">
Learn more</a>
  </p>
</div>
```

Hello, world!

This is a simple hero unit, a simple jumbotron-style component for calling extra attention to featured content or information.

It uses utility classes for typography and spacing to space content out within the larger container.

[Learn more](#)

See more at: <https://getbootstrap.com/docs/4.0/components/jumbotron/>

- JS is a **dynamic programming language**
- Functions in JS
- JS objects hold **key-value pairs**
- Bootstrap is the most popular front-end **component library**

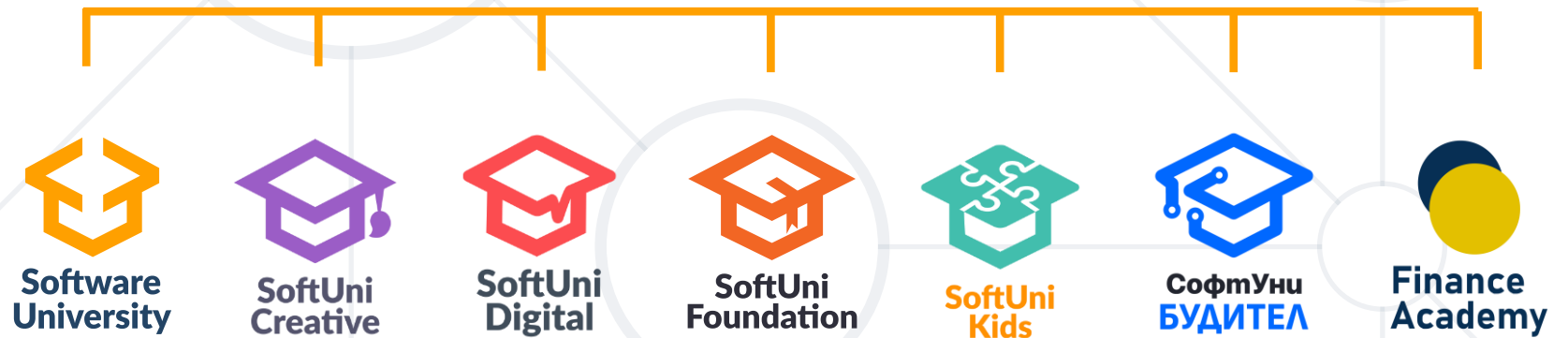




# Questions?



SoftUni



# SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [about.softuni.bg](http://about.softuni.bg)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)



- 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>

