# JavaScript

## jQuery, AJAX

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

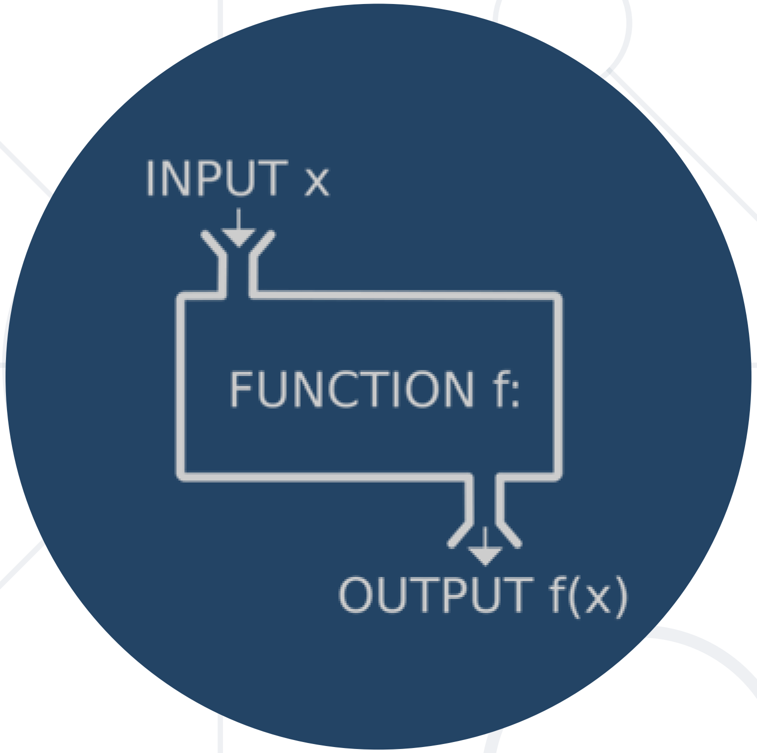
## Table of Contents

## Have a Question?

SoftUni Foundation

sli.do

#java-web



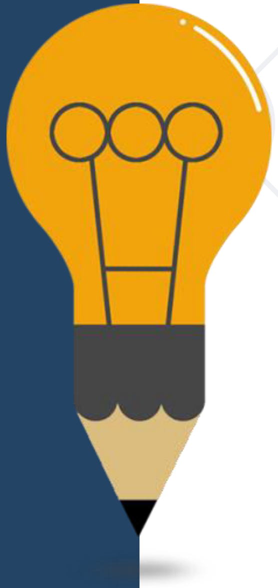
# JavaScript

## Functions - Syntax, Invocation, Return, Functions as values

# JavaScript Syntax

The JavaScript syntax is similar to C#, Java and PHP

- Operators, Variables, Conditional statements, loops, functions, arrays, objects and classes



Declare a variable with let

Conditional statement

```
let a = 5;
let b = 10;
if (b > a) {
  console.log(b);
}
```

Body of the conditional statement

## Functions in JS

- Why Functions?
  - You can **reuse** code, define **once**, use **many times**.
- **Function** = block of code
  - Can take **parameters** and return **result**

Function name: use **camelCase**

Function **parameters**: use **camelCase**

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

The **{** stays at the same line

```
printStars(10);
```

**Invoke** the function

## Functions in JS (2)

```
function sum (a, b) {
  console.log(a + b);
}
sum (5, 6);             // 11
```

**Invoke** the function with different parameters value

```
function sum (a, b = 3) {
  console.log(a + b);
}
sum (11);               // 14
```

**Default** function parameters

```
function sum (a, b) {
  return a + b;
}
let c = sum (5.8, 3);
console.log (c);        // 8.8
```

**Return** ends function execution

# Function Declaration, Expression, Arrow

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

Function **Declaration**

```
let solve = function walk() {
    console.log('walking');
}
solve();         // walking
```

Function **Expression**

```
let solve = () => {
    console.log('walking');
}
solve();         // walking
```

**Arrow** function

## Function Invocation

- The code inside a function is executed when the function is **invoked** with **()** operator

```
function write(name) {
     console.log(`I am ${name}`);
}
write('George');       // I am George
```

**Invoke** the function **after** declaration

```
write('George');       // I am George
function write(name) {
     console.log(`I am ${name}`);
}
```

**Invoke** the function **before** declaration

# Function Invocation (2)

```
(function (count) {
  for (let i = 1; i <= count; i++)
    console.log('+'.repeat(i));
})(4);
```

Anonymous **self-invoking** function

```
+
++
+++
++++
```

```
let f = (function () {
  let x = 0;
  return function() { console.log(++x); }
})(); f(); f(); f(); f();
```

This is called "**closure**" (a state is closed inside)

```
1
2
3
4
```

## Function Return

- Return statement **ends** function execution

```
function getRectArea(width, height) {
  if (width > 0 && height > 0) {
    return width * height;
  }
  return 0;
}
console.log(getRectArea(3, 4));          // 12
console.log(getRectArea(-3, 4));         // 0
```

```
let result = function (a, b) {
      return a % b;
};
console.log(result(10, 3));              // 1
```

## Variables Holding Functions

- In JS variables can hold functions as their values

```
let f = function(x) { return x * x; }
console.log(f(3)); // 9
console.log(f(5)); // 25
f = function(x) { return 2 * x; }
console.log(f(3)); // 6
console.log(f(5)); // 10
f = undefined;
console.log(f(3)); // TypeError: f is not a function(…)
```

# Functions as Parameters

SoftUni Foundation

```
function repeatIt(count, func) {
  for (let i = 1; i <= count; i++)
    func(i);
}

let starsFunc = function(i) {
  console.log('**'.repeat(i))
};

repeatIt(3, starsFunc);

repeatIt(3, function(x) {

      console.log(2 * x);

});
```

```
**
****
******
```

```
2
4
6
```

## jQuery

## jQuery Selectors

jQuery selection is the same as the **querySelector** in Vanilla JS

- All selector

```
$('*') // Selects all elements
```

- Class selector

```
$('.class') // Select by class name
```

- Element selector

```
$('section') // Select by tag name
```

- Id selector

```
$('#id') // Selects a element by given id
```

- Multi-selector

```
$('selector1, selector2') // Combined
```

## Adding Elements with jQuery

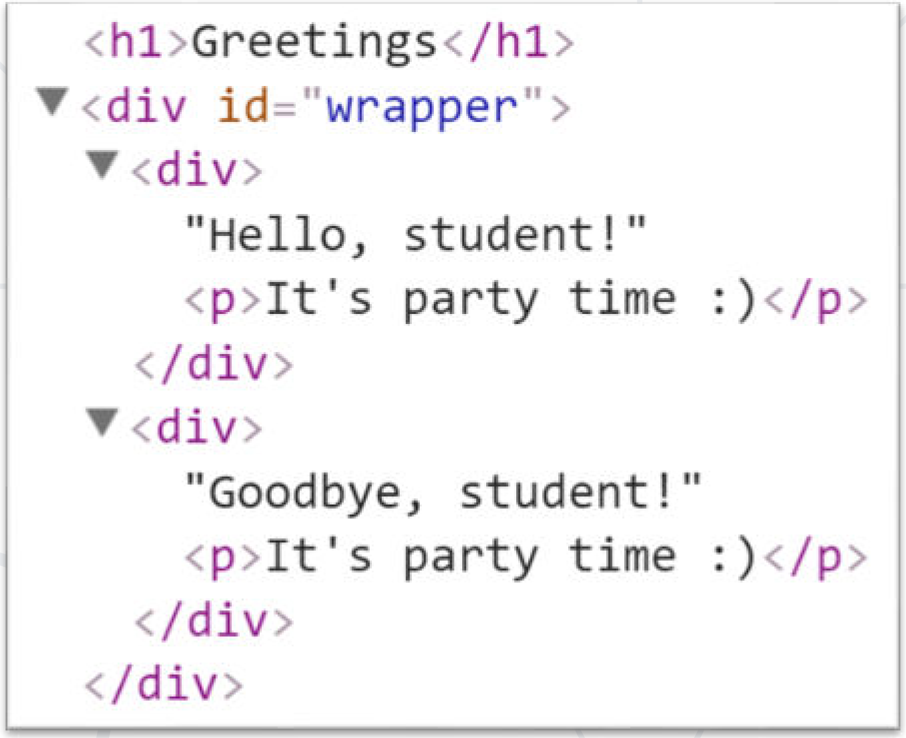
Select the parent element, then use:

- **append() / prepend()**
- **appendTo() / prependTo()**

```html
<div id="wrapper">
  <div>Hello, student!</div>
  <div>Goodbye, student!</div>
</div>
```

```javascript
$('#wrapper div').append("<p>It's party time :)</p>");
```

```javascript
$('<h1>Greetings</h1>').prependTo('body');
```



```
  <h1>Greetings</h1>
▼ <div id="wrapper">
  ▼ <div>
      "Hello, student!"
      <p>It's party time :)</p>
    </div>
  ▼ <div>
      "Goodbye, student!"
      <p>It's party time :)</p>
    </div>
  </div>
```

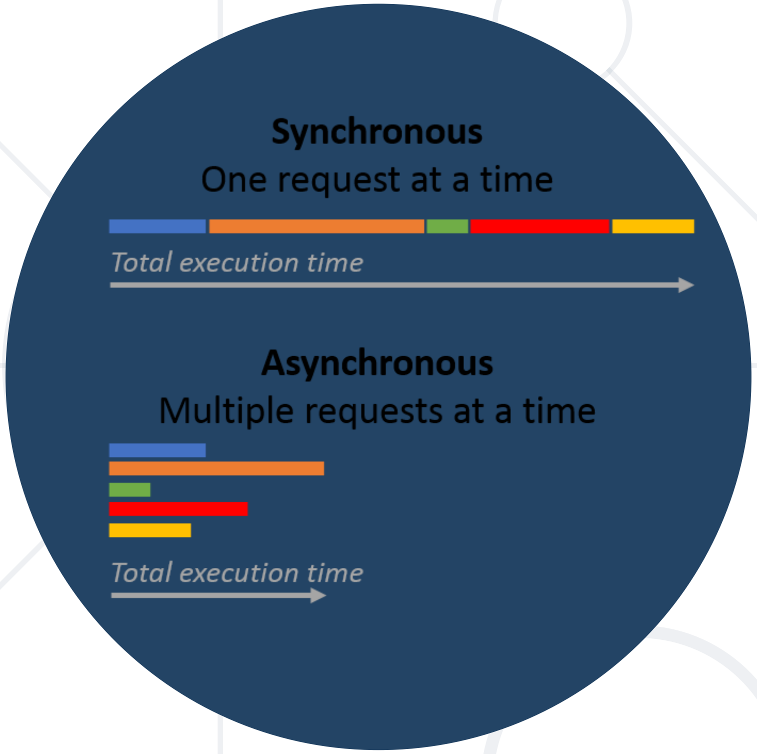
## jQuery Events: Attach / Remove

- Attaching events on certain elements

```
$('a.button').on('click', buttonClicked);
function buttonClicked() {
  $('.selected').removeClass('selected');
  $(this).addClass('selected');
  // "this" is the event source (the hyperlink clicked)
}
```

- Removing event handler from certain elements

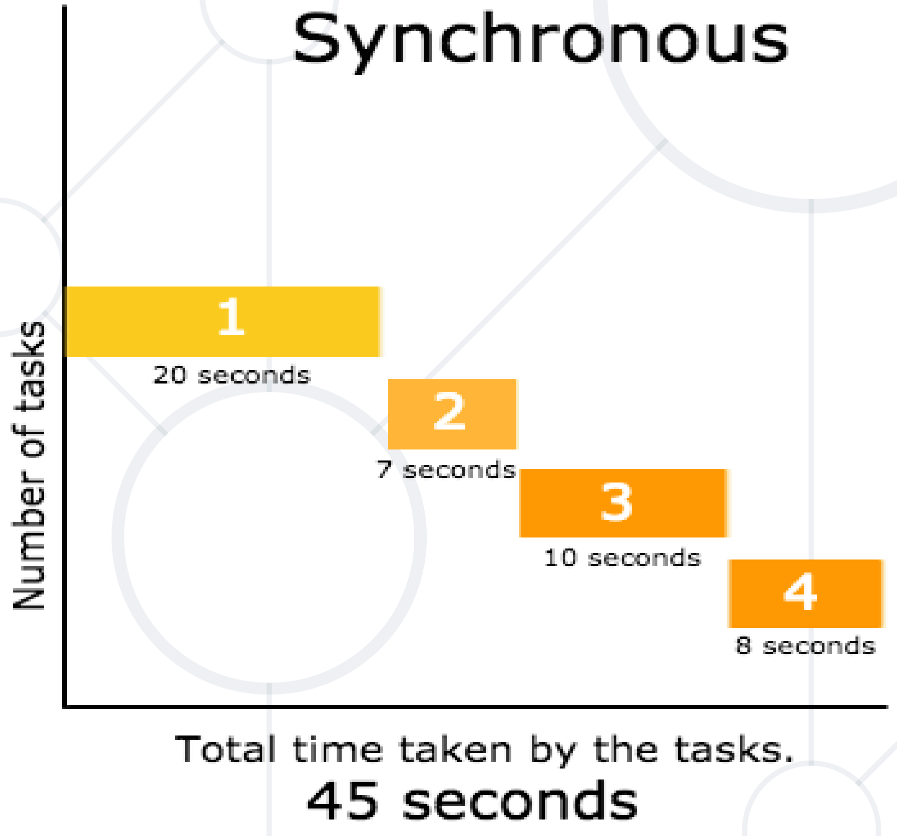
```
$('a.button').off('click', buttonClicked);
```

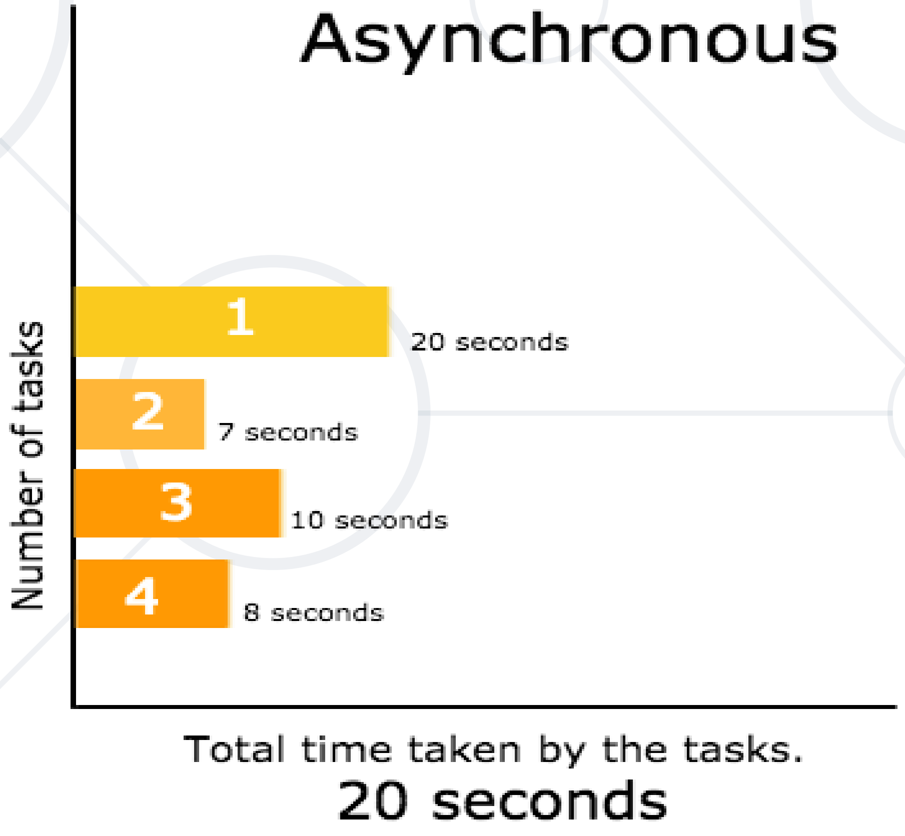


# Asynchronous Programming

# Asynchronous Programming

- **Asynchronous programming** deals with the needs to run several tasks (pieces of code) in parallel, in the same time





## Asynchronous Programming

- Synchronous programming
  - Each line of code is executed only when the preceding line has finished it’s work
- Asynchronous Programming
  - In asynchronous programs, you can have two lines of code (L1 followed by L2), where L1 schedules some task to be run in the future, but L2 runs before that task completes
  - The tasks may finish in order of execution but it is not guaranteed

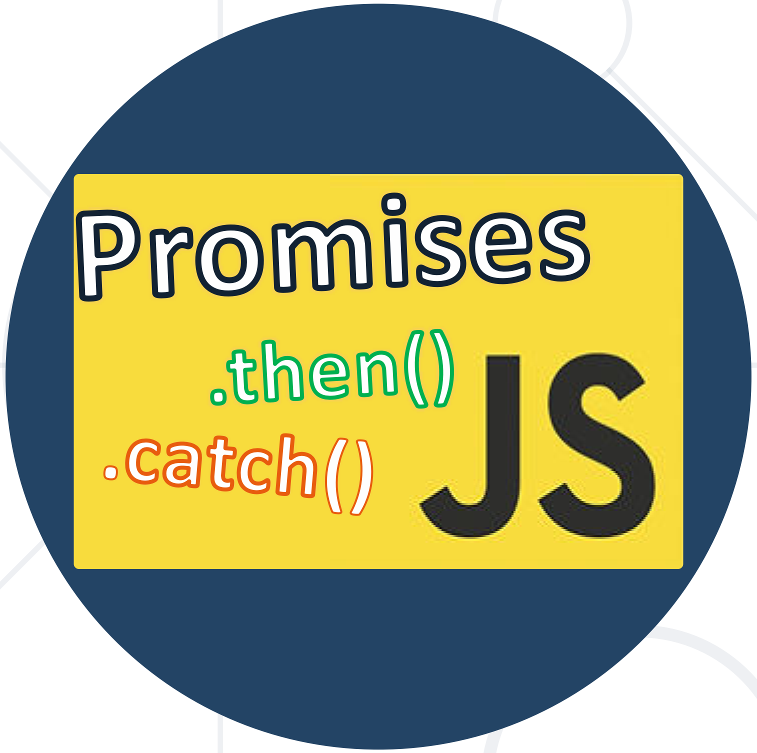
## Asynchronous Programming – Example

```
// Say "Hello."
console.log('Hello.');
// Say "Goodbye" two seconds from now.
setTimeout(function() {
  console.log('Goodbye!');
}, 2000);
// Say "Hello again!"
console.log('Hello again!');
```

```
function getData() {
  var data;
  $.get('example.php',
    function(response) {
      data = response;
    }
  );
  return data;
}

var data = getData();
console.log('The data is: ' + data);
```

- What is wrong with these examples?



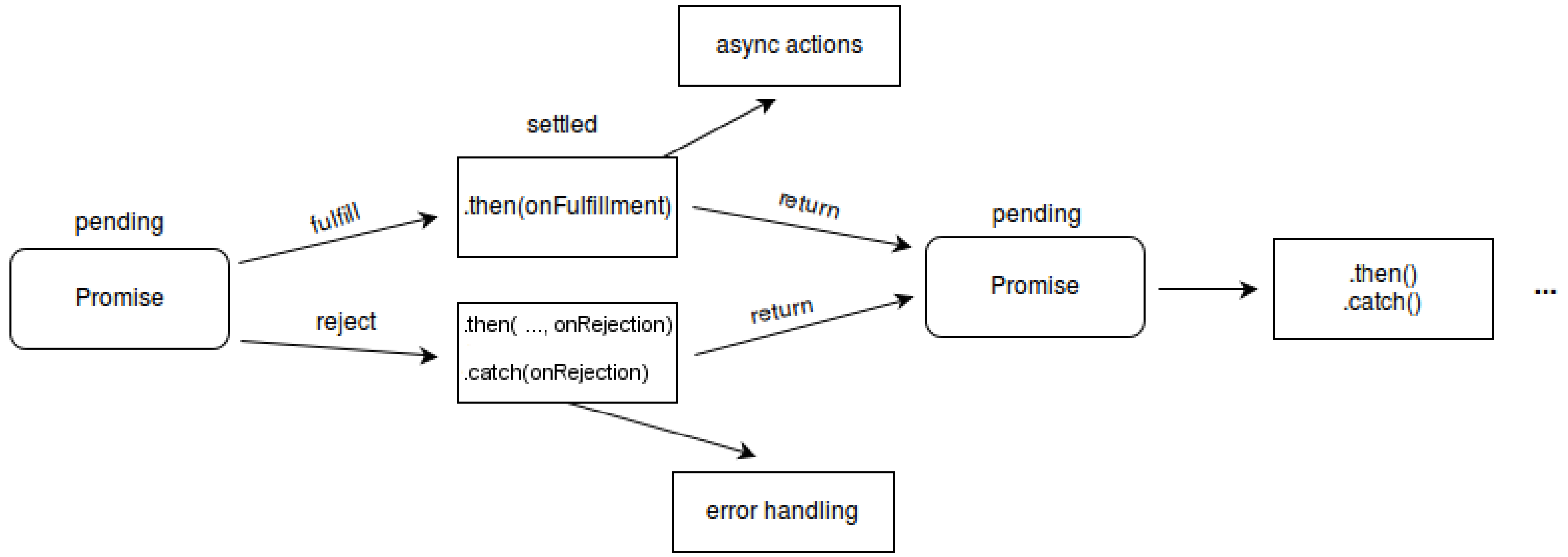
## Promises in JS

### Objects Holding Asynchronous Operations

# What is a Promise?

- A promise is an object holding an asynchronous operation
  - A result which may be available now, or in the future, or never
  - Promises let asynchronous methods return values like synchronous methods, instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future
- Promises may be in one of these states:
  - **Pending** – operation still running (unfinished)
  - **Fulfilled** – operation finished (and the result is available)
  - **Failed** – operation is failed (and an error is available)
- Promises in JS use the **Promise** object

# What is a Promise?



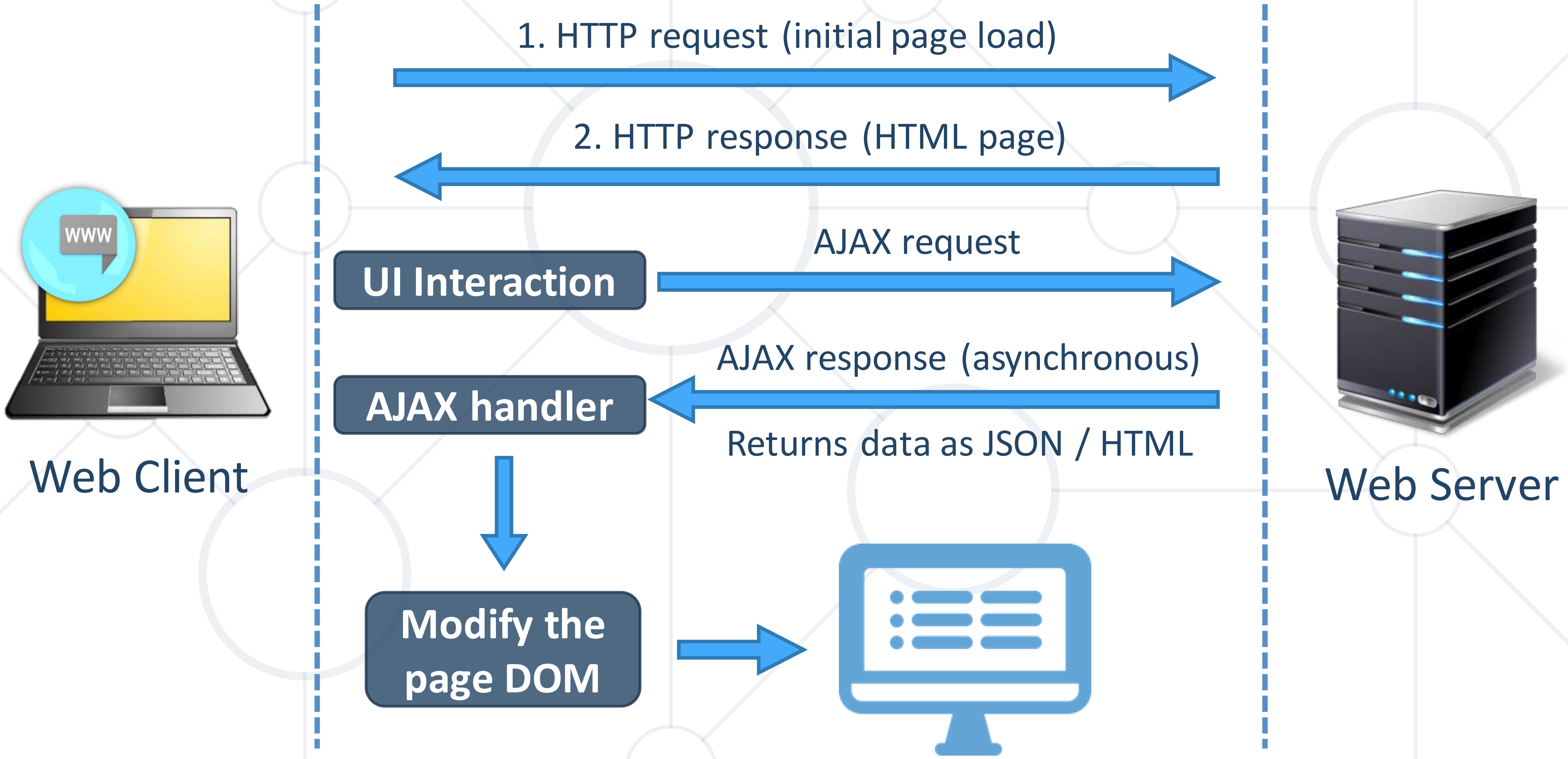
## AJAX

Asynchronous JavaScript And XML

## What is AJAX?

- AJAX == **Asynchronous JavaScript And XML**
  - A broad group of Web technologies that can be used to implement a Web application that communicates with a server in the background, without interfering with the current state of the page
    - HTML (or XHTML) and CSS for presentation
    - The Document Object Model (DOM) for dynamic display of and interaction with data
    - JSON or XML for the interchange of data, and XSLT for its manipulation
    - The XMLHttpRequest object for asynchronous communication
    - JavaScript to bring these technologies together

# AJAX: Workflow


1. HTTP request (initial page load)
2. HTTP response (HTML page)
UI Interaction
AJAX request
AJAX response (asynchronous)
AJAX handler
Returns data as JSON / HTML
Modify the page DOM
Web Client
Web Server
www

## jQuery AJAX

- **jQuery** simplifies how developers make AJAX calls
- Low-Level Interface
  - jQuery.ajax() - Perform an asynchronous HTTP (Ajax) request.
  - jQuery.ajaxPrefilter() - Handle custom Ajax options or modify existing options before each request is sent and before they are processed by $.ajax().
  - jQuery.ajaxSetup() - Set default values for future Ajax requests. Its use is not recommended.
  - jQuery.ajaxTransport() - Creates an object that handles the actual transmission of Ajax data.

## jQuery AJAX

SoftUni Foundation

- Shorthand Methods
  - jQuery.get() - Load data from the server using a HTTP GET request.
  - jQuery.getJSON() - Load JSON-encoded data from the server using a GET HTTP request
  - jQuery.getScript() - Load a JavaScript file from the server using a GET HTTP request, then execute it.
  - jQuery.post() - Load data from the server using a HTTP POST request.
  - .load() - Load data from the server and place the returned HTML into the matched element.

# jQuery AJAX

SoftUni Foundation

- Global Ajax Event Handlers
  - .ajaxComplete() - Register a handler to be called when Ajax requests complete
  - .ajaxError() - Register a handler to be called when Ajax requests complete with an error
  - .ajaxSend() - Attach a function to be executed before an Ajax request is sent
  - .ajaxStart() - Register a handler to be called when the first Ajax request begins
  - .ajaxStop() - Register a handler to be called when all Ajax requests have completed
  - .ajaxSuccess() - Attach a function to be executed whenever an Ajax request completes successfully

## jQuery – GET & POST

```
$.ajax('myservice/username', {
    data: {
        id: 'some-unique-id'
    }
})
.then(
    function success(name) {
        alert('User\'s name is ' +
name);
    },

    function fail(data, status) {
        alert('Request failed.
Returned status of ' + status);
    }
);
```

```
var newName = 'John Smith';

$.ajax('myservice/username?' + $.param({id: 'some-
unique-id'}), {
    method: 'POST',
    data: {
        name: newName
    }
})
.then(
    function success(name) {
        if (name !== newName) {
            alert('Something went wrong.  Name is
now ' + name);
        }
    },

    function fail(data, status) {
        alert('Request failed.  Returned status of
' + status);
    }
);
```

## Fetch API

## Fetch API

- Fetch provides a generic definition of Request and Response objects
- They can be used:
  - When they are needed
  - For service workers
  - Cache API
  - Similar things that handle or modify requests and responses

# Fetch API (Demo)

```java
@GetMapping("/")
public ModelAndView index(ModelAndView modelAndView) {
    modelAndView.setViewName("index");
    return modelAndView;
}

@GetMapping(value = "/fetch", produces = "application/json")
@ResponseBody
public Object fetchData() {
    return new ArrayList<Product>() {{
        add(new Product(){{
            setName("Chewing Gum");
            setPrice(new BigDecimal(1.00));
            setBarcode("133242556222");
        }});
        ...
    }};
}
```

HomeController.java

```java
public class Product {
    private String name;
    private BigDecimal price;
    private String barcode;

    // Getters & Setters
    ...
}
```

Product.java

## Fetch API (Demo)

- Now let’s head to the **view**
  - There is no need for a separate **.js** file for one-time use

index.html

```
...
<div class="container-fluid">
    <h1 class="text-center mt-5 display-1">Data Fetch</h1>
    <div class="data-container mt-5"></div>
    <div class="button-holder mt-5">
        <button id="fetch-button" class="btn btn-info">Fetch Data</button>
        <button id="clear-button" class="btn btn-secondary">Clear Data</button>
    </div>
</div>
<script>
    // jQuery Event handlers
    $('#fetch-button').click(() => {...}); // Fetch and render the data
    $('#clear-button').click(() => $('.data-container').empty()); // Clear the data
</script>
```

# Fetch API (Demo)

```
$('#fetch-button').click(() => {
    fetch('http://localhost:8000/fetch') // Fetch the data (GET request)
        .then((response) => response.json()) // Extract the JSON from the Response
        .then((json) => json.forEach((x, y) => { // Render the JSON data to the HTML
                if (y % 4 === 0) {
                    $('.data-container').append('<div class="row d-flex justify-content-
around mt-4">');
                }

                let divColumn =
                    '<div class="col-md-3">' +
                    '<h3 class="text-center font-weight-bold">' + x.name + '</h3>' +
                    '<h4 class="text-center">Price: $' + x.price + '</h4>' +
                    '<h4 class="text-center">Barcode: $' + x.barcode + '</h4>' +
                    '</div>';

                $('.data-container .row:last-child').append(divColumn);
            }));
});
```

## AJAX & Fetching Data

### Live Demonstration

## Summary

- JavaScript supports conditional statements and loops
- JavaScript objects hold **key-value pairs**
- jQuery is a fast, small, and feature-rich JavaScript library that can easily manipulate HTML documents
- **Asynchronous programming** deals with the needs to run several tasks (pieces of code) in parallel
- AJAX == **Asynchronous JavaScript And XML**
- Fetch provides a generic definition of Request and Response objects

# Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

# SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech we know sports

telenor

SoftwareGroup doing it right

NETPEAK

SmartIT

Postbank Решения за твоето утре

SUPER HOSTING ®.BG

INDEAVR Serving the high achievers

INFRAGISTICS®

LIEBHERR

æternity

codexio

# SoftUni Organizational Partners

SoftUni Foundation

IS ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

WORLD OF MYTHS

Lukanet.com

codexio

# Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

## License