WEB DESIGN AND PROGRAMMING

INTRODUCTION TO FRONT-END TECHNOLOGIES 4

FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ

ASST. PROF. DR. SAMET KAYA

skaya@fsm.edu.tr

kysamet@gmail.com

Front-End Technologies Overview

- HTML - Structure and Content
- CSS - Design and Styling
- JavaScript - Interactivity and Behavior
- Bootstrap - Rapid Development Framework
- Thymeleaf - Server-Side Template Engine

HTML: HyperText Markup Language

- Developed by Tim Berners-Lee at CERN (early 1990s)
- Markup language, not a programming language
- Defines semantic structure of web content
- Uses tags between angle brackets: **<tag>content</tag>**
- Foundation of every web page

HTML Document Structure

```
<!DOCTYPE html>

<html lang="tr">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-
width, initial-scale=1.0">

    <title>Page Title</title>

</head>

<body>

    <h1>Hello World!</h1>

    <p>This is first page.</p>

</body>

</html>
```

- **<!DOCTYPE html>** - HTML5 declaration
- **<html>** - Root element
- **<head>** - Meta-information
- **<body>** - Visible content

Basic HTML Elements: Text Structure

- Headings: **<h1>** to **<h6>**
  - **<h1>** - Highest level (use once per page)
  - **<h2>** to **<h6>** - Hierarchical subsections
- Paragraphs: **<p>** tag
- Text Emphasis:
  - **<strong>** - Strong importance (bold)
  - **<em>** - Emphasis (italic)
- Semantic tags preferred over visual tags

Why Do Enterprise Frameworks Exist?

- Key Motivations
  - Abstraction of Infrastructure Problems: Frameworks handle repetitive infrastructure concerns, allowing developers to focus on business logic rather than plumbing code.
  - Standardized Programming Models: Enable team collaboration and long-term maintainability through consistent patterns and conventions.
  - Cloud-Native Readiness: Provide built-in support for containers, Kubernetes, CI/CD pipelines, and observability requirements.
  - Reduced Time-to-Market: Pre-built solutions for common problems accelerate development cycles and reduce bugs.

Hyperlinks and Images

- **Hyperlinks (<a> tag):**

```
<a href="https://www.abacus.ai/">Visit
Abacus.AI </a>
```

```
<a href="/about.html">About Us</a>
```

```
<a href="#contact">Goto Contact </a>
```

- **Images (<img> tag):**
  - src - Image file path
  - alt - Alternative text (accessibility & SEO)

```
<img  src="images/view.jpg"  alt="sunny
day" width="500" height="300">
```

HTML Lists

- Unordered Lists (<ul>):

```
<ul>
    <li>Apple</li>
    <li>Banana</li>
</ul>
```

- Ordered Lists (<ol>):

```
<ol>
    <li>Fill The Form</li>
    <li>Clict to Submit</li>
</ol>
```

- Definition Lists (<dl>):

```
<dl>
    <dt>Fill The Form</dt>
    <dd>Enter your info</dd>
    <dt>Click to Submit</dt>
    <dd>Press the submit.</dd>
</dl>
```

Tables for Structured Data

```
<table>
    <thead>
        <tr>
            <th>Name</th>
            <th>Department</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Ahmet Yılmaz</td>
            <td>Engineering</td>
        </tr>
        <tr>
            <td>Ayşe Kaya</td>
            <td>Manager</td>
        </tr>
    </tbody>
</table>
```

- **Key Elements:**
  - **<table>** - Container
  - **<thead>, <tbody>, <tfoot>** - Semantic sections
  - **<tr>** - Table row
  - **<th>** - Header cell
  - **<td>** - Data cell

Forms: Collecting User Input

```
<form action="/register" method="POST">
    <label for="username">User Name:</label>
    <input type="text" id="username"
name="username" required>

    <label for="email_addres">E-mail:</label>
    <input type="email" id="email_addres"
name=" email_addres " required>

    <button type="submit">Register</button>
</form>
```

❖ **Key Attributes:**

- ❖ **action** - Where to send data
- ❖ **method** - How to send (GET/POST)
- ❖ **type** - Input type
- ❖ **required** - Validation

Semantic HTML: Adding Meaning to Structure

```
<header>
    <h1>Web Site</h1>
    <nav>
        <ul>
            <li><a href="/">Main Page</a></li>
            <li><a href="/abotu">About</a></li>
            <li><a href="/contact">Contact</a></li>
        </ul>
    </nav>
</header>
<main>
    <article>
        <h2>Article Title</h2>
        <p> Article Content...</p>
    </article>
</main>
<footer>
    <p>&copy; 2025</p>
</footer>
```

- **Semantic Elements:**
  - **<header>** - Header content
  - **<nav>** - Navigation links
  - **<main>** - Main unique content
  - **<section>** - Thematic section
  - **<article>** - Independent content
  - **<aside>** - Related side content
  - **<footer>** - Footer information

CSS - Design and Styling

- **CSS**: Cascading Style Sheets
- Controls visual appearance of HTML elements
- **Defines**: colors, fonts, sizes, positioning, animations
- "**Cascading**" = multiple rules can apply with priority order
- Separates content (HTML) from presentation (CSS)

Three Ways to Include CSS

```
<head>
    <link rel="stylesheet" href="styles.css">
</head>
```

- **External** - Separate .css file (recommended)

```
<head>
    <style>
        p {
            color: blue;
            font-size: 16px;
        }
    </style>
</head>
```

- **Internal** - <style> tag in <head>

```
<p  style="color:  blue;  font-size:  16px;">Blue
Text</p>
```

- **Inline** - style attribute in HTML tag

CSS Selectors: Targeting Elements

- Element Selector:

```
p {color: black;}
```

- Class Selector (.):

```
.highlight { background-color: yellow; }
```

- ID Selector (#):

```
#title { font-size: 24px;}
```

- Combined Selectors:

```
div p { margin: 10px; }  /* Descendant */
div > p { padding: 5px; } /* Child */
a:hover { color: red; }   /* Pseudo-class */
```

Typography and Text Properties

```
.mtext {
    color: #333;
    font-family: Arial, sans-serif;
    font-size: 16px;
    font-weight: bold;
    font-style: italic;
    text-align: center;
    text-decoration: underline;
    line-height: 1.5;
    letter-spacing: 2px;
}
```

- **Key Properties:**
  - Color, font family, size, weight
  - Alignment, decoration, spacing
  - Line height for readability

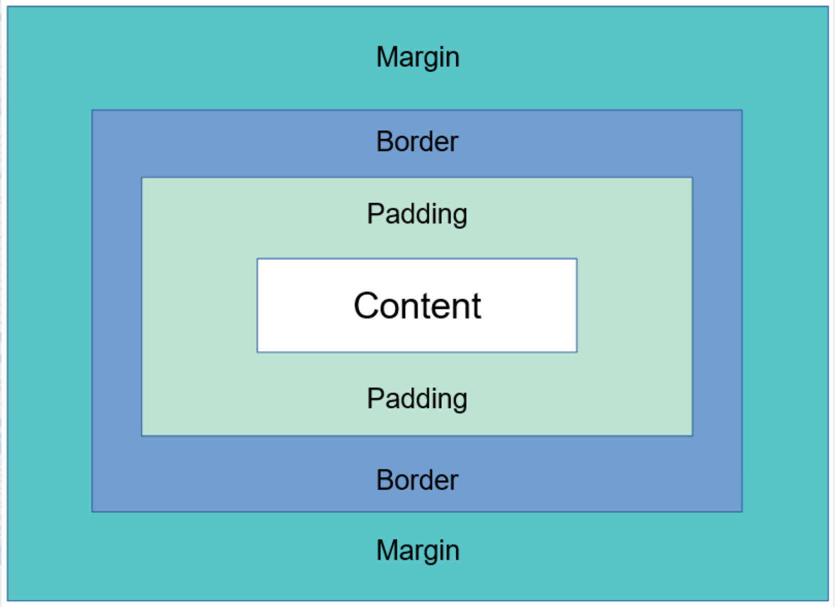
CSS Colors and Backgrounds

```
.mbox{
    background-color: #f0f0f0;
    background-image: url(imagex.jpg');
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
}
```

- **Color Formats:**
  - **Name**: red, blue, green
  - **Hex**: #FF0000, #00F
  - **RGB**: rgb(255, 0, 0)
  - **RGBA**: rgba(255, 0, 0, 0.5) (with opacity)
  - **HSL**: hsl(0, 100%, 50%)

The Box Model: Understanding Element Spacing

```
.box {
    width: 300px;
    height: 200px;
    padding: 20px;      /* Inner space */
    border: 2px solid black;
    margin: 10px;       /* Outer space */
}
```



- **Four Layers:**
  - **Content** - The actual content
  - **Padding** - Space between content and border
  - **Border** - The element's border
  - **Margin** - Space between element and others

Display: Controlling Element Behavior

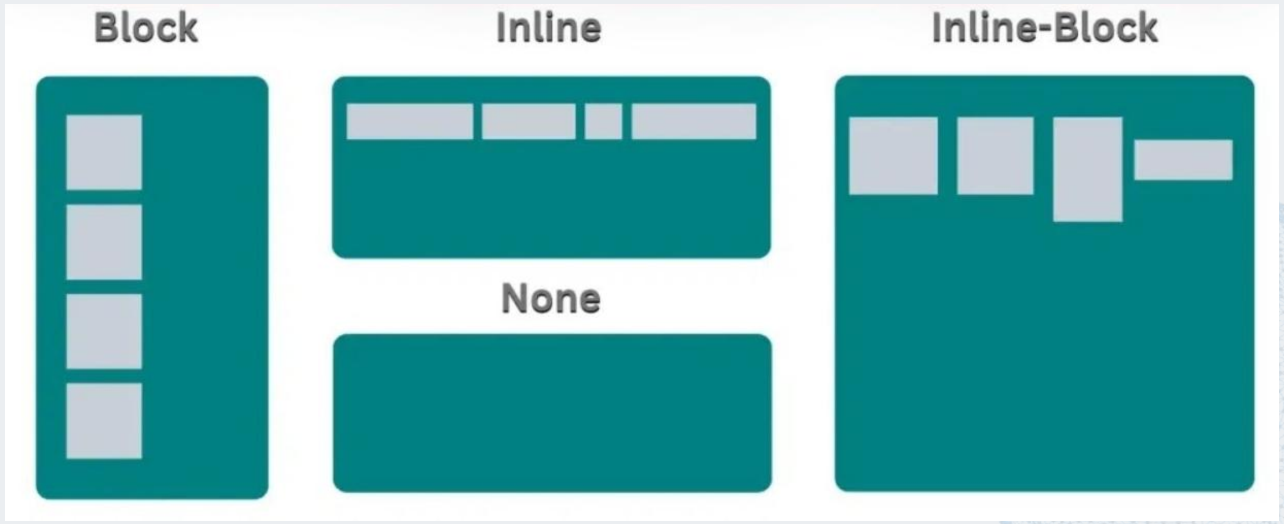
```
/* Full width, new line */
.block {
    display: block;
}

/* Content width, same line */
.inline {
    display: inline;
}

/* Hybrid */
.inline-block {
    display: inline-block;}

/* Hides element */
.hidden {
    display: none;
}
```

- ❖ **Display Types:**
  - ❖ **Block** - Takes full width, starts on new line
  - ❖ **Inline** - Takes only content width, stays in line
  - ❖ **Inline-block** - Combination of both
  - ❖ **None** - Completely hidden



Position: Controlling Element Placement

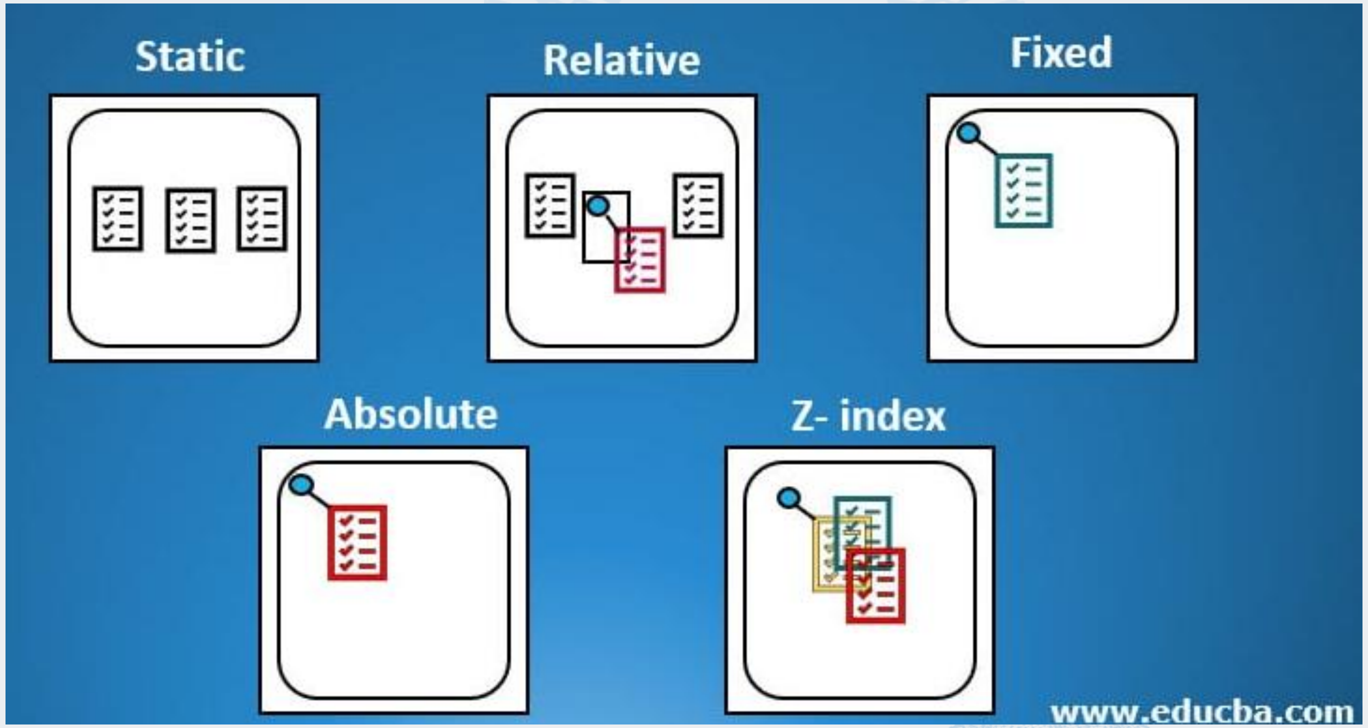
```
/* Default */
.static { position: static; }

/* Relative to normal position */
.relative {
    position: relative;
    top: 10px; left: 20px;
}

/* Relative to positioned parent */
.absolute {
    position: absolute;
    top: 0; right: 0;
}

 /* Fixed to viewport */
.fixed {
    position: fixed;
    bottom: 0; right: 0;
}

/* Sticky on scroll */
.sticky {
    position: sticky;
    top: 0;
}
```



Flexbox - Modern Layout System

```css
.container {
    display: flex;
    flex-direction: row;           /* row, column */
    justify-content: center;       /* Main axis alignment */
    align-items: center;           /* Cross axis alignment */
    flex-wrap: wrap;               /* Wrap to new line */
    gap: 10px;                     /* Space between items */
}

.item {
    flex: 1;                       /* Grow to fill space */
    flex-grow: 1;
    flex-shrink: 1;
    flex-basis: 200px;
}
```

- **Key Properties**:
  - Direction, justification, alignment
  - Wrapping, gaps, flexible sizing
  - One dimentional

CSS Grid - Two-Dimensional Layouts

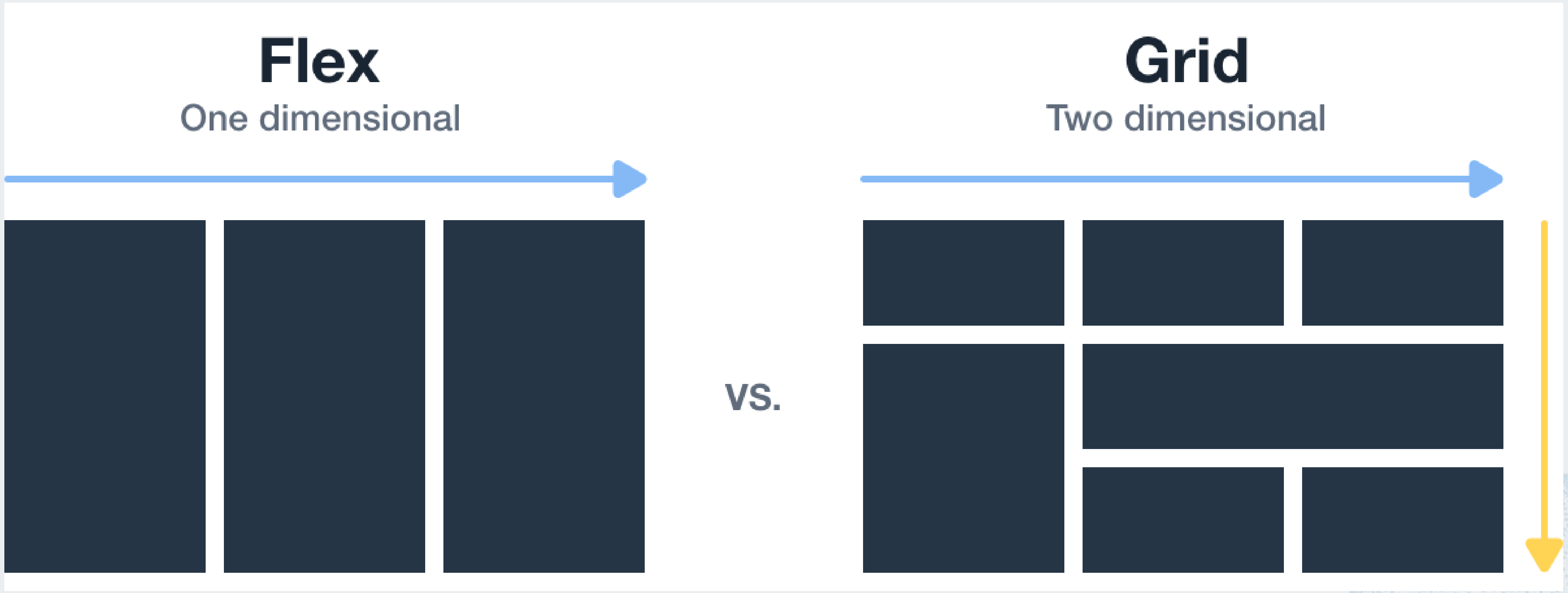
```
.grid-container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;  /* 3 columns */
    grid-template-rows: auto 1fr auto;   /* 3 rows */
    gap: 20px;

    grid-template-areas:
        "header header header"
        "sidebar main aside"
        "footer footer footer";
}

.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.main { grid-area: main; }
```

- **Key Properties**:
  - Complex layout system.
  - Two dimentional

Flexbox vs Grid



Responsive Design: Adapting to Screen Sizes

```css
/* Mobile-first approach */
.container {
    width: 100%;
    padding: 10px;
}

/* Tablet */
@media (min-width: 768px) {
    .container { width: 750px; }
}

/* Desktop */
@media (min-width: 1024px) {
    .container { width: 960px; }
}

/* Large screens */
@media (min-width: 1200px) {
    .container { width: 1140px; }
}
```

- **Responsive Units:**
    - **%** - Relative to parent
    - **em** - Relative to parent font-size
    - **rem** - Relative to root font-size
    - **vw/vh** - Viewport width/height

CSS Transitions and Animations

```
/*Transitions:*/
.button {
    background-color: blue;
    transition: background-color 0.3s ease;
}
.button:hover {
    background-color: darkblue;
}


/*Animations:*/
@keyframes slide {
    0% { transform: translateX(0); }
    50% { transform: translateX(100px); }
    100% { transform: translateX(0); }
}

.animated {
    animation: slide 2s infinite;
}
```

JavaScript: The Brain and Muscles of the Web

- Developed by Brendan Eich (1995, in 10 days!)
- Client-side programming language
- Runs in the user's browser
- Makes pages dynamic and interactive
- **Modern uses:**
  - **Browser (front-end)**
  - **Server (Node.js)**
  - **Mobile apps (React Native)**
  - **Desktop apps (Electron)**

Including JavaScript in HTML

1. External File (Best Practice):

```
<script src="script.js"></script>
```

2. Internal Script:

```
<script>
    console.log('Page loaded');
</script>
```

3. Inline (Avoid):

```
<button
onclick="alert('Hello')">Click</butto
n>
```

❖ **Best Practice:**

- ❖ Place <script> tags before closing </body>
- ❖ Keeps structure, presentation, and behavior separated
- ❖ Improves code reusability and maintainability

JavaScript Variables and Data Types

```
//Variable Declaration:
let variable = 'Can change';         // Block-scoped
const constant = 'Cannot change';  // Block-scoped, immutable
var old = 'Avoid using';             // Function-scoped (legacy)

//Primitive Data Types:
let name = "John";               // String
let age = 25;                    // Number
let isActive = true;             // Boolean
let notDefined;                  // Undefined
let empty = null;                // Null
let unique = Symbol('id');       // Symbol
let bigNum = 123n;               // BigInt

//Reference Types:
let arr = [1, 2, 3];             // Array
let obj = {name: 'John'};        // Object
let func = function() {};        // Function
```

JavaScript Operators

```
//Arithmetic:
5 + 3    // 8 (addition)
5 - 3    // 2 (subtraction)
5 * 3    // 15 (multiplication)
5 / 3    // 1.666... (division)
5 % 3    // 2 (modulus/remainder)
5 ** 3   // 125 (exponentiation)

//Comparison:
5 == '5'    // true (loose equality - avoid)
5 === '5'   // false (strict equality - use this)
5 !== '5'   // true (strict inequality)
5 > 3       // true

//Logical:
true && false   // false (AND)
true || false   // true (OR)
!true           // false (NOT)

//Assignment:
x += 5;   // x = x + 5
x++;      // x = x + 1
```

Conditional Logic: If/Else Statements

```
//If-Else:
let age = 18;

if (age >= 18) {
    console.log('You are an adult');
} else if (age >= 13) {
    console.log('Teenager');
} else {
    console.log('Child');
}

// Ternary operator (shorthand)
let status = age >= 18 ? 'Adult' : 'Not
Adult';
```

```
//Switch Statement:
switch (day) {
    case 'Monday':
        console.log('Start of week');
        break;
    case 'Friday':
        console.log('End of week');
        break;
    default:
        console.log('Weekday');
}
```

Loops: Executing Code Repeatedly

```
//For Loop:
for (let i = 0; i < 5; i++) {
    console.log(i);  // 0, 1, 2, 3, 4
}

//While Loop:
let i = 0;
while (i < 5) {
    console.log(i);
    i++;
}
```

```
//For-Of Loop (Arrays):
let fruits = ['apple', 'pear',
'banana'];
for (let fruit of fruits) {
    console.log(fruit);
}

//For-In Loop (Objects):
let person = {name: 'John', age: 25};
for (let key in person) {
    console.log(`${key}:
${person[key]}`);
}
```

Functions - Reusable Code Blocks

```
//Function Declaration:
function sum(a, b) {
    return a + b;
}

//Function Expression:
const multiply = function(a, b) {
    return a * b;
};

//Arrow Function (ES6):
const divide = (a, b) => a / b;
const square = x => x * x;
const greet = () =>
console.log('Hello');
```

```
//Default Parameters:
function sayHello(name = 'Guest') {
    return `Hello ${name}`;
}

//Rest Parameters:
function sum(...numbers) {
    return numbers.reduce((sum, num)
=> sum + num, 0);
}
```

Arrays: Working with Lists of Data

```
//Array
let fruits = ['apple', 'pear', 'banana'];

// Access
console.log(fruits[0]);  // 'apple'

// Add/Remove

    // Add to end
    fruits.push('strawberry’);

    // Add to start
    fruits.unshift('cherry');

    // Remove from end
    fruits.pop();

    // Remove from start
    fruits.shift();
```

```
//Powerful Array Methods:
let numbers = [1, 2, 3, 4, 5];

numbers.map(x => x * x);
// [1, 4, 9, 16, 25]

numbers.filter(x => x % 2 === 0);
// [2, 4]

numbers.reduce((sum, x) => sum + x, 0);
// 15

numbers.find(x => x > 3);
// 4

numbers.some(x => x > 3);
// true

numbers.every(x => x > 0);
// true
```

Objects: Structured Data Collections

```javascript
let person = {
    name: 'John',
    age: 25,
    city: 'Istanbul',
    greet: function() {
        return `Hello, I'm ${this.name}`;
    }
};

// Access
    // Dot notation
    console.log(person.name);
    // Bracket notation
    console.log(person['age']);

// Add/Update
person.job = 'Engineer';
person.age = 26;

// Delete
delete person.city;
```

```javascript
//Object Methods:

    // Returns keys
    Object.keys(person);

    // Returns values
    Object.values(person);

    // Returns [key, value] pairs
    Object.entries(person);

// Destructuring
let {name, age} = person;

// Spread operator
let newPerson = {...person, country: 
'Turkey'};
```

DOM Manipulation - Changing the Page

```
//Selecting Elements:
let element =
document.getElementById('title');
let first = document.querySelector('.box');
let all = document.querySelectorAll('.box');

//Changing Content:
element.textContent = 'New text';
element.innerHTML = '<strong>Bold
text</strong>';

//Changing Styles:
element.style.color = 'red';
element.style.backgroundColor = 'yellow';
```

```
//Class Operations:
element.classList.add('active');
element.classList.remove('active');
element.classList.toggle('active');

//Creating/Removing Elements:
let newDiv =
document.createElement('div');
newDiv.textContent = 'New content';
document.body.appendChild(newDiv);
element.remove();
```

Events: Listening to User Actions

```
let button =
document.querySelector('#button');

button.addEventListener('click',
function(event) {
    console.log('Button clicked');
    // Prevent default behavior
    event.preventDefault();
});

// Arrow function syntax
button.addEventListener('click', (e) => {
    console.log('Clicked');
});
```

```
// Click
element.addEventListener('click', handler);
// Double-click
element.addEventListener('dblclick',
handler);
// Mouse over
element.addEventListener('mouseenter',
handler);
// Key press
element.addEventListener('keydown',
handler);
// Form submit
element.addEventListener('submit', handler);
// Input change
element.addEventListener('change', handler);
// Focus
element.addEventListener('focus', handler);
// Lose focus
element.addEventListener('blur', handler);
```

Async JavaScript: Non-Blocking Operations

- The Problem:
  - JavaScript is single-threaded
  - Long operations would freeze the page
  - Solution: Asynchronous programming

Async JavaScript: Callbacks

```
//Callbacks:
function fetchData(callback) {
    setTimeout(() => {
        callback('Data received');
    }, 1000);
}

fetchData((data) => {
    console.log(data);
});

//Callback Hell:
getData(function(a) {
    getMoreData(a, function(b) {
        getMoreData(b, function(c) {
            // Nested callbacks become hard to read
        });
    });
});
```

Promises: Cleaner Asynchronous Code

```
let promise = new Promise((resolve, reject) => {
    setTimeout(() => {
        let success = true;
        if (success) {
            resolve('Operation successful');
        } else {
            reject('Error occurred');
        }
    }, 1000);
});

promise
    .then(result => console.log(result))
    .catch(error => console.error(error))
    .finally(() => console.log('Operation complete’));

fetchUser()
    .then(user => fetchPosts(user.id))
    .then(posts => displayPosts(posts))
    .catch(error => console.error(error));
```

Async/Await: Synchronous-Looking Async Code

```
async function fetchData() {
    try {
        let response = await
fetch('https://api.example.com/data');
        let data = await response.json();
        console.log(data);
    } catch (error) {
        console.error('Error:', error);
    }
}

fetchData();
```

```
// GET request
fetch('https://api.example.com/users')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error:',
error));

// POST request
fetch('https://api.example.com/users', {
    method: 'POST',
    headers: {'Content-Type':
'application/json'},
    body: JSON.stringify({name: 'John',
age: 25})
})
    .then(response => response.json())
    .then(data => console.log(data));
```

Local Storage - Browser Data Storage

```
// Save data
localStorage.setItem('name', 'John');
localStorage.setItem('user', JSON.stringify({
    name: 'John',
    age: 25
}));

// Read data
let name = localStorage.getItem('name');
let user = JSON.parse(localStorage.getItem('user'));

// Remove data
localStorage.removeItem('name');
localStorage.clear();  // Remove all
```

- **Use Cases:**
  - User preferences (theme, language)
  - Shopping cart items
  - Form data persistence
  - Simple caching
- **Limitations:**
  - Only stores strings (use JSON.stringify/parse for objects)
  - Limited to ~5-10MB
  - Not secure (don't store sensitive data)

Thymeleaf: Java Template Engine

- **What is Thymeleaf?**
  - Modern server-side template engine for Java
  - Perfect integration with Spring Framework
  - Natural templating - valid HTML that works in browsers
  - Fills HTML templates with dynamic content
- **Advantages:**
  - Natural templating (HTML validity)
  - Easy Spring Boot integration
  - Powerful expression language
  - Internationalization (i18n) support
  - Layout systems

Installation Thymeleaf (Spring Boot):

- TemplatesLocation:
    - src/main/resources/templates/

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Thymeleaf Syntax and Expressions

Namespace Declaration:

```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Thymeleaf Example</title>
</head>
<body>
    <h1 th:text="${title}">Default
Title</h1>
</body>
</html>
```

Controller Example:

```
@GetMapping("/")
public String home(Model model) {
    model.addAttribute("name", "John");
    model.addAttribute("age", 25);
    return "index";
}
```

Variable Expressions (${}):

```
<p th:text="${name}">Name will appear
here</p>
<p th:text="${age}">Age will appear here</p>

<!-- String concatenation -->
<p th:text="'Hello ' + ${name}">Hello</p>

<!-- Literal substitution (cleaner) -->
<p th:text="|Hello ${name}, you are
${age}|">Hello</p>
```

Thymeleaf Object Selection

Controller:

```
@GetMapping("/user")
public String user(Model model) {
    User user = new User("John",
"john@email.com", 25);
    model.addAttribute("user", user);
    return "user";
}
```

Selection Expressions (*{}):

```
<div th:object="${user}">
    <p th:text="*{name}">Name</p>
    <p th:text="*{email}">Email</p>
    <p th:text="*{age}">Age</p>
</div>
```

Thymeleaf Link Expressions

```html
<!-- Static URL -->
<a th:href="@{/about}">About</a>

<!-- URL with path variable -->
<a th:href="@{/user/{id}(id=${user.id})}">Profile</a>

<!-- URL with query parameters -->
<a th:href="@{/search(q=${query},page=${page})}">Search</a>
<!-- Result: /search?q=test&page=1 -->

<!-- Static resources -->
<link th:href="@{/css/style.css}" rel="stylesheet">
<script th:src="@{/js/script.js}"></script>
<img th:src="@{/images/logo.png}" alt="Logo">
```

Thymeleaf Conditionals

```
<p th:if="${user.age >= 18}">You are an adult</p>
<p th:unless="${user.age >= 18}">You are not an adult</p>

<!-- Null check -->
<div th:if="${user != null}">
    <p th:text="${user.name}">Name</p>
</div>

<!-- Switch -->
<div th:switch="${user.role}">
    <p th:case="'ADMIN'">Administrator</p>
    <p th:case="'USER'">User</p>
    <p th:case="*">Guest</p>
</div>
```

Thymeleaf Loops

```
<table>
    <thead>
        <tr>
            <th>No</th>
            <th>Product</th>
            <th>Price</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="product, iterStat : ${products}">
            <td th:text="${iterStat.count}">1</td>
            <td th:text="${product.name}">Product name</td>
            <td th:text="${product.price}">Price</td>
        </tr>
    </tbody>
</table>
```

ASST. PROF. DR. SAMET KAYA

FATIH SULTAN MEHMET VAKIF UNIVERSITY

skaya@fsm.edu.tr

kysamet@gmail.com