COMP7270/7980
Final Exam Revision Part 1

ZHANG Ce

Final Exam

- It will be 3 hours long
- There will be 10 questions; each may carry a few sub-questions
  - 10 marks for each question
- The exam is a CLOSED book test
  - Calculator can be used, but not necessary

Topics Covered

- There won’t be any questions that were not covered by the tutorials/slides
- Topic Distribution
    1. HTML and CSS
    2. Responsive Web Design/Bootstrap
    3. JavaScript
    4. HTML Form and Input Validation
    5. Express.js
    6. MEAN Stack/MongoDB CRUD
    7. Web Technologies, including Restful API/AJAX
    8. Vue.js
    9. Working with Data
    10. Token-based Authentication and Ionic Mobile App Development

HTML and CSS

- What is the basic component of HTML?

```
<html>

<head>
   <title> This is a starting page </title>
</head>

<body>
   <h1 style="text-align:center"> This is a starting page </h1>
   Click <a href="http://www.comp.hkbu.edu.hk"> Here </a>
   to go to Computer Department of HKBU <br>
</body>

</html>
```

HTML and CSS

- What is the basic component of HTML?

  Links

  - Create a named div inside an HTML document:
    `<div id="cp3"> Chapter 3 </div>`
  - Create a link to the “Chapter 3” inside the same document:
    `<a href="#cp3"> Go to Chapter 3 </a>`
  - Or, create a link to the “Chapter 3” from another page:
    `<a href="anchor1.html#cp3"> Go to Chapter 3 </a>`

HTML and CSS

- What is the basic component of HTML?

  Tables

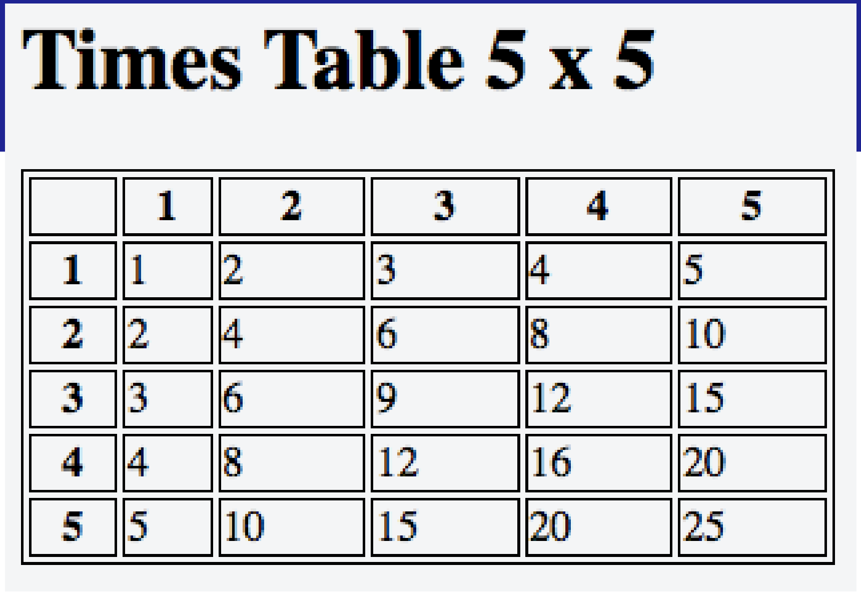
  - Tables are defined with <table> tag.
  - A table is divided into **rows** with <tr> tag.
  - Each row is divided **data cells** with <td> tag.
  - <td> tag can contain text, links, images, lists, forms, other tables, etc.
  - <th> tag stands for **table header** in which text element is displayed as **bold and centered**.

HTML and CSS

- What is the basic component of HTML?

  Tables

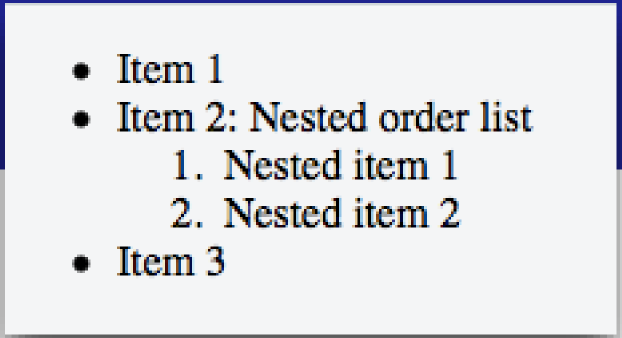
```
<!DOCTYPE html>
<html>
<head>
  <title>Times Table</title>
  <style>
    table, th, td {
      border: 1px solid black;
    }
  </style>
</head>
<body>
  <h1>Times Table 5 x 5</h1>
  <table style="width:50%">
    <tr><td></td><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr>
    <tr><th>1</th><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr>
    <tr><th>2</th><td>2</td><td>4</td><td>6</td><td>8</td><td>10</td></tr>
    <tr><th>3</th><td>3</td><td>6</td><td>9</td><td>12</td><td>15</td></tr>
    <tr><th>4</th><td>4</td><td>8</td><td>12</td><td>16</td><td>20</td></tr>
    <tr><th>5</th><td>5</td><td>10</td><td>15</td><td>20</td><td>25</td></tr>
  </table>
</body>
</html>
```



Times Table 5 x 5

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

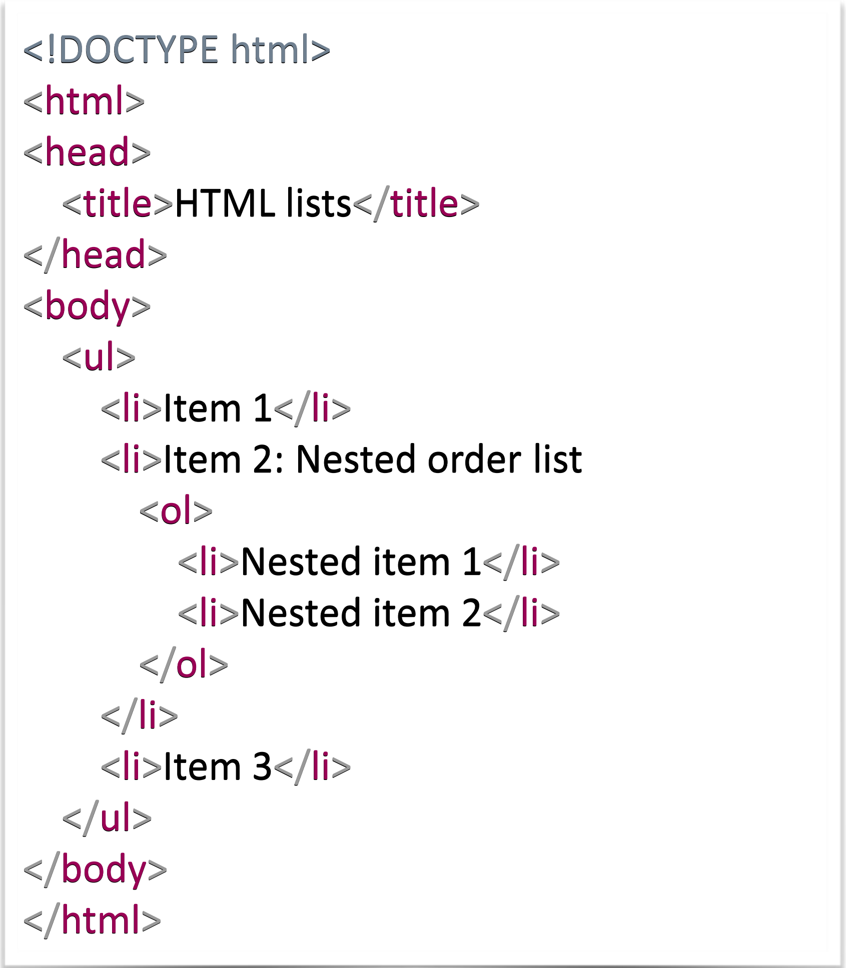
HTML and CSS



- Item 1
- Item 2: Nested order list
  1. Nested item 1
  2. Nested item 2
- Item 3

- What is the basic component of HTML?

  Lists
  - **Unordered list**
    - Defined with <ul> tag
    - Each item starts with <li> tag
  - **Ordered list**
    - Defined with <ol> tag
    - Each item starts with <li> tag
  - **List can be nested**
  - Reverse the ordered list and start the numbering from 10?
    - <ol start="10" reversed>
    - </od>



```
<!DOCTYPE html>
<html>
<head>
  <title>HTML lists</title>
</head>
<body>
  <ul>
    <li>Item 1</li>
    <li>Item 2: Nested order list
      <ol>
        <li>Nested item 1</li>
        <li>Nested item 2</li>
      </ol>
    </li>
    <li>Item 3</li>
  </ul>
</body>
</html>
```

HTML and CSS

- CSS
  - **Styles** defined how to display HTML elements
    - Specify display details once for any element.
    - Styles can be saved in external .css files.
    - Change presentation of all pages in **one single file**.
  - Where to put CSS?
    - **External** style sheet
      - Style applies to many pages, each page must link with <link> tag inside the head section
    - **Internal** style sheet
      - For a single document has a unique style, specified using <style> tag
    - **Inline** style
      - Style tag using style attribute

HTML and CSS

- How CSS is inserted:
  - External
  - Internal
  - Inline

```
<html>
<head>
  <link rel="stylesheet" href="external.css">
  <style>
    p { color:#ff0033; }
  </style>
</head>
<body>
  <p style="color:#ff0033;"> Some text. </p>
</body>
</html>
```

HTML and CSS

- Two main parts: Selectors { declarations }

- **Selectors**
- Specify the HTML elements to be styled.
    - Multiple selectors are separated with a comma.

- **Declarations**
    - Each declaration consists of a property and a value.
    - Multiple declarations are separated with a semi-colon.
- Comment enclosed between /* and */

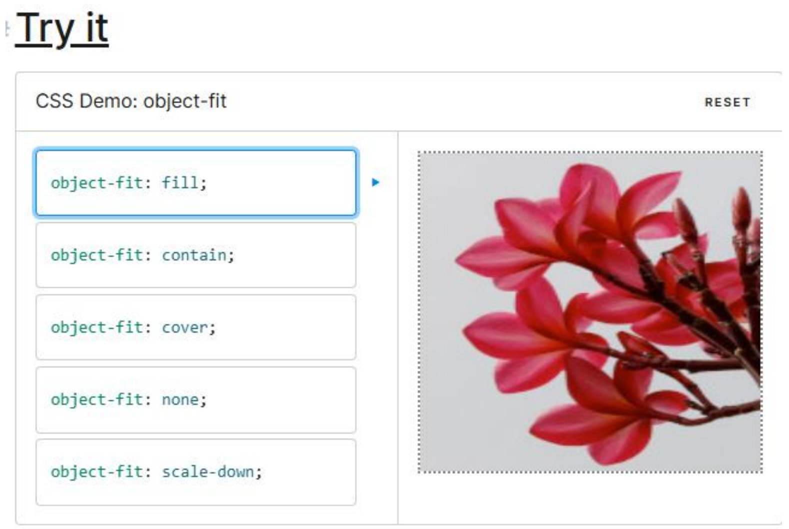
HTML and CSS

- Some CSS properties
  - font-family: specifies the font for an element.
  - font-weight: sets how thick or thin characters in text should be displayed.
  - front-style: specifies the font style for a text.
  - font-size: sets the size of a font.

```css
body {
   background-color: black;
   color: white;
   font-family: times, arial, serif;
}
h1 {
   text-align: center;
   text-transform: uppercase;
   text-decoration: underline;
}
h2 {
   font-weight: bold;
   font-style: oblique;
}
```

HTML and CSS

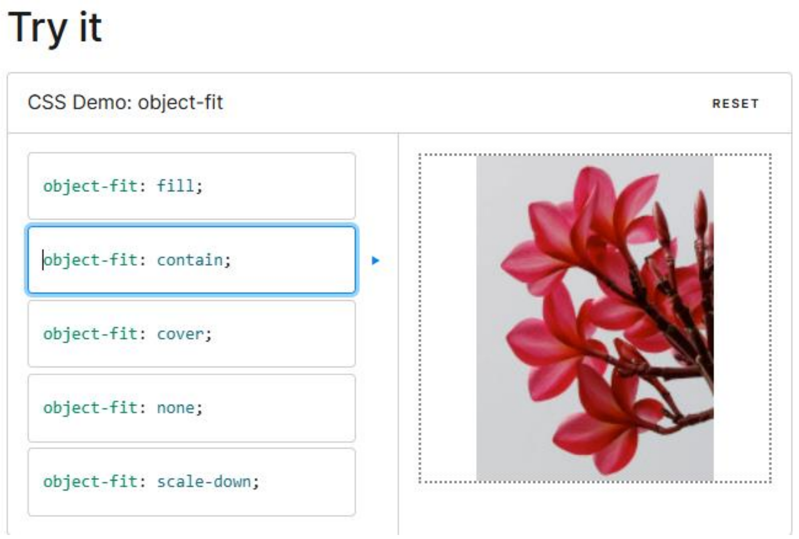
- CSS
  - Object-fit
    - Fill
    - Contain
    - Cover
    - None



Try it

CSS Demo: object-fit

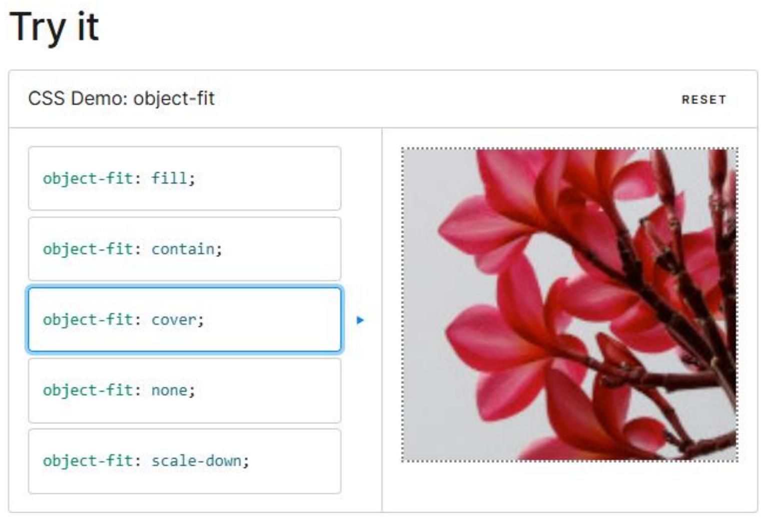
```
object-fit: fill;
object-fit: contain;
object-fit: cover;
object-fit: none;
object-fit: scale-down;
```



Try it

CSS Demo: object-fit

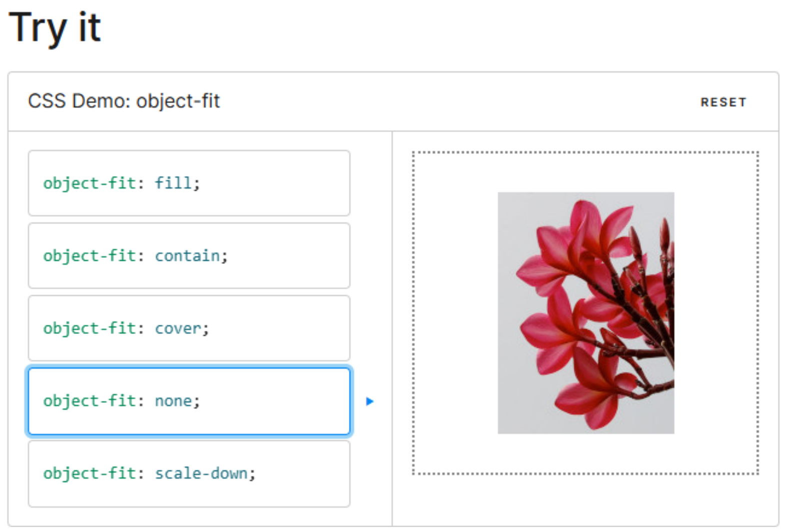
```
object-fit: fill;
object-fit: contain;
object-fit: cover;
object-fit: none;
object-fit: scale-down;
```



Try it

CSS Demo: object-fit

```
object-fit: fill;
object-fit: contain;
object-fit: cover;
object-fit: none;
object-fit: scale-down;
```



Try it

CSS Demo: object-fit

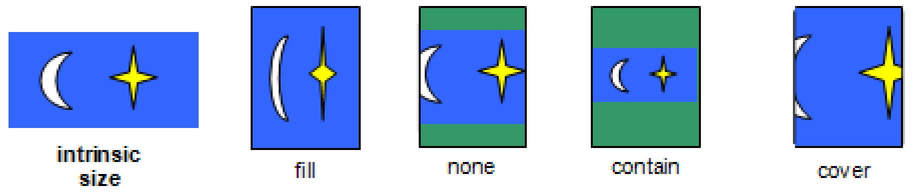
```
object-fit: fill;
object-fit: contain;
object-fit: cover;
object-fit: none;
object-fit: scale-down;
```

https://developer.mozilla.org/en-US/docs/Web/CSS/object-fit

HTML and CSS

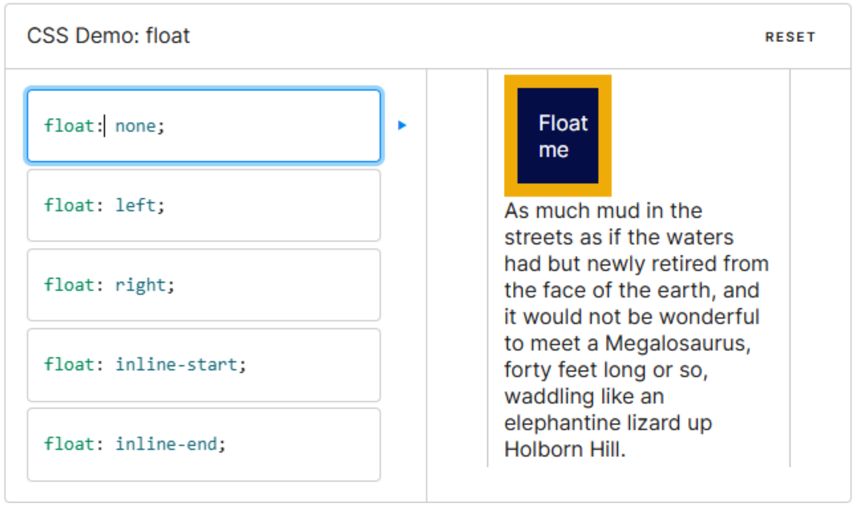
- CSS
  - Object-fit
    - Fill
    - Contain
    - Cover
    - None

- `fill`: The image will be **stretched to fill the entire content box**, potentially **distorting its aspect ratio**.
- `contain`: The image will be **scaled down** as much as possible to **fit within the content box** while **maintaining its aspect ratio**.
- `cover`: The image will be **scaled up** as much as possible to **fill the entire content box** while **maintaining its aspect ratio**. Any **overflow will be clipped**.
- `none`: The image will be **displayed at its original size**, ignoring the size of the content box.



HTML and CSS

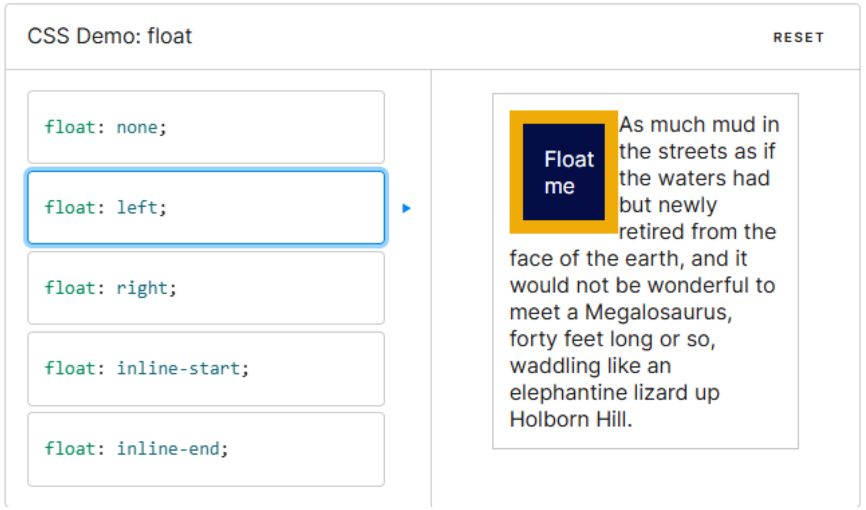
- CSS
  - Floating



CSS Demo: float

```
float: none;
float: left;
float: right;
float: inline-start;
float: inline-end;
```

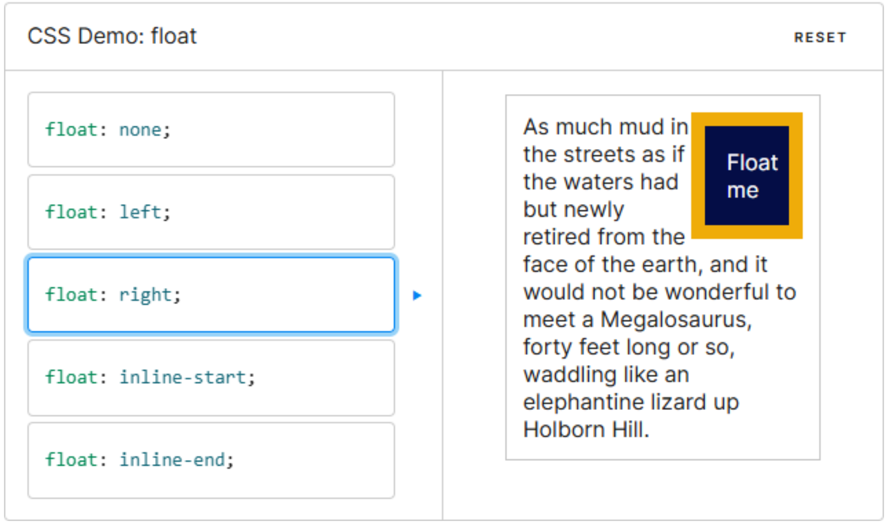
As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.



CSS Demo: float

```
float: none;
float: left;
float: right;
float: inline-start;
float: inline-end;
```

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.



CSS Demo: float

```
float: none;
float: left;
float: right;
float: inline-start;
float: inline-end;
```

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

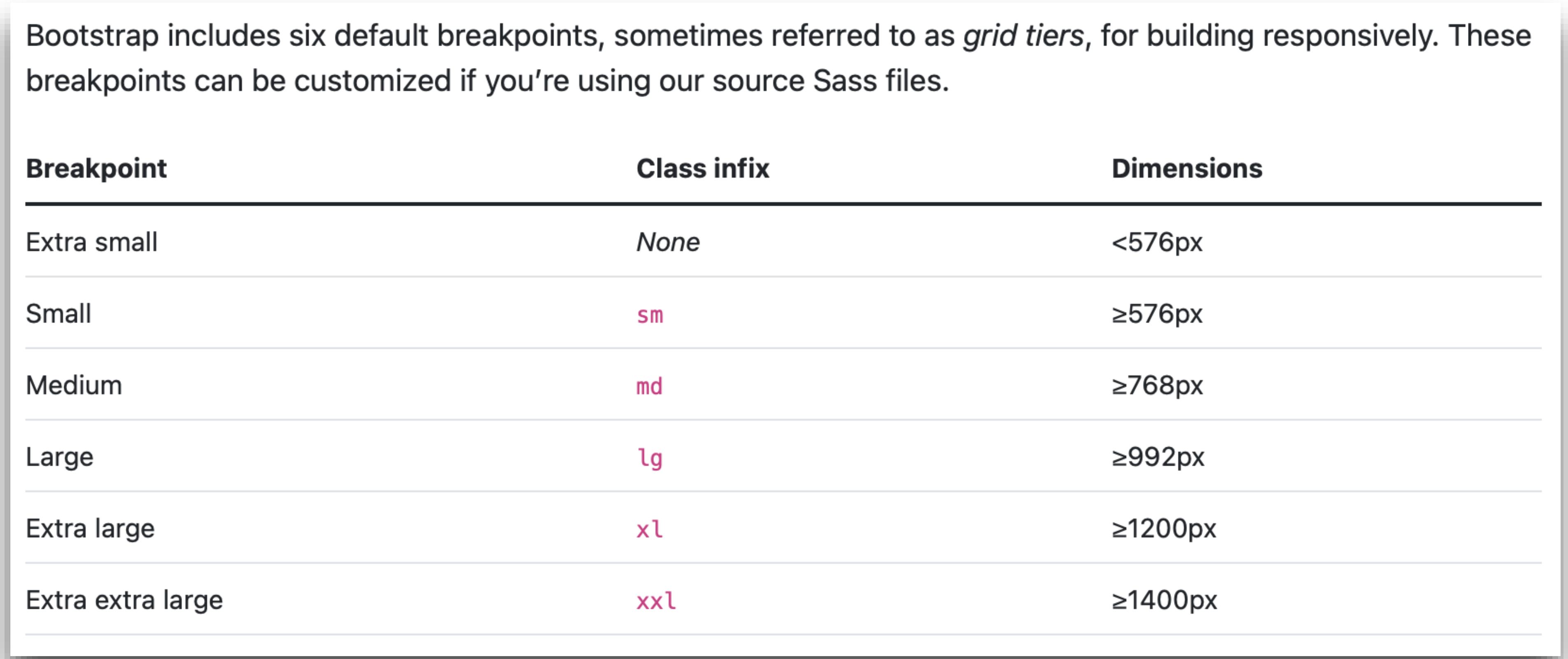
https://developer.mozilla.org/en-US/docs/Web/CSS/float

Responsive Web Design/Bootstrap

- Bootstrap Grid System
  - Bootstrap provides a **responsive grid system** that helps in creating flexible and responsive layouts. The grid system is based on a **12-column layout**, allowing developers to easily organize and structure content **across different screen sizes**.
  - **Grid Classes**: Bootstrap provides **CSS classes** that define the layout and behavior of columns within the grid system. You can assign these classes to HTML elements to specify how they should behave at various screen sizes. The most commonly used grid classes are **col-xs-***, **col-sm-***, **col-md-***, and **col-lg-***, where * represents **the number of columns an element should span**.

Responsive Web Design/Bootstrap

- Bootstrap Grid System



Bootstrap includes six default breakpoints, sometimes referred to as *grid tiers*, for building responsively. These breakpoints can be customized if you're using our source Sass files.

Breakpoint	**Class infix**	**Dimensions**
Extra small	*None*	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

Responsive Web Design/Bootstrap

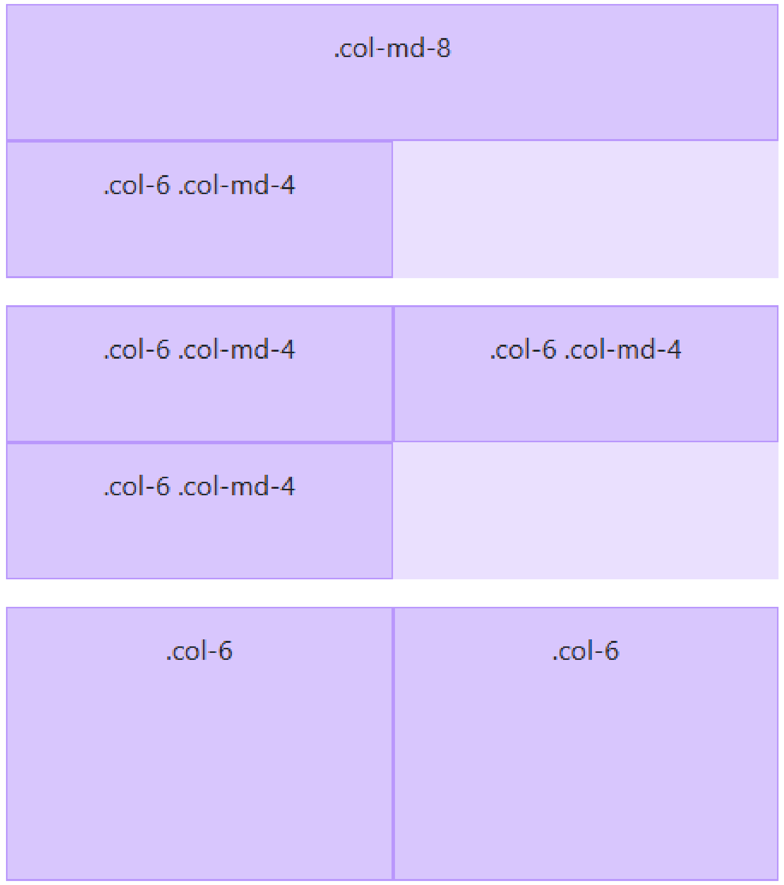
- If the following code is run on a small-sized screen, what will be shown?

```
<div class="container text-center">
  <div class="row">
   <div class="col-md-8">.col-md-8</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  </div>
  <div class="row">
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  </div>
  <div class="row">
   <div class="col-6">.col-6</div>
   <div class="col-6">.col-6</div>
  </div>
</div>
```

Responsive Web Design/Bootstrap

- If the following code is run on a small-sized screen, what will be shown?

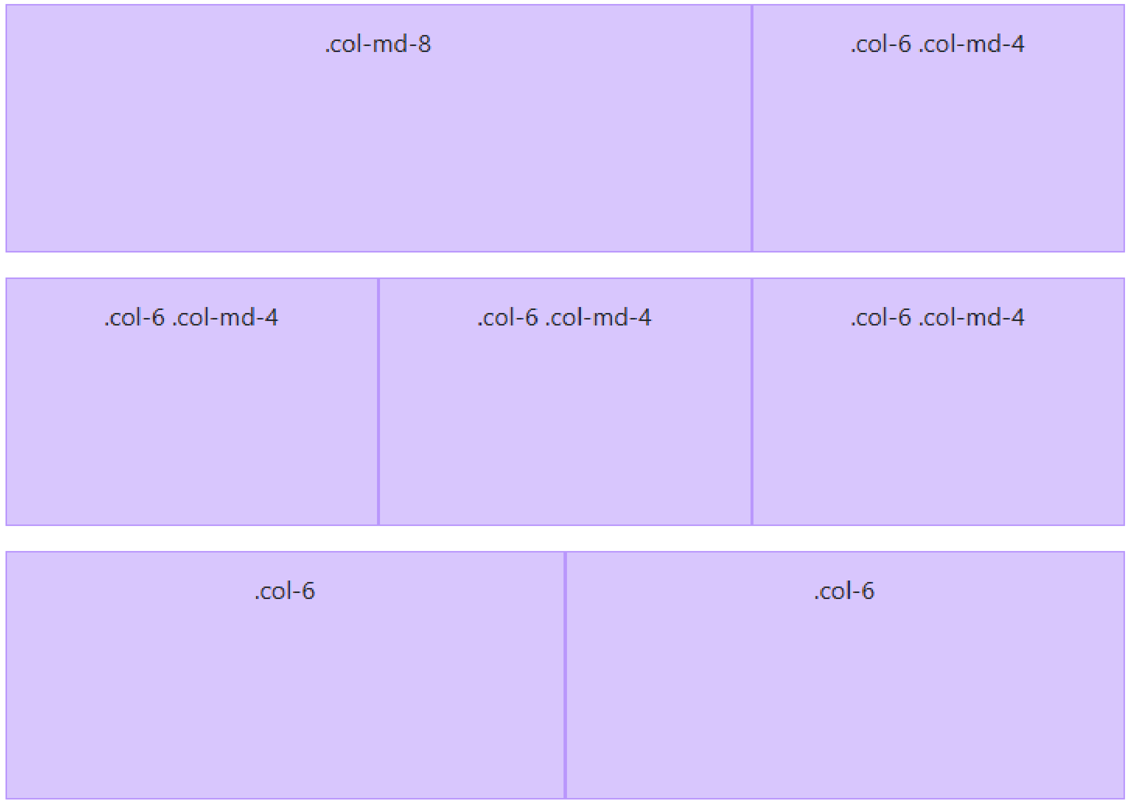
```
<div class="container text-center">
  <div class="row">
   <div class="col-md-8">.col-md-8</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
 </div>
  <div class="row">
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
 </div>
  <div class="row">
   <div class="col-6">.col-6</div>
   <div class="col-6">.col-6</div>
 </div>
</div>
```



Responsive Web Design/Bootstrap

- If the following code is run on a desktop computer, what will be shown?

```
<div class="container text-center">
  <div class="row">
   <div class="col-md-8">.col-md-8</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
 </div>
  <div class="row">
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
 </div>
  <div class="row">
   <div class="col-6">.col-6</div>
   <div class="col-6">.col-6</div>
 </div>
</div>
```



Responsive Web Design/Bootstrap

- Hide the first element for small-sized screen and show it for medium-sized screen

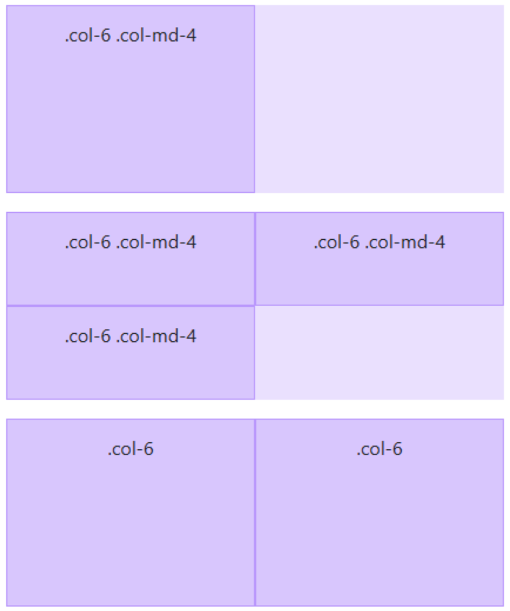
```
<div class="container text-center">
  <div class="row">
   <div class="col-md-8 d-none d-md-block">.col-md-8</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
 </div>
  <div class="row">
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
 </div>
  <div class="row">
   <div class="col-6">.col-6</div>
   <div class="col-6">.col-6</div>
 </div>
</div>
```

Responsive Web Design/Bootstrap

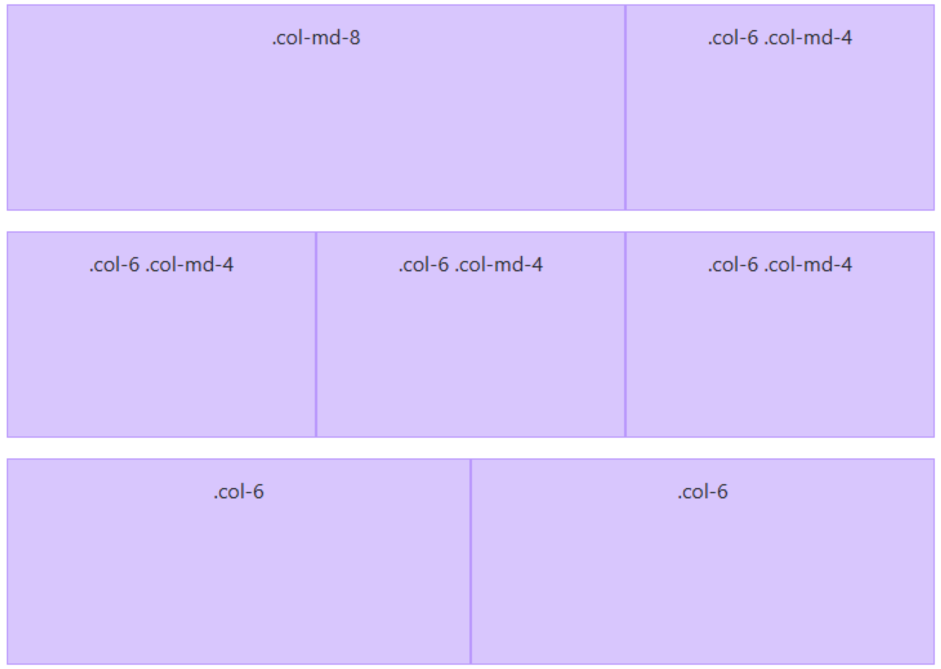
- Hide the first element for small-sized screen and show it for medium-sized screen

```
<div class="container text-center">
  <div class="row">
   <div class="col-md-8 d-none d-md-block">.col-md-8</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  </div>
  <div class="row">
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
   <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  </div>
  <div class="row">
   <div class="col-6">.col-6</div>
   <div class="col-6">.col-6</div>
  </div>
</div>
```

Small-sized screen



Medium-sized screen



JavaScript

- **React to events**
  - Execute when something happens, like when a page has finished loading or when a user clicks on an HTML element.
  - e.g., onchange, onclick, etc
- **Read and write HTML elements**
  - Read and change the content of an HTML element

```
function teamSelected(team) {
    var superheroElem = document.getElementById("superhero");
    superheroElem.options.length = 0;
}
```

JavaScript

- Button interaction
  - To add an interactive effect to your web page using JavaScript, we can create a button formatted by Bootstrap with an onclick attribute assigned to it. Here's an example of how you can create the section:

```
<div class="container my-4">
    <div class="row">
        <button class="col btn btn-primary" onclick="buttonClicked(this)">
            I'm a button, Click me!
        </button>
    </div>
</div>
```

Add the following inside the `<script>` element:

```
function buttonClicked(elem) {
    console.log("console output");
    elem.innerHTML = "Don't click me!";
}
```

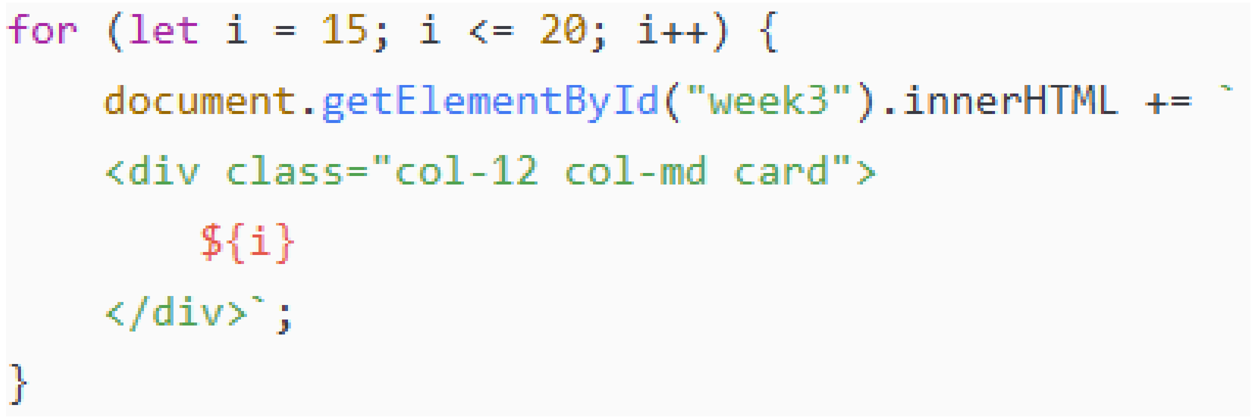
JavaScript

- Array
  - An array is used to store and **manage a collection of elements in a specific order**.
  - In JavaScript, arrays allow you to group multiple values under a single name and access those values using **numerical indices**.
- for…of
  - We can use a **for…of** loop to iterate through all the elements of an array

```
const avengers = ['Iron Man', 'Captain America',
                  'Thor', 'Hulk', 'Black Widow', 'Hawkeye'];
for (var avenger of avengers) {
  console.log(avenger);
}
```

JavaScript

- Another for loop:

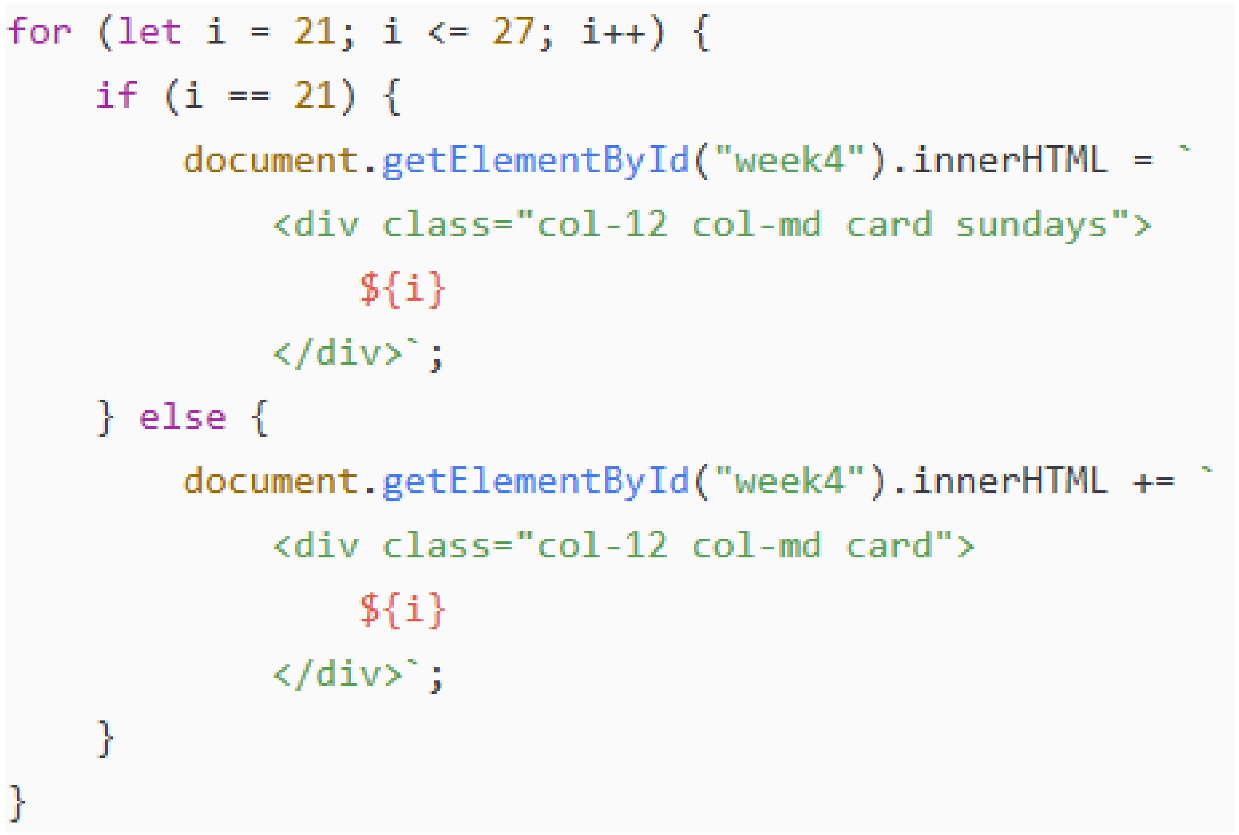


```
for (let i = 15; i <= 20; i++) {
    document.getElementById("week3").innerHTML += `
    <div class="col-12 col-md card">
        ${i}
    </div>`;
}
```

- String interpolation:
  - Use backticks: ``
  - Insert a variable: ${i}

JavaScript

- If-Else



```
for (let i = 21; i <= 27; i++) {
    if (i == 21) {
        document.getElementById("week4").innerHTML = `
            <div class="col-12 col-md card sundays">
                ${i}
            </div>`;
    } else {
        document.getElementById("week4").innerHTML += `
            <div class="col-12 col-md card">
                ${i}
            </div>`;
    }
}
```

JavaScript

- Read and write HTML elements

```
document.getElementById("myid").innerHTML = "Inner content";
```

- QuerySelector & QuerySelectorAll
  - These methods return element(s) that **match a specified CSS selector(s)**.

Element Selector	`document.querySelector('td');`
Id Selector	`document.querySelector('#id2');`
Class Selector	`document.querySelectorAll('.sundays');`
Attribute Selector	`document.querySelectorAll('ol[start="5"]');`

JavaScript

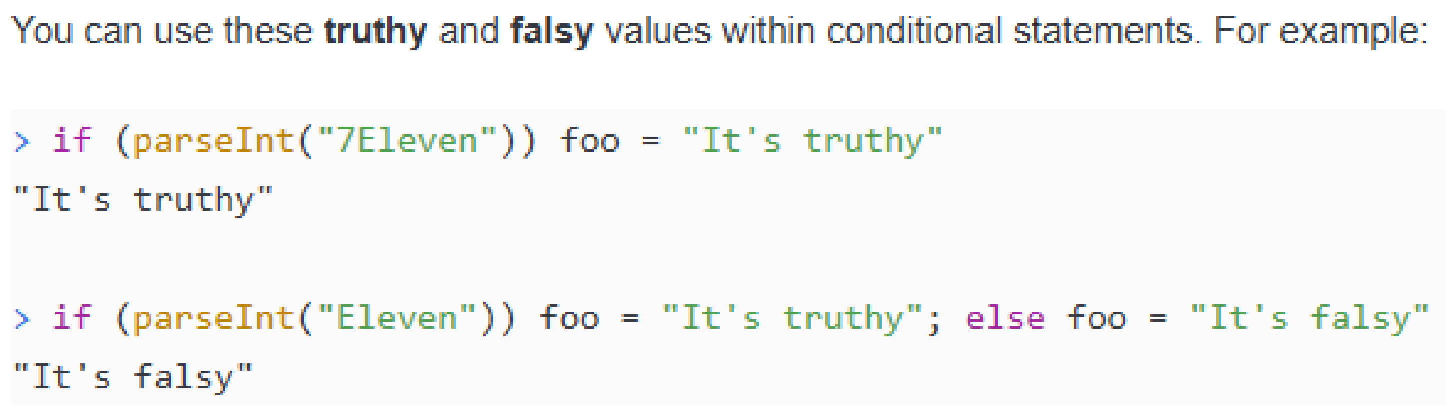
- To parse a string into a number, we can use the parseInt() function. numVariable = parseInt(stringValue);

```
var person = {
  name: "John Doe",
  age: 30,
  occupation: "Software Engineer",
  isStudent: false,
  hobbies: ["reading", "playing guitar", "hiking"],
  address: {
    street: "123 Main St",
    city: "Anytown",
    country: "USA"
  }
};
```

- Object:
  - A collection of properties
  - Each property is **identified by a key value** and **can hold values of any type**, including other objects.
  - To access this **person's city**, we can use **person.address.city**

JavaScript

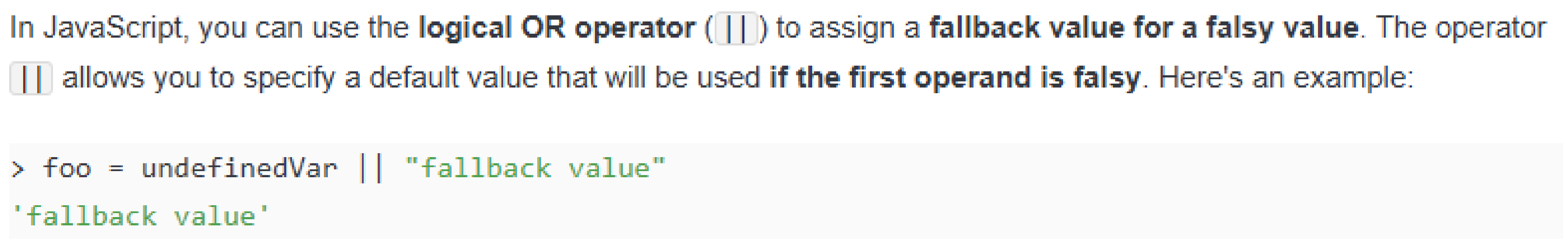
- Falsy Values: a value is either "**truthy**" or "**falsy**"



You can use these **truthy** and **falsy** values within conditional statements. For example:

```
> if (parseInt("7Eleven")) foo = "It's truthy"
"It's truthy"

> if (parseInt("Eleven")) foo = "It's truthy"; else foo = "It's falsy"
"It's falsy"
```

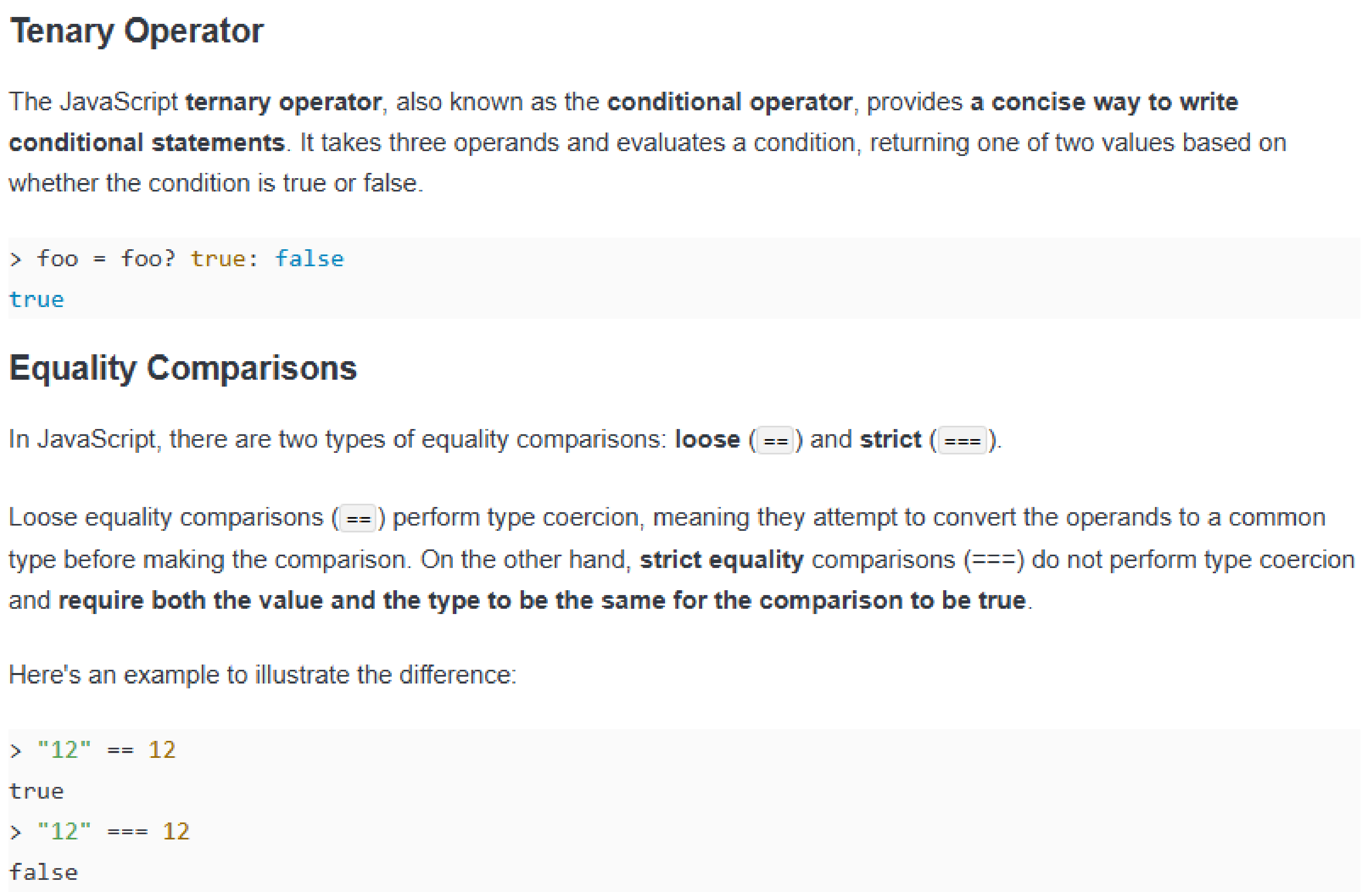
- Fallback value



In JavaScript, you can use the **logical OR operator** (`||`) to assign a **fallback value for a falsy value**. The operator `||` allows you to specify a default value that will be used **if the first operand is falsy**. Here's an example:

```
> foo = undefinedVar || "fallback value"
'fallback value'
```

JavaScript



Tenary Operator

The JavaScript **ternary operator**, also known as the **conditional operator**, provides **a concise way to write conditional statements**. It takes three operands and evaluates a condition, returning one of two values based on whether the condition is true or false.

```
> foo = foo? true: false
true
```

Equality Comparisons

In JavaScript, there are two types of equality comparisons: **loose** (`==`) and **strict** (`===`).

Loose equality comparisons (`==`) perform type coercion, meaning they attempt to convert the operands to a common type before making the comparison. On the other hand, **strict equality** comparisons (===) do not perform type coercion and **require both the value and the type to be the same for the comparison to be true**.

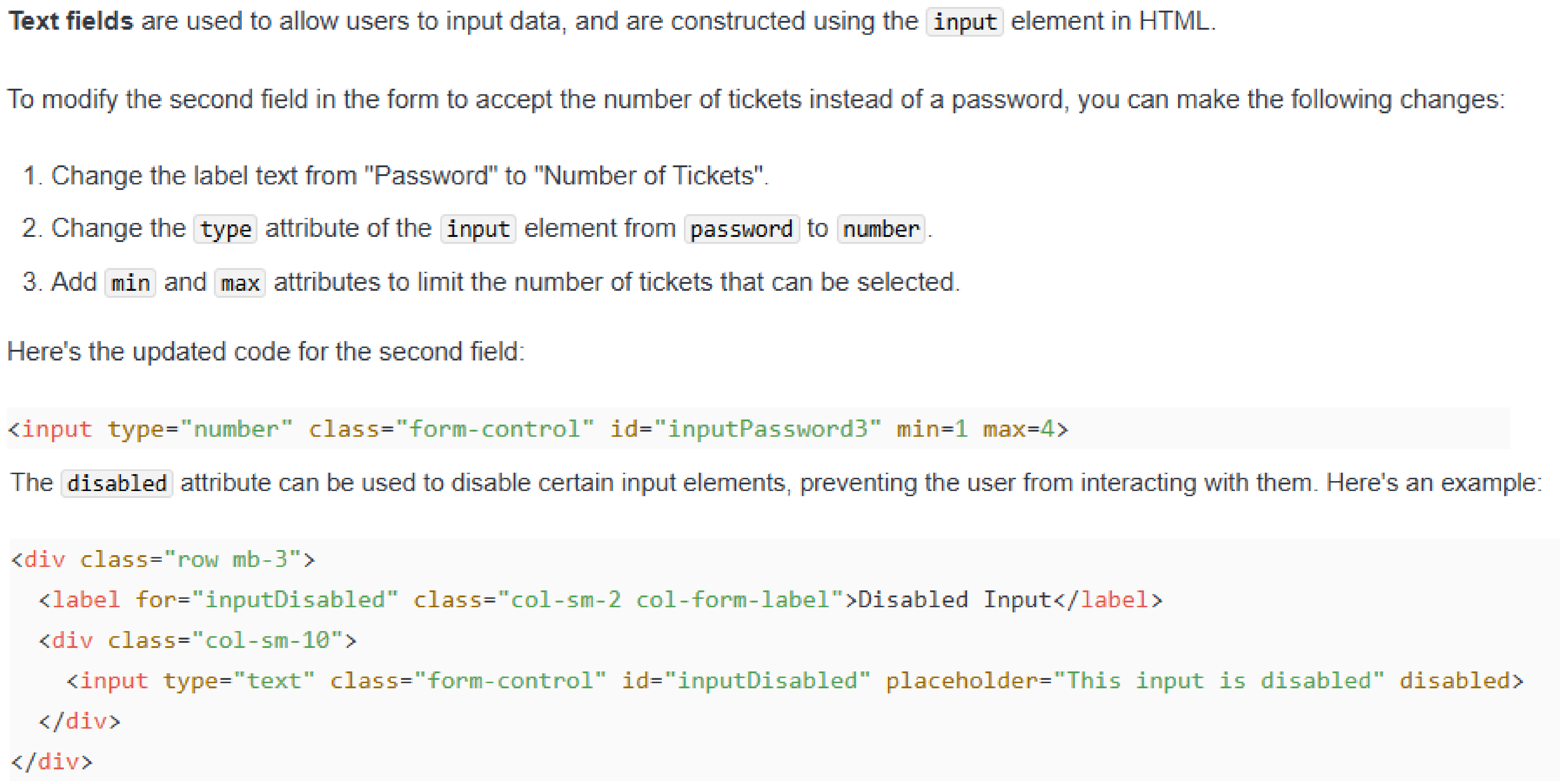
Here's an example to illustrate the difference:

```
> "12" == 12
true
> "12" === 12
false
```

HTML Form and Input Validation

- Input Element: The **<input>** element is used to create **text fields, checkboxes, radio buttons, and more**. It allows users to input data. The type attribute determines the specific type of input element.
- Type Attribute: The **type attribute** of the <input> element specifies the type of input control to display. Some common values include text for text fields, password for password fields, number for numeric input, checkbox for checkboxes, and radio for radio buttons.
- Select Element: The **<select>** element creates **a dropdown menu or list box**. It allows users to select options from a predefined list.
  The **<option>** elements inside the <select> element represent the **available choices**.

HTML Form and Input Validation



Text fields are used to allow users to input data, and are constructed using the `input` element in HTML.

To modify the second field in the form to accept the number of tickets instead of a password, you can make the following changes:

1. Change the label text from "Password" to "Number of Tickets".
2. Change the `type` attribute of the `input` element from `password` to `number`.
3. Add `min` and `max` attributes to limit the number of tickets that can be selected.

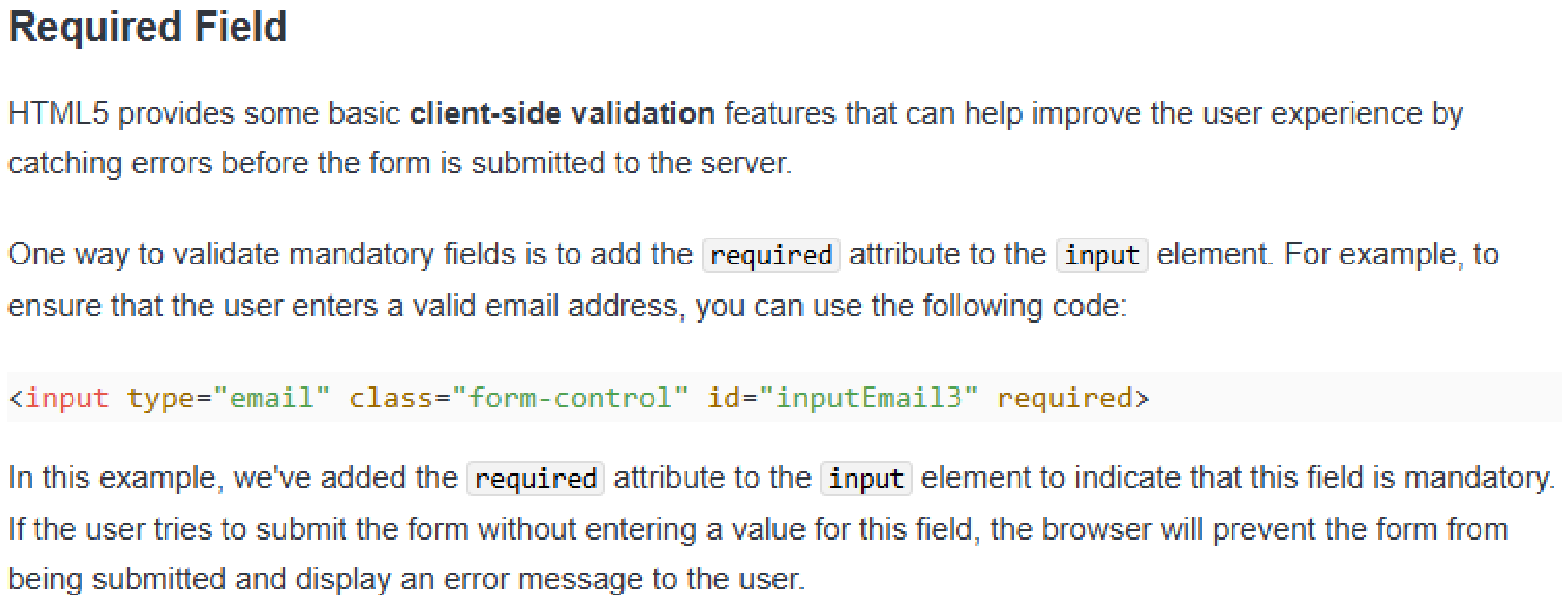
Here's the updated code for the second field:

```
<input type="number" class="form-control" id="inputPassword3" min=1 max=4>
```

The `disabled` attribute can be used to disable certain input elements, preventing the user from interacting with them. Here's an example:

```
<div class="row mb-3">
  <label for="inputDisabled" class="col-sm-2 col-form-label">Disabled Input</label>
  <div class="col-sm-10">
    <input type="text" class="form-control" id="inputDisabled" placeholder="This input is disabled" disabled>
  </div>
</div>
```

HTML Form and Input Validation



Required Field

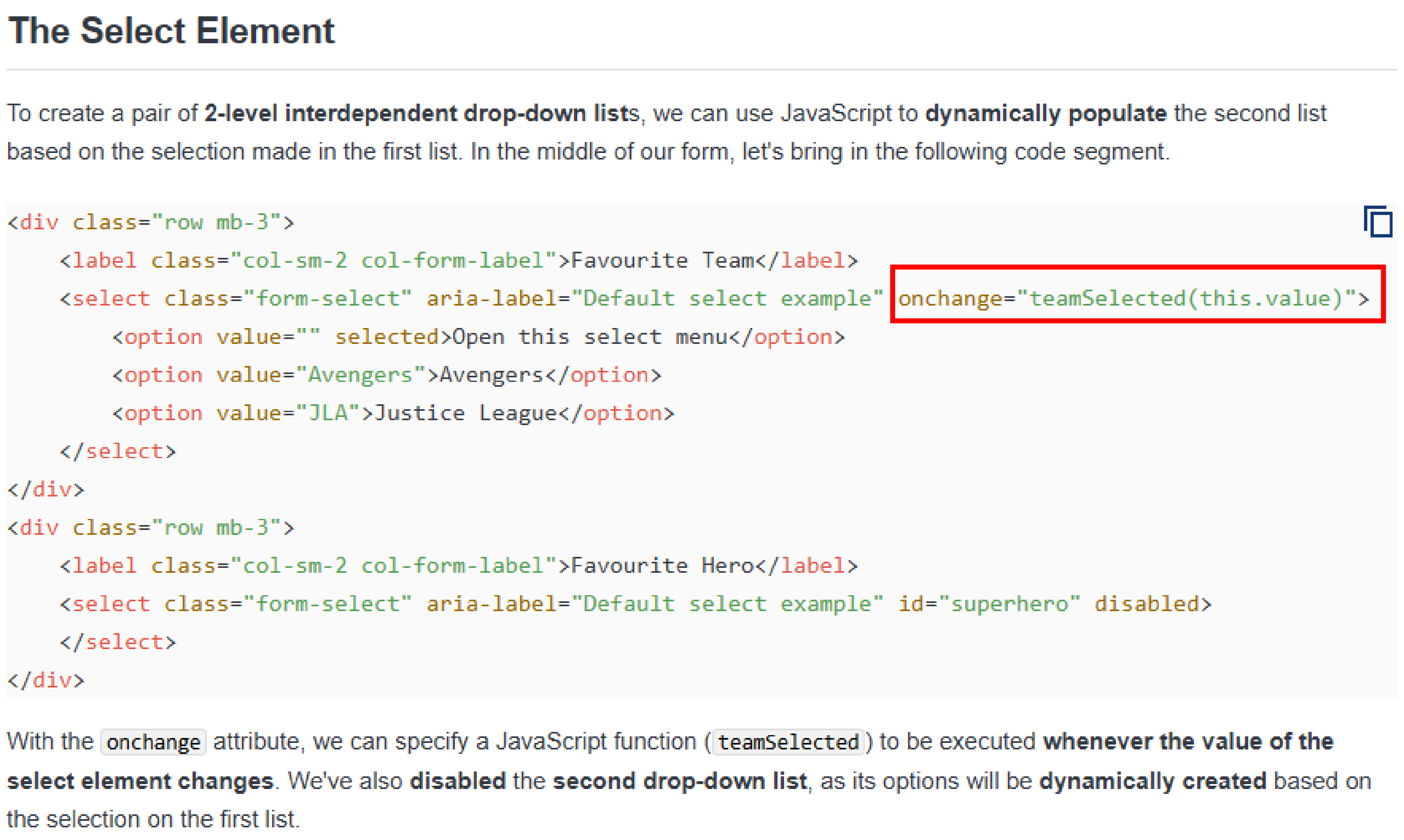
HTML5 provides some basic **client-side validation** features that can help improve the user experience by catching errors before the form is submitted to the server.

One way to validate mandatory fields is to add the `required` attribute to the `input` element. For example, to ensure that the user enters a valid email address, you can use the following code:

```
<input type="email" class="form-control" id="inputEmail3" required>
```

In this example, we've added the `required` attribute to the `input` element to indicate that this field is mandatory. If the user tries to submit the form without entering a value for this field, the browser will prevent the form from being submitted and display an error message to the user.

HTML Form and Input Validation



The Select Element

To create a pair of **2-level interdependent drop-down list**s, we can use JavaScript to **dynamically populate** the second list based on the selection made in the first list. In the middle of our form, let's bring in the following code segment.

```html
<div class="row mb-3">
    <label class="col-sm-2 col-form-label">Favourite Team</label>
    <select class="form-select" aria-label="Default select example" onchange="teamSelected(this.value)">
        <option value="" selected>Open this select menu</option>
        <option value="Avengers">Avengers</option>
        <option value="JLA">Justice League</option>
    </select>
</div>
<div class="row mb-3">
    <label class="col-sm-2 col-form-label">Favourite Hero</label>
    <select class="form-select" aria-label="Default select example" id="superhero" disabled>
    </select>
</div>
```

With the `onchange` attribute, we can specify a JavaScript function (`teamSelected`) to be executed **whenever the value of the select element changes**. We've also **disabled** the **second drop-down list**, as its options will be **dynamically created** based on the selection on the first list.

HTML Form and Input Validation

Dynamic Options

To create an array of superheroes for each team, we can define the arrays inside the script tag. Here's an example:

```
<script>
  const avengers = ['Iron Man', 'Captain America', 'Thor', 'Hulk', 'Black Widow', 'Hawkeye'];
  const justiceLeague = ['Superman', 'Batman', 'Wonder Woman', 'Flash', 'Aquaman', 'Cyborg'];
</script>
```

In this example, we've defined two arrays, `avengers` and `justiceLeague`, that contain the names of the superheroes for each team.

Next, we define the `teamSelected()` function, which will be called whenever the value of the `team` select element changes.

```
function teamSelected(team) {
    alert(team);
}
```

Here, we use an `alert` dialog box to display the selected value.

Next, we obtain a reference to the second drop-down list. The last line basically **clears all the options in the second drop-down list**.

```
var superheroElem = document.getElementById("superhero");
superheroElem.options.length = 0;
```

Then, **based on the option selected**, we will **populate data to the second `select` element accordingly**. We need a **conditional statement** (`if`) here.

```
if (team == "Avengers") {

    for (var hero of avengers) {
        var option = document.createElement("option");
        option.text = hero;
        option.value = hero;
        superheroElem.add(option);
    }

    superheroElem.disabled = false;

} else if (team == "JLA") {

    for (var hero of justiceLeague) {
        var option = document.createElement("option");
        option.text = hero;
        option.value = hero;
        superheroElem.add(option);
    }

    superheroElem.disabled = false;

} else {

    superheroElem.disabled = true;
}
```

HTML Form and Input Validation

- Button Element: The **<button>** element creates a clickable button.
  - The type attribute specifies the behavior of the button.
  - **type="submit"** is used to create **a submit button that triggers form submission**.

HTML Form and Input Validation

```
<form action="https://www.httpbin.org/post" method="POST">
  <input name="email" type="email">

  <input name="numTickets" type="number" min=1 max=4>

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

- Form is used to pass data to a server.
- The **submit button** will trigger the submission.
- action specifies **where** the form data will be submitted to.
- method specifies the **HTTP request method** for sending form data.

HTML Form and Input Validation

```
<input name="numTickets" type="number">
```

- Name attribute
  - The name attribute specifies the name of an <input> element.
  - The name attribute could be used to reference elements in client-side JavaScript, or to **reference form data in form submission**.
    - Note: Only form elements with a name attribute will be included in form submission.

HTML Form and Input Validation

- Checkbox
  - When this checkbox is clicked, this form element name, **together with its value**, is submitted to the server.
  - If the box is NOT clicked, this form element **won’t be submitted**.
  - To check the box, put the checked attribute in the opening tag.

Express.js

- Express is a fast, minimalistic, and flexible **web application framework for Node.js**. It provides a robust set of features and utilities for building web applications and APIs. Express.js is built on top of the Node.js core HTTP module, **simplifying the process of handling HTTP requests and responses**.
- **Routing**: Express.js allows developers to **define routes for handling specific HTTP requests** (such as GET, POST, PUT, DELETE) and their corresponding actions.

```
router.post('/form', function (req, res) {

   var response = {
      header: req.headers,
      body: req.body
   };

   res.json(response);
});
```

Express.js

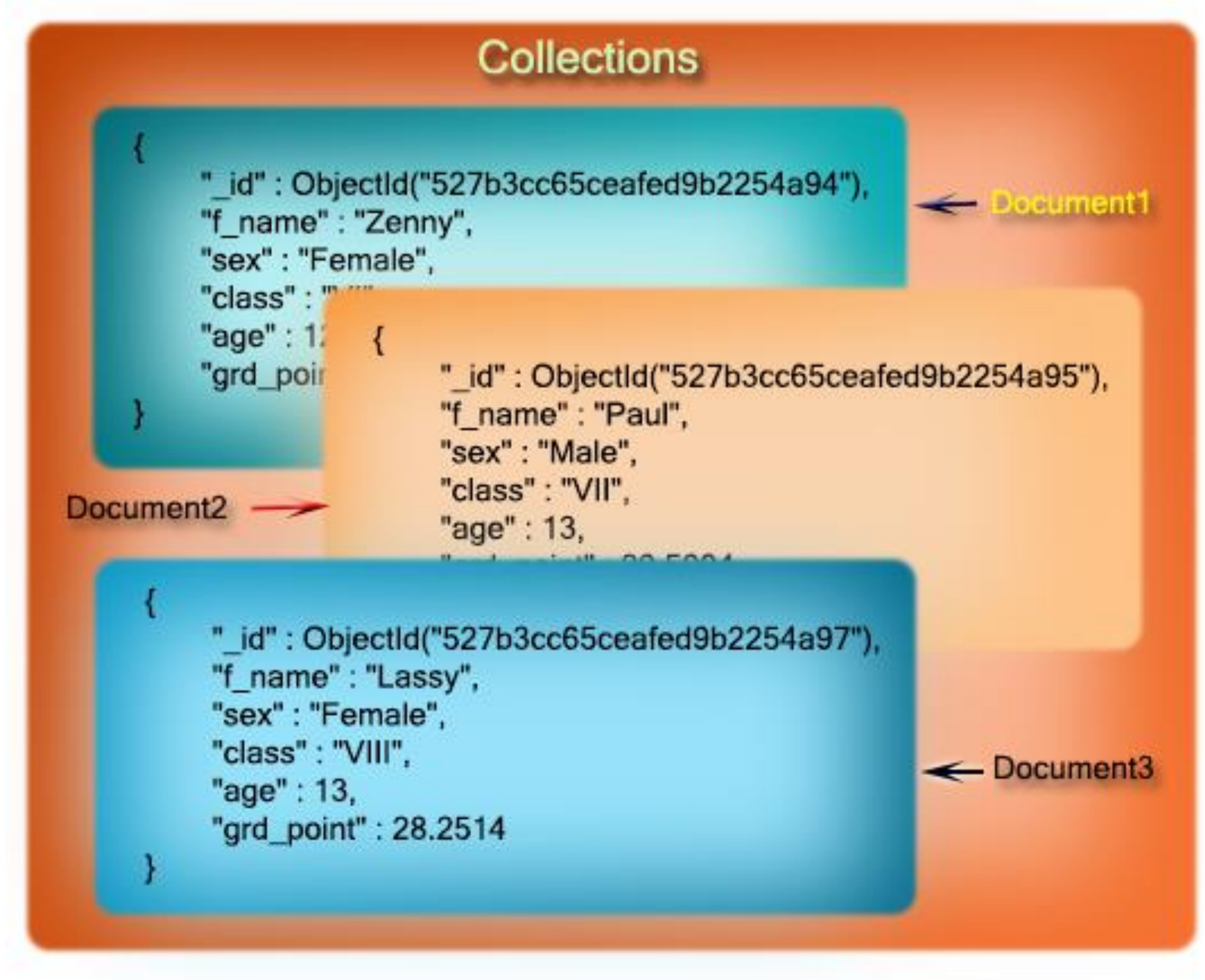
- **Template Engines**: Express supports various template engines, such as Pug (formerly known as Jade), **EJS (Embedded JavaScript)**, Handlebars, and more. Template engines enable the **dynamic generation of HTML** or other markup languages, simplifying the process of **rendering views and generating dynamic content**.

```
<table>
  <% for (var booking of bookings) { %>

    <tr>
      <td><%= booking.email %></td>
      <td><%= booking.numTickets %></td>
    </tr>

  <% } %>
</table>
```

```
<table>
          <tr>
            <td>tony@stark.com</td>
            <td>2</td>
          </tr>
          <tr>
            <td>bruce@wayne.com</td>
                 <td>1</td>
          </tr>
</table>
```

Response to client

MongoDB



```
{
    "_id" : ObjectId("527b3cc65ceafed9b2254a94"),
    "f_name" : "Zenny",
    "sex" : "Female",
    "class" :
    "age" : 1
    "grd_poi
}
```

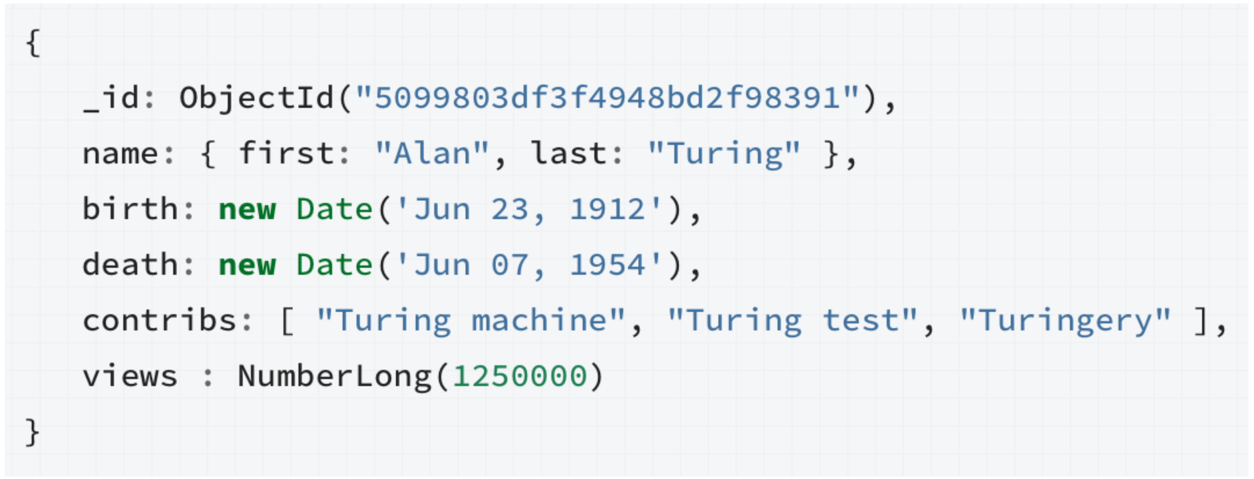
```
{
    "_id" : ObjectId("527b3cc65ceafed9b2254a95"),
    "f_name" : "Paul",
    "sex" : "Male",
    "class" : "VII",
    "age" : 13,
```

```
{
    "_id" : ObjectId("527b3cc65ceafed9b2254a97"),
    "f_name" : "Lassy",
    "sex" : "Female",
    "class" : "VIII",
    "age" : 13,
    "grd_point" : 28.2514
}
```

Relational	NoSQL
Database	Database
Table	**Collection**
Row	**Document**
Column	Field

MongoDB

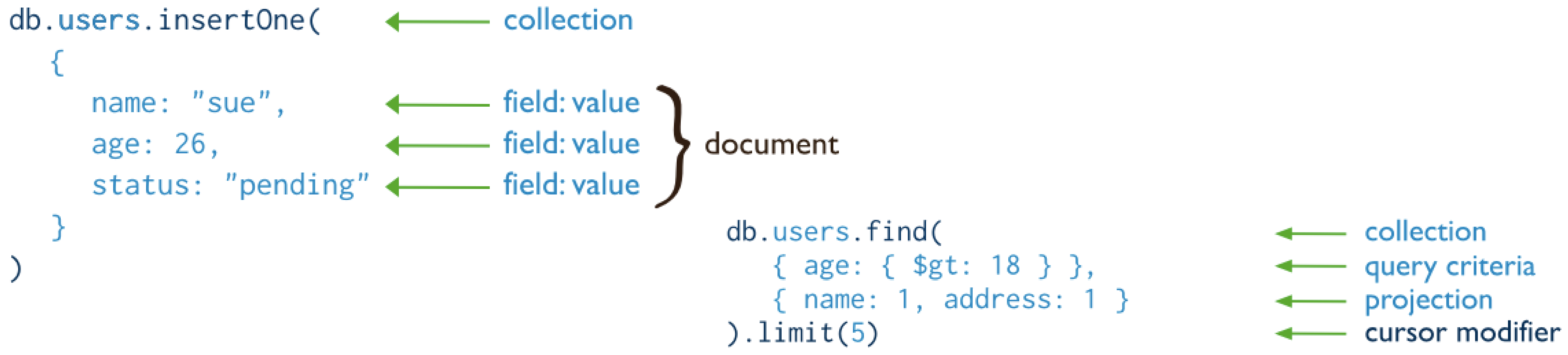
- _id field is reserved for **primary key** in MongoDB.
- MongoDB uses ObjectId as the default value of _id field of each document, which is generated while the creation of any document.



```
{
    _id: ObjectId("5099803df3f4948bd2f98391"),
    name: { first: "Alan", last: "Turing" },
    birth: new Date('Jun 23, 1912'),
    death: new Date('Jun 07, 1954'),
    contribs: [ "Turing machine", "Turing test", "Turingery" ],
    views : NumberLong(1250000)
}
```

MongoDB

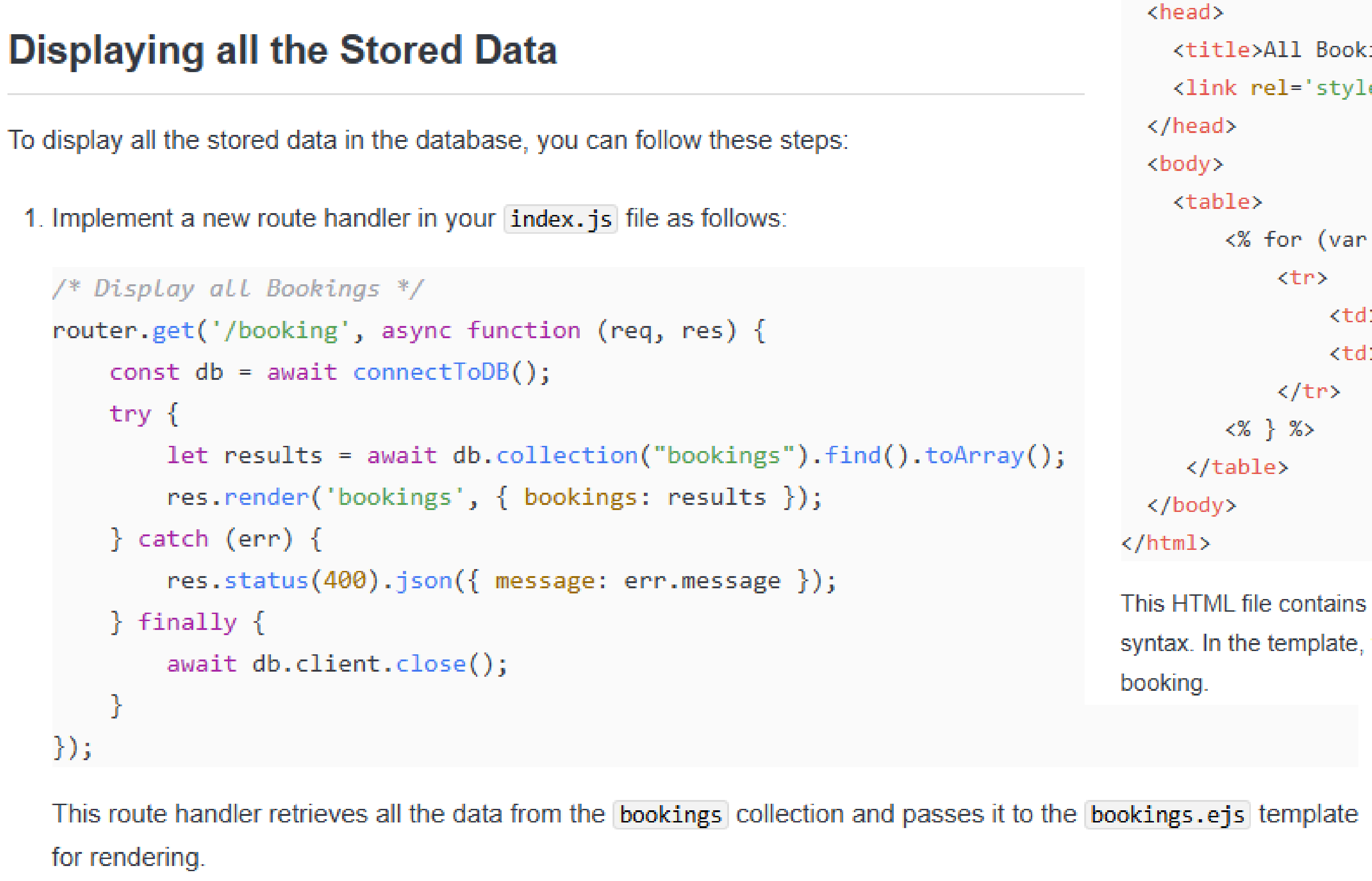
- MongoDB provides a powerful and **flexible query language** that supports a wide range of operations for retrieving, modifying, and aggregating data. The query language includes capabilities for filtering, sorting, joining, and performing complex aggregations, making it suitable for a variety of use cases.



```
db.users.insertOne(
   {
      name: "sue",
      age: 26,
      status: "pending"
   }
)
```

```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

MongoDB



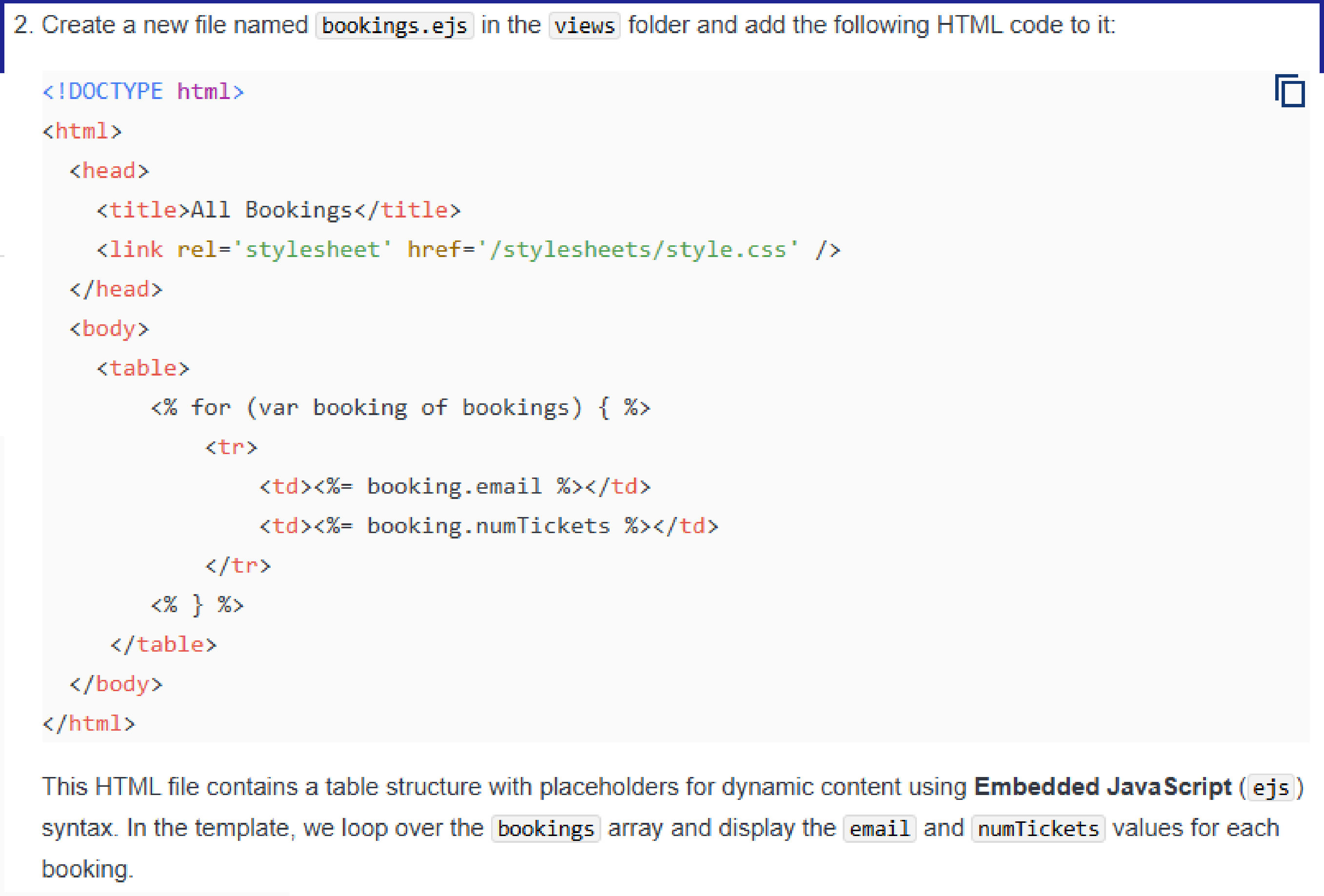
Displaying all the Stored Data

To display all the stored data in the database, you can follow these steps:

1. Implement a new route handler in your `index.js` file as follows:

```
/* Display all Bookings */
router.get('/booking', async function (req, res) {
    const db = await connectToDB();
    try {
        let results = await db.collection("bookings").find().toArray();
        res.render('bookings', { bookings: results });
    } catch (err) {
        res.status(400).json({ message: err.message });
    } finally {
        await db.client.close();
    }
});
```

This route handler retrieves all the data from the `bookings` collection and passes it to the `bookings.ejs` template for rendering.



2. Create a new file named `bookings.ejs` in the `views` folder and add the following HTML code to it:

```
<!DOCTYPE html>
<html>
  <head>
    <title>All Bookings</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <table>
        <% for (var booking of bookings) { %>
            <tr>
                <td><%= booking.email %></td>
                <td><%= booking.numTickets %></td>
            </tr>
        <% } %>
     </table>
  </body>
</html>
```

This HTML file contains a table structure with placeholders for dynamic content using **Embedded JavaScript** (`ejs`) syntax. In the template, we loop over the `bookings` array and display the `email` and `numTickets` values for each booking.

MongoDB

```
/* Display all Bookings */
router.get('/booking', async function (req, res) {

  let results = await db.collection("bookings").find().toArray();

  res.render('bookings', { bookings: results });

});
```

The await operator is used here to ensure **the methods resolve** before we move on to the next line.
“await” can only exist **inside** an async function, so it **would not block other functions** (router handlers), from running concurrently.

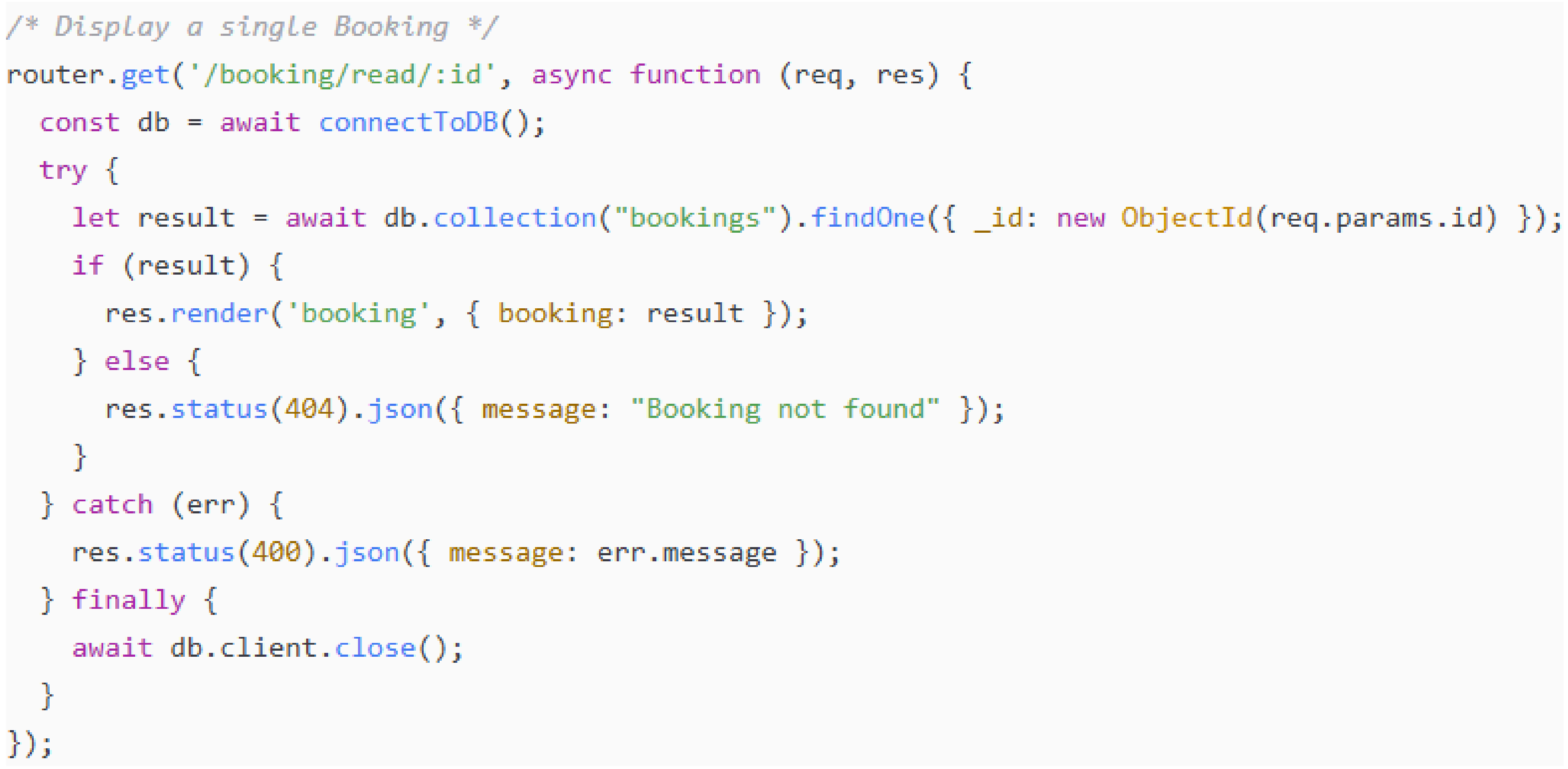
MongoDB

4. Implement the following code in the route handler for **storing data in the server-side**:

```
/* Handle the Form */
router.post('/booking', async function (req, res) {
  const db = await connectToDB();
  try {
    req.body.numTickets = parseInt(req.body.numTickets);
    req.body.terms = req.body.terms? true : false;
    req.body.created_at = new Date();
    req.body.modified_at = new Date();

    let result = await db.collection("bookings").insertOne(req.body);
    res.status(201).json({ id: result.insertedId });
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

Express.js & CRUD Operations

- Read



```
/* Display a single Booking */
router.get('/booking/read/:id', async function (req, res) {
  const db = await connectToDB();
  try {
    let result = await db.collection("bookings").findOne({ _id: new ObjectId(req.params.id) });
    if (result) {
      res.render('booking', { booking: result });
    } else {
      res.status(404).json({ message: "Booking not found" });
    }
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

Express.js & CRUD Operations

- Update

```
// Update a single Booking
router.post('/booking/update/:id', async function (req, res) {
  const db = await connectToDB();
  try {
    req.body.numTickets = parseInt(req.body.numTickets);
    req.body.terms = req.body.terms? true : false;
    req.body.superhero = req.body.superhero || "";
    req.body.modified_at = new Date();

    let result = await db.collection("bookings").updateOne({ _id: new ObjectId(req.params.id) }, { $set: req.body });

    if (result.modifiedCount > 0) {
      res.status(200).json({ message: "Booking updated" });
    } else {
      res.status(404).json({ message: "Booking not found" });
    }
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

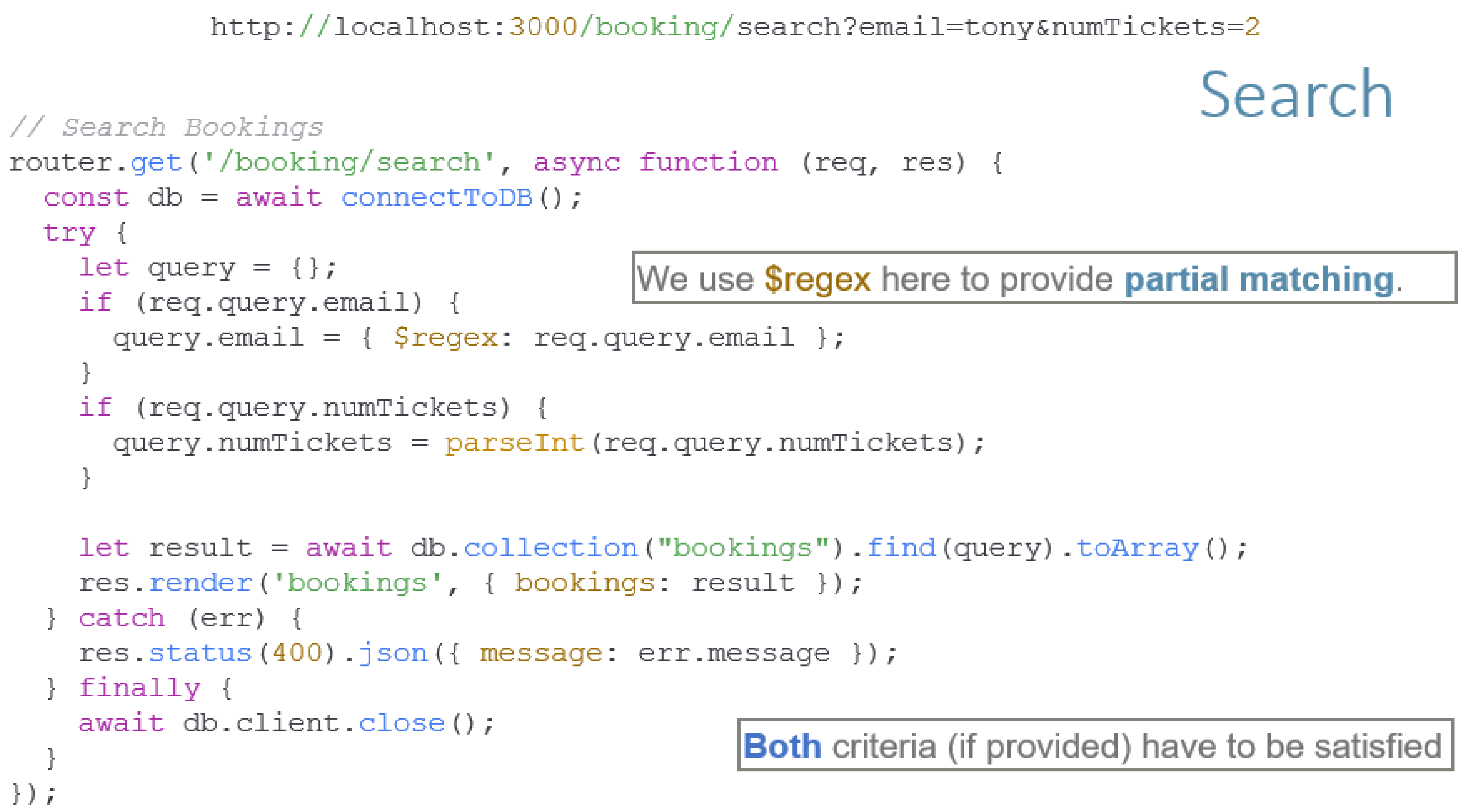
Express.js & CRUD Operations

- Delete

```
// Delete a single Booking
router.post('/booking/delete/:id', async function (req, res) {
  const db = await connectToDB();
  try {
    let result = await db.collection("bookings").deleteOne({ _id: new ObjectId(req.params.id) });
    if (result.deletedCount > 0) {
      res.status(200).json({ message: "Booking deleted" });
    } else {
      res.status(404).json({ message: "Booking not found" });
    }
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

Express.js & CRUD Operations

- Search



```
http://localhost:3000/booking/search?email=tony&numTickets=2
```

Search

```
// Search Bookings
router.get('/booking/search', async function (req, res) {
  const db = await connectToDB();
  try {
    let query = {};
    if (req.query.email) {
      query.email = { $regex: req.query.email };
    }
    if (req.query.numTickets) {
      query.numTickets = parseInt(req.query.numTickets);
    }

    let result = await db.collection("bookings").find(query).toArray();
    res.render('bookings', { bookings: result });
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

We use **$regex** here to provide **partial matching**.

Both criteria (if provided) have to be satisfied

Express.js & CRUD Operations

- Pagination

```
// Pagination based on query parameters page and limit, also returns total number of documents
router.get('/booking/paginate', async function (req, res) {
  const db = await connectToDB();
  try {
    let page = parseInt(req.query.page) || 1;
    let perPage = parseInt(req.query.perPage) || 10;
    let skip = (page - 1) * perPage;

    let result = await db.collection("bookings").find().skip(skip).limit(perPage).toArray();
    let total = await db.collection("bookings").countDocuments();

    res.render('bookings', { bookings: result, total: total, page: page, perPage: perPage });
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
  finally {
    await db.client.close();
  }
});
```

Number of items per page

Number of items to be skipped

page and perPage may have been modified in the route handler

```
http://localhost:3000/booking/paginate?perPage=2&page=2
```

Express.js & CRUD Operations

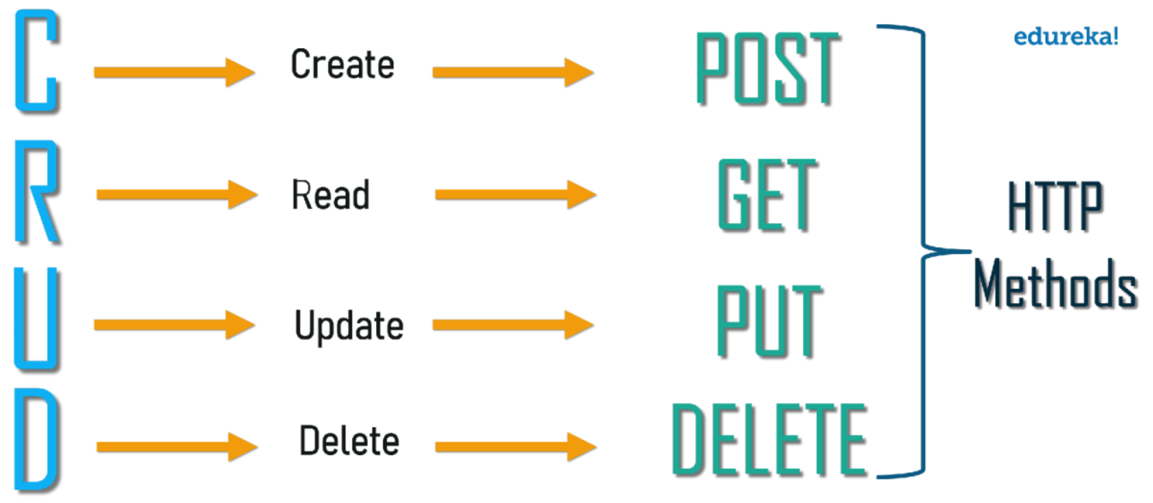
- Utilize the **template engine** to calculate **the total number of pages**.

```
<% for (let i = 1; i <= Math.ceil(total / perPage); i++) { %>
  <% if(i === page) { %>
    <span><%= i %></span>
  <% } else { %>
    <a href="/booking/paginate?page=<%= i %>&perPage=<%= perPage %>"><%= i %></a>
  <% } %>
<% } %>
```

Restful API/AJAX

- A **RESTful API** (Representational State Transfer API) is an architectural style and approach for **designing web services** that enable communication between systems over the internet.
- It follows a set of principles and constraints, which allow for the **creation, retrieval, update, and deletion (CRUD) operations** on **resources**.

Restful API/AJAX



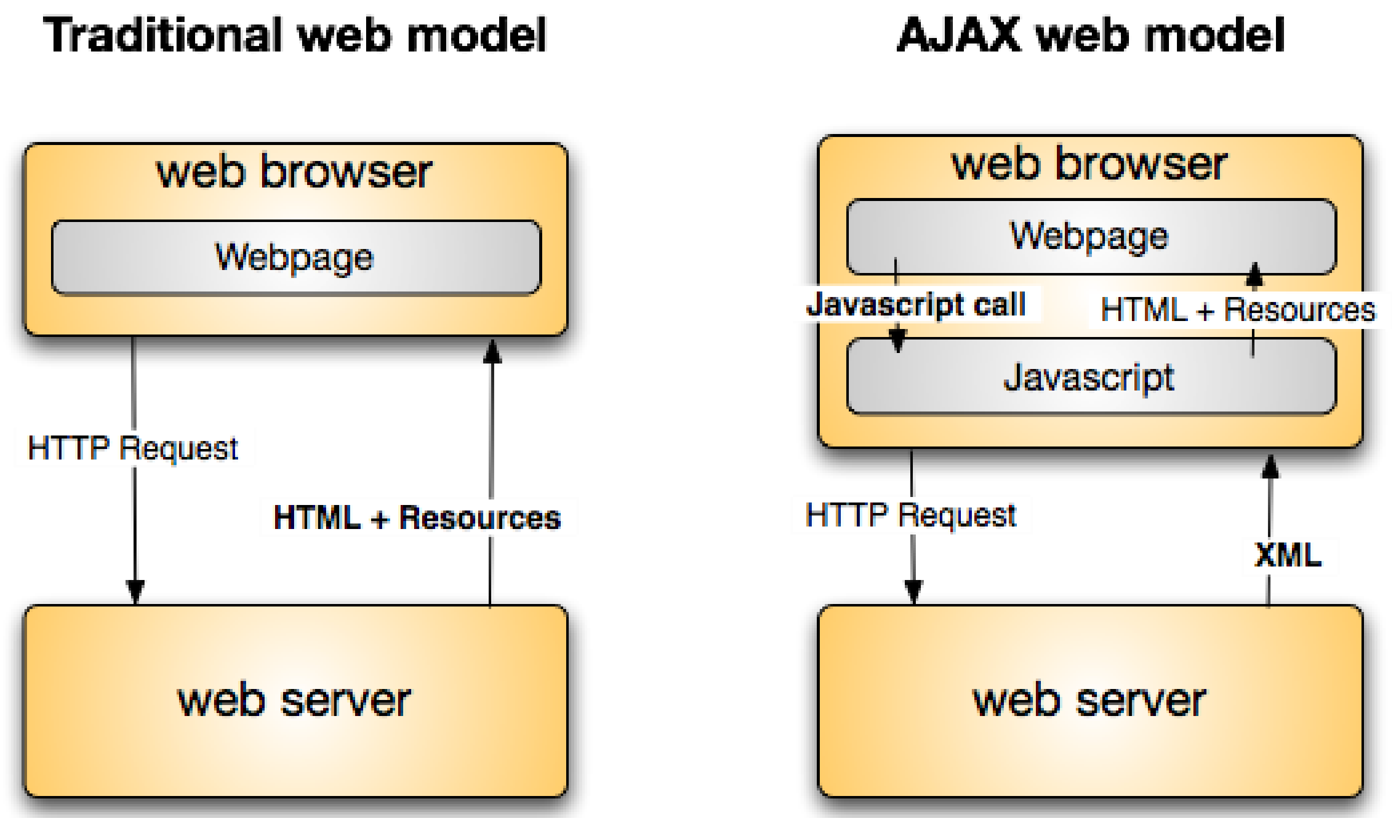
- Uniform Interface:
  - RESTful APIs use a uniform set of **well-defined methods** and standard HTTP verbs such as GET, POST, PUT, PATCH, and DELETE to perform **CRUD operations** on resources. These methods provide a consistent and predictable way to interact with the API.

Restful API/AJAX

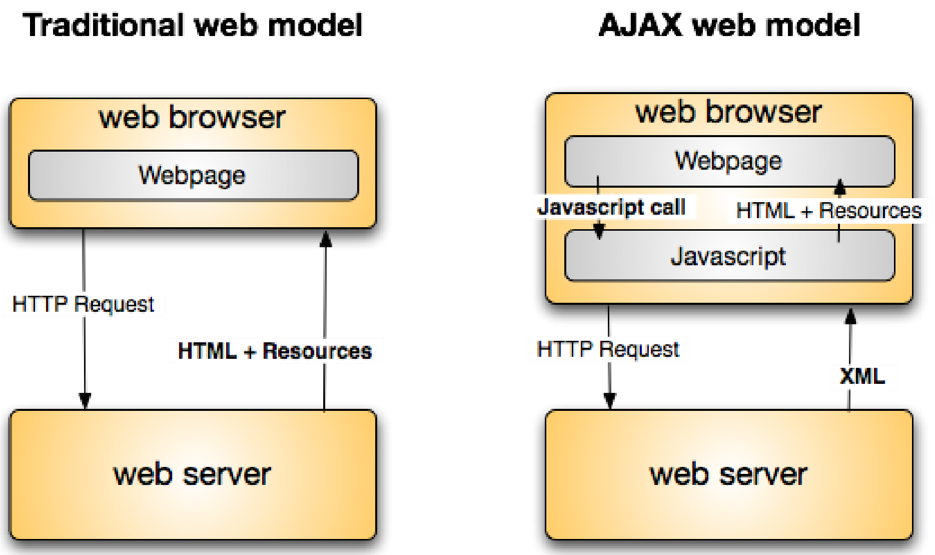
Non-RESTful (current implementation)	RESTful
POST /booking	**POST** /bookings
GET /booking/**read**/:id	**GET** /bookings/:id
POST /booking/**update**/:id	**PUT** /bookings/:id
POST /booking/**delete**/:id	**DELETE** /bookings/:id

Restful API/AJAX

- Asynchronous JavaScript and XML
- The HTTP **request** is now initiated by client-side **JavaScript**.
- Server **response** only contains data (like **XML**), but not the entire page.



Restful API/AJAX



- **XML** was once the go-to choice of data exchange format.
  - Now, **JSON** is more often used as the response data format.
- The client-side JavaScript will receive this data response and **update the current web page**.
  - As such, we are able to update a web page **without reloading it**.

Restful API/AJAX

- Benefits of AJAX
- Ajax can **reduce the traffic travels** between the client and the server.
  - Usually, only data (in JSON or XML formats) will be sent, **HTML and CSS codes are not transmitted**.
- The **server response time is faster** so increases performance and speed.
  - **Data aren't processed in the server side**, like being included in html.

Restful API/AJAX

- Fetch API
  - The Fetch API provides a global fetch method
  - Making an API call with fetch()

```
const response = await fetch(`/api/bookings?page=${page}&perPage=${perPage}`);
```

  - fetch doesn't perform automatic transformations, so we have to convert JSON data with response.json():

```
// convert the response to json
const json = await response.json();
```

Restful API/AJAX

- The fetch function returns a **Promise** that **resolves** to a Response object representing the server's response to the request.
- The response.json() method is called on the Response object to extract the JSON data from the response. **This method also returns a Promise** that resolves to the parsed JSON data.
- **The function awaits the resolution** of the response.json() Promise and assigns the parsed JSON data to the json variable.

Restful API/AJAX

```
// an async function to fetch bookings and metadata from the backend
async function getBookings(page, perPage) {
    // fetch the bookings
    const response = await fetch(`/bookings?page=${page}&perPage=${perPage}`);
    // convert the response to json
    const json = await response.json();
    // return the json
    return json;
}
```

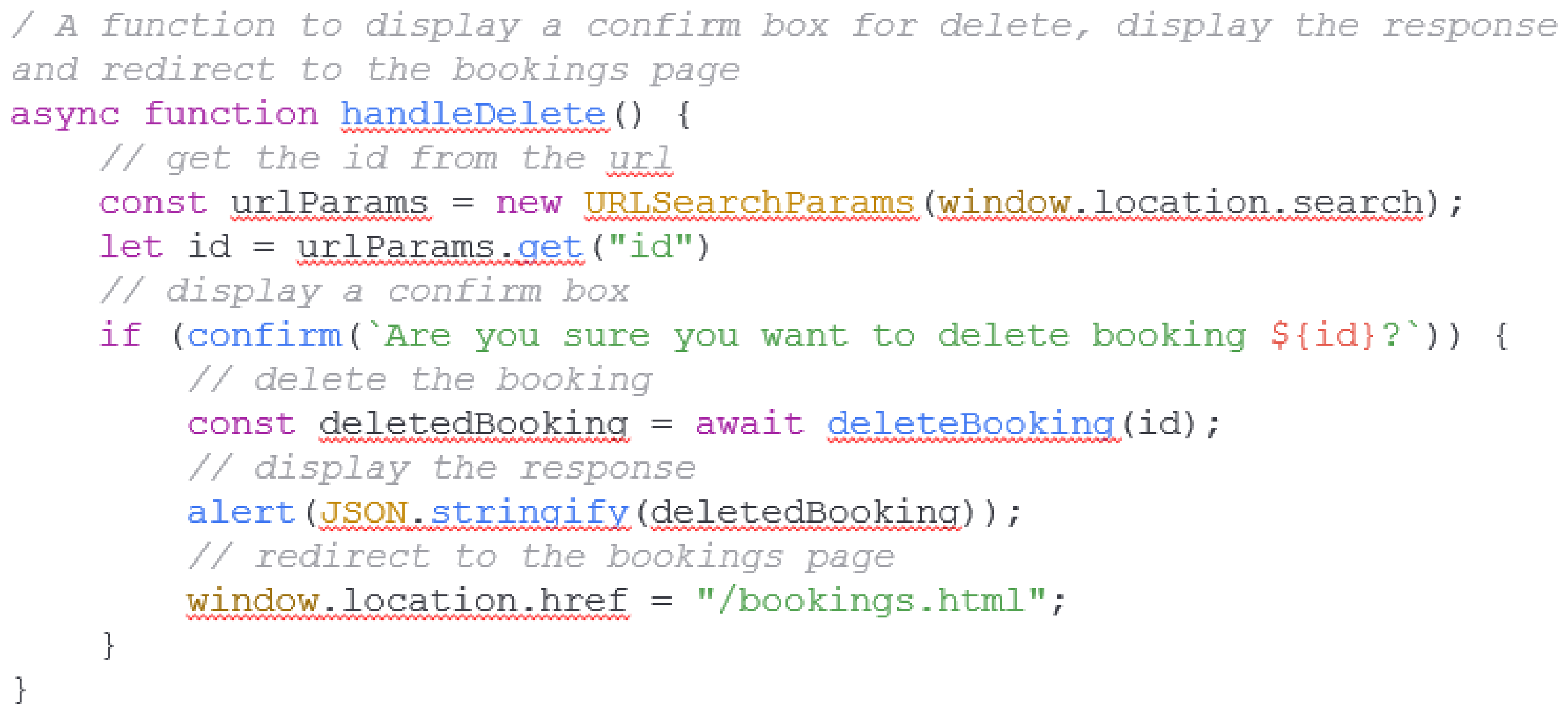
```
// A function to update a booking with www-form-urlencoded data
async function updateBooking(id, booking) {

    const response = await fetch(`/bookings/${id}`, {
        method: 'PUT',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
        },
        body: new URLSearchParams(booking)
    });
    // convert the response to json
    const json = await response.json();
    // return the json
    return json;
}
```

Restful API/AJAX

- When a form is submitted using the **application/x-www-form-urlencoded** format, the form data is encoded in a **key-value pair format**, where each field name and its corresponding value are joined by an equal sign (=), and multiple pairs are separated by an ampersand (&). For example, a key-value pair name=John would be URL-encoded as **name=John**.
- This format is commonly used in HTML forms and is the **default format when submitting HTML forms** without specifying an explicit enctype attribute.

Restful API/AJAX

- Client-side redirect
  - The window.location.href can redirect the user to a new URL.



```
/ A function to display a confirm box for delete, display the response
and redirect to the bookings page
async function handleDelete() {
    // get the id from the url
    const urlParams = new URLSearchParams(window.location.search);
    let id = urlParams.get("id")
    // display a confirm box
    if (confirm(`Are you sure you want to delete booking ${id}?`)) {
        // delete the booking
        const deletedBooking = await deleteBooking(id);
        // display the response
        alert(JSON.stringify(deletedBooking));
        // redirect to the bookings page
        window.location.href = "/bookings.html";
    }
}
```