# Internet Technologies

Event-driven programming in JavaScript (Exercises)

University of Cyprus
Department of Computer Science

# Accessing HTML elements in JavaScript

- Via the querySelector() function:

```
document.querySelector('css selector');
```
JS

  - Returns the **first** element that matches the given CSS selector.

- Via the querySelectorAll() function:

```
document.querySelectorAll('css selector');
```
JS

  - Returns **all** elements that match the given CSS selector.

- `.getElementById()` and `.getElementsByClassName()` are also available but are very specific to ids and classes respectively

## Accessing HTML elements in JavaScript

JS

```
// Returns the DOM object for the HTML element
// with id="button", or null if none exists. **
let element = document.querySelector('#button');

// Returns a list of DOM objects containing all
// elements that have a "quote" class AND all
// elements that have a "comment" class.
let elementList = document.querySelectorAll('.quote, .comment');
```

** `let element = document.getElementById('button');` does the same job but works only for ids.

## Adding event listeners

- Each DOM object has the following method defined:

Called using: object.addEventListener()

```
.addEventListener(event name, function name);
```

  - **event name** is the string name of the JavaScript event you want to listen to -- common ones: `click, focus, blur, change, mouseover, keyup,` etc
  - **function name** is the name of the JavaScript function (handler) you want to execute when the event fires
    - ➢you can have multiple handlers for a single event

## HTML Element Attributes and DOM Object Properties

- Roughly every **attribute** on an HTML element is a **property** on its respective DOM object...

HTML

```html
<img src="puppy.jpg" />
```

JS

```js
const element = document.querySelector('img');
element.id = 'hello';
element.src = 'kitten.jpg'; // change image
```

# DOM Object Properties for all HTML elements

| Property | Description |
|---|---|
| `id` | Gets/sets the value of the id attribute of the element, as a string |
| `innerHTML` | Gets/sets the raw HTML between the starting and ending tags of an element, as a string (parses content as HTML source code) |
| `textContent` | Gets/sets the text content of a node and its descendants. (This property is inherited from Node) (parses content as text only) |
| `classList` | An object containing the classes applied to the element |
| `setAttribute` | Sets the value of attribute on the specified element |

# Other DOM Object Properties Examples

HTML
```html
<div class="myclass"></div>
```

JS
```javascript
const element = document.querySelector('.myclass');
element.id = 'myid';
```

HTML
```html
<div class="myclass" id="myid"></div>
```

JS
```javascript
element.innerHTML = '<em>Hello World</em>';
```

HTML
```html
<div class="myclass" id="myid"><em>Hello World</em></div>
```

*Hello World*

# Other DOM Object Properties Examples

HTML
```
<div class="myclass"></div>
```

JS
```
const element = document.querySelector('.myclass');
element.id = 'myid';
```

HTML
```
<div class="myclass" id="myid"></div>
```

JS
```
element.textContent = '<em>Hello World</em>';
```

HTML
```
<div class="myclass" id="myid">&lt;em&gt;Hello World&lt;/em&gt;</div>
```

<em>Hello World</em>

# Adding and removing classes

- You can control classes applied to an HTML element via `classList.add` and `classList.remove`:

HTML
```html
<img class="hidden" src="puppy.jpg"></div>
```

JS
```js
const image = document.querySelector('img');
// Adds a CSS class called "active".
image.classList.add('active');
// Removes a CSS class called "hidden".
image.classList.remove('hidden');
```

HTML
```html
<img class="active" src="puppy.jpg"></div>
```

Important:
The `classList.toggle` function is a part of the ClassList API and is a convenient way to add or remove a class from an element's list of classes. If the class is present, it gets removed; if not, it gets added.

# Add elements via DOM

- We can create elements dynamically and add them to the web page via `createElement` and `appendChild`:

```
new_elem = document.createElement(tag string);
other_elem.appendChild(new_elem);
```
JS

- Technically you can also add elements to the webpage via `innerHTML`, but it poses a security risk.
- Suggestion: Try not to use innerHTML like this: `element.innerHTML = '<h1>Hooray!</h1>'`

# Example: Add elements via DOM

HTML
```
<div class="row1"></div>
<div class="row2"></div>
```

JS
```
const newHeader = document.createElement('h1');
newHeader.textContent = 'Hooray!';
const element = document.querySelector('div.row1');
element.appendChild(newHeader);
```

HTML
```
<div class="row1"><h1>Hooray!</h1></div>
<div class="row2"></div>
```

## Remove elements via DOM

- We can also call remove elements from the DOM by calling the `remove()` method on the DOM object:

```js
const element = document.querySelector('img');
element.remove();
```

JS

- And actually setting the innerHTML of an element to an empty string is a fine way of removing all children from a parent node:
- This is fine and poses no security risk:
  `element.innerHTML = '';`

## Adding and removing attributes

HTML
```html
<button id="ok">OK</button>
```

JS
```js
const button= document.querySelector("#ok");
button.setAttribute("type", "submit");
button.setAttribute("disabled", "");
```

$\Downarrow$

HTML
```html
<button "type"="submit" id="ok" disabled>OK</button>
```

## Adding handler functions via attributes

using the onclick attribute

HTML
```
<button id="ok">OK</button>
```

JS
```
function myFunction() {
  console.log("Button pressed!")
}
const button= document.querySelector("#ok");
button.setAttribute("onclick", "myFunction()");
```

HTML
```
<button "onclick"="myFunction()" id="ok">OK</button>
```

## Adding handler functions via attributes

using the onclick attribute (can provide fixed input parameters)

HTML
```
<button id="ok">OK</button>
```

JS
```
function myFunction(param) {
  console.log("Button pressed with input: " + param)
}
const button= document.querySelector("#ok");
button.setAttribute("onclick", "myFunction(10)");
```

HTML
```
<button "onclick"="myFunction()" id="ok">OK</button>
```

# HTML input elements

- Single-line text input:

```html
<input type="text" />
```
HTML

- Multi-line text input:

```html
<textarea></textarea>
```
HTML

- In JavaScript, you can read and set the input text via `inputElement.value`

```js
// change event is fired when element loses focus (user leaves element)
// keyup event is fired after every key press
const input = document.querySelector('input');
input.addEventListener('change', myFunction);
```
JS

# HTML input elements

Codepen

- Checkbox:

HTML

```
<label><input type="checkbox" name="size" value="done"/>Option</label>
```

☐ Option

JS

```
const checkbox = document.querySelector('input');
checkbox.addEventListener('change', function() {
  console.log(checkbox.checked);
  console.log(checkbox.value);
})
```

- Select: Option A ▾

HTML

```
<select>
  <option value="1" selected>Option A
  </option>
  <option value="2">Option B</option>
  <option value="3">Option C</option>
</select>
```

JS

```
const selectElem = document.querySelector('select');
selectElem.addEventListener('change', function() {
  const index = selectElem.selectedIndex;
  console.log('index selected: ' + index);
  console.log('option value selected: ' +
selectElem.options[index].value);
  console.log('option text selected: ' +
selectElem.options[index].text);
})
```

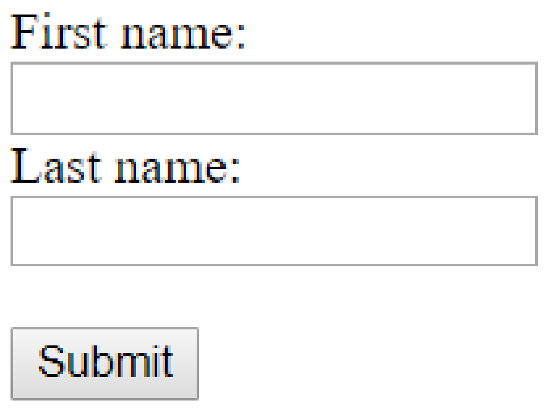
## Form submit

- What if you want to have a form with input elements that can be submitted after you click "enter"?

# Form submit

1. Wrap your input elements in a `<form>` :

```
<form>                                                HTML
  First name:<br/>
  <input type="text" id="fname"/><br/>
  Last name:<br/>
  <input type="text" id="lname"/><br/><br/>
  <input type="submit" value="Submit"/>
</form>
```



- You need to use `<input type="submit">` or `<button type="submit">` instead of `type="button"` to capture "submit" event

## Form submit

2. Listen for the "`submit`" event on the form element:

```js
const form = document.querySelector('form');
form.addEventListener('submit', onFormSubmit);
```
JS

- When you use **`type="submit"`** on `<input>` or on `<button>` the "`submit`" event will fire on form element. When `type="button"` the "`submit`" event is not fired on form element.

## Form submit

3. Capture input data:

```js
function onFormSubmit(event) {
  event.preventDefault();
  const name = document.querySelector('#fname');
  const surname = document.querySelector('#lname');
  console.log(name.value+" "+surname.value);
}
```

- The page will refresh (and data on form will be automatically reset) on submit event unless you explicitly prevent it
  - You may prevent the default action before handling the event through `event.preventDefault()`:

## AJAX

- Asynchronous JavaScript and XML
- Can be used to download/upload data from/to a server in the background (Asynchronous)
- Allows dynamically updating a page without making the user wait and without refreshing the page
- Implemented in vanilla JavaScript using **XMLHttpRequest** or the **Fetch API**

Download data

## Example using XMLHttpRequest (GET)

JS

```
// Set up our HTTP request
const xhr = new XMLHttpRequest();
// Setup our listener to process completed requests
xhr.onreadystatechange = function () {
        // Only run if the request is complete (xhr.readyState = 4)
        if (xhr.readyState !== 4) return;
        // Process our return data
        if (xhr.status >= 200 && xhr.status < 300) {
                // What to do when the request is successful
                console.log(JSON.parse(xhr.responseText));
        } else {
                // What to do when the request has failed
                console.log('Error: ', xhr);
        }
};
// Create and send a GET request
// The first argument is the post type (GET, POST, PUT, DELETE, etc.)
// The second argument is the endpoint URL
xhr.open("GET", "https://api.openaq.org/v1/countries");
xhr.setRequestHeader("Accept", "application/json");
xhr.send();
```

To test in VSCode you need to install xhr2 package via terminal since XMLHttpRequest is a built-in object in web browsers:

```
node install xhr2
```

and then include this line in the first line of your script:

```
const
XMLHttpRequest =
require('xhr2');
```

**Download data**

## Example using Fetch API (GET)

JS

```
fetch("https://api.openaq.org/v1/countries", {
  method: "GET",
  headers: {
    "Accept": "application/json",
  }
})
.then(
  response => { // handle the response
    if (response.status !== 200) {
      console.log('Status Code: ' + response.status);
      return;
    }
    // Parse response as JSON (no need to call JSON.parse())
    response.json().then(
        data => {
            console.log(data);
        }
    );
  } // end of response
) // end of then
.catch( error => { // handle the error
    console.log('Error: ', error);
});
```

## Example using XMLHttpRequest (POST)

**JS**

```
// Set up our HTTP request
const xhr = new XMLHttpRequest();
// Setup our listener to process completed requests
xhr.onreadystatechange = function () {
        // Only run if the request is complete
        if (xhr.readyState !== 4) return;
        // Process our return data
        if (xhr.status >= 200 && xhr.status < 300) {
                console.log(JSON.parse(xhr.responseText));
        } else {
                console.log('error', xhr);
        }
};
// Create and send a POST request. The second argument is the endpoint
URL which will receive the POST message (e.g. a PHP file on the same
server and folder)
xhr.open('POST', 'https://cs.ucy.ac.cy/~csp5pa1/test.php');
xhr.setRequestHeader("Content-type", "application/json");
data = {};
data.name = "John";
data.surname = "Smith";
xhr.send(JSON.stringify(data));
```

- Create a JavaScript object with two properties (name, surname).
- Convert JS object to JSON string.
- Send JSON string to the predefined endpoint.

**Upload data**

## Example using Fetch API (POST)

JS

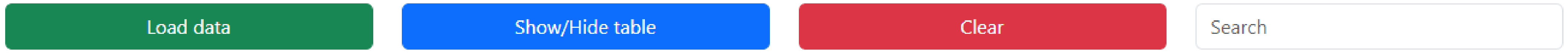
```
fetch('https://cs.ucy.ac.cy/~csp5pa1/test.php', {
  method: "POST",
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
})
.then(
  response => { // handle the response
    if (response.status !== 200) {
      console.log('Status Code: ' + response.status);
      return;
    }
    // Parse response as JSON (no need to call JSON.parse())
    response.json().then(
        data => {
            console.log(data);
         }
    );
  } // end of response
)
.catch( error => { // handle the error
    console.log('Error: ', error);
});
```

```
data = {};
data.name = "John";
data.surname = "Smith";
```

# Exercise 1

- Create the following web page using Bootstrap classes exclusively (avoid creating your custom .css file)
  - CDN-based Boostrap .css and .js files are imported in the given html file

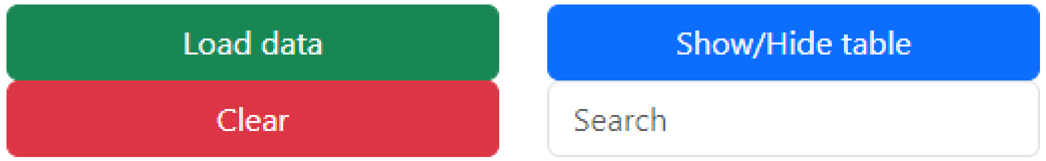
≥768px



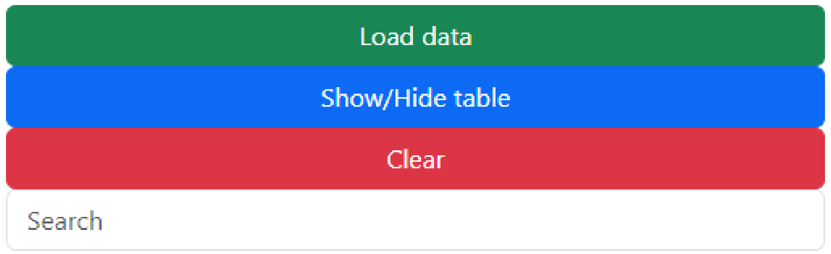
Color classes: .btn-success .btn-primary .btn-alert

Buttons and input element have 100% width (use .w-100 class)

≥576px

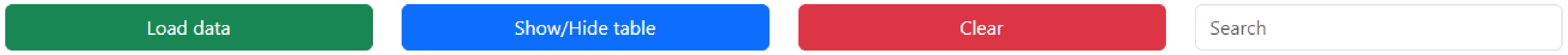


<576px

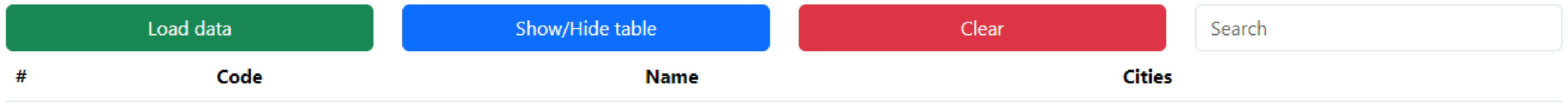


# Exercise 1

- By default, the empty table is hidden (.d-none class is applied, see .html)



When the “Show/Hide table” button is clicked, the table appears/disappears (consider classList.toggle)

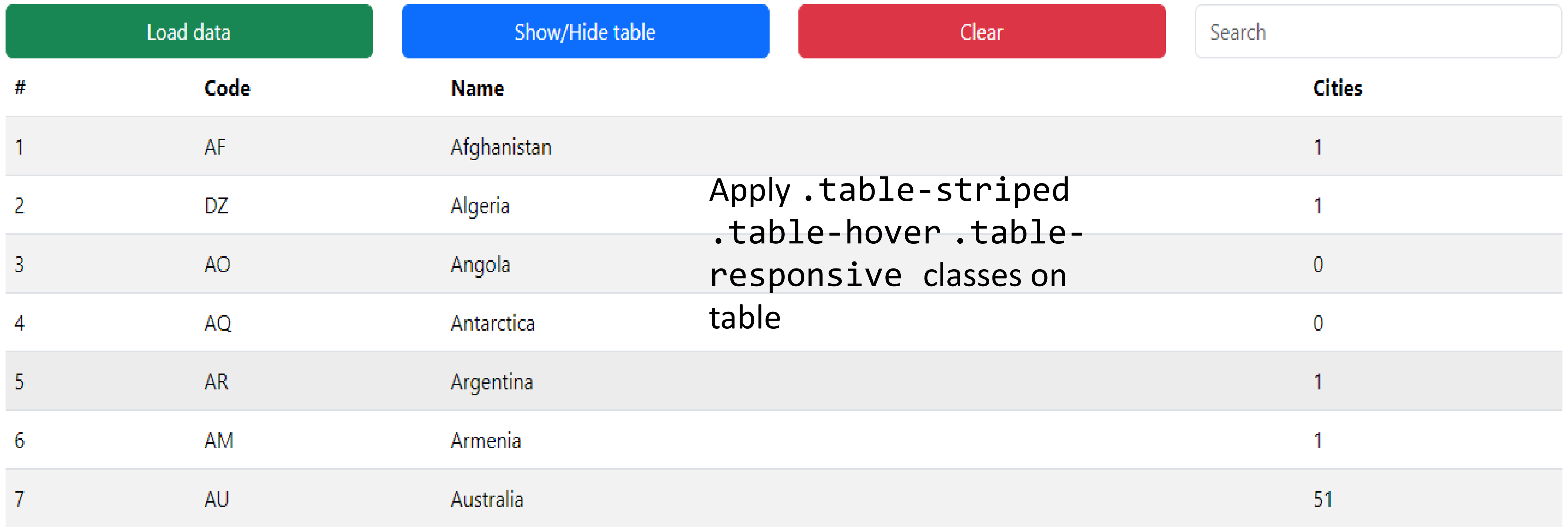


| # | Code | Name | Cities |
|---|---|---|---|

This is table heading (source code provided in .html)

# Exercise 1

- When "Load data" button is clicked, AJAX call (XMLHttpRequest or Fetch API) sends a GET message to https://cs.ucy.ac.cy/courses/EPL425/labs/Lab8/countries.php
- Data received, stored in variable (to enable searching) & displayed in table

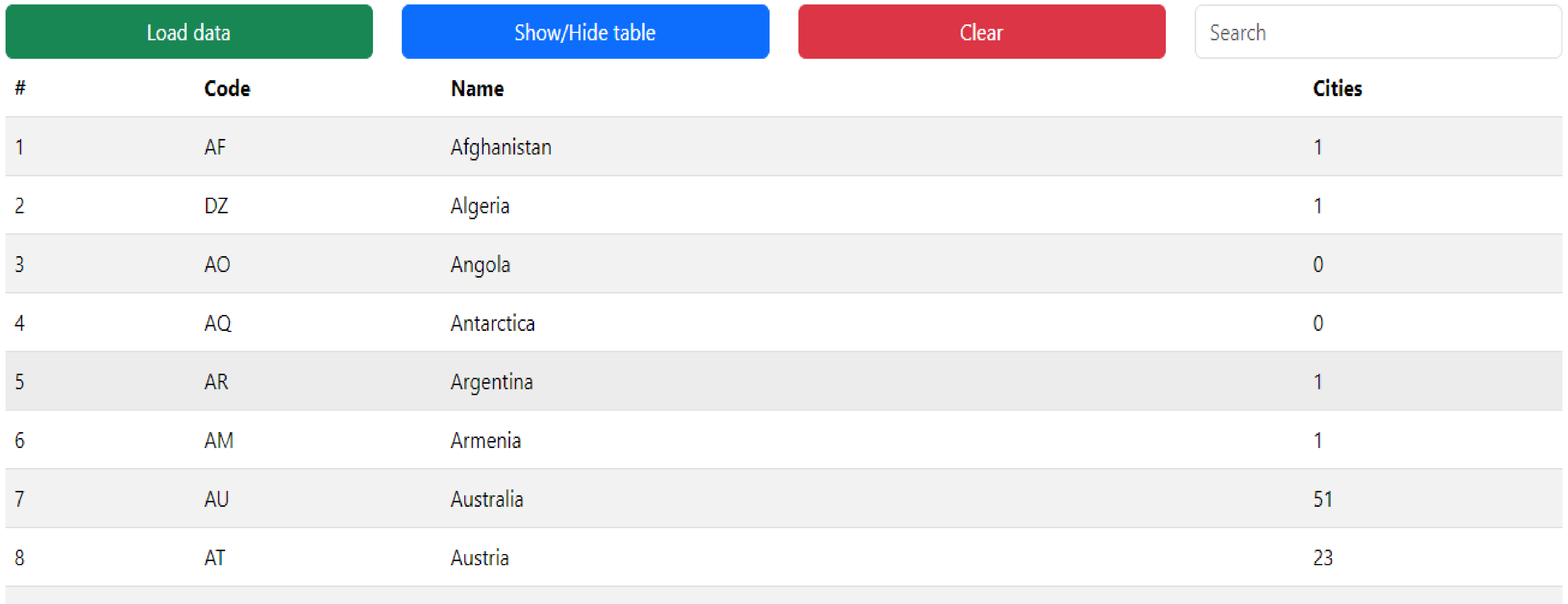


| # | Code | Name | Cities |
|---|---|---|---|
| 1 | AF | Afghanistan | 1 |
| 2 | DZ | Algeria | 1 |
| 3 | AO | Angola | 0 |
| 4 | AQ | Antarctica | 0 |
| 5 | AR | Argentina | 1 |
| 6 | AM | Armenia | 1 |
| 7 | AU | Australia | 51 |

Apply `.table-striped .table-hover .table-responsive` classes on table

# Exercise 1

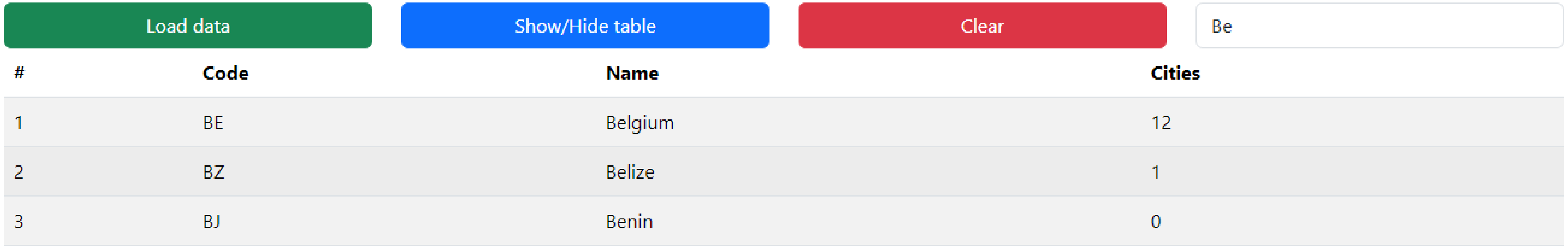
- When "Clear" button is clicked, table contents and data variable are cleared, and table is heading gets hidden (.d-none class)



| # | Code | Name | Cities |
|---|---|---|---|
| 1 | AF | Afghanistan | 1 |
| 2 | DZ | Algeria | 1 |
| 3 | AO | Angola | 0 |
| 4 | AQ | Antarctica | 0 |
| 5 | AR | Argentina | 1 |
| 6 | AM | Armenia | 1 |
| 7 | AU | Australia | 51 |
| 8 | AT | Austria | 23 |

## Exercise 1

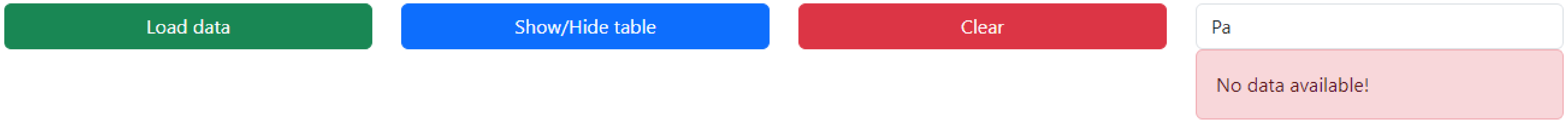
- While user keeps typing in the search box, data displayed in table are updated responsively without losing focus on the input element.



| # | Code | Name | Cities |
|---|---|---|---|
| 1 | BE | Belgium | 12 |
| 2 | BZ | Belize | 1 |
| 3 | BJ | Benin | 0 |

## Exercise 1

- When user starts typing when data is not available (prior loading or after clearing), an appropriate alert message is displayed below the input element (use the boostrap related classes for the alert message).



No data available!

- The alert message is disappeared when data is loaded ("Load data" button is clicked) or after clearing ("Clear" button is clicked)