Front End Web Development I
CSC102

JavaScript CRUD Function

Student data entry

Prepared by Mr. Kinley Tshering

Content

- Designing interactive web form
- Describe JavaScript function
- Define DOM
- Click Event handlers
- Use DOM methods and properties
- Read Data from HTML form
- Inserting record in the table
- Resetting the form to initial state
- Creating Edit function
- Creating Delete function
- Introduction to form validating

Designing Interactive Web Form with JavaScript

- Create an index.html file and save it.
- Fill up the content in index.html file as given below
- Add style.css file in the <head> section

  <link rel="stylesheet" href="style.css">
- Add an external script.js file in the <body>, or in the <head> section of an HTML page enclosed between <script> and </script> tag.

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

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Student Form</title>
</head>
<body>
    <div class="container1">
        <h3>Student Detail Entry</h3>
        <form action="/" method="get">
            <div class="row">
                <div class="col-1">
                    <label >Student ID : </label>
                </div>
                <div class="col-2">
                    <input type="text" id="id" name="stdid" placeholder="STUDENT ID">
                </div>
            </div>
            <div class="row">
                <div class="col-1">
                    <label for="name"> Student Name :* </label>
                </div>
                <div class="col-2">
                    <input type="text" id="name" name="stdtname" placeholder="STUDENT NAME">
                </div>
            </div>
            <div class="row">
                <div class="col-1">
                    <label for="email">Student Email : </label>
                </div>
                <div class="col-2">
                    <input type="email" id="email" name="stdemail" placeholder="EMAIL">
                </div>
            </div><br>
            <div><input type="submit" value="ADD"></div>
```

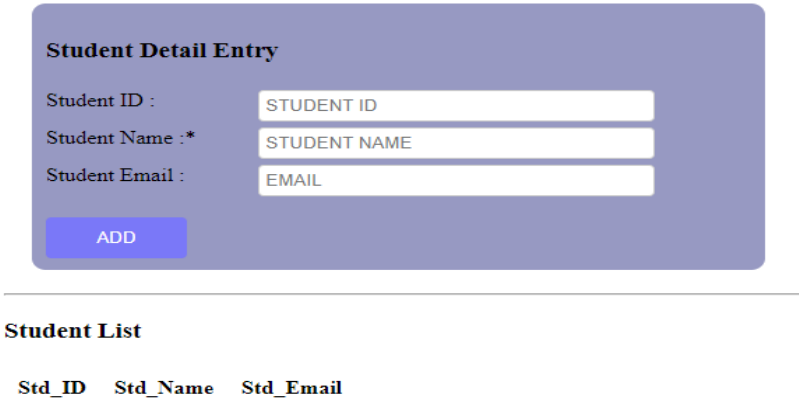
```
        </form>
      </div><hr>
      <div class="container2 list"><h3>Student List</h3>
          <table id="stdList">
              <thead>
                <tr>
                    <th>Std_ID</th>
                    <th>Std_Name</th>
                    <th>Std_Email</th>
                    <th></th>
                    <th></th>
                </tr>
              </thead>
              <tbody>
              </tbody>
          </table>
      </div>
      <script src="script.js"></script>
</body>
</html>
```

Creating style.css file

- Create a style.css file and save it.
- Fill up the content in style.css file as given below

```
.container1{
    width: 50%;
    padding: 10px;
    margin: 20px;
    background-color: rgb(151, 153, 194);
    border-radius: 10px;
}
input {
    width: 90%;
    padding: 5px;
    border: 1px solid #ccc;
    border-radius: 4px;
}
input[type=submit] {
    width: 20%;
    background-color: #7a78f8;
    color: white;
    padding: 10px;
    border: none;
    border-radius: 4px;
    float: left;
```

```
}
.col-1{
    float: left;
    width: 30%;
    margin-top: 5px;
}
.col-2{
    float: left;
    width: 70%;
    margin-top: 5px;
}
.row:after {
    content: "";
    display: table;
    clear: both;
}
#button-1{
    background-color: rgb(241, 44, 44);
    padding-right: 10px;
}
#button-2{
    background-color: rgb(90, 241, 44);
    padding-right: 10px;
}
table{ border-collapse: collapse;}
td {
    border: 1px solid #dddddd;
    padding-right: 10px;
}
.hide{ display: none;}
label.validation-error{
    color: rgb(248, 130, 5);
    margin-left: 5px;
}
a{
    cursor: pointer;
    color: rgb(63, 6, 252);
    margin-right: 4px;
}
/* Responsive layout - when screen size is 1100 at max*/
@media screen and (max-width: 1100px) {
    .container1 { width: 60%; margin-top: 0;}
    .col-1{ width: 30%;}
    .col-2{ width: 60%;}
}
/* Responsive layout - when screen size is 800 at max*/
@media screen and (max-width: 800px) {
    .container1 { width: 80%; margin-top: 0;}
    .col-1, .col-2{ width: 100%;}
}
```



Implementing JavaScript CRUD

```
<form action="/" method="get" onsubmit="event.preventDefault();onFormSubmit();" autocomplete="off">
```

- The `autocomplete` attribute allows the browser to predict the value. When a user starts to type in a field, the browser should display options to fill in the field, based on earlier typed values.
- The `event.preventDefault()` method cancels the default action that belongs to the event or prevents a link from opening the URL.
- Add `onsubmit` event in html form to execute a JavaScript when a form is submitted.
- `onFormSubmit();` is the user defined function in javascript. To invoke this function, `onsubmit` event is used in the form to call the function when the form is being submitted.
- The `onFormSubmit()` function is declared in script.js file as shown below.

```
function onFormSubmit() {
}
```

JavaScript Function Definition:

> JavaScript function is a block of code to perform a particular task and is executed when something invokes/calls it.

Syntax;

Function is defined with the `function` keyword, followed by a **name**, followed by parenthesis **()**.
Parenthesis may include parameters separated by commas.

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Reading data from HTML Form

- To read/retrieve data from HTML form we define separate function called `readFormData()`

```
function readFormData(){
    var formData = {};
    formData["id"] = document.getElementById("id").value;
    formData["name"] = document.getElementById("name").value;
    formData["email"] = document.getElementById("email").value;
    return formData;
}
```

- `var formData` means declaring `formData` variable to store form data.

DOM (Document Object Model)

The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

The `getElementById()` is a DOM method used to return the element that has the ID attribute with the specified value. method returns `null` if no elements with the specified ID exist.

The `value` property sets or returns the value of the value attribute of a text field.

- `formData["id"] = document.getElementById("id").value` finds the HTML element with the `id` as `"id"` and assigns it to the `formData` variable as `formData["id"]`.

- `formData["name"] = document.getElementById("name").value` finds the HTML element with the `id` as `"name"` and assigns it to the `formData` variable as `formData["name"]`.

- `formData["email"] = document.getElementById("email").value` finds the HTML element with the `id` as `"email"` and assigns it to the `formData` variable as `formData["email"]`.

- `return formData` returns value back to caller i.e. `readFormData()` as shown below.

```
function onFormSubmit() {
        var formData = readFormData();
}
```

- Now data retrieved from form is all stored in variable `formData`.

Inserting Record into the table

- To insert data into the `<tbody>` inside the HTML table we define a separate function called `insertNewRecord(data)` with the single parameter.

```
function insertNewRecord(data){
    var table = document.getElementById("stdList").getElementsByTagName('tbody')[0];
    var newRow = table.insertRow(table.length);
    cell1 = newRow.insertCell(0);
    cell1.innerHTML = data.id;
    cell2 = newRow.insertCell(1);
    cell2.innerHTML = data.name;
    cell3 = newRow.insertCell(2);
    cell3.innerHTML = data.email;
}
```

- `var table = document.getElementById("stdList").getElementByTagName('tbody')[0];` it finds the HTML elements by tag name i.e. tbody of table with the id `"stdList"`.

- `var newRow = table.insertRow(table.length);` makes/inserts the row of the table inside tbody of table (`insertRow()` function) with the length of the row.

- With the `insertCell()` function we can insert the cell of the table with the index starting from 0.
- The `innerHTML` property modifies the content of an HTML element, here we insert data inside tbody element using the parameter `data` as shown in the given code.

Resetting the Form

- To reset the form to its initial value we define a function called `resetForm()` with the single parameter.

```
function resetForm(){
    document.getElementById("id").value = "";
    document.getElementById("name").value = "";
    document.getElementById("email").value = "";
}
```

- To clear the text boxes in form after submitting we use the `getElementById` method to find and set the value to an empty string.
- Function call is made in `onFormSubmit` after the `readFromData` function is called as shown below.

```
function onFormSubmit() {
    var formData = readFormData();
    resetForm();
}
```

Editing table data

- We insert an extra two columns for edit and delete operation inside `insertNewRecord()` function.

```
function insertNewRecord(data){
    var table =
document.getElementById("stdList").getElementsByTagName('tbody')[0];
    var newRow = table.insertRow(table.length);
```

```
    cell1 = newRow.insertCell(0);
    cell1.innerHTML = data.id;
    cell2 = newRow.insertCell(1);
    cell2.innerHTML = data.name;
    cell3 = newRow.insertCell(2);
    cell3.innerHTML = data.email;
    cell4 = newRow.insertCell(3);
    cell4.innerHTML = '<a onClick="onEdit(this)">Edit</a>';
    cell5 = newRow.insertCell(4);
    cell5.innerHTML = '<a onClick="onDelete(this)">Delete</a>';
}
```

- `onClick` event executes a JavaScript when the user clicks on an element.
- `onEdit(this)` is the JavaScript **user defined function** to be executed when a user clicks on an Edit link.
- The `this` parameter is passed for reference to the `td` cell HTML element on function invocation as shown below.

```
function onEdit(td){
    selectedRow = td.parentElement.parentElement;
    document.getElementById("id").value = selectedRow.cells[0].innerHTML;
    document.getElementById("name").value = selectedRow.cells[1].innerHTML;
    document.getElementById("email").value = selectedRow.cells[2].innerHTML;
}
```

- `onEdit(td)` is the function with the single parameter td.
- Declare `selectedRow` variable globally and set initial value as `null`.
- `selectedRow` is the variable to store the corresponding HTML row.

```
var selectedRow = null
```

- `td.parentElement.parentElement` : td will have corresponding cell HTML and with parentElement property will have the exact element from the row. That corresponding tr element can be return using .parentElement

- The parentElement property returns the parent element of the specified element.

- To migrate table data back to form for editing
    - `document.getElementById("id").value` will bring that text box with the id "`id`". And `selectedRow.cells[0].innerHTML` will select row with first cell index 0 (`id`) with the innerHTML which will be exact studentId.

    - `document.getElementById("name").value` will bring that text box with the id "`name`". And `selectedRow.cells[1].innerHTML` will select row with first cell index 1 (`name`) with the innerHTML which will be exact studentName.

    - `document.getElementById("email").value` will bring that text box with the id "`email`". And `selectedRow.cells[2].innerHTML` will select row with first cell index 2 (`email`) with the innerHTML which will be exact studentEmail.

- After form is being edited form data needs to be updated for that we define `updateFormData` function with parameter as formData
- Inorder to update student detail in the table based on new values from form we use `updateFormDataForm(formData)` function.

```
function updateFormData(formData){
    selectedRow.cells[0].innerHTML = formData.id;
    selectedRow.cells[1].innerHTML = formData.name;
    selectedRow.cells[2].innerHTML = formData.email;
}
```

- `selectedRow.cells[0].innerHTML` selects the row, index 0 with innerHTML. And `formData.id` updates/saves studentId.
- `selectedRow.cells[1].innerHTML` selects the row, index 1 with innerHTML. And `formData.id` updates/saves studentName.
- `selectedRow.cells[2].innerHTML` selects the row, index 2 with innerHTML. And `formData.id` updates/saves studentEmail.
- It is necessary to reset `selectedRow` to null after resetForm as shown below

```
function resetForm(){
    document.getElementById("id").value = "";
    document.getElementById("name").value = "";
    document.getElementById("email").value = "";
```

```
    selectedRow = null;
}
```

- We use JavaScript conditional statements (if else) to perform different actions based on conditions.
- Execute `insertNewRecord(formData)` function if selectedRow is null Else perform `updateFormData(forData)` function.

```
function onFormSubmit() {
        var formData = readFormData();
        if (selectedRow == null)
            insertNewRecord(formData);
        else
            updateFormData(formData);
        resetForm();
}
```

Deleting table data

- `onDelete(this)` is the JavaScript user defined function to be executed when a user clicks on an Delete link.

```
function onDelete(td){
        row = td.parentElement.parentElement;
        document.getElementById("stdList").deleteRow(row.rowIndex);
        resetForm();
}
```

- Inorder to delete a corresponding row from an HTML table first store corresponding row elements in the row using

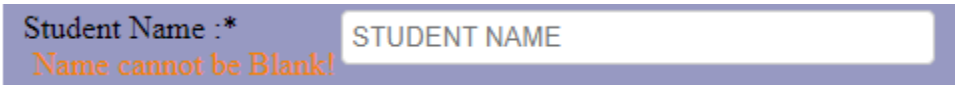
  `row = td.parentElement.parentElement;`

- The `deleteRow()` method removes the row at the specified index from a table.
- `document.getElementById("stdList")` gets the HTML table with `id="stdList"` and `deleteRow(row.rowIndex)` function deletes the row with the corresponding students row index.
- After deletion form needs to be reset and is done by calling `resetForm()` function.

- Before deleting the data, `confirm()` method can be used to confirm the operation with a message inside the method as shown below.

```
function onDelete(td){
    if (confirm('Are You Sure to DELETE this record')){
        row = td.parentElement.parentElement;
        document.getElementById("stdList").deleteRow(row.rowIndex);
        resetForm();
    }
}
```

Form Validation



- When you try to submit the empty form field the function alerts a message, and returns false, to prevent the form from being submitted or being deleted.
- In index.html file

```
<div class="row">
                <div class="col-1">
                    <label for="name"> Student Name :* </label></br>
<label class="validation-error hide" id="nameValidationError">Name cannot be Blank!</label>
                </div>
                <div class="col-2">
                    <input type="text" id="name" name="stdtname" placeholder="STUDENT NAME">
                </div>
            </div>
```

- In style.css file

```
.hide{ display: none;}
```

- In script.js file

```
function validate() {
    isValid = true;
    if(document.getElementById("name").value == ""){
        isValid = false;
        document.getElementById("nameValidationError").classList.remove("hide");
```

```
    }else{
        isValid = true;
        if(document.getElementById("nameValidationError").classList.contains("hide"));
        document.getElementById("nameValidationError").classList.add("hide");
    }
    return isValid;
}
```

- `isValid = true` boolean value is set to true initially
- `if(document.getElementById("name").value == "")` means that the text field with id="name" is empty then we set the boolean value to false. Then we have to show the label.
- To show the label, we have to remove the hide class from the classList and this can be done by `document.getElementById("nameValidationError").classList.remove("hide");`
- The `classList` property returns the CSS classnames of an element.
- In `else` part we do the reverse operation to `if`.
- Initialized `isValid` to true then we check whether we have hide class in this label if there is no hide class we will add the hide class.
- `validate()` function calls should be made before form submission as shown below.

```
function onFormSubmit() {
    if (validate()) {
        var formData = readFormData();
        if (selectedRow == null)
            insertNewRecord(formData);
        else
            updateRecord(formData);
        resetForm();
    }
}
```

#####Thankyou#####