158.258 WEB DEVELOPMENT

JavaScript and Client-side Scripting

School of Mathematical & Computational Sciences
College of Sciences, Massey University
(AKLI, DISD & MTUI)

Revised 2022-08-11

Learning Objectives

After studying this topic, you should be able to:

1. Develop simple JavaScript programs, involving *decision* and *iterative* structures and functions
2. Work with different types of data, including dates, arrays and maps *(Data types and structures)*.
3. Debug JavaScript programs
4. Use client-side scripts in web forms
5. Access HTML elements from JavaScript
6. Display one page from another and share data using query string
7. Use some common *jQuery* functions

Scripting

Client-side scripting

- Code written in interpretive scripting languages like JavaScript, PHP and CGI
- Runs on the client when the browser receives the web page from the server
- Makes web pages more interesting, user-friendly and interactive
- Users can view the script

Server-side scripting

- Code written in scripting languages or programming languages like C# and PHP
- Runs on the server
- Users won't be able to view the script in the browser
- More secure

Components of a Web Project

Diagram illustrating the components of a Web project — should be familiar from past slides on Web application architecture.

(End: Section 1)

Section 2: Example — JavaScript Welcome Program

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      document.write("Welcome to <strong>JavaScript</strong> Programming!")
    </script>
  </head>
  <body>
    ...
  </body>
</html>
```

...

The JavaScript code runs in the browser to dispplay the following:

Welcome to **JavaScript** Programming!

Example - A simple JavaScript program

Purpose	Compute and display the total cost of a reservation
Output	Total cost
Input data	Cost per ticket, Number of tickets
Process *(how)*	Get input data and compute the total cost. Total Cost = Cost per Ticket × Number of Tickets

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      // Declare variables to hold data
      let costPerTicket, tickets, totalCost;
      // Get input data
      costPerTicket = 17.5;
      tickets = prompt("Number of tickets: ");
      // Compute total cost
      totalCost = costPerTicket * tickets;
      // Display result
      document.write("Total Cost: " + totalCost)
    </script>
  </head>
</html>
```

Execute the above JavaScript code by clicking the button: Compute Total Cost

Section 3: JavaScript Data Types

A table of data types in the JavaScript language

Data Type	Description
`Number`	Includes whole numbers and decimal numbers, e.g., 17.5
`string`	A text consisting of a set of characters like the performance name "Rock of Ages" or the zip code "54901". A `string` is enclosed in a pair of double or single quotes.
`boolean`	Represents Boolean data that have values *true* or *false*
`object`	An object has a set of properties representing one or more related data items, and methodfs or functions that perform actions on the object. ▪ `array` is a special tyoe of object that can store a list of data items. ▪ `Date` and `function` are also objects. ▪ `array` is an object that has the value *null*, which means "nothing".
`undefined`	A variable that is not assigned a value has the value "undefined"
`function`	A *function* is a block of statements that does a particular task.
`BigInt`	*BigInt* can store very large integers.

Data Type	Description
`symbol`	*symbol* type is used to create a unique identifier for an object.

Variables in JavaScript

Purpose

- To store data items temporarily while the program is running

Creating variables

- using `var`
  - *`var costPerTicket;`*
  - *`var costPerTicket, tickets, totalCost;`*
- using `let`
  - *`let index;`*

Naming variables

- Must start with a letter, dollar sign ($) or underscore (_), and can contain only letters, digits, dollar signs and underscores
- Must use the Camel Case notation: If a variable name is composed from multiple words, all

words after the first one must start with a capital letter, e.g., `costPerTicket`.

`let` vs `var`

- In JavaScript, both the keywords `var` and `let` are used to declare variables.
- The `let` keyword was introduced in the later version of JavaScript known as ES6(ES2015).
  And it's the preferred way to declare variables.*

`let`	`var`
▪ `let` is block-scoped.	▪ `var` is function scoped.
▪ `let` does not allow to redeclare variables.	▪ `var` allows to redeclare variables.
▪ Hoisting does not occur in `let`.	▪ Hoisting occurs in `var`.

Working with variables

- Assigning values to variables
  - *variableName = value*
  - *Example: costPerTicket = 17.5;*
- Until a variable is assigned a value, it has the value, undefined.
- An undeclared variable will have the value, NaN.
- You may declare and assign values in one statement
  - *let costPerTicket = 17.5, tickets = 2;*

(End: Section 3)

Section 4: Doing Calculations in JavaScripts

A table of mathematical operators in the JavaScript language

Operator	Operation	Example
*	Multilication	Multiply 17.5 by 3: `17.5 * 3`
/	Division	Divide 17.5 by 3: `17.5/3`
+	Addition	Add 17.5 and 0.55: `17.5 + 0.55`
-	Subtraction	Subtract 1.25 from 12.5: `12.5 - 1.25`
-	Unary minus	Negative 2.3: `-2.5`
%	Modulus *(remainder)*	32 modulo 10: `32%10` Result is 2

Precedence of operations

- Operations within parentheses are done first.
- Multiplications and divisions are done before additions and subtractions,
- Calculations take place from left to right.

Mixing String and Numbers

- JavaScript won't raise an error if you mix strings and numbers in an expression
- "+" operator
- JavaScript will convert the number to a string and concatenate the two strings together
  - *6 + "2" -> 62*
  - *"HeLLo" + 23 -> Hello23*
- Operators other than "+" (e.g., multiplication and division)
  - *The string is converted to a number*
    - 6 * "2" -> 12
- The `Number()` function: *converts a string to a number*

```
<script>
  let orderTotal = prompt("Please enter order total");
  let salesTax = prompt("Pleae enter sales tax");
  let amtDue = Number(orderTotal) + Number(salesTax);
  window.alert("Amount Due: " + amtDue);
</script>
```

Execute the above JavaScript code by clicking the button: Compute Amount Due

Debugging: Syntax Errors

- **Syntax errors** are caused by not following the rules of the language
  - *`let costPerTicket tickets, totalCost;` (Syntax error is a missing comma)*

Table presenting the shortcut keys for identifying syntax errors

Browsers	Keyboard Shortcut	How to Display Errors
Chrome	▪ Windows: `Ctrl-Shift-I` the select the tab `Console` ▪ MacOS: `Option-Cmd-I`	Open the browser menu and select `More Tools` > `Developer tools`. Finally, select the tab, `Console`.
Firefox	▪ Windows: `Ctrl-Shift-K` ▪ Mac: `Option-Cmd-K`	Open the browser menu and select `Web Developer` > `Web Console`.
Safari	Mac: `Option-Cmd-C`	▪ Select `Safari, Preferences, Advanced`. ▪ Check `Show Develop menu` in menu bar. ▪ From the `Develop` option, choose `Show JavaScript Console`
Edge	Windows: `F12` > `Ctrl-2`	▪ Open the menu and select `Developer Tools`. ▪ Select the tab `Console`

Debugging: Run-time and logical errors

- **Run-time errors:** identified at run-time
- Examples:

    1. invalid key word

    2. accessing the value of an undeclared variable without assigning any value to it.

    - *`totalCost = CostPerTicket * tickets;` (Hint: capital letter on first variable is an error)*

```
 Uncaught ReferenceError: CostPerTicket is not defined at
Sandbox.html:72
```

3. assigning a value to an undeclared variable in strict mode.

    - *`strict mode: use strict;`*

- **Logical errors:** causes the program to produce incorrect or unintended results

  Example: A syntactically correct formula that is wrong from the point of view of the application domain, e.g., computing costPerTicket by dividing. *Not detected by JavaScript language engine.*

Section 5: Client-side Scripting: Reservation Form

Screenshot of the reservation form. Hear is the real one on the instructor's server.

- HTML forms send data to the server using an HTTP request:
- It's possible for forms to send the HTTP request via a **client-side script** commonly written in **JavaScript**, especially `XMLHttpRequest`.
- HTML forms now mostly send data without loading a new document when response data is received.
- Modern UI's use HTML forms only to collect input from the user, and *not for data submission:*
  - *When the user tries to send the data, the Web application transmits the data asynchronously in the*

background, updating only the parts of the UI that require changes.

- *Sending data asynchronously is generally called AJAX — "Asynchronous JavaScript And XML".*

(Mozilla Developer Network (MDN), Sending forms through JavaScript, Accessed: 2022-08-07)

JavaScript functions

function name(*optional parameters separated by commas*){
  (code to be executed)
}

```
function ComputeCost(){
  "use strict";
  // Declare variables to hold data
  let costPerTicket=17.5, tickets, totalCost;
  tickets = window.prompt("Number of tickets: ");
  // Compute total costs
  totalCost = costPerTicket * Number(tickets);
  // Display result
  window.alert("Total cost: " + totalCost.toFixed(2));
}
```

- Calling a function in HTML code:

```
<button type="button" id="cost" onclick="ComputeCost()">Compute Cost</button>
```

- Since these are HTML slides, you may execute this function by clicking the following button:+

Compute Cost

`document.getElementById()` function

- Accesses an element by specifying the id of the element
- Returns the specified element as an `Element` object, if it exists; if not, it returns `null`
- Examples:

```
let tickets = document.getElementById("tickets").value;
document.getElementById("totalCost").value = totalCost;
document.getElementById("msg2").innerHTML="Please click Continue button to proceed";
```

HTML Events — Used in JavaScript

A Table of events occuring at HTML elements.

Event	When event takes place
onblur	when an element loses focus
onchange	For an input element, onchange is fired when the content of the element changes and the element loses focus. For a select element, onchange is fired when the selection changes.
onlclick	When the element is clicked
onfocus:	When an elementy gets focus
oninput	Immediately after the value of an <input> or <textarea> changes
onload:	When the browser finishes loading a page
onmouseover	When mouse is moved over an element

- These events are attributes of HTML elements and their values could be styles or JavaScript functions, e.g.:

```
<input type="number" onfocus="this.style.backgroundColor='lightgrey'" />
```

(End: Section 5)

Section 6: JavaScript Decision or Selection Structures

- The simple **if** statement

```
if (condition){
  Code (one or more statements) to obe executed, if condition is true.
}
```

```
let discount = 0;
if (discount >= 100){
  discount = 10;
}
```

Relational or Conditional Operators

A table of relational or conditional operators

Relational Operator	Meaning	Example
>	greater than	if (roomCapacity > enrollment)
<	less than	if (dateDue < DateTime.Today)
>=	greater than or equal to	if (examScore >= 90)
<=	less than or equal to	if (dateOfBirth <= DateTime.Parse("12/31/1980")

==	equal to	▪ if (city == "Oshkosh") // city is string type
		▪ if (inState==true) // inState is Boolean type
		▪ // The expression if (inState ==true) may be replaced
		// by if (inState), because inState itself is true or false

!=	not equal to	if (middleInitial != "C")

if-else Statement

```
if (condition){
  // code to be executed,
  // if condition is true
} else{
  // code to be executed,
  // if condition is false
}
```

```
let discount;
if (subTotal >= 100){
  discount = 0;
}else{
  discount = 0;
}
```

if-else-if statement

```
if (condition1) {

  // code to be executed
  // if condition1 is true

} else if (condition2) {

  // code to be executed if
  // condition1 is false and condition2 is true

} else {

  // code to be executed if both
  // condition1 and condition2 are false

}
```

- Example:

```
let discount;
if (subTotal >= 100){
  discount = 10;
} else if (subTotal >= 50) {
  discount = 5;
} else {
  discount = 0;
}
```

- This code is an implementation of the following discount policy:

Condition	Discount
Total cost is 100 or above	10
Total cost is less than 100 but is 50 or above	5
Total cost less than 50	0

The *switch* statement

```
switch (test variable or expression) {
  case value1:
    /*
      Code to be executed if value1 matches the
      value of the test variable or expression.
    */
    break;
  case value2:
    /*
      Code to be executed if value2 matches the
      value of the test variable or expression.
    */
    break;
    ...
  default:
    /*
      Code to be executed if none of the values match
      the value of the test variable or expression.
    */
}
```

Example — switch statement — the `basePrice()` function

```
function BasePrice() {
  let basePrice = 0;
  let performance =
    document.getElementById("performance").value;
  switch (performance){
    case "Young Irelanders":
      basePrice = 65;
      break;
    case "Justin Hayward":
      basePrice = 50;
      break;
    case "TedcOshkosh":
      basePrice = 45;
      break;
    case "OSO Orchestral":
      basePrice = 35;
      break;
    default:
      alert("Invalid performance");
  }
  document.getElementById("basePrice").innerHTML= basePrice;
}
```

Section 7: Global Variables

- Declared outside the functions
- Accessible from all functions
- They do not retain their values after a page refresh.

```
// Declare global variables to hold prices
var basePrice = 0, zonePrice = 0, discount = 0;

function computerCost(){

  "use strict";

  // Declare variables to hold data
  let costPerTicket, tickets, totalCost;

  //Get numbebr of tickets
  tickets = document.getElementById("tickets").valueAsNumber;

  // Compute total cost
  costPerTicket = basePrice + zonePrice - discount;
  totalCost = costPerTicket * tickets;

  // Display result
  document.getElementById("totalCost").innerHTML = totalCost.toFixed(2);

  // Modify instructions for the user
  document.getElementById("msg1").innerHTML = "";
  document.getElementById("msg2").innerHTML = "Please, click Continue button to proceed.";

  // Enable Continue button
```

```
  document.getElementById("continue").disabled = false;
}
```

Sharing data using the `sessionStorage` object

- `SessionStorage` object can be accessed using the `sessionStorage` property of the `window` object:
  - *Lets you store data in the browser.*
  - *Lasts for an entire session.*
  - *The value stored in the sessionStorage survives a page refresh.*
  - *Data can be stored only as strings.*
- Storing data in `sessionStorage`:
  - *`sessionStorage.key = value`*
  - *or, `sessionStorage.setItem("key", "value")`*
- Example:
  - *`sessionStorage.Discount = discount;  // discount is a variable`*
  - *or, `sessionStorage.setItem("Discount", discount);`*

JavaScript Logical operators

Logical Operator	Name	Example	What it means
&&	Conditional And	`if( (ACT >=20) && (GPA>=2.5))` The second Boolean expression is evaluated only if the first one is true.	If both expressions are true, "&&" returns true; if not, "&&" returns false. That is: ▪ true&&true yields true ▪ true&&false yields false ▪ false&&true yields false ▪ false&&false yields false
&	And	`if( (ACT >=20) & (GPA >=2.5))` Both expressions are always evaluated.	Same as "&" (i.e., if both expressions are true, "&" returns true; if not, "&" returns false).
||	Conditional Or	`if ((ACT <20) || (GPA <2.5))` The second Boolean expression is evaluated only if	If either expression is true, "||" returns true; if not, "||" returns false. That is:

Logical Operator	Name	Example	What it means
		the first one is false.	▪ true||true yields true ▪ true||false yields true ▪ false||true yields true ▪ false||false yields false
|	Or	`if ((ACT <20) | (GPA <2.5))`Both expressions are always evaluated.	Same as "||" (i.e., if either expression is true, "|" returns true; if not, "|" returns false).
!	Not	`if (!(ACT >= 20))`	if (ACT >= 20), "!" returns false; if not, "!" returns true.

(End: Section 7)

Section 8: Iteration structures — The `while` loop

- The code within the loop is executed as long as the condition is true

```
while (condition){
  // statements too be executed
}
```

```
<script>
  let password;
  let count = 0;
  while (password != "guest"){
    count = count + 1;
    if (count > 3){
      alert("Sorry, only three attempts allowed");
      exit;
    }
    password = prompt("Please, enter password");
  }
  document.write("You may proceed");
</script>
```

The `do-while` loop

- Checks the condition at the end of loop
- The loop is always executed the first time

```
do {
  // statements to be executed
}
while (condition)
```

```
<script>
  let password;
  do {
    password = prompt("Please, enter password");
  }
  while (password != "guest");
  document.write("You may proceed")
</script>
```

The for loop

- The for loop is particularly suitable for performing a specified number of iterations

```
for (initialisation statement, Boolean expression, increment statement) {
  // statements to be executed
}
```

```
<script>
  let yearlyInvestment = prompt("Enter yearly investment");
  let yearlyInvestment = number(yearlyInvestment);
  let percentGrowth = 5, investmentValue = 0;
  for (let year = 1; year <= 30; year++) {
    investmentValue = (investmentValue + yearlyInvestment) * (1 + percentGrowth/100);
    document.write("Year: " + year + "Investment value: " + investmentValue.toFixed(2) + "<br />");
  }
</script>
```

(End: Section 8)

Section 9: Arrays

Arrays let you store multiple values of the same type or different types, in a single variable.
Creating arrays

```
let array name = [value-1, value-2, ... ];

// or

let array name = new Array(value-1, value-2, ... );
```

For example:

```
let zones = ["suite","premium","circle","balcony"];

// or

let zones = new Array("suite","premium","circle","balcony");
```

Accessing the elements of an array

```
let zone = zones[2]; // zone will have the value “circle”
zones[3] = “gallery”;
```

```
for (i = 0; i < zones.length, i++)
  document.write(zones[i] + "<br />");
```

Methods of arrays

Method	Example
`concat()`	`zones-1.concat(zones-2)` – Concatenates the arrays, zones-1 and zones-2
`sort()`	`zones.sort()` – Sorts the array
`toString()`	`zones.toString()` – Returns a comma separated string.

The Map Object

- The Map object lets you store a set of **key-value** pairs
- Using the Map object:

```
let map name = new Map([ [key-1, value-1], [key-2, value-2], [key-3, value-3], …])
let zonePrices = new Map([["suite", 40], ["premium", 25], ["circle", 15],["balcony", 0]]);
document.write(zonePrices.get("premium")); // displays the price for premium zone
```

- Accessing every element of the object:

```
for (let [zone, price] of zonePrices)
  document.write(zone + ": " + price + "<br />");
```

The ZonePrice() function

```
function ZonePrice() {
  // Find selected zone
  let radioList, button, zone;
  radioList = document.getElementsByName("zone");
  for (button of radioList) {
    if (button.checked)
      zone = button.value;
  }
  // Compute zone price
  switch (zone) {
    ...
    ...
  }
  // Statement to display zone price in the output field, goes here
}
```

(End: Section 9)

Section 10: Working With Dates

- The `Date` type can be used to store a date along with a time
- The `Date` type is an object
- Creating a `Date` object

```
let performDate = new Date("12/14/2022");
let performDateTime = new Date("12/14/2022 2:30");
let performDate = new Date(document.getElementById("performDate").value);
```

Methods of the Date object

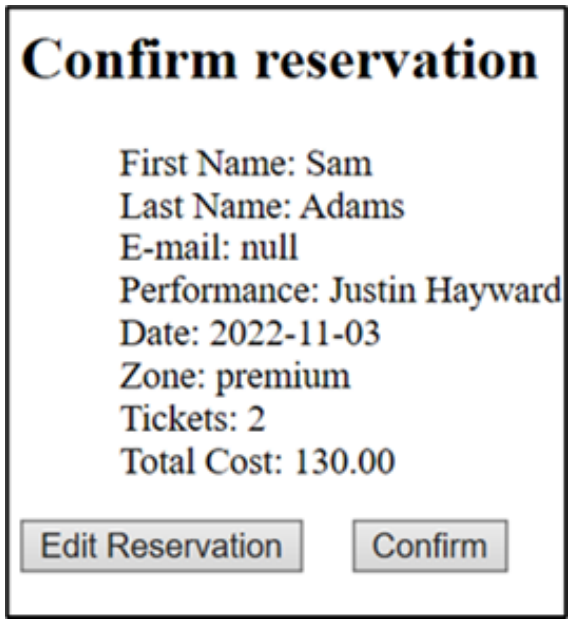
Method	Description
`getDate()`	Returns the day of the month as a number, 1 – 31.
`getDay()`	Returns the day of the week as a number, 0 – 6.
`getFullYear()`	Returns the year
`getMonth()`	Returns the month as a number, 0 – 11.
`toDateString()`	Converts the date portion of a Date object into a string.
`toString()`	Converts a Date object to a string.

Validating the performance date

```
function validateDate() {
  let performDate = new Date(document.getElementById("performDate").value);
  if (performDate < new date())
    alert("Date must be current or future date");
}
```

```
<label for = "performDate">Date:</label>
<input type="date" name="performDate" id="performDate" required onblur="validateDate()"><br /><br />
```

Sharing data with Confirmation page



Method	Description
Get method that uses Query string	If you set the method attribute of a form element to get, the browser will send the input data to the page specified in the action attribute, by appending the data to the URL as query string. A drawback of using query string is that it is not suitable for sending sensitive data.
Post method that uses HTTP request	If you set the method attribute to post, the input data is sent by adding it to the body of the Http request, making it more suitable for sensitive and larger sets of data. However, retrieving the values is not as simple as with query string.
Session variables	Session variables store data in the server memory. Therefore, retrieving the data involves using server-side code. Session variables are discussed

Method	Description
	in more detail and used in chapters 9 and 17.
localStorage and sessionStorage objects	You can use the sessionStorage object, which lasts only for a session, or the localStorage, which spans multiple sessions, to save data in the browser as key-value pairs, and retrieve from any page. Similar to using query string, these methods are not suitable to share sensitive data.
Hidden fields	The hidden field (<input type="“hidden”">) is an input field that is not visible to the user when the form is displayed in a browser.

Retreiving data from query string

- The `window.location.search` property
  - *The search property of the `window.location` object returns the entire query string as a single string*

```
<body>
```

```
  <h2>Confirm Reservation</h2>
  <p id="qs"></p>
  <script>
    // Get the entire query string
    let queryString = location.search;
    document.getElementById("qs").innerHTML = "Query String: " + "<br />" + queryString;
  </script>
</body>
```

- `URLsearchParams` object
  - *The `get()` method of `URLSearchParms` gets the value for any specific variable (key) from the query string*

```
<script>
  // get the entire query string
  let queryString = location.search;
  // Create URLSearchParams object using the query string
  let urlParams = new URLSearchParams(queryString);
  // Get first name
  let firstName = urlParams.get("fName");
  document.getElementById("qs").innerHTML = "First Name: " + firstName;
</script>
```

Concatenating data items together

```
// get each data item and concatenate them together to form a single string
let resData ="";
resData += "First Name: " + urlParams.get("fName") + "<br />";
resData += "Last name: " + urlParams.get("lName") + "<br />";
resData += "E-mail: " + urlParams.get("email") + "<br />";
resData += "Performance: " + urlParams.get("performance") + "<br />";
resData += "Date: " + urlParams.get("performDate") + "<br />";
resData += "Zone: " + urlParams.get("zone") + "<br />";
resData += "Tickets: " + urlParams.get("tickets") + "<br />";
resData += "Total Cost: " + urlParams.get("totalCost") + "<br />";
// Display the concatenated string in the <p> element
document.getElementById("qs").innerHTML = resData;
```

Displaying one page from another

- `window.history` object: `history.back()` and `history.forward()` methods

```
<button type="button" onclick="history.back()">Edit Reservation</button>
```

- anchor (<a>) element

```
<a href="Reservation-2.html"> <input type="button" value="Edit Reservation"/></a>
```

- window.location object

```
<button type="button" onclick= "GoToReservation()">Edit Reservation</button>
```

```
<script>
  function goToReservation() {
    location.href = "reservation-2.html";
  }
</script>
```

(End: Section 10)

Section 11: jQuery Library

- jQuery: *a library of JavaScript functions that perform many common tasks*
- How to use jQuery:
  - *download the jQuery library into the project folder*
  - *use it from a Content Delivery Network (CDN) like Google or Microsoft*
- jQuery methods:
  - *performs an action (e.g., change the background color or content) on a group of selected elements (e.g., all input fields or headings):*

```
$(selector).action
```

- Types of actions:
  - *Add/change HTML elements and*

If you don't want to download and host jQuery yourself, you could include it from a CDN (Content Delivery Network), for example, Google hosts the jQuery Library as illustrated below.

```
<head>

  <!-- Linking to local jquery library -->
  <script src="js/jquery-3.6.0.min.js" />

  <!-- Linking to jquery library in CDN -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0
        /jquery.min.js" />

</head>
```

attributes

- *Find HTML elements based on their position relative to other elements*
- *Produce effects*
- *Provide AJAX (Asynchronous JavaScript and XML) functionality*

jQuery Methods — I: text(), html() and val()

- **`text()` method:** sets or returns the text content of the selected element(s)

```
// JavaScript code to set the text content of selected element
document.getElementById("msg2").innerHTML = "Plese click Continue button";

// jQuery code to set the text content of selected element
$("#msg2").text("Please click Continue button");
```

- **`html()` method:** similar to the `text()` method, except that you can set an html instead of text

```
// JavaScript code to insert HTML code into selected element
document.getElementById("msg2").innerHTML = "<strong>Please click Continue button to proceed</strong>";

// jQuery code to insert HTML code into selected element
$("#msg2").html("<strong>Please click Coninue button</strong>");
```

- **`val()` method:** sets or returns the value of form fields

```
<!-- HTML -->
<input type="number" name="tickets" id="tickets" ... value="1" />

<script>
// jQuery
$("#tickets").val("0");
</script>
```

jQuery Methods — II: Selecting multiple elements

- Example: Reset the text content of every element whose class attribute is set to `output`.

```
    $("output").text("0");
```

- The JavaScript function that uses this JQuery method is as follows:

```
<script>
  function resetOutput() {
    $("output").text("0");
  }
</script>
```

- The function is then invoked from the form's `onreset` attribute as follows:

```
<form action="Confirmation.html" id="reservation" onsubmit="return ValidData()“ method="get" onreset="ResetOutput()">
```

jQuery Methods — III: Adding elements

- The `after()` and `before()` methods
- Example: Add a element and the content following the <output> element (`#totalCost`)

```
$("#totalCost").after("<br><br><div id='msg2'>Please click Continue button</div>“)
```

jQuery Methods — IV: Event Methods

- Event methods: *take actions in response to events*
- Examples: `click()`, `focus()`, `change()`, `submit()`

```
    $(selector).eventmethod (function(){code})
```

- Example: *Change the background color for <input> fields whose value was changed.*

```
<script>
  $(document).ready(
    function () {
      $("input").change(
        function () {
          $(this).css("background-color", "lightgreen");
        }
      )
    }
  );
</script>
```

(End: Section 11)

Section 12: Topic Summary

In this topic, we covered the following aspects of the JavaScript language:

1. Simple JavaScript programs, involving *decision* and *iterative* structures and functions
2. Different types of data, including dates, arrays and maps *(Data types and structures)*.
3. Debugging JavaScript programs
4. Client-side scripts in web forms
5. Accessing HTML elements from JavaScript
6. Displaying one page from another and sharing data using query string
7. Some common *jQuery* functions