CoSc3081
Web Programming

Instructor: Zinabu H.

zinabuscholar@gmail.com
zinabu@aku.edu.et

2024

Chapter 4

Client Side Scripting (JavaScript)

- JavaScript Basics
- Array in JavaScript
- JavaScript Functions
- JavaScript DOM
- Form Processing in JavaScript
- JavaScript BOM

Introduction to JavaScript

- JavaScript is a popular programming language that has a wide range of applications.
- JavaScript was previously used mainly for making webpages **interactive** such as form validation, animation, etc.
- Nowadays, JavaScript is also used in many other areas such as **server-side development**, **mobile app development** and so on.

Introduction to JavaScript

- Run a JavaScript in several ways
  - Using console tab of web browsers
  - Using Node.js
  - By creating web pages

JavaScript Comments

- JavaScript comments are hints that a programmer can add to make their code easier to read and understand. They are completely ignored by JavaScript engines.
- There are two ways to add comments to code:
  - // - Single Line Comments
  - /* */ -Multi-line Comments

JavaScript Input Output

- JavaScript can take inputs using
  - prompt('msg')
  - HTML element
- JavaScript can "display" data in different ways
  - Writing into an HTML element, using innerHTML.
  - Writing into the HTML output using document.write().
  - Writing into an alert box, using window.alert().
  - Writing into the browser console, using console.log()

JavaScript Data Types and Variables

- There are different types of data that we can use in a JavaScript program

Data Types	Description	Example
String	represents textual data	'hello', "hello world!" etc
Number	an integer or a floating-point number	3, 3.234, 3e-2 etc.
BigInt	an integer with arbitrary precision	900719925124740999n , 1n etc.
Boolean	Any of two values: true or false	true and false
undefined	a data type whose variable is not initialized	let a;
null	denotes a null value	let a = null;
Symbol	data type whose instances are unique and immutable	let value = Symbol('hello');
Object	key-value pairs of collection of data	let student = { };

JavaScript Data Types and Variables…

JavaScript Variables and Constants

- In programming, a variable is a container (storage area) to hold data
- In JavaScript, we use either var or let keyword to declare variables.
- **var** is used in the older versions of JavaScript
- **let** is the new way of declaring variables starting ES6 (ES2015).
- **Note:** It is recommended we use let instead of var. However, there are a few browsers that do not support let

JavaScript Data Types and Variables…

Rules for Naming JavaScript Variables

- Variable names must start with either a letter, an underscore _ , or the dollar sign $
- JavaScript is case-sensitive. So y and Y are different variables
- Keywords cannot be used as variable names
- Note:
  - Though you can name variables in any way you want, it's a good practice to give a descriptive variable name. If you are using a variable to store the number of apples, it better to use apples or numberOfApples rather than x or n.

JavaScript Data Types and Variables…

JavaScript Constants

- The **const** keyword was also introduced in the ES6(ES2015) version to create constants

```
const x = 5;
```

JavaScript Data Types and Variables…

Data Type Conversion

- Convert string to Number

```
let myStrNum = '5';
```

- Convert number to string

```
let myNum  = 5;
```

- Convert String to decimal

```
let  myStrNum  = '5.235';
```

JavaScript Operators

What is an Operator?

- In JavaScript, an operator is a special symbol used to perform operations on operands (**values** and **variables**).
- **JavaScript Operator Types**
  - Assignment Operators
  - Arithmetic Operators
  - Comparison Operators
  - Logical Operators
  - Bitwise Operators
  - String Operators
  - Other Operators

JavaScript Operators

JavaScript Assignment Operators

- The commonly used assignment operator is =.
- You will understand other assignment operators such as +=, -=, *=,/= ,%=,**=.
- once we learn arithmetic operators.
- ** Exponentiation Assignment

```
a **= 2; // a = a**2
```

JavaScript Operators : Comparison

- Comparison operators compare two values and return a boolean value, either **true** or **false**

Operator	Description	Example
==	Equal to: returns true if the operands are equal	x == y
!=	Not equal to: returns true if the operands are not equal	x != y
===	Strict equal to: true if the operands are equal and of the same type	x === y
!==	Strict not equal to: true if the operands are equal but of different type or not equal at all	x !== y
>	Greater than: true if left operand is greater than the right operand	x > y
>=	Greater than or equal to: true if left operand is greater than or equal to the right operand	x >= y
<	Less than: true if the left operand is less than the right operand	x < y
<=	Less than or equal to: true if the left operand is less than or equal to the right operand	x <= y

JavaScript Operators …

JavaScript Logical Operators

- Logical operators perform logical operations and return a boolean value, either **true** or **false**

Operator	Description	Example
&&	**Logical AND**: true if both the operands are true, else returns false	x && y
||	**Logical OR**: true if either of the operands is true; returns false if both are false	x || y
!	**Logical NOT**: true if the operand is false and vice-versa.	!x

JavaScript Operators …

JavaScript Bitwise Operators

- Bitwise operators perform operations on binary representations of numbers

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left shift
>>	Sign-propagating right shift
>>>	Zero-fill right shift

JavaScript Operators …

JavaScript String Operators

- In JavaScript, you can also use the **+** operator to concatenate (join) two or more **strings**

```
// concatenation operator
console.log('hello' + 'world');

let a = 'Web';

a += ' Programming';  // a = a + ' Programming';
console.log(a);
```

JavaScript Operators …

Other JavaScript Operators

Operator	Description	Example
,	evaluates multiple operands and returns the value of the last operand.	let a = (1, 3 , 4); // 4
?:	returns value based on the condition	(5 > 3) ? 'success' : 'error'; // "success"
delete	deletes an object's property, or an element of an array	delete x
typeof	returns a string indicating the data type	typeof 3; // "number"
void	discards the expression's return value	void(x)
in	returns true if the specified property is in the object	prop in object
instanceof	returns true if the specified object is of of the specified object type	object instanceof object_type

Control Structure

- In computer programming, there may arise situations where you have to run a block of code among more than one alternatives
- In JavaScript, there are three forms of the if...else statement.
  - **if** statement
  - **if...else** statement
  - **if...else if...else** statement

The syntax of the if statement is:

```
if (condition) {
    // the body of if
}
```

Control Structure

The syntax of the if .. else statement is:

```
if (condition) {
   // block of code if condition is true
} else {
    // block of code if condition is false
 }
```

The syntax of the if...else if...else statement is:

```
if (condition1) {
    // code block 1
} else if (condition2){
    // code block 2
} else {
    // code block 3
}
```

Control Structure

JavaScript switch Statement

- If you need to make a choice between more than one alternatives based on a given test condition, the switch statement can be used

The syntax of the switch statement is:

```
switch(variable/expression) {
    case value1:
        // body of case 1
        break;
    case value2:
        // body of case 2
        break;
    case valueN:
        // body of case N
        break;
    default:
        // body of default
}
```

Control Structure

JavaScript loops

- In programming, loops are used to repeat a block of code
- In JavaScript, there are three forms of loops
  - **for** loop
  - **do while** loop
  - **while** loop

The syntax of the **for** loop is:

```
for (initialExpression; condition; updateExpression) {
    // for loop body
}
```

Control Structure

JavaScript loops

The syntax of the **while** loop is:

```
while (condition) {
  // body of loop
}
```

The syntax of the **do-while** loop is:

```
do{
  // body of loop
} while (condition)
```

Note: do...while loop is similar to the while loop. The only difference is that in do…while loop, the body of loop is executed at **least once**.

Control Structure

for Vs while Loop

- A for loop is usually used when the number of **iterations is known**
- And while and do...while loops are usually used when the number of **iterations are unknown**.

Control Structure

JavaScript break Statement

- The break statement is used to terminate the loop immediately when it is encountered

The syntax of the break statement is:

```
break [label];
```

Note:

label is optional and rarely used.

The break statement is almost always used with decision-making statements.

Control Structure

JavaScript continue Statement

- The continue statement is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.

The syntax of the break statement is:

```
continue [label];
```

Note:

label is optional and rarely used.

The break statement is almost always used with decision-making statements.

Array in JavaScript

- An array is an **object** that can store multiple values at once.

Create an Array

- We can create an array by placing elements inside an array literal [], separated by commas.
- Example

```
const numbers = [10,21,19,70,18];
```

Access Elements of an Array

- Each element of an array is associated with a number called an **index**. The index specifies the position of the element inside the array.

```
Example: console.log(numbers[2]);
```

Array in JavaScript…

Add Element to an Array

- We can add elements to an array using built-in methods like **push()** and **unshift()**
- The push() method adds an element at the **end** of the array.

  Example

```
numbers.push(29);
```

- The unshift() method adds an element at the **beginning** of the array.

  Example

```
numbers.unshift(11);
```

Array in JavaScript…

Change the Elements of an Array

- We can add or change elements by accessing the index value

Example

```
numbers[1]=27;
```

Remove Elements from an Array

- We can remove an element from any specified index of an array using the **splice()** method.

Example

```
// remove one element at the index 2
numbers.splice(2, 1);
```

Array in JavaScript...

splice() Syntax

```
arr.splice(start, deleteCount, item1, ..., itemN)
```

splice() Parameters

- The splice() method takes in:
  - ***start*** - The index from where the array is changed.
  - ***deleteCount*** (optional) - The number of items to remove from start.
  - ***item1, ..., itemN*** (optional) - The elements to add to the start index. If not specified, splice() will only remove elements from the array.

Array in JavaScript…

Length of an Array

- We can find the length of an array using the length property

```
arr.length;
```

Array in JavaScript…

Array Methods

Method	Description
concat()	Joins two or more arrays and returns a result.
indexOf()	Searches an element of an array and returns its position.
find()	Returns the first value of an array element that passes a test.
findIndex()	Returns the first index of an array element that passes a test.
forEach()	Calls a function for each element.
includes()	Checks if an array contains a specified element.
sort()	Sorts the elements alphabetically in strings and in ascending order.
slice()	Selects the part of an array and returns the new array.
splice()	Removes or replaces existing elements and/or adds new elements.

Array in JavaScript…

JavaScript Multidimensional Array

- A multidimensional array is an array that contains another array.

Create a Multidimensional Array

For example

```
let studentsData = [['Hagos', 24],
                    ['Sara', 23],
                    ['Berhe', 21]];

let student1 = ['Hagos', 24];
let student2 = ['Sara', 23];
let student3 = ['Berhe', 21];
// multidimensional array
let studentsData = [student1, student2, student3];
```

Array in JavaScript…

Access Elements of a Multidimensional Array

- You can access the elements of a multidimensional array using indices (0, 1, 2 …)

For example

```
let studentsData = [[‘Hagos', 24],
                    ['Sara', 23],
                    [‘Berhe', 21]];

 // access the first item
 console.log(studentsData[0]); // [“Hagos", 24]
```

Array in JavaScript…

Add an Element to a Multidimensional Array

- You can use the Array's **push()** method or an indexing notation to add elements to a multidimensional array.

For example

```
  //outer Array
  studentsData.push([Tadesse', 24]);
   console.log(studentsData);

    //Inner Array
let studentsData = [[‘Hagos', 24], ['Sara', 23]];
studentsData[1][2] = ‘CS';
console.log(studentsData);
        // [[”Hagos", 24], ["Sara", 23, ”CS"]]
```

Array in JavaScript…

Remove an Element from a Multidimensional Array

- You can use the Array's **pop()** method to remove the element from a multidimensional array.

For example

```
//Remove Element from Outer Array
studentsData.pop();
 console.log(studentsData);  //['Hagos', 24],

 //Remove Element from Inner Array
let studentsData = [['Hagos', 24], ['Sara', 23]];
studentsData[1].pop();
console.log(studentsData);
      // [["Hagos", 24], ["Sara", 23, "CS"]]
```

Array in JavaScript…

Iterating over Multidimensional Array

- You can iterate over a multidimensional array using the Array's **forEach()** method to iterate over the multidimensional array

For example

```
let studentsData = [[‘Hagos', 24], ['Sara',23],];

// iterating over the studentsData
studentsData.forEach((student) => {
    student.forEach((data) => {
        console.log(data);
    });
});
```

JavaScript Functions

- A function is a block of code that performs a specific task.

Declaring a Function

The syntax to declare a function is:

```
function nameOfFunction () {
    // function body
}
```

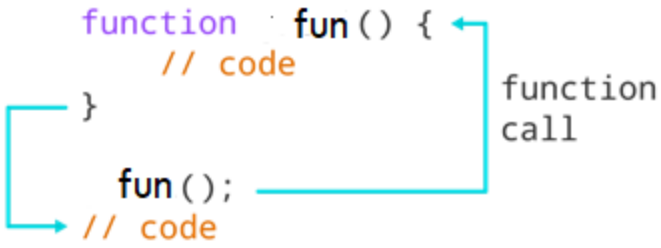
Note:

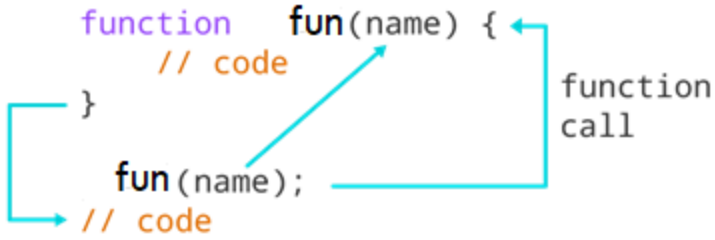
- A function is declared using the function keyword.
- The basic rules of naming a function are similar to naming a variable. It is better to write a descriptive name for your function. For example, if a function is used to add two numbers, you could name the function add or addNumbers

JavaScript Functions

Calling a Function

```
nameOfFunction() ;
```





Note:

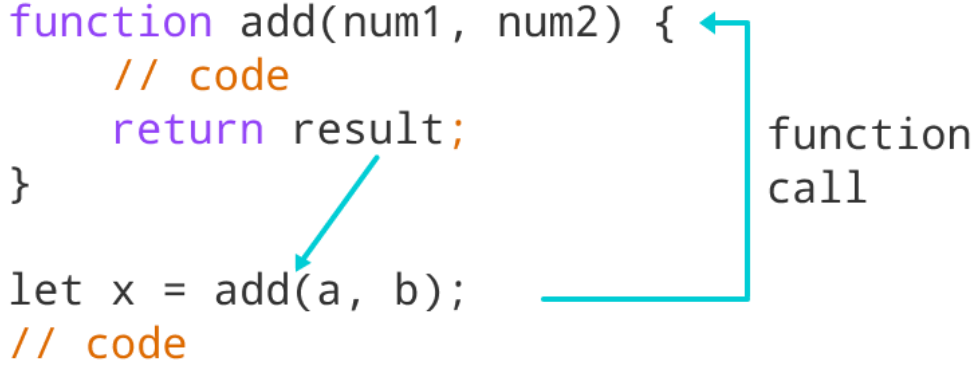
A function can also be declared with parameters.

A parameter is a value that is passed when declaring a function.

JavaScript Functions

Function return

- The return statement can be used to return the value to a function call.



Benefits of Using a Function

- Function makes the code reusable. You can declare it once and use it multiple times.
- Function makes the program easier as each small task is divided into a function.
- Function increases readability.

JavaScript Functions

CallBack Function

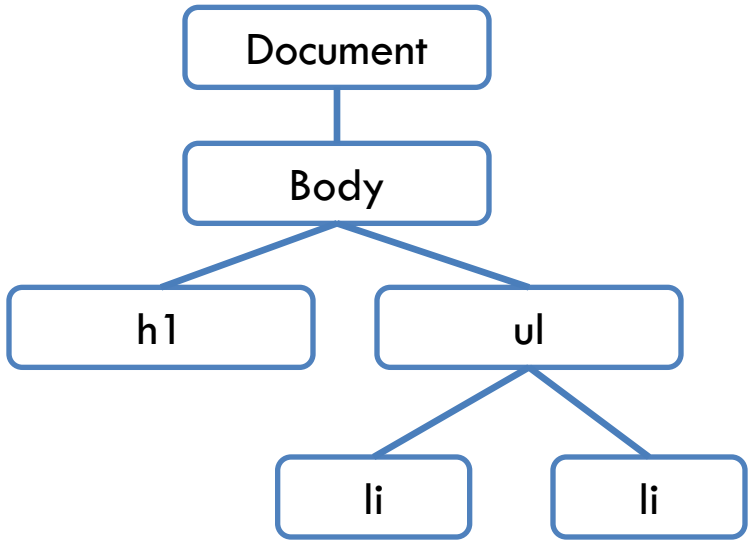
- When we provide function as an (argument) to other function that function is known as callback function.

Example on board

JavaScript DOM

- When a web page is **loaded**, the browser creates a Document Object Model of the page.
- The **HTML DOM** model is constructed as a tree of **Objects:**

```
<body>
 <h1>JS DOM</h1>
 <ul>
     <li>1</li>
  <li>2</li>
  </ul>
</body>
```



JavaScript DOM

Accessing HTML elements in JavaScript

- **DOM Selectors**

- getElementById()
  - document.getElementById() method returns an Element object that represents an HTML element with an id that matches a specified string.
  - If the document has no element with the specified id, the document.getElementById() returns null.

Syntax

```
const element = document.getElementById(id);
```

JavaScript DOM

Accessing HTML elements in JavaScript

- **DOM Selectors**
  - getElementByName()
    - The getElementsByName() accepts a name which is the value of the **name** attribute of elements and returns a live **NodeList** of elements

Syntax

```
const element = document.getElementByName(name);
```

JavaScript DOM

Accessing HTML elements in JavaScript

- **DOM Selectors**
  - getElementByTagName()
    - It is a method of the document object or a specific DOM element.
    - This method accepts a tag name and returns a live **HTMLCollection** of elements with the matching tag name in the order which they appear in the document.
    - The HTMLCollection is an array-like object.

Syntax

```
const elem = document.getElementByTagName(tagname);
```

JavaScript DOM

Accessing HTML elements in JavaScript

- **DOM Selectors**

- getElementByClassName()
  - The getElementsByClassName() method returns an array-like of objects of the child elements with a specified class name.
  - The getElementsByClassName() method is available on the document element or any other elements.

Syntax

```
let elem = document.getElementByClassName(name);
```

JavaScript DOM

Accessing HTML elements in JavaScript

- **DOM Selectors**
  - querySelector()
    - The querySelector() method allows you to select the first element that matches one or more CSS selectors.

Syntax

```
let element = parentNode.querySelector(selector);
```

JavaScript DOM

Event in JavaScript

- An event is an action that occurs in the web browser, which the web browser feedbacks to you so that you can respond to it.
- Each event may have an event handler which is a block of code that will execute when the event occurs.
- An event handler is also known as an event listener. It listens to the event and executes when the event occurs.

JavaScript DOM

Event in JavaScript

- Suppose you have a button with the id **btn**:

```
<button id="btn">Click Me!</button>
```

- To define the code that will be executed when the button is clicked: **addEventListener()** method

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

function display() {
    alert('It was clicked!');
}

btn.addEventListener('click',display);
```

JavaScript DOM

Event in JavaScript

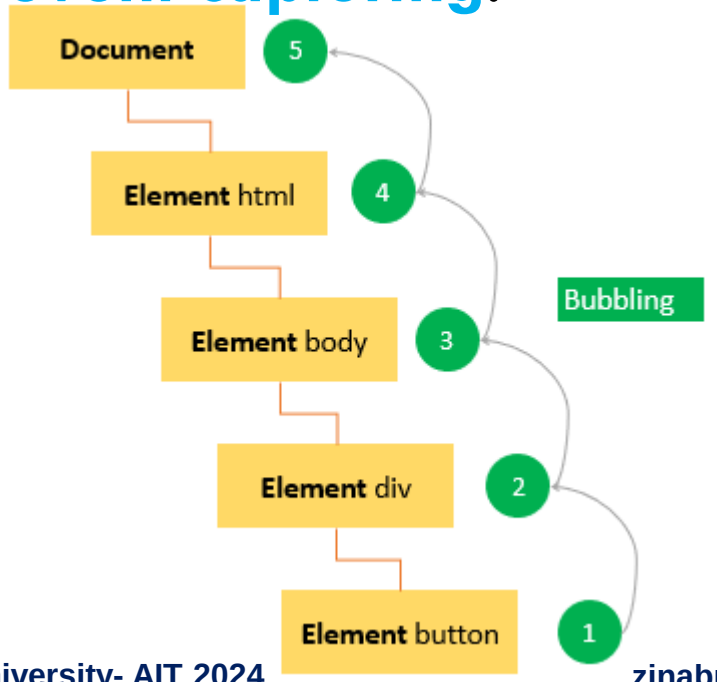
- A shorter way to register an event handler is to place all code in an **anonymous function**

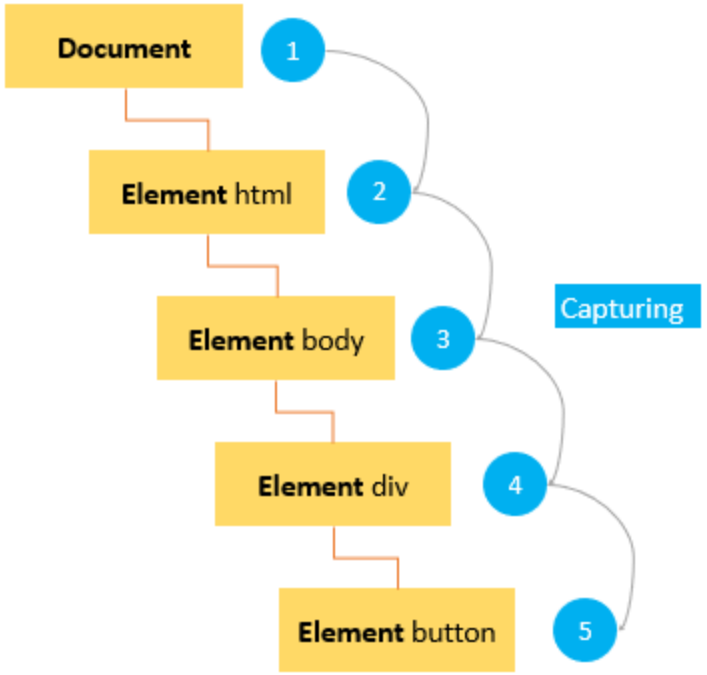
```
let btn = document.querySelector('#btn');

btn.addEventListener('click',function() {
    alert('It was clicked!');
});
```

JavaScript DOM

Event in JavaScript : Event flow

- Event flow explains the order in which events are received on the page from the element where the event occurs and propagated through the DOM tree.
- There are two main event models: **event bubbling** and **event capturing**.





JavaScript DOM

CSS in JavaScript

- The HTML DOM allows JavaScript to change the style of HTML elements.
- To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property = new style
```

- Using Events
  - The HTML DOM allows you to execute code when an event occurs.

JavaScript DOM

Handling Exception in JavaScript

- The following example attempts to call the add() function that doesn't exist:

```
let result = add(10, 20);
console.log(result);
```

- And the JavaScript engine issues the following error:

```
Uncaught TypeError: add is not a function
```

- Sometimes, you want to handle the error and continue the execution. To do that, you use the try...catch statement.

JavaScript DOM

Handling Exception in JavaScript

- syntax:

```
try {
  // code may cause error
} catch(error){
  // code to handle error
} finally {
// code to execute whether exceptions
occur or not
}
```

- Basically, the error object has at least two properties:
  - name specifies the error name.
  - message explains the error in detail.

JavaScript DOM

Handling Exception in JavaScript

- The throw statement allows you to throw an exception. Here's the syntax of the throw statement:

```
throw expression;
```

JavaScript DOM

Handling Exception in JavaScript

- **throw a user-defined exception**
- First, define the user-defined exception that **extends** the **Error** class:

```
class NumberError extends Error {
  constructor(value) {
    super(`"${value}" is not a valid number`);
    this.name = 'InvalidNumber';
  }
}
```

Form Processing Using JavaScript

Steps

- Get the form elements
- Add an event listener to the form element
- Prevent the default form submission behavior
- Access the form data
- Perform any necessary validation on the form data
- Process the from data as needed

JavaScript BOM

- The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.
- The BOM provides you with objects that expose the web browser's functionality.
  - **Window**
  - **Location**
  - **Cookies**

JavaScript BOM

JavaScript Windows

- The window object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.

Window size

The window object has four properties related to the size of the window:

- The **innerWidth** and **innerHeight** properties return the size of the page viewport inside the browser window (not including the borders and toolbars).

JavaScript BOM

JavaScript Windows

- Window size

  The window object has four properties related to the size of the window:

  - The **innerWidth** and **innerHeight** properties return the size of the page viewport inside the browser window (not including the borders and toolbars).
  - The **outerWidth** and **outerHeight** properties return the size of the browser window itself.

JavaScript BOM

JavaScript Windows

- window.open() - open a new window
- window.close() - close the current window
- window.resizeTo() - resize the current window
- window.moveTo() - move the current window
- window.alert(message)- method to display information that you want users to acknowledge.
- window.confirm() - invoke a dialog with a question and two buttons OK and Cancel
- window.prompt()-shows a dialog that prompt the user to enter a text

JavaScript BOM

JavaScript Location

- The Location object represents the current location (URL) of a document.
  - location.href returns the href (URL) of the current page
  - location.hostname returns the domain name of the web host
  - location.pathname returns the path and filename of the current page
  - location.protocol returns the web protocol used (http: or https:)

JavaScript BOM

JavaScript Cookies

- Cookies let you store user information in web pages.
- Cookies were invented to solve the problem "how to remember information about the user":
  - When a user visits a web page, his/her name can be stored in a cookie.
  - Next time the user visits the page, the cookie "remembers" his/her name.
- JavaScript can create, read, and delete cookies with the **document.cookie** property.

JavaScript BOM

JavaScript Cookies

- Cookies let you store user information in web pages.
- Cookies were invented to solve the problem "how to remember information about the user":
  - When a user visits a web page, his/her name can be stored in a cookie.
  - Next time the user visits the page, the cookie "remembers" his/her name.
- JavaScript can create, read, and delete cookies with the **document.cookie** property.

End of Chapter 4

Aksum University- AIT

2024