IT1100 - Internet and Web Technologies

Lecture 05
JavaScript – Part II

SLIIT
FACULTY OF COMPUTING

Content

- JavaScript Strings
- Arrays in JS
- DOM API
- Event handling

JavaScript Strings

JavaScript Strings

- JavaScript strings are used for storing and manipulating text.
- zero or more characters written inside quotes , using single or double quate.
- var Description= “IWT"
  var Description = ‘Lecture 05’
  var Description = 'He is called “Mahela”’
- You can use quotes inside a string, as long as they don't match the quotes surrounding the String

Length of a String

```
var txt = "lets watch legend playing";

var Length=txt.length;
```

Escape Character

- Since strings must be written within quotes, JavaScript will misunderstand the following string:

```
var s1 = "IWT Lecture 05 “JavaScript” Part 2.”;
var s2 = "IWT Lecture 05 'JavaScript' Part 2”;
var s3 = "IWT Lecture 05 \JavaScript\ Part 2";
```

To avoid that problem, use the backslash escape character.

```
var s1 = "IWT Lecture 05 \"JavaScript\" Part 2";

var s2 = "IWT Lecture 05 \'JavaScript\' Part 2";

var s3 = "IWT Lecture 05 \\JavaScript\\ Part 2";
```

String Search Methods

- The indexOf() method returns the index (the position) of the first occurrence

```
var text = "this lecture is JavaScript lecture";
text.indexOf("lecture")
```

- The lastIndexOf() method returns the index of the last occurrence of a specified text in a string

```
var text = "this lecture is JavaScript lecture";
text.lastIndexOf("lecture")
```

Both methods return -1 if the text is not found

Converting Variables to Numbers

- The Number() method returns a number converted from its argument.

```
Number(“10") // returns 10
Number(“IWT") // returns NaN
```

- The parseInt() parses a string and returns a whole number.

```
parseInt("10");        // returns 10
parseInt("10.33");     // returns 10
```

Arrays in JS

JavaScript Arrays

- An array is a special variable, which can hold more than one value at a time.
  - var *array_name* = [*item1*, *item2*, ...];
  - var cars = [“Toyota", "Volvo"];

How to insert new elements

- Cars[2]=“BMW”;

How to display element ant it’s value

- Document.write(Cars[0]);

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The best way to loop through an array is using a standard for loop:</p>
<p id="demo"></p>
<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

for (i = 0; i < fLen; i++) {
  document.write(fruits[i]+"</br>");
}
</script>
</body>
</html>
```

JavaScript Arrays

The best way to loop through an array is using a standard for loop:

Banana
Orange
Apple
Mango

output

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The best way to loop through an array is using a standard for loop:</p>
<p id="demo"></p>
<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;
text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

JavaScript Arrays

The best way to loop through an array is using a standard for loop:

- Banana
- Orange
- Apple
- Mango

output

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For/In Loop</h2>
<p>The for/in statement loops through the properties of an object.</p>

<script>
var txt = "";
var person = ["John","Doe","James"];
var x;
for (x in person) {
  txt = txt + person[x] + " ";
}
document.write(txt);
</script>
</body>
</html>
```

JavaScript For/In Loop

The for/in statement loops through the properties of an object.

John Doe James

output

Functions in JS

Functions

- A function is a group of reusable code which can be called anywhere in your program.
- This eliminates the need of writing the same code again and again.
- It helps programmers in writing modular codes.
- Functions allow a programmer to divide a big program into a number of small and manageable functions.

Function Definition

- Before we use a function, we need to define it.
- The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

```
<script>
function function_name (parameter-list)
{
    statement(s)
}
</script>
```

Calling a Function

- To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the code.

```
<html>
<head>
<script>
function sayHello()
{
   document.write ("Hello  there!");
}
</script>
</head>
<body>
<script>
    sayHello();
</script>
</body>
</html>
```

output

Hello there!

Function Parameters

- Till now, we have seen functions without parameters.
- But there is a facility to pass different parameters while calling a function.
- These passed parameters can be captured inside the function
- Any manipulation can be done over those parameters.
- A function can take multiple parameters separated by comma.

Function Parameters Example

```
<html>
<head>
<script type="text/javascript">
function sayHello(name, age)
{
document.write (name + " is " + age + " years old.");
}
</script></script>
</head>
<body>
<script>
    sayHello('Zara', 7);
</script>
</body>
</html>
```

output

Zara is 7 years old.

The return Statement

- A JavaScript function can have an optional **return** statement.
- This is required if you want to return a value from a function.
- **This statement should be the last statement in a function**.
- For *example*, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

The return Statement

```
<html>
<head>
<script>
function concatenate(first, last)
{
var full;
full = first + last;
return full;
}
function secondFunction()
{
var result;
result = concatenate('Zara ', 'Ali  Khan');
document.write (result );
}
</script>
</head>
<body>
<script>
secondFunction();
</script>
</body>
</html>
```

output

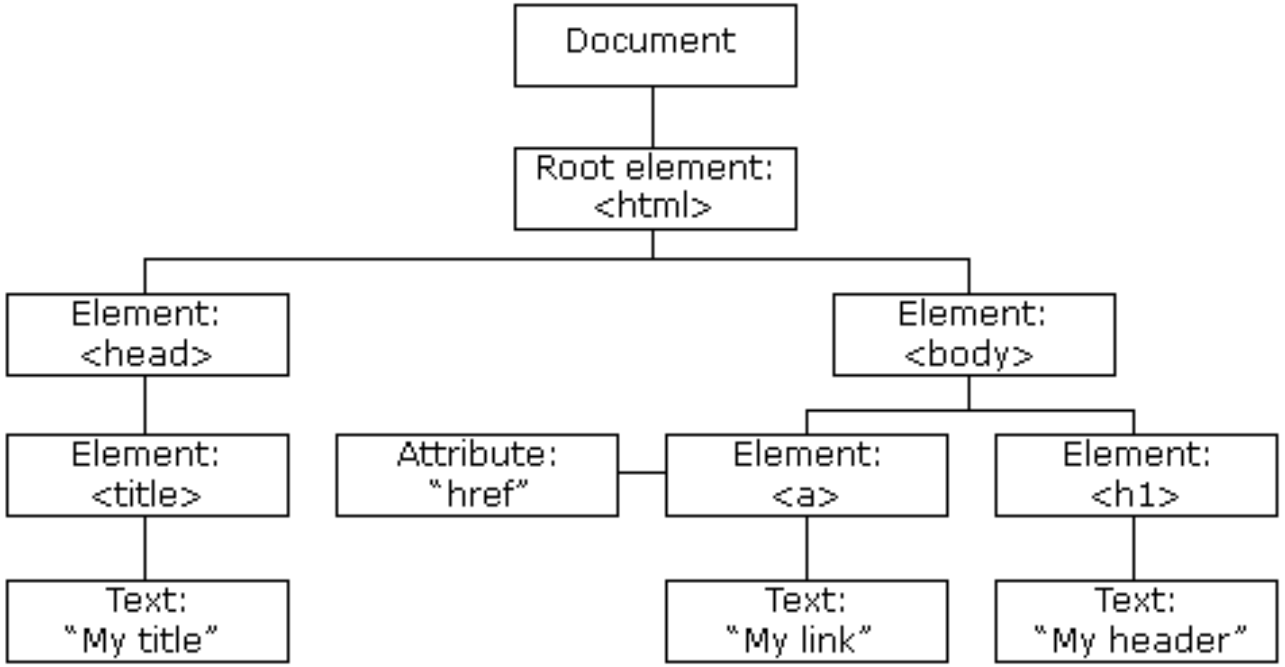
Zara Ali Khan

Document Object Model

Document Object Model

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:



Document Object Model

In the DOM, all HTML elements are defined as objects.

The programming interface is the properties and methods of each object.

```
<p id="para1"></p>
<script>
document.getElementById("para1").innerHTML = "Hello World!";
</script>
```

Method

Property

DOM Methods

Method	Description
close()	Closes the output stream previously opened with document.open()
getElementsByName()	Accesses all elements with a specified name
getElementById()	Accesses the element with the specified id
getElementsByClassName()	Accesses all elements with a specified class name
getElementsByTagName()	Accesses all elements with a specified tag name
open()	Opens an output stream to collect the output from document.write() or document.writeln()
write()	Writes HTML expressions or JavaScript code to a document
writeln()	Same as write(), but adds a newline character after each statement

DOM API

```
<form>
        <input type="text" id="txtName">
        <div id="divOutput"></div>
</form>
-----------------------------------------------------
//Read the value
var name = document.getElementById("txtName ").value;

//Display output
document.getElementById("divOutput").innerHTML = "Hello "+name;
```

Event Handling

Event Handling

- Event handling is used to implement responses for the user events
  - Click, type, select, drag and drop, etc…

Ex:

- Read form values and validate before submitting the form and display proper error messages

Event Handling

- Event handlers are used to handle the events, when the events are triggered

- There 2 main ways of developing event handlers in JS

  1. DOM level 0 inline event handlers

  2. Event registration using the **addEventListener()** function

DOM level 0 inline event handlers

- DOM allows to assign events to HTML elements using JavaScript:

- HTML event attributes are used.
  - **onclick, onload, etc…**

<button onclick="**alert('Hello');**">Try it</button>

DOM level 0 inline event handlers

- If there is more code to write, it is good to implement a function and call that function in the event handler

```
<button onclick="myFunction();">Try it</button>
<script>
function myFunction() {
        alert("Do whatever needed in this function");
}
</script>
```

DOM level 0 inline event handlers

```
<html>
<head>
<script>
function myFunction() {
   document.getElementById("demo").innerHTML = document.getElementById("inTxt").value;
}
</script>
</head>
<body>

<p id="demo"></p>

<input type="text" id="inTxt">

<button onclick="myFunction()">Click Me</button>
</body>
</html>
```

Event registration using addEventListener()

- It is good the separate the JS from HTML as much as possible, towards increasing the modifiability.

- By using the **addEventListener()** function, we can eliminate the HTML event attributes

Event registration using addEventListener()

```
<button id="btnTest">Try it</button>
<script>
var btn = document.getElementById("btnTest");
btn.addEventListener("click", function() {
    alert("Do whatever needed in this function");
}
);
</script>
```

Event registration using addEventListener()

```
<html>
<head>
</head>
<body>
<input type="text" id="inTxt">

<button id="myBtn">Click Me</button>

<p id="demo"></p>
<script>
document.getElementById("myBtn").addEventListener("click", function(){
    document.getElementById("demo").innerHTML = document.getElementById("inTxt").value;
});
</script>
</body>
</html>
```

Summary

- JavaScript Arrays
- String and Numerical methods
- DOM API
- Event handling