CUI Abbottabad

Department of Computer Science

Web Technologies

Lecture -11-[CLO-2]
Javascript-Part3

COMSATS University Islamabad, Abbottabad Campus

Agenda

- JavaScript Functions
- Defining User-defined Function in JavaScript
- Invoking/calling User-defined Function
- Tips about function calling
- Function Overloading
- Function without parameters
- Function with parameter
- Use of input fields to capture values and send to function
- Returning value from function
- Use of Math Built-in Functions
- Random Numbers Generation

Functions in Javascript

- A function also known as a *method*.
- A function is a group of reusable code which can be called anywhere in your program.
- This eliminates the need of writing same code again and again.
- You can divide your big program in a number of small and manageable functions.
- You have seen functions like *alert()* and *confirm()*. We are using these function again and again, but their coding(definition) have been written in core JavaScript only once. These functions are called **pre-defined/built-in methods/built-in-functions**.
- Note that we only call (invoke) the **built-in methods in our** program whenever we need them.
- built-in methods are already defined and stored in JavaScript core library.

Functions in Javascript

User-defined Function:

- JavaScript allows us to write our own functions as well. These functions are called **user-defined functions.**
- User have to defined the function either inside <script></script> tag in html page or create separate file of .js type and define function there.
- Once a user defined its own function/s then these can be called(invoke) anywhere in the html document
- Note that a JavaScript function is executed when "something" invoks it (calls it).

Defining a User-defined Function

General Syntax :

```
<script type="text/javascript">
function name (parameter1, parameter2,…) //local variable
{
  //code(Statements) to be executed
}
</script>
```

Defining a User-defined Function

- The parentheses may include parameter names separated by commas: (***parameter1***, ***parameter2***, ...)
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function **parameters** when it is invoked.
- Inside the function, the arguments (the parameters) behave as **local variables**.

Defining a User-defined Function

Example-1----Function without Parameters list

- A simple function that takes no parameters called sayhello is defined here:

```
<script type="text/javascript">
function sayHello()
{
alert("Hello there");
}
</script>
```

Empty parenthesis()

- The above function will always show the same message"Hello there" whenever it is invoked.

Calling a Function

- The code inside the function will execute when "something" **invokes** (calls) the function.
- Function can be called by any of the following:
  - When an event occurs (for example-when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

- To invoke/call a function somewhere later in the script, you would simple need to write the name of that function followed by (), for example here is call to the function we just defined earlier:

```
<script type="text/javascript
sayHello();
 </script>
```

Important about Function calling

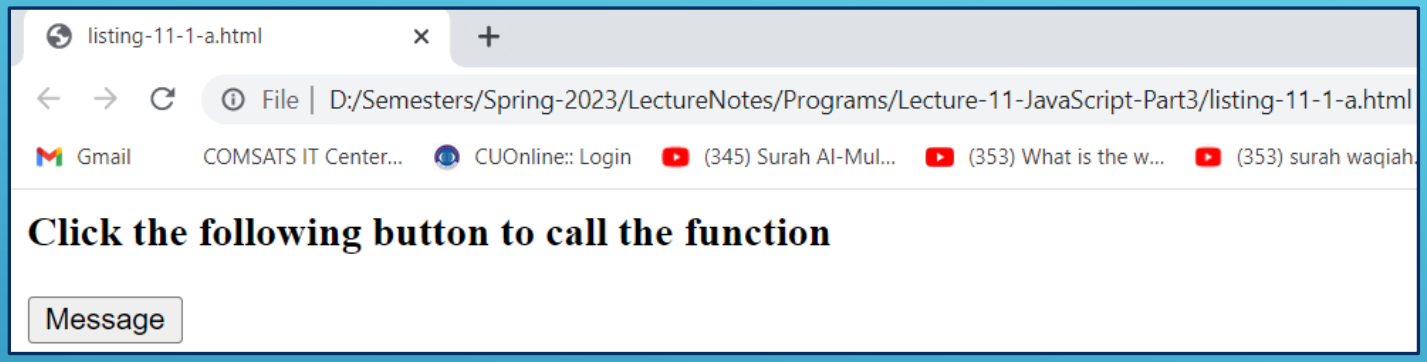
- Note that function call is dependent on function definition:
  - If we defined a function with empty parenthesis () then function call will only need functionName()
  - If we defined a function with one or more parameters in parenthesis () then we have to pass same number of argument/values during function call inside the ().
  - if we defined a function which is not returning any value to calling point then function call should not be inside assignment or in output statement
  - Similarly, if we defined a function which is returning any value to calling point then function call should be inside assignment or in output statement

Function Example-1

```html
<html>
<head>
    <script type="text/javascript">
        function sayHello()
        {
            alert("Welcome in JavaScript Functions!");
        }
    </script>
</head>

<body>
    <h3>Click the following button to call the function</h3>

    <input type="button" onclick="sayHello()" value="Message">

</body>
</html>
```

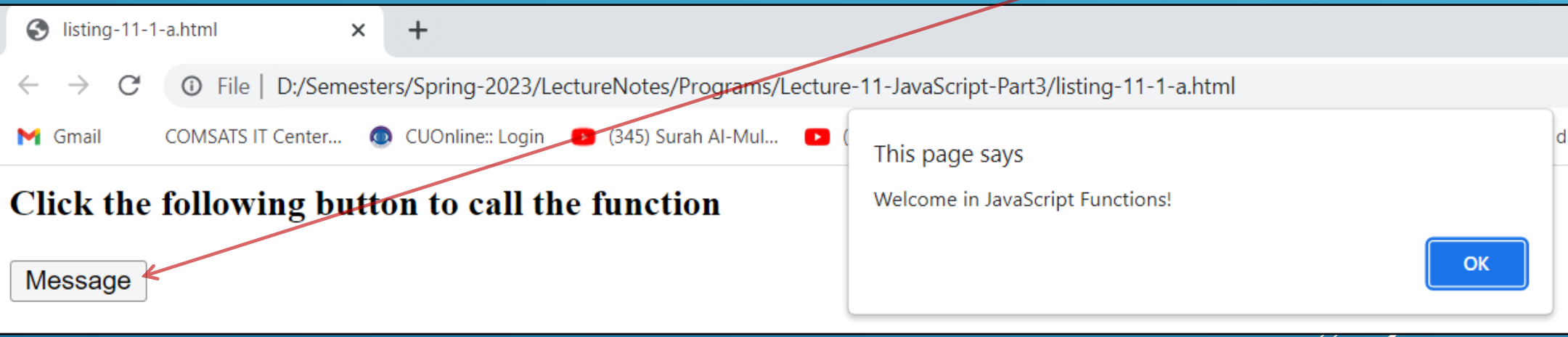
Listing-11-1-a.html

Function Example-1-a



Output-19.1(a)

Click on button will call sayhello() function , and function displays alert() dialog with greetings



Output-11-1-a(b)

Javascript does not support Function Overloading Natively

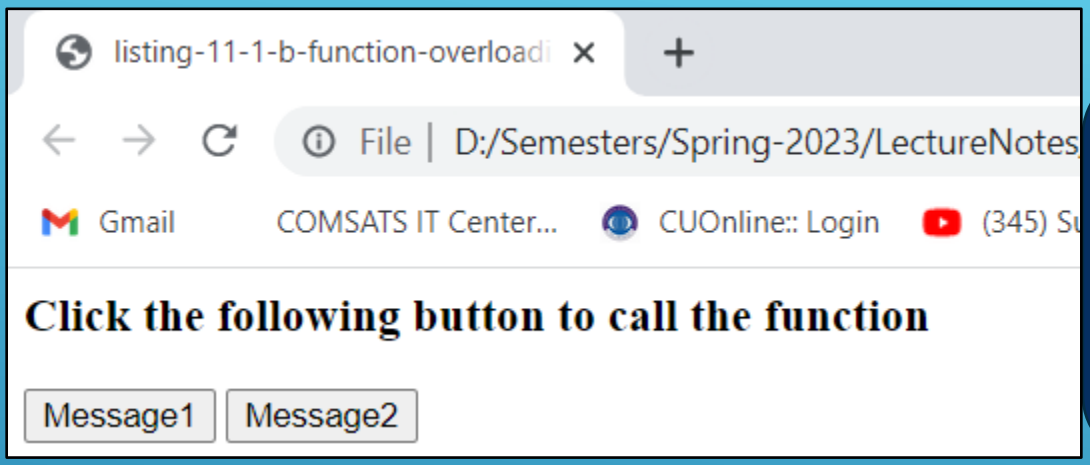
- Note that javascript does not support function overloading:
  - As you know that function overloading is “more than one functions with same name but with different arguments”
  - If we add functions with the same name and different arguments, it considers the last defined function..
  - Consider the next example-Listing-11-1-b-function-Overloading.html, in this example the second definition of sayHello() will be considered by javascript. That is why when the user click the first button with value “message1” it will return ---”**undefined for you**” because the this button does not passing any value to function hence will show “undefined” in place of myval
  - For the second button it will show “**best wishes** for you” because “best wishes” was passed to function as argument for myval.

Javascript does not support Function Overloading Natively

```html
<html>
<head>
    <script type="text/javascript">
        function sayHello()
        {
            alert("Welcome in JavaScript Functions!");
        }
        function sayHello(myval)
        {
            alert(myval+" for you");
        }
    </script>
</head>

<body>
    <h3>Click the following button to call the function</h3>

    <input type="button" onclick="sayHello()" value="Message1">
    <input type="button" onclick="sayHello('\nBest Wishes')" value="Message2">

</body>
</html>
```

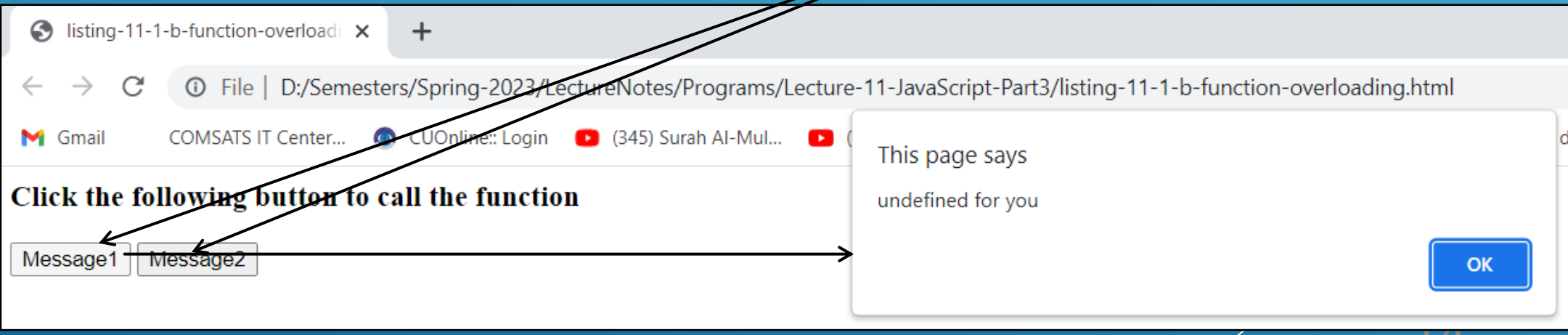
Listing-11-1-b-function-Overloaing.html

Javascript does not support Function Overloading Natively



Click on any button will call next sayhello() function with argument, and function displays alert() dialog with greetings, here Message1 is clicked even then it called second sayhello() function with arguments

Output-11-1-b-function-Overloading(a)



Output-11-1-b-function-Overloading(b)

Function Parameters…Example-2

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

Example-2---------Function with parameters:

- Let us do a bit modification in our *sayHello()* function. This time it will take two parameters:

Function Parameters ...Example-2.

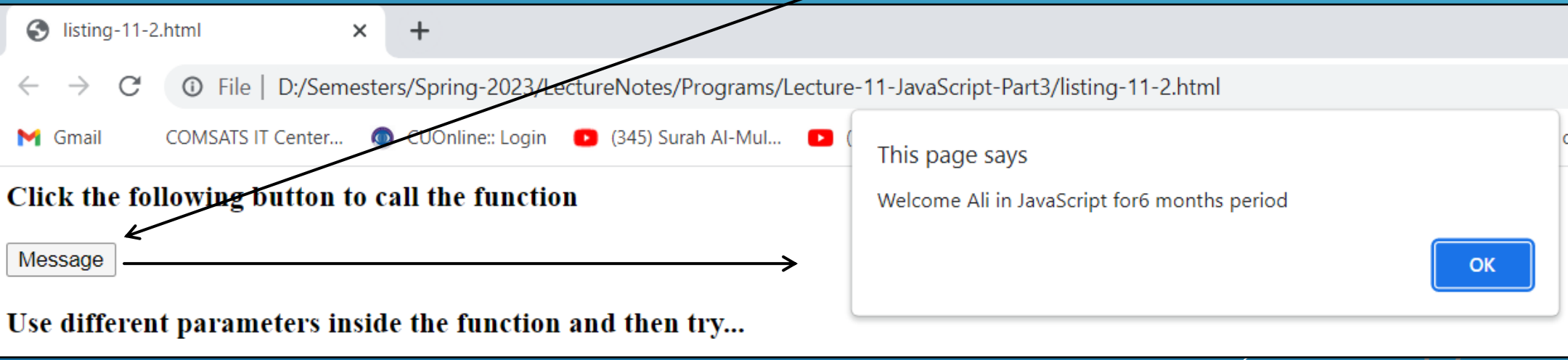
```html
<html>
<head>
    <script type="text/javascript">
        function sayHello(name,duration)
        {
            alert("Welcome "+name+ " in JavaScript for"+ duration+" months period");
        }
    </script>
</head>

<body>
    <h3>Click the following button to call the function</h3>
    <form>
        <input type="button" onclick="sayHello('Ali',6)" value="Message">
    </form>
    <h3>Use different parameters inside the function and then try...</h3>

</body>
</html>
```

Listing-11-2

Function Parameters...Example-2

Output-11-2.html(a)

Click on button will call sayhello() function , and function displays name and duration send by function call



Output-11-2(b)

Function Parameters…Example-3

Example-3--------Capturing form textbox data and send to function as parameter

In next program (Listing 11-3), User input its name in the text field, when we click the button with caption "Display Message" then following operations will be performed in sequence:

1. Capture value from text box
2. send the entered name from text box to function displaymessage() as parameters
3. Function will display the name through alert() dialog box

Function Parameters…Example-3

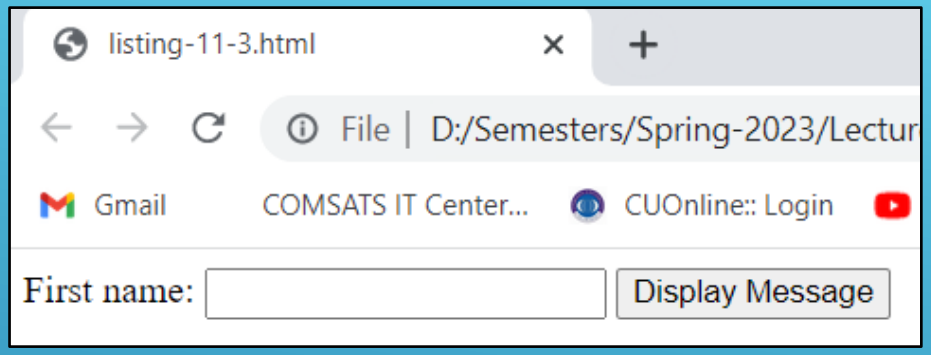
- On button click the value from text box is captured by form through text box name and send to displaymessagefunction() to display the name in message.
- Note that in following onclick event:

onclick="displaymessage(form.yourname.value)“

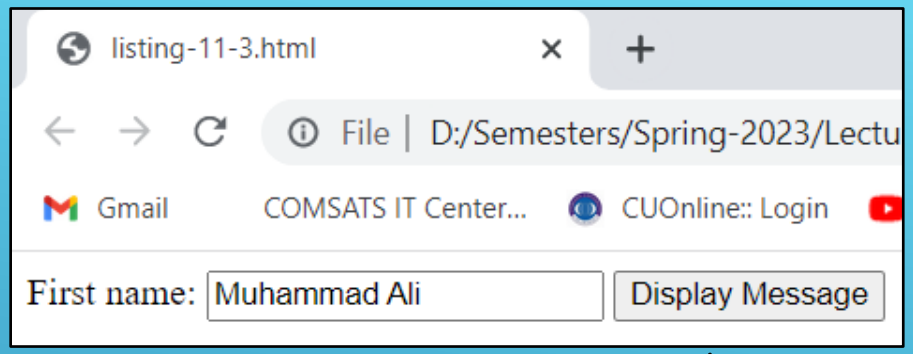
```html
<html>
<head>
    <script type="text/javascript">
        function displaymessage(firstname)
        {
         alert("Welcome " + firstname + ", hope you like functions");
        }
    </script>
</head>

<body>
    <form>
        First name: <input type="text" name="yourname" />
        <input  type="button" onclick="displaymessage(form.yourname.value)"
               value="Display Message"      />
    </form>

</body>
</html>
```

Listing-11-3.html

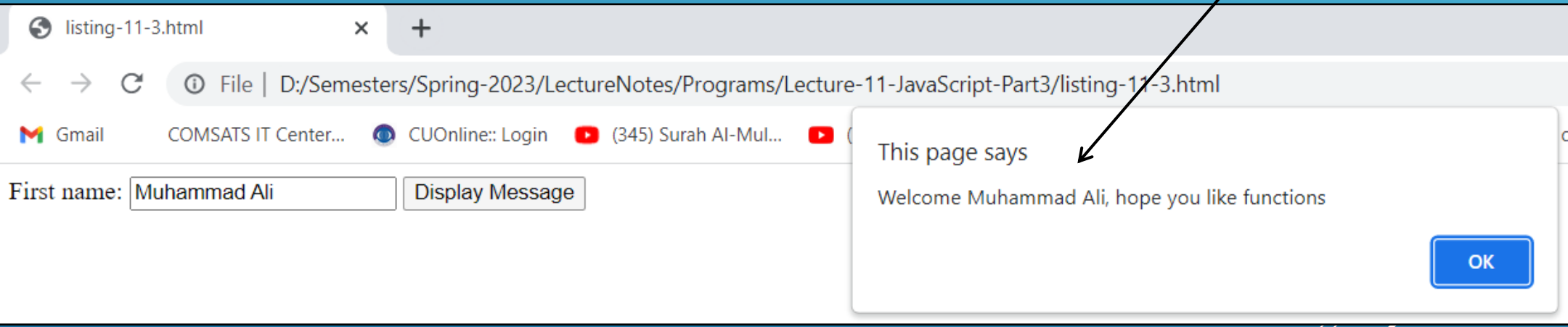
Function Parameters...Example-3



Output-11-3(a)



Output-11-3(b)



Output-11-3(c)

Return value from function…Example-4

- 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 from the function to return their multiplication result at the point where the function was invoked(called) in your calling program.

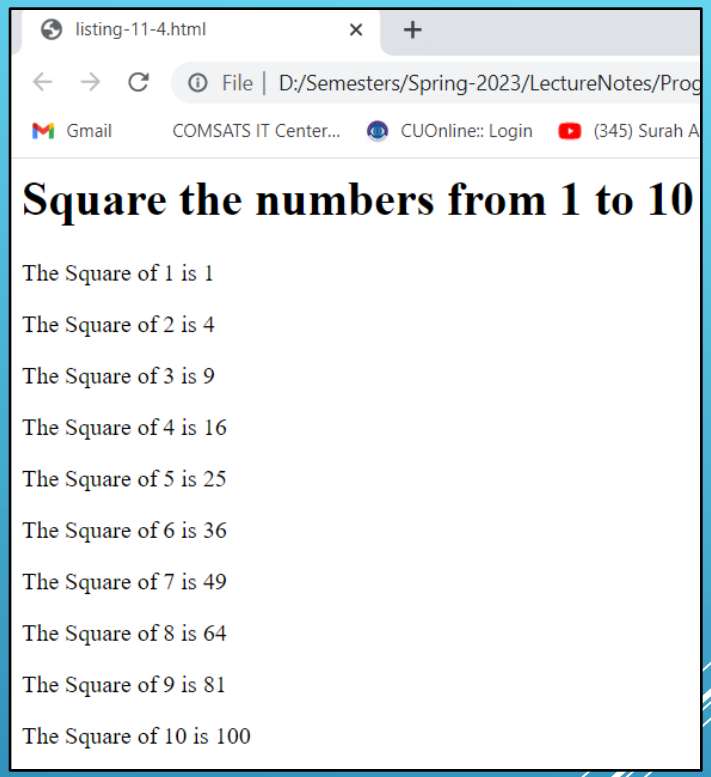
Example-4--------Returning a value from fruntion definition

- In the following program listing 11-4.html, we are calling function square() from inside the for-loop 10 times(line#6) and every time passing a number to sqaure(), in response the line# 10 function is returning the square of that number back on calling point at line#6

Return value from function…Example-4

```html
<html>
<head>
    <script type="text/javascript">
        document.write("<h1>Square the numbers from 1 to 10</h1>");
        for(var x=1; x<=10;++x)
            document.writeln("<p>The Square of"+x+"is"+square(x)+"</p>");

        function square(y)
        {
            return y*y;
        }
    </script>
</head>

<body>

</body>
</html>
```

Listing-11-4



Result of function square() is returning back to calling point of function

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST)

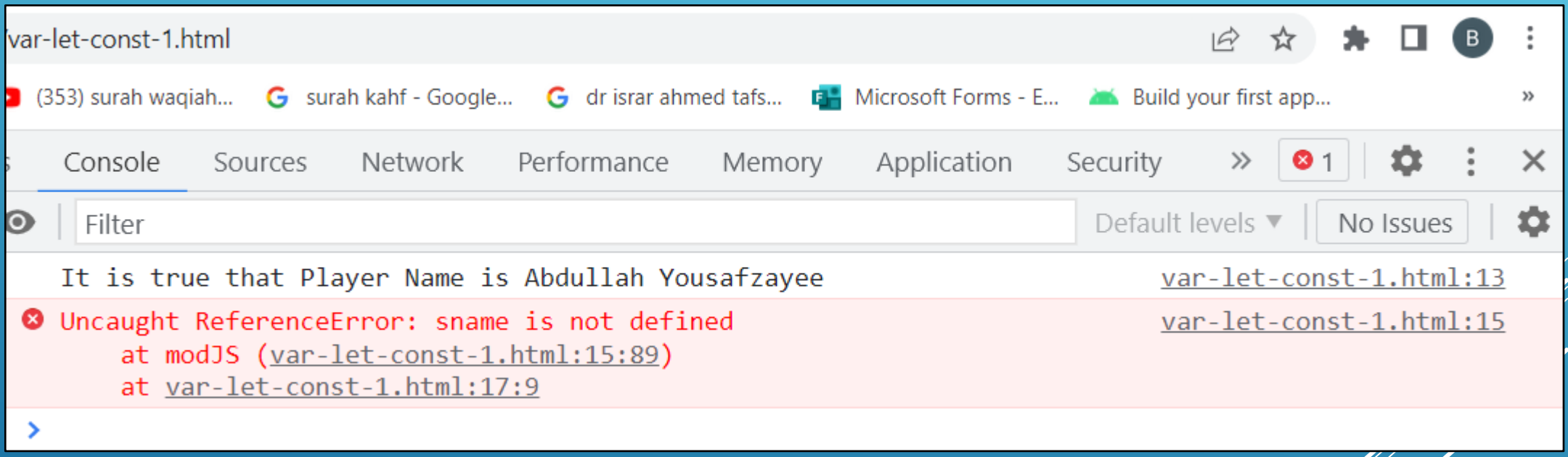
- Code inside the curly braces{} from Line#8-line#16 is called function scope
- Code inside the curly braces{} from line#10-line#14is called block scope

```
1   <!DOCTYPE html>
2   <!-- var-let-const-1.html -->
3   <html>
4   <head></head>
5       <body>
6       <script>
7           function modJS(ok)
8           {
9               if(ok)
10              {   let name="Abdullah";
11                  const sname="Yousafzayee";
12
13                  console.log("It is "+ok+" that Player Name is "+name+" "+sname);
14              }
15              console.log(" Again saying that it is "+ok+" that Player Name is "+name+" "+sname);
16          }
17          modJS(true);
18      </script>
19      </body>
20  </html>
```

Var-let-const-1.html

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST)

Note: “let” and “const” have block scope, that is why they are accessible only inside the if-block (where they are declared). The “let” and “const” do not accessible outside the if-block hence we found error at line#15



Output:var-let-const-1.html

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST)

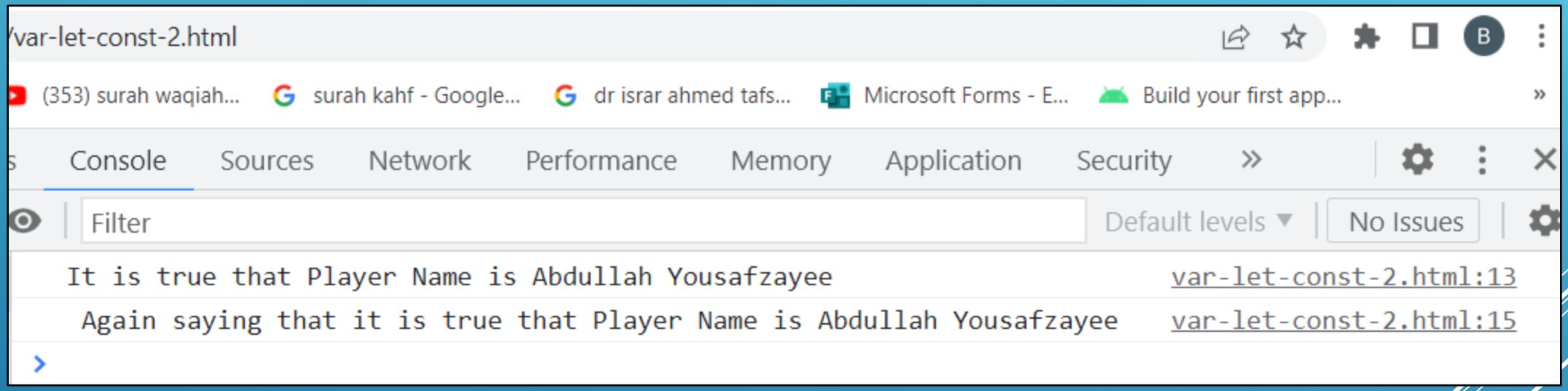
- Code inside the curly braces{} from Line#8-line#16 is called function scope
- Code inside the curly braces{} from line#10-line#14is called block scope

```
1   <!DOCTYPE html>
2   <!-- var-let-const-2.html -->
3   <html>
4   <head></head>
5       <body>
6       <script>
7           function modJS(ok)
8           {
9               if(ok)
10              {   var name="Abdullah";
11                  var sname="Yousafzayee";
12
13                  console.log("It is "+ok+" that Player Name is "+name+" "+sname);
14              }
15              console.log(" Again saying that it is "+ok+" that Player Name is "+name+" "+sname);
16          }
17          modJS(true);
18      </script>
19      </body>
20  </html>
```

Var-let-const-2.html

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST)

- Note that “var” has function scope that is why it is accessible inside (line#13)and outside(line#15) the if block in the function scope.



Output:var-let-const-2.html

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST).....

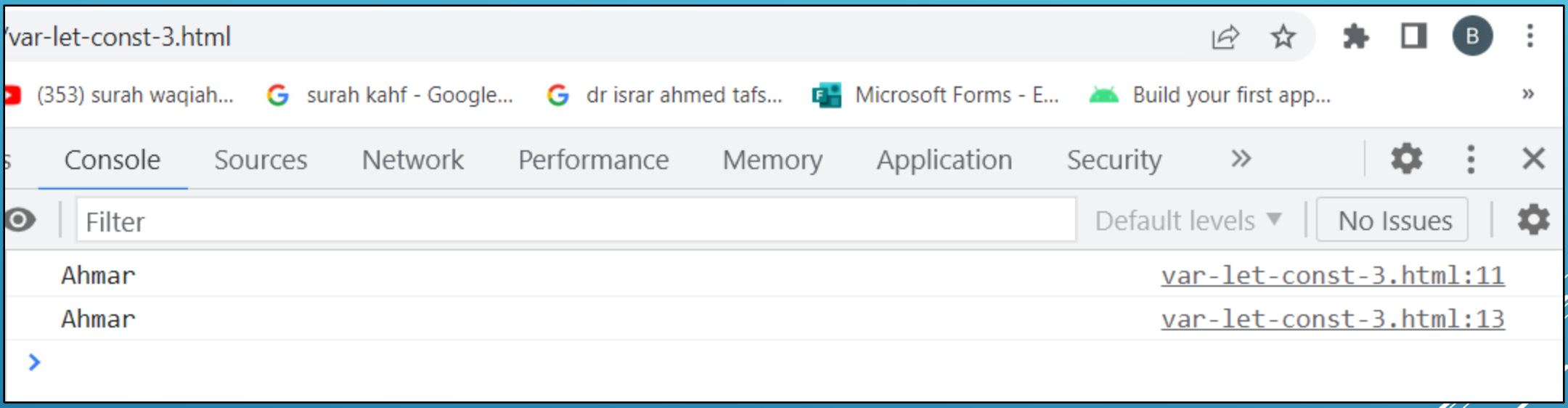
- We declare the variable with same name “whoWillWinToday” using “var” outside(line#7) and inside(line#10) of if-block

```
1   <!DOCTYPE html>
2   <!-- var-let-const-3.html -->
3   <html>
4   <head></head>
5       <body>
6       <script>
7           var whoWillWinToday="Saad";
8           if(true)
9               {
10                  var whoWillWinToday="Ahmar";
11                  console.log(whoWillWinToday+"\n");
12              }
13          console.log(whoWillWinToday);
14          </script>
15      </body>
16  </html>
```

Var-let-const-3.html

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST).....

- Note that “whoWillWinToday” displayed most updated value inside and outside the block scope.



Output:var-let-const-3.html

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST).....

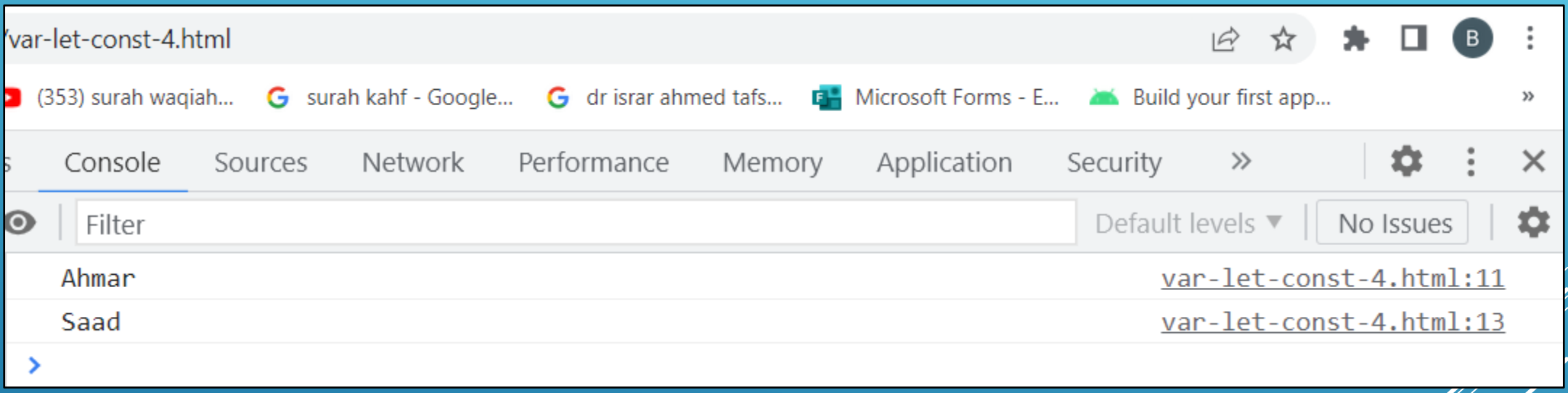
- We declare the variable with same name “whoWillWinToday” using “let” outside(line#7) and inside(line#10) of if-block

```
1   <!DOCTYPE html>
2   <!-- var-let-const-4.html -->
3   <html>
4   <head></head>
5       <body>
6       <script>
7           let whoWillWinToday="Saad";
8           if(true)
9               {
10                  let whoWillWinToday="Ahmar";
11                  console.log(whoWillWinToday+"\n");
12              }
13          console.log(whoWillWinToday);
14          </script>
15      </body>
16  </html>
```

Var-let-const-4.html

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST).....

- Note that “whoWillWinToday” displayed at line#11 the value which defined inside the if-block and displayed at line#13 the value defined outside the if-block.



Output:var-let-const-4.html

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST).....

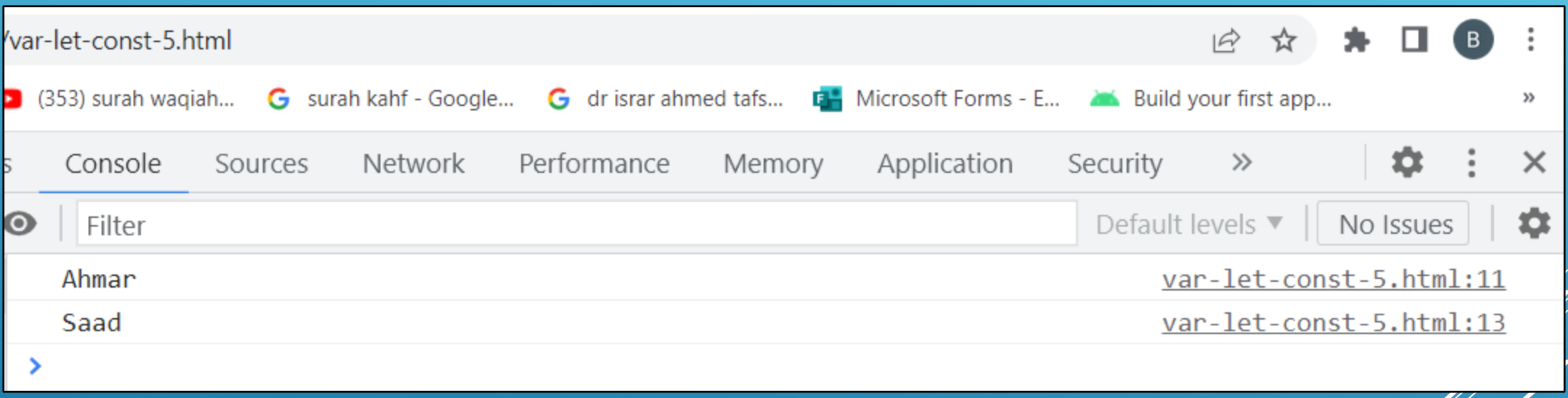
- We declare the variable with same name “whoWillWinToday” using “let” outside(line#7) and inside(line#10) of if-block

```
1   <!DOCTYPE html>
2   <!-- var-let-const-5.html -->
3   <html>
4   <head></head>
5       <body>
6       <script>
7           const whoWillWinToday="Saad";
8           if(true)
9               {
10                  const whoWillWinToday="Ahmar";
11                  console.log(whoWillWinToday+"\n");
12              }
13          console.log(whoWillWinToday);
14          </script>
15      </body>
16  </html>
```

Var-let-const-5.html

FUNCTION SCOPE VS BLOCK SCOPE (VAR, LET, CONST).....

- Note that “whoWillWinToday” displayed at line#11 the value which defined inside the if-block and displayed at line#13 the value defined outside the if-block.



Output:var-let-const-5.html

FUNCTION RETURNING A MAXIMUM VALUE USING MATH.MAX() BUILT-IN FUNCTION

Example-5--------Use of math library built-in method to return value from frunction definition

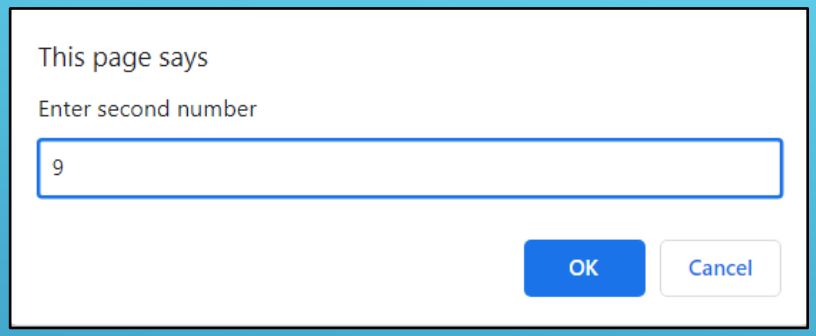
- In the following program listing 11-5.html:
  - first we are asking user to input three integer values(line#4 to line#6),
  - convert the input to integer format (line#7 to line#9)
  - Call(invoke) the user-defined function maximum() at line#11 and pass the these three integers to maximum()
  - Inside the maximum() function we further call at line#15 Math.max() function to find maximum value among these numbers.
  - Finally the resultant maximum among three integer numbers returns back to line#11(calling point) and we displayed them as output.

FUNCTION RETURNING A MAXIMUM VALUE USING MATH.MAX() BUILT-IN FUNCTION

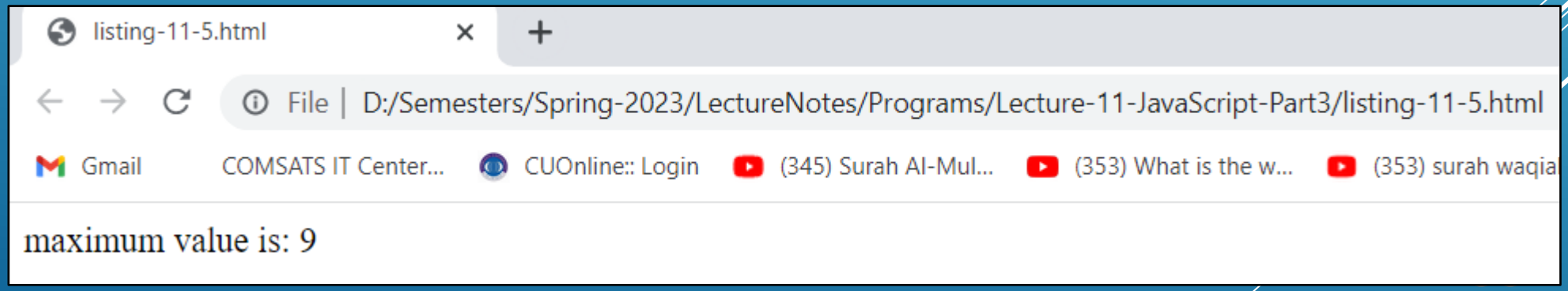
```
1   <html>
2   <head>
3       <script type="text/javascript">
4           var input1=window.prompt("Enter first number","0");
5           var input2=window.prompt("Enter second number","0");
6           var input3=window.prompt("Enter third number","0");
7           var value1=parseFloat(input1);
8           var value2=parseFloat(input2);
9           var value3=parseFloat(input3);
10
11          var maxValue = maximum(value1,value2,value3);
12          document.writeln("maximum value is: "+maxValue);
13
14          function maximum(x,y,z)
15          {   return Math.max(x,Math.max(y,z));   }
16
17      </script>
18  </head>
19
20  <body>
21
22  </body>
23  </html>
```

Listing-11-5.html

FUNCTION RETURNING A MAXIMUM VALUE USING MATH.MAX() BUILT-IN FUNCTION



Output
Listing-11-5.html



RANDOM NUMBER GENERATION

- In many applications we need random numbers generation again and again, for example roll a dice, lucky draws, random images etc.
- In JavaScript we use Math.random() built-in method for random number generation.

Consider the following statement:

```
var randomValue = Math.random();
```

Method random generates a floating-point value from 0.0 up to, but *not* including, 1.0.(0.0----<1)

In our applications we can change the range of random numbers by adding and multiplying suitable numbers with Math.random() method

RANDOM NUMBER GENERATION

Scaling and Shifting Random Numbers

- In order to generate random numbers according to application demand, we scale or shift the range of numbers.
- Let us take the example of roll a six-sided die, every time we roll the die then we want to show a number from 1 to 6 only but the Math.random() method generates from 0.0 to less than 1.0.
- Hence we need to scale and shift the range of numbers as follows:

```
Math.floor( 1 + Math.random() * 6 )
```

- The preceding expression multiplies the result of a call to Math.random() by 6 to produce a value from 0.0 up to less than 6. This is called *scaling* the range of the random numbers.
- Next, we add 1 to the result to *shift* the range of numbers to produce a number in the range 1.0 up to 6, but not including, 7.0.

RANDOM NUMBER GENERATION

Scaling and Shifting Random Numbers

- Finally, we use method **Math.floor()** to determine the closest integer *not greater than* the argument's value—for example, Math.floor(1.75) is 1 and Math.floor(6.75) is 6.

Example-6----------Random number Generation

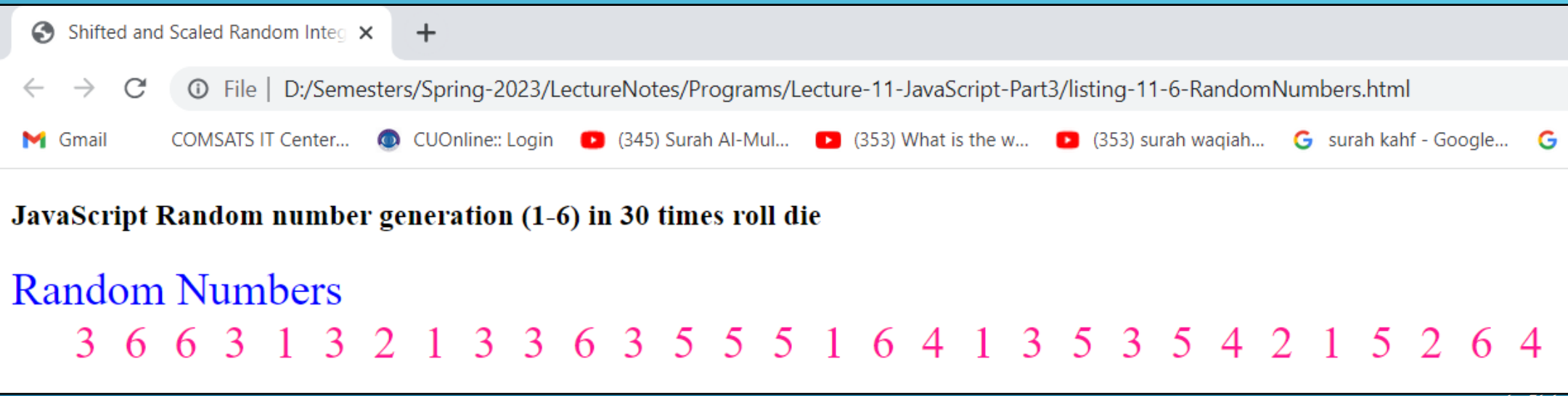
- Listing 11-6.html line#22, confirms that the results are in the range 1 to 6.
- To add space between the values being displayed, we output each value as an li element in an ordered list.
- The CSS style in line 11 places a margin of 10 pixels to the right of each li and indicates that they should display **inline** rather than vertically(on separate lines) on the page.

RANDOM NUMBER GENERATION

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset = "utf-8">
    <title>Shifted and Scaled Random Integers</title>
    <style type = "text/css">
         p, ol      { margin: 0;
                  font-size:4ex;
                }
         li         { display: inline;
                  margin-right: 10px;
                  color:deeppink; }
    </style>
 </head>
<body>
    <h3>JavaScript Random number generation (1-6) in 30 times roll die</h3>
    <script>
     var value;
     document.writeln( "<p style='color:blue;'>Random Numbers</p><ol>" );

     for ( var i = 1; i <= 30; ++i )
     {  value = Math.floor( 1 + Math.random() * 6 );
        document.writeln( "<li >" + value + "</li>" );
     }
document.writeln( "</ol>" );
</script>
</body>
</html>
```

Listing-11-6.html

RANDOM NUMBER GENERATION



Output-11-6.html

DISPLAY RESULT OF FUNCTION IN PLACE OF HTML ELEMENT WITH SPECIFIC ID

getElementById():

- getElementById() built-in method is used to capture the html element with given id.
- In the following example listing- 11-7.html in line#16 we are going to use getElementById() built-in method to capture the html element which has “demo” as id, line#9
- We will display the result of function in place of element having that id at line#9.
- In listing 11-7.html we are going to convert temperature from Fahrenheit into equivalent temperature in celsius .

DISPLAY RESULT OF FUNCTION IN PLACE OF HTML ELEMENT WITH SPECIFIC ID

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function to convert from Fahrenheit to Celsius:</p>
<p> The result of 77 Fahrenheit in Celsius is: </p>
<p id="demo"></p>

<script>
function toCelsius(f) {
                        return (5/9) * (f-32);
                      }

document.getElementById("demo").innerHTML = toCelsius(77);
</script>

</body>
</html>
```

Listing-11-7.html

DISPLAY RESULT OF FUNCTION IN PLACE OF HTML ELEMENT WITH SPECIFIC ID

listing-11-7-TestGetElementById.

File | D:/Semesters/Spring-2023/LectureNotes/Programs/Lecture-11-JavaScript-Part3/listing-11-7-TestGetElementById.html

Gmail COMSATS IT Center... CUOnline:: Login (345) Surah Al-Mul... (353) What is the w... (353) surah waqiah... surah kahf -

JavaScript Functions

25

This example calls a function to convert from Fahrenheit to Celsius:

The result of 77 Fahrenheit in Celsius is:

Output-11-7.html