|  |
| --- |
| WEB-007 JavaScript |
| Course companion |

**Contents**

Module 3. Data Types. 3

Module 4. Variables and scopes. 6

Module 10. Classes and Prototypes. 20

Module 12. Document Object Model (DOM). 22

Objectives 22

Module 13. CSS. 24

Module 15. Forms. 26

Module 16. Browser window management. 29

Module 18. Events. 30

# Module 3. Data Types.

## Objectives

Get familiar with JavaScript data types and learn data types conversion principles, understand ways to pass data by reference.

## Tasks

### Task 1. Converting numbers to strings.

Learn following conversion techniques:

1. String and number concatenation:

|  |
| --- |
| var n = 100;  var s = n + " bottles of beer on the wall."; |

1. Concatenating number n with an empty string;
2. Calling String(number);
3. Calling toString();
4. Calling toString() specifying the number base:

|  |
| --- |
| var n = 17;  binary\_string = n.toString(2); |

1. Given n= 123456.789, use toFixed(), toExponential(), toPrecision() to get the following results:
   1. "123457";
   2. "123456.79";
   3. "1.2e+5";
   4. "1.235e+5";
   5. "1.235e+5";
   6. "123456.8".

### Task 2. Converting string to numbers.

1. In var product = "21" \* "2"; what is going to be the type of product variable, and what its value is going to be?
2. Try this conversion technique  
   var number = string\_value - 0;
3. Try this conversion technique Number(stringValue);
4. What the following code fragments will evaluate to?
5. parseInt("3 blind mice");
6. parseFloat("3.14 meters");
7. parseInt("12.34");
8. parseInt("0xFF");
9. parseInt("11", 2);
10. parseInt("ff", 16);
11. parseInt("zz", 36);
12. parseInt("077", 8);
13. parseInt("077", 10);
14. parseInt("eleven");
15. parseFloat("$72.47");

### Task 3. Boolean conversions.

Evaluate x in Boolean context.

1. Assuming that x‘s value is:

* 1. Undefined;
  2. null;

1. x = “aaa”;
2. x = 7;
3. x = 0;
4. x = Infinity;
5. x - это объект;
6. Learn converting with Boolean(x);
7. Learn converting with x!!

### Task 4. Passing values of primitive types.

1. What the values of n and m are going to be?

|  |
| --- |
| var n = 1;  var m = n;  function add\_to\_total(total, x)  {  total = total + x;  }  add\_to\_total(n, m); |

### Task 5. Working with variables of reference types.

In the following code:

|  |
| --- |
| 1. var xmas = new Date(2007, 11, 25); 2. var solstice = xmas; 3. solstice.setDate(21); 4. xmas.getDate( ); 5. xmas == solstice 6. var xmas = new Date(2007, 11, 25); 7. var solstice\_plus\_4 = new Date(2007, 11, 25); 8. xmas != solstice\_plus\_4 |

1. *What line 4 call will evaluate to?*
2. *What line 5 call will evaluate to?*
3. *What line 8 call will evaluate to?*
4. Draw conclusions

# Module 4. Variables and scopes.

## Objectives

Consolidate the understanding of global and local variables’ visibility scopes. Repeat the concept of variable declaration. Pay attention to the needs of memory deallocation.

## Tasks

### Task 1. Local and global visibility scopes.

Specify, what will be output as a result of executing the following code:

|  |
| --- |
| scope = "global";  function checkscope( ) {  scope = "local";  document.write(scope);  myscope = "local";  document.write(myscope);  }  checkscope( );  document.write(scope);  document.write(myscope); |

Review the following code, specify what will be output during execution:

|  |
| --- |
| var scope = "global scope";  function checkscope( ) {  var scope = "local scope";  function nested( ) {  var scope = "nested scope";  document.write(scope);  }  nested( );  }  checkscope( ); |

Explain the results from the point of view of call chain.

**Module 5. Expressions and operators.**

## Objectives

Learn the rules of data type implicit conversion. Master identity and equality operators. Logical operators. Modulo operator.

## Tasks

### Task 1. Using equality(==) and identity(===) operators.

1. Explain the results of comparisons using implicit data type conversion rules:
   1. alert(null == undefined);
   2. alert(null === undefined);
   3. alert(5 == "5");
   4. alert(5 === "5");
   5. alert("true" == true);
   6. alert(1 == true);
   7. alert("1" == true);
   8. alert(1 === true);
   9. alert("0" == false);
   10. alert("0" === false);
   11. var x = { id: 123, name: "Blah" };
   12. var y = { id: 456, name: "Not Blah, definitely" };
   13. alert(x == "[object Object]");
   14. alert(y == "[object Object]");
   15. alert(x == y);
   16. alert(1 != true);
   17. alert(0 != false);
   18. alert(5 != "5");
   19. var x = new Object();
   20. alert(x != "[object Object]");
2. Equality operator here is a better fit because field value is a string:  
   if(document.getElementById("age").value== 20);
3. Compare equality and identity operators in the comparisons:

|  |
| --- |
| var k;  if (k==null) {  alert("k must be defined " + k);  } |

and

|  |
| --- |
| var k = null;  if (k==null) {  alert("k must be defined " + k);  } |

### Task 2. String and arithmetic operators.

1. What is the value of variable х, how comparison is performed, what is the type of x:
   1. x = "The answer is " + 42;
   2. x = 42 + " is the answer";
   3. x = "37" - 7;
   4. x = "37" + 7;
   5. x = 1 + 2;
   6. x = "11" < "3";
   7. x = "11" < 3;
   8. x = "one" < 3;
   9. x = 1 + 2 + " blind mice";
   10. x = "blind mice: " + 1 + 2;

### Task 3. Modulo operator.

1. What is the value of:

1. 5%3
2. 7%3
3. -3.93%2
4. *How to generate 0,1,0,1,0,1… sequence using %?*   
   This mechanics might be used to produce “zebra striped” tables.

**Module 6. Statements.**

## Objectives

Improving switch, do-while, and for-in using skills, working with labels, throwing and catching exceptions.

## Tasks

### Task 1. switch statement.

1. Specify whether the following code is correct or not whether will line 6 execute.

If yes, then explain why. Explain, how to change the code so that “Question” were not shown.

|  |
| --- |
| 1. var x = 10; 2. switch (x) { 3. case 10: 4. x = 15; 5. case 15: 6. alert(“Question”); 7. } |

### Task 2. do-while statement.

1. Given array **а**, write a function that will print message «Empty Array» if the array is empty and print array elements if the array is not empty. Do the exercise using do-while.

### Task 3. for-in.

1. Client-side JavaScript (in browser) has access to document object. Print all the properties of document object.

### Task 4. Labeled break statement.

1. Review the code, specify the expected output:

|  |
| --- |
| outerloop:  for(var i = 0; i < 10; i++) {  innerloop:  for(var j = 0; j < 10; j++) {  if (j > 3) break; // Quit the innermost loop  if (i == 2) break innerloop; // Do the same thing  if (i == 4) break outerloop; // Quit the outer loop  document.write("i = " + i + " j = " + j + "<br>");  }  }  document.write("FINAL i = " + i + " j = " + j + "<br>"); |

### Task 5. Throwing exceptions.

Explain what is going to be output and why.

|  |
| --- |
| <script type="text/javascript">  try {  alert(variable);  }  catch (ex) {  alert(ex);  }  alert("Here");  </script> |

### Task 6. Catching exceptions.

Explain what is going to be output and why.

|  |
| --- |
| <script type="text/javascript">  function f() {  try{  throw 1;  }catch(ex) {  return 2;  }finally{  alert("Here!");  }  }  alert(f());  </script> |

**Module 7. Objects and Arrays.**

## Objectives

Exercise creating objects, review different ways of accessing objects’ properties. Reinforce following skills: creating arrays, using length property, iterating through arrays, working with arrays as with objects.

## Tasks

### Task 1. Creating objects and working with objects’ properties.

1. Create an object without properties;
2. Ensure that when a is an object and it has no property named prop, then a.prop returns undefined;
3. Use an undeclared variable in a random program location: alert(nosuchprop). Make sure that the action results in runtime exception;

### Task 2. Object as an associative array.

1. Create an empty object, assign it to variable named obj.
2. Add property prop to object obj.
3. Make sure that the property is defined for the object using for/in. Print all obj properties.
4. Assure that the property is defined for the object using hasOwnProperty.
5. Delete the property using operator delete;
6. Check if prop is deleted using for/in and hasOwnProperty.

### Task 3. Creating arrays.

### Explain what the following code does:

|  |
| --- |
| a = new Array({"attr1":"text1","attr2":"text2"}, {"attr1":"text3","attr2":"text4"}); |

### Task 4. Array length.

1. Explain what is the array length in each line and how much memory the arrays use:

|  |
| --- |
| var a = []  a[1] = 1  a[999999] = 2 |

1. Make sure that the following methods of adding an element to the end of the array are equivalent:

|  |
| --- |
| a[a.length] = "new element"  a.push("new element") |

### Get the last array element using method pop. Check what happened with the array length;

### Use array.length = 0 to clear an array.

### Task 5. Iterating array elements.

Let’s consider the following sparse array:

|  |
| --- |
| **var arr = []**  **arr[1] = 123**  **arr[9999] = 456** |

Iterate through array (print 123 and 456), using for/in language construct. What the following code will output

|  |
| --- |
| **arr.hasOwnProperty("1")**  **arr.hasOwnProperty(9999)**  **arr.hasOwnProperty(9998)** |

### Task 6. Arrays are objects.

Define if the following code is correct. If yes, explain what will be output:

|  |
| --- |
| <script type="text/javascript">  a = new Array();  a[-2]=-2;  a[-1]=-1;  a[0]=0;  a[1]=1;  a[2]=2;  alert("Length: " + a.length);  for (var i=0; i<a.length; i++) {  alert(a[i]);  }  alert(a[a[-2]+a[1]]);  alert(a[3]);  </script> |

**Module 9. Functions.**

## Objectives

Using function declaration skills, reviewing functions scopes details, working with functions arguments, using clojures.

## Tasks

### Task 1. Function definitions.

1. Compare the examples showcasing two ways of creating a function. Answer if the second example is correct and explain the behavior.

|  |
| --- |
| /\* function sum is defined below \*/  var a = sum(2,2)    function sum(x,y) {  return x+y  } |

|  |
| --- |
| /\* explain the behavior of the following line of code \*/  var a = sum(2,2)    var sum = function(x,y) {  return x+y  } |

### Task 2. Function return value.

1. What is going to be assigned to variable x?

|  |
| --- |
| function f(p) {  if (p>0) return p;  }  x = f(5);  alert(x);  x = f(-5);  alert(x); |

### Task 3. Function visibility scope.

1. The following example showcases that JS has no concept of function overloads. The last function declaration is used.   
   What the following code is going to output?

|  |
| --- |
| function func(param) {alert("First declaration");}function func() {alert("Second declaration");}func(2); |

### Task 4. Function visibility scope.

1. What the two code samples below output? Explain the way the function variable declarations are done:

|  |
| --- |
| function f() {  p = 5;  }  f();  alert(p); |

|  |
| --- |
| function f() {  var p = 5;  }  f();  alert(p); |

### Task 5. Functions arguments.

Answer questions for the given function:

|  |
| --- |
| var run = function(distance, speed) {  speed = speed || 10;  var time = distance / speed;  return time;  } |

1. Are calls run(), run(10), run(10,5), run(10, 5, 2, 7) syntactically correct?
2. What the results are going to be for any of the calls above (or an exception will be thrown)?
3. Add the alert statement showing the number of the function arguments to the code above.

### Task 6. Functions arguments.

1. Implement function f() which takes arbitrary number of parameters and count arithmetic average of their values (do not implement argument validity checks for brevity);
2. When called like f(1,2,3), your function should return 2;
3. For the array m = [1,2,3], count arithmetic average of its elements using your brand new function f(). Hint: use apply.

### Task 7. Using objects to pass named parameters to functions.

1. Let's consider the function:

|  |
| --- |
| function resize(toWidth, toHeight, saveProportion, animate) {  // default values  saveProportions = saveProportions || true  animate = animate || true  toHeight = toHeight || ...  } |

1. Calls looking like resize(100, null, null, true); are not giving any information about what value is corresponding to what parameter. When there are many parameters, this might become a problem. The better way of implementing such function is, arguably, as follows:

|  |
| --- |
| function resize(setup) {  // default values  var saveProportions = setup.saveProportions || true  var animate = setup.animate || true  var toHeight = setup.toHeight || ...  } |

1. Now the function call looks like:   
   resize({toWidth: 100, animate: true}).

### Task 8. Function closures.

1. What the following code outputs? Explain the result.

|  |
| --- |
| function makeShout() {  var phrase = "Wassup!"  var shout = function() {  alert(phrase)  }  phrase = "Ready!"  return shout  }  shout = makeShout()  // what is the output?  shout() |

### Task 9. Function closures.

1. Using closures, write function that will make it possible to calculate sum of two numbers using code like this:

sum(1)(3) = 4.  
Hint: return function.

### Task 10. Function closures.

1. Review the following function:

|  |
| --- |
| function makeProperty(o, name, predicate) {  var value;  o["get" + name] = function() { return value; };  o["set" + name] = function(v) {  if (predicate && !predicate(v))  throw "set" + name + ": invalid value " + v;  else  value = v;  };  } |

The function can be used to create properties of the object o with “getter” and “setter” methods and built-in value validation for the “setters”.

1. Review following code and explain what it does:

|  |
| --- |
| var o = {};  makeProperty(o, "Name", function(x) { return typeof x == "string"; });  o.setName("Frank");  print(o.getName());  o.setName(0); |

# 

# Module 10. Classes and Prototypes.

## Objectives

Learn principles of creating classes and JavaScript inheritance model.

## Tasks

### Task 1. Implementing class Complex.

We are going to implement class Complex representing complex number. We’ll define methods magnitude (square root of x\*x+y\*y), negative, add, multiply (x1\*x2-y1\*y2+(x1\*y2+x2\*y1)\*i), equals, toString, valueOf (real part of the complex number) in class Complex, using prototype. Methods magnitude, add, multiply should not change the inner state of our object as we are going the treat our complex numbers as immutable. Define constant Complex.ZERO as a property of the class.

Files **Complex.js** and **example01.html** contain the full solution code (can be used for reference).

1. Review example01.html.   
   Note that class Complex contains methods magnitude (square root of x\*x+y\*y), negative, add, multiply (x1\*x2-y1\*y2+(x1\*y2+x2\*y1)\*i), equals, toString, valueOf;
2. Implement Complex constructor taking real and imaginary parts of the complex number as parameters;
3. Define the methods above using prototype.
4. Define Complex.ZERO, as a class’s property returning value 0;
5. Pay attention to the fact that for an expression like ”string” + complexNumber valueOf() is called, as it’s defined for our object. Delete or comment out the valueOf(), assure that toString() is called after the deletion instead.

### Task 2. Implementing class PositionedRectangle.

1. Given class Rectangle:

|  |
| --- |
| function Rectangle(w, h) {  this.width = w;  this.height = h;  }  Rectangle.area = function( ) { return this.width \* this.height; } |

Implement class PositionedRectangle that makes it possible to specify the lelf bottom corner of the rectangle as well as its width and height.

PositionedRectangle must inherit from Rectangle, specifically area() method should be inherited. Class PositionedRectangle must additionally define method contains(), returning true when a given point is located inside the rectangle.

1. When in trouble, please don’t hesitate to refer to **Rect.js**.
2. Implement the constructor;

Class PositionedRectangle should call Rectangle constructor, passing width and height of the rectangle as parameters.   
The constructor should define additional properties for PositionedRectangle. Name the coordinates of the left bottom corner x and y.

Use constructor chaining principle;

1. Set Rectangle as the prototype for PositionedRectangle.
2. Assign correct value to the field PositionedRectangle.prototype.constructor;
3. Define method contains(), returning true, when a given point is located inside the rectangle;
4. Use **Example02.html**, to test your code.

# Module 12. Document Object Model (DOM).

# Objectives

Master DOM usage: querying, creating and modifying elements;

## Tasks

### Task 1. Querying DOM element.

1. Using this HTML fragment:

|  |
| --- |
| <div id="codesection">  <ul>  <li><a href="http://www.sitepoint.com/" id="splink"> SitePoint</a>  </li>  <li><a href="http://www.yahoo.com/" id="yalink"> Yahoo!</a></li>  </ul>  </div> |

1. Create an HTML document, insert the HTML fragment into it, create a form and a button.
2. Declare a function and assign it as the button’s click hander.
3. In the button click handler function you have just created add an alert call showing the value of href attribute of the first a tag (it’s going to be <http://www.sitepoint.com/>), querying the a tag from DOM by id.

### Task 2. Modifying element’s attribute.

1. Expand the previous example by adding a new button with a new click handler function.
2. Inside the new click handler implement code changing the value of href attribute of the first a tag from <http://www.sitepoint.com/> to <http://www.google.com/> .

### Task 3. Changing text node’s value.

1. In the previous example using sitepoint\_link.childNodes[0] output the text of the first link (SitePoint);
2. Find out that sitepoint\_link.childNodes[0] return a text node (figure out how to output the node’s type);
3. Implement code changing the text of the text node (use nodeValue attribute) to “Google”.

### Task 4. Changing element’s position inside DOM.

1. Review the following code fragment:

|  |
| --- |
| <div id="codesection">  <p id="codepara">  <a href="http://www.google.com/" id="splink">Google</a>  </p>  <ul>  <li></li>  <li><a href="http://www.yahoo.com/" id="yalink"  >Yahoo!</a></li>  </ul>  </div> |

1. Query DOM and assign the element with **id** codepara to a variable;
2. Query DOM and assign the element with **id** yalink to a variable;
3. Using para.appendChild mode the link element with adderess yahoo.com from the list to the paragraph.

### Task 5. Creating elements.

1. Expand the previous example adding the fragment

<a href="http://www.linux.org/">The Linux operating system</a>;

1. Implement code moving this link to a position before “yalink” link using para.insertBefore(element, anchorElement).

### Task 6. Creating dynamic content.

1. Review the code inside file **Examples/Module11/example04.html**
2. The JS code extracts all the H1 headers’ texts inside the body of the document, creates contents and creates links to the corresponding sections from the contents.

# Module 13. CSS.

## Objectives

Developing styles management skills, attributes’ specification. Z-index, constructing a menu.

## Tasks

### Task 1. Positioning rules. position:relative.

1. Open **Examples/Module12/Example12.html** in browser, review its source code.
2. Open **Exercises/Module12/Exercise01.html** starter file in editor.
3. Note how headers are positioned.
4. Using inline styles, move Header1 right by 20px and move Header2 by 20px. Use position:relative;
5. Using position:absolute; place one header on the top of the other.
6. Refer to complete code in **Example12.html** when in trouble.

### Task 2. z-index attribute.

1. Open **Example13.html** in browser, review its source code.
2. Open **Exercise02.html** starter file in editor.
3. Using class x and property z-index and position CSS rule, place the bulb image to background. That is, the header should be on top of the image.
4. Refer to complete code in **Example12.html** when in trouble.
5. Then, using z-index, place the image on top of the header.

### Task 3. Changing style dynamically.

1. Open **Example14.html** in browser, review its source code. Trigger onmouseover and onmouseout switching the background color.
2. Open **Exercise03.html** starter file in editor. Note the 3 squares and onmouseover and onmouseout handlers calling bgChange() function that takes color name its parameter.
3. Implement function bgChange() that should change the page’s background to the color that was passed using the parameter.
4. Refer to complete code in **Example14.html** when in trouble.

### Task 4. Changing style dynamically. Changing image size.

### Open **Example15.html** in browser, hover over the bulb image with mouse pointer, review its source code.

1. Open **Exercise04.html** starter file in editor.
2. Implement moveover() and moveback(); functions.
3. moveover()should query the image and set its size to 200x360.

Then, it should query for an element with id=”text1” and set its innerHtml property to "Enlarge image!";

1. moveback() should query the image and set its size to 100х180.
2. Then, it should query for an element with id=”text1” and set its innerHtml property to "Original size";
3. Refer to complete code in **Example15.html** when in trouble.

### Task 5. Building a menu.

### Open Example16.html in browser, play with the menu and review its source code.

### Open **Exercise05.html** starter file in editor.

1. Implement functions showMenu() and hideMenu(), that get corresponding submenu by id and set visibility property with values «visible» and «hidden» when needed.
2. Specify following rules for menu CSS class for table tags:
   1. Absolute positioning to turn off the default positioning (free floating) relatively to the main menu table cell;
   2. Hide the submenu element (visibility attribute).
3. Now menu mechanics works. Use CSS to prettify the appearance of the menu (refer to the complete code in **Example16.html**),
4. Refer to complete code in **Example16.html** when in trouble.
5. Review an example of more sophisticated menu mechanics in **Example17.html** .
6. (Optional) Implement such mechanics by yourself.

# Module 15. Forms.

## Objectives

Creating forms, forms validation, forms mechanics.

## Tasks.

### Task 1. Creating forms, working with form events.

1. Open **Example01.html** in browser, review its source code, note how different form’s elements are created.
2. Open **Exercise01.html** starter file in editor.
3. Create function addhandlers(form) that iterates through form elements and sets function report() as event handler for onclick, onchange, onfocus, onblur, onselect;
4. Create function report() that prints the event happened, event target element name, and element’s value;
5. Refer to complete code in **Example01.html** when in trouble.

### Task 2. Elements mechanics.

1. Open **Example02.html** in browser, review its source code.
2. Open **Exercise02.html** starter file in editor.
3. Set the select element’s handler so that when a browser is selected in combo box, the selected element is shown in the text field.
4. Refer to complete code in **Example02.html** when in trouble.

### Task 3. Moving focus between elements.

1. Open **Example03.html** in browser, review its source code. Pay attention to how HTML attributes size, tabindex, maxlength for text form elements.
2. Open **Exercise03.html** starter file in editor.
3. Implement function checkLen() that:
4. Checks that current field’s value’s length is maximum for the field: y.length==x.maxLength
5. Determines the index of the next text field: var next=x.tabIndex;
6. Checks is there is such field in the document: next<document.getElementById("myForm").length
7. Switches focus to it: element.focus().
8. Refer to complete code in **Example03.html** when in trouble.

### Task 4. Useful scripts.

1. Open **Example04.html.** Review the code and explain what the code does.

### Task 5. Form validation.

Implement text fields validation using regular expression set as a value in HTML attribute.

1. Open **Example06.html** in browser, review its source code.

This is form fields validation sample:

1. First field permits any non-space characters;
2. Second field’s value must be a valid e-mail address;
3. Third field’s value must consist of 5 digits;
4. The last field’s value can be any text.
5. Open **Exercise04.html** starter file in editor.
6. Implement function init() that:
   1. Iterates through all forms and elements inside them collection only text fields;
   2. Retrieves pattern and required HTML attributes and if validation is required, sets the onchange handler as

e.onchange = validateOnChange;

* 1. Set form's submit hander as follows:

f.onsubmit = validateOnSubmit.

1. Function validateOnChange() that:
   1. Checks if the element’s value satisfies the regular expression value.search(pattern) == -1;
   2. Set  
      textfield.className = invalid|valid

according to the check result.

1. Function validateOnSubmit() that:
2. Iterates over all form’s text elements, calling e.onchange( ); for each of them;
3. Checks if e.className == "invalid" is true. If any of the elements is found to fail validation, call alert(“Validation error.”); and cancels the form submit.

# Module 16. Browser window management.

## Objectives

Learning how to use functions setTimeout() and clearTimeout(), opening new windows, navigating to new URL, Screen object properties.

## Tasks

### Task 1. Using setTimeout().

1. Create an HTML file.

2. Add two buttons and a text field.

3. When the first button is clicked, an timer calling timedCount() should start. timedCount() writes the count to the text, increments its value, and then sets itself to be executed after 1 second.   
The timer can be set like this: t=setTimeout("timedCount()",1000);

4. The second button stops the timer-based behavior.

It’s possible to pass variable t as argument to window.clearTimeout();

1. Implement the same behavior using setInterval();
2. Refer to complete code in **Example01.html** when in trouble.

### Task 2. Opening new window.

1. Create an HTML file.

2. Add a button and set function open\_win() as its click handler.

open\_win() opens new window for location <http://www.google.com> setting window decoration as follows: toolbar=yes, menubar=yes, scrollbars=yes, width=400, height=400;

3. Test other possible decoration values;

4. Refer to complete code in **Example02.html** when in trouble.

### Task 3. Свойство window.location.

1. Create an HTML file.

2. Add 2 butttons;

3. The first button should case an alert showing current URL;

4. The second button should cause redirect to http://www.google.com, using window.location;

5. Refer to complete code in **Example03.html** when in trouble.

### Task 4. Screen object.

Review code in **Example04.html** to figure out possible Screen properties.

# Module 18. Events.

## Objectives

## Master event binding models, events bubbling and events capturing.

## Tasks

### Task 1. Legacy model.

1. Create an HTML file.

2. Add a button and add a hander to it. The handler should contain code alert(“1”);

3. In code immediately after setting the first handler, set another handler containing code alert(“2”);

4. Note that the second handler replaces the first handler.

### Task 2. DOM Level 2 event model. Assigning two handlers.

DOM Level 2 defines new way of assigning event handlers.

https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

1. Create an HTML file.

1. Create a button with id=myElement;
2. Using DOM Level 2, implement 2 different event handlers and add them to the button’s click event. The first handler contains code alert('Thank you!');, the second contains alert('Thank you very much!');
3. You can refer to complete solution in **Exercise01.html**, note that the file also contains code supported by legacy IE browsers.

### Task 3. DOM Level 2. Event firing order. Event bubbling propagation, stopping events propagation.

1. Open **Exercise04.html** in browser, showcasing event bubbling, review its source code.
2. Implement code, preventing bubbling on the second element.
3. Modify the code so that all the elements capture the event.

### Task 4. DOM Level 2. Preventing default actions.

### Let’s consider the link:

|  |
| --- |
| <a href="http://www.google.com" id="myElement2"> Press me<a/> |

1. Open **Exercise03.html** in browser, showcasing event bubbling, review its source code.
2. Register a handler firing when the link is clicked, use the legacy model: myElement2.onclick=handler;
3. Access the clicked element and display the link with an alert.
4. Cancel navigating to the new URL on the link’s click using DOM Level 2 model preventDefault();

**Module 19. Cookies. (Module15)**

## Objectives

Learn how to set, read and delete cookies.

## Tasks.

### Task 1. Working with cookies.

1. Review code in **Cookie.js** and figure out how the methods in the file work.
2. Create and HTML with 3 buttons «Check cookie», «Set cookie» and «Delete cookie» and define 3 onclick event handlers for them.
3. When «Check cookie» button is clicked, the handler should check if cookie username is set. If yes, then output the value with alert, if no, then show an alert “User name is not set”.
4. When «Set cookie» is pressed, a prompt (use prompt()) is shown and the value received from user is saved as a cookie with name **username**;
5. When «Delete cookie» is clicked, the cookie **username** is deleted;
6. Use Cookie.js file when doing the exercise;
7. Complete code is in **Example01.html**.