|  |
| --- |
| [[https://myetudes.org/etudes-melete-tool/images/printer.png](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385324) Send to Printer](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385324) | [Close Window](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385324) |
|  | |
| 13. Namespace Considerations  13.1. Global Variables and Functions as Properties  *Copyright (c) 2014, Rula Khayrallah*  In a JavaScript program, when we declare a variable outside of any function, that variable becomes a property of the global object.  Named functions that we declare in our program are also properties of the global object.   **In client side JavaScript that global object is window.**  All we have to do to see the potential problem is open a web page in Firefox, go to the Firefox console and type 'window.' :  Firebug brings up a list of all the enumerable properties defined on window.  Some of these properties are global functions such as alert and setInterval and some are global variables.  In addition, when we use the id attribute in our HTML document, a property corresponding to that id is added to the window object.  So in our calculator example from module 10, when we added the ids "first" and "second" to our input elements,  the window object got two properties window.first and window.second whose value is the corresponding input element.   All of **this makes it harder to name global variables and functions in our program that do not conflict with existing property names.**  We’ll look at two approaches to mitigate the problem next.  13.2. Object as Namespace  *Copyright (c) 2014, Rula Khayrallah*  One way to minimize the risk of name collision is to create a single global variable for our application: var  myApp = {};  **The object referenced by that variable then becomes the container for our application.**  **Our global variables will then be defined as properties of myApp and our functions will be defined as methods of myApp.** The only variable that will be added to the global namespace is myApp.  In client side JavaScript, the only property that will be added to the window object is myApp.  Let’s go back to our add.js program that we used with our calculator.html document.  Even though we did not define any global variables there, we had two functions, update() and help(), that were added to the global namespace.  We can define them  as methods of myApp as follows:  var myApp = {};  myApp.update = function () {     // Get the two input numbers     var firstNumber = Number(document.getElementById('first').value);     var secondNumber = Number(document.getElementById('second').value);     // Then  compute the sum     var myAnswer = firstNumber + secondNumber;     // And write it in the 'answer' element     document.getElementById('answer').textContent = myAnswer;  };  myApp.help = function () {     // check that there is an answer currently displayed     var currentAnswer = Number(document.getElementById('answer').textContent);     if (currentAnswer) {         document.getElementById('answer').textContent =             currentAnswer +             '=' +             document.getElementById('first').value +             '+' +             document.getElementById('second').value;     }  };  document.getElementById('first').addEventListener('input', myApp.update, false);  document.getElementById('second').addEventListener('input', myApp.update, false);  document.getElementById('answer').addEventListener('mouseover', myApp.help, false);    Note that we have to specify myApp.update and myApp.help now in the addEventListener() invocation.  If we have global variables in our program, we can also define them as properties of myApp:  myApp.myGlobalVar =  …    Note that another way to define the methods on myApp is within the curly braces as follows:  var myApp = {     update: function ()  {  // Get the two input numbers  var firstNumber = Number(document.getElementById('first').value);  var secondNumber = Number(document.getElementById('second').value);  // Then  compute the sum  var myAnswer = firstNumber + secondNumber;  // And write it in the 'answer' element  document.getElementById('answer').textContent = myAnswer;  },  help: function () {  // check that there is an answer currently displayed  var currentAnswer = Number(document.getElementById('answer').textContent);  if (currentAnswer) {  document.getElementById('answer').textContent =  currentAnswer +  '=' +  document.getElementById('first').value +  '+' +  document.getElementById('second').value;  }  };  document.getElementById('first').addEventListener('input', myApp.update, false);  document.getElementById('second').addEventListener('input', myApp.update, false);  document.getElementById('answer').addEventListener('mouseover', myApp.help, false);    13.3. Function as Namespace  *Copyright (c) 2014, Rula Khayrallah*  Another approach to minimize the risk of name collision is to **use a function as a container for all our code and then invoke that function.**  The function is then just a temporary namespace for our code.  Let’s go back to our initial calculator example and implement that approach:  **function myModule() {**  function update() {      // Get the two input numbers      var firstNumber = Number(document.getElementById('first').value);      var secondNumber = Number(document.getElementById('second').value);      // Then  compute the sum      var myAnswer = firstNumber + secondNumber;      // And write it in the 'answer' element      document.getElementById('answer').textContent = myAnswer;  };  function help() {      // check that there is an answer currently displayed      var currentAnswer = Number(document.getElementById('answer').textContent);      if (currentAnswer) {          document.getElementById('answer').textContent =              currentAnswer +              '=' +              document.getElementById('first').value +              '+' +              document.getElementById('second').value;      }  };  // Define our event listeners  document.getElementById('first').addEventListener('input', update, false);  document.getElementById('second').addEventListener('input', update, false);  document.getElementById('answer').addEventListener('mouseover', help, false);  }  **myModule();**  **We have to make sure we call that function that we just declared.**  The only addition to the global namespace here is myModule.  If we don't want to even add one name to the global namespace, **we can define an anonymous function and invoke it** as follows.  **Note that for this to work, the whole thing has to be enclosed in parentheses.**  **(**function(){  function update() {      // Get the two input numbers      var firstNumber = Number(document.getElementById('first').value);      var secondNumber = Number(document.getElementById('second').value);      // Then  compute the sum      var myAnswer = firstNumber + secondNumber;      // And write it in the 'answer' element      document.getElementById('answer').textContent = myAnswer;  };  function help() {      // check that there is an answer currently displayed      var currentAnswer = Number(document.getElementById('answer').textContent);      if (currentAnswer) {          document.getElementById('answer').textContent =              currentAnswer +              '=' +              document.getElementById('first').value +              '+' +              document.getElementById('second').value;      }  };  // Define our event listeners  document.getElementById('first').addEventListener('input', update, false);  document.getElementById('second').addEventListener('input', update, false);  document.getElementById('answer').addEventListener('mouseover', help, false);  }**())**; |  |