Telerik Academy

# JavaScript: Good Practices

## Learn how to write good JavaScript


Telerik Software Academy
Learning & Development Team
http://academy.telerik.com

Telerik Academy

# Table of Contents

- Naming in JavaScript
- Scoping
  - Global and function
  - Fake block scope
- Duplicated Object Identifiers
- The this object
  - In global, function and object scope
  - In event handler
- Variables
- Strict mode

Telerik Academy

# Naming in JavaScript

# Naming

- In JavaScript almost everything is camelCase
  - Variables, methods, properties
  - Objects, modules

```
var number = 5;
function printMsg(message){ … }
var arr = [];
arr.toString();
var controls = (function(){ … } ());
```

Telerik Academy

# Naming: Function Constructors

- The only exception to the rule is function constructor
  - Function constructors use PascalCase
- Function constructors are more special than the other objects
  - They are meant to be called with `new`
  - Without `new`, `this` has an incorrect value
- Unfortunately, JavaScript has no way to restrict a call to a function constructor without `new`
  - All we have to do is prey the developer sees the visual difference

Telerik Academy

# Scoping

## Global, Function and Object

# Scoping

- JavaScript has only two types of scope
  - Global scope and function scope
    - Function scope may be called object scope when used with new
  - There is no block scope in JavaScript
    - { and } do not define a scope
    - Use IIFE to define scope
- All JavaScript code, in all files, share the same global scope

# Scoping

- Everything inside an `if-else`, `for` or `while` "block scope" is actually outside this block

```
if (false) {
  var count = 15;
  function printMsg(message) {
    console.log("Message: " + message + "!");
  };
}
printMsg(count)
//outputs Message: undefined!
```

both count and printMsg are defined

- Both `printMsg` and `count` are declared
  - Yet, count has no value, because the execution flow cannot reach the initialization

Telerik Academy

# Fake "Block" Scope

Live Demo

# Function Scope

- Function scope is the only scope where variables are temporary
  - A variable, declared with var, cannot exist outside of its function scope

```
(function(){
  if (false) {
    var count = 15;
    function printMsg(message) {
      console.log("Message: " + message + "!");
    };
  }
}());
printMsg(count);
//ReferenceError: printMsg is not defined
```

Telerik Academy

# Function Scope

## Live Demo

Telerik Academy

# Duplicated Object Identifiers

Telerik Academy

# Duplicated Object Identifiers

- Due to the shared global scope in JavaScript, it is a common case to have a duplicated object identifiers
  - What happens if two or more libraries/frameworks/js files have a function with the same name?
  - Which one is the actual?

Telerik Academy

# Duplicated Object Identifiers (2)

- The solution to preventing duplicated identifiers is using function scope, or maybe a module
  - Do the stuff you need and expose only the meaningful pieces

```
jsConsole.write("Message");
document.write("Message");
database.write("Message");
```

Telerik Academy

# Duplicated Object Identifiers

Live Demo

Telerik Academy

# The this Object

Telerik Academy

# The this Object

- The this object has a different value depending on the scope
  - In function scope
  - In object scope
  - In global scope
  - In event handlers

# this in Global Scope

- In the global scope `this` means the global scope
  - i.e. window

```
console.log( this === window) //logs true
```

- These work exactly the same when in global scope

```
this.message = "Hello";
```

```
var message = "Hello";
```

```
window.message = "Hello";
```

Telerik Academy

# this in Global Scope

Live Demo

# this in Function Scope

- this in function scope almost always means the this of the parent of the function
  - If the function is in the global scope this means the global scope
  - In object scope – this means the object itself
    - Later in this presentation

```
(function createAndSetVariable(number){
  this.number = number;
}(5));
console.log(number); //logs 5
```

this means window

Telerik Academy

# this in Function Scope

Live Demo

Telerik Academy

# this in Object Scope

- Object scope is created when a function is called with new
  - The rules that apply are the same as with regular function call
    - Except for the value of this

```
function Person(fname, lname){
 this.fname = fname;
 this.lname = lname;
}
var person = new Person();
var invalidPerson = Person();
```

this means an instance of the person object

this means the window

- Always beware of PascalCase-named functions
  - There is a reason for that!

Telerik Academy

# this in Object Scope

Live Demo

Telerik Academy

# this in Event Handlers

- this in an event handler means the DOM element that the event was fired on
  - i.e. if a click event fires on a anchor element, this means the clicked anchor element

```
var button = document.getElementById("the-button");
function onButtonClick(ev){
 console.log(this === button); //logs true
}
button.addEventListener("click", onButtonClick, false);
```

```
var usernameTb = document.getElementById("tb-user");
function onUsernameTbChange(ev){
 console.log(this === usernameTb); //logs true
}
usernameTb.addEventListener("change", onUsernameTbChange, false);
```

Telerik Academy

# this in Event Handlers

Live Demo

Telerik Academy

# Variables

# Variables

- Declare all the variables in the beginning of the scope
  - Even if they will not be used yet
  - This prevents lots of error-prone code
- Many ways to structure variables declaration:

```
function something(){
  var number;
  var word;
  var eventNumbers;
…
}
```

```
function something(){
  var number,
      word,
      eventNumbers;
…
}
```

# Variables

- Declare all the variables in the beginning of the scope
  - Even if they will not be used yet
  - This prevents lots of error-prone code
- Many ways to structure variables declaration:

```
function something(){
  var number;
  var word;
  var eventNumbers;
…
}
```

```
function something(){
  var number,
      word,
      eventNumbers;
…
}
```

Both work exactly the same

Telerik Academy

# Variables

## Live Demo

Telerik Academy

# Strict Mode

# Strict Mode

- Strict mode is a nice subset of the JavaScript functionality
  - Removes some of the bad parts of JavaScript
  - Adds parts of yet-to-be ECMAScript versions
- Strict mode changes both syntax and runtime behavior
  - Makes changes to the syntax to prevent silent errors
  - Restricts functionality to remove bad JS
  - Makes the transition to new JS features more seamless

# Strict Mode Usage

- Strict mode can be used either for the whole script or per-function
  - If used for the whole scripts, everything is in strict mode
    - Not a good idea, since a third-party script may fail in strict mode
  - Better use IIFE and per-function strict mode
    - That way only your code will run in strict mode

Telerik Academy

# Strict Mode Properties

- Some of the characteristics of Strict mode:
  - Converts silent errors to exceptions
    - Trying to change the value of `document`
    - Deleting the prototype of an object
  - Makes `this` undefined inside a function scope
    - In a `function scope`, `this` is equal to `undefined`, instead of the parent this object
  - Forbids octal syntax
  - Prevents variable declaration without var

Telerik Academy

# Strict Mode

Live Demo

Telerik Academy

# JavaScript Execution

Telerik Academy

# JavaScript Execution

- As we know, JavaScript executes per-line-reached basis
  - The execution flow goes from top to bottom
    - Imagine all loaded JavaScript files, merged together in one really big JavaScript file
  - A JavaScript line of code is executed, when it is reached in the execution process
- Yet execution may take time
  - Time that is not pleasant to the user

Telerik Academy

# JavaScript Execution (2)

- A common approach is to start execution of JavaScript, when the web page is ready
  - And there is an event for that

```
window.onload = function(){
  //do the code preparations
}
```

- Or, if using jQuery, we can use its load event

```
$(document).ready(function(){});
$(function(){});
```

- Loading the script at the end of the load time, ensures that all the DOM is already rendered

Telerik Academy

# JavaScript Load in the HTML file

Telerik Academy

# JavaScript Load in the HTML File

- A common questions is "Where to load the JavaScript?"
  - Load it in the header?
  - Load it at the end of the body element?
  - Load it anywhere in the document
- All JavaScript files have the same global scope, so it really doesn't matter?
  - No, it does matter

Telerik Academy

# JavaScript Load in the HTML File (2)

- There are two common places to load the JavaScript files
  - In the header
  - At the end of the body element
- What is really the difference?
  - Put simply – Performance

Telerik Academy

# JavaScript Load in the HTML File (3)

- Loading of large script file at the document header, freezes the web page, and makes it unusable
  - Better show something to the user
    - Like the rendered HTML and CSS
  - And then load your JavaScript

Telerik Academy

# JavaScript: Good Practices


http://academy.telerik.com