JAVASCRIPT CODING STANDARDS

Version 1.2.0

ENG1003

Monash University | 25 Exhibition Walk, Clayton Campus, Monash University 3800

Table of Contents

Purpose

This document serves as the JavaScript coding standards document for all documentation and coding done for ENG1003: Engineering Mobile Apps at Monash University. You should follow the advice in this document for your Assignment work.

The version of this document is 1.2.0.

This document was last updated on the March 22, 2016.

Coding Standards

Indentation

Criteria:

- Blocks of code should be clear and human-readable.
- All content within a block ({}) should be indented to the right, including function bodies, loop bodies and blocks in if-else-if statements.

See example below.

```
function someFunc(x, y)
{
    if (x > 100)
    {
        if (y > 100)
        {
            doThing(x, y);
        }
        else
        {
            doThing(x, 100 - y);
        }
    }
    else if (x>50)
    {
        if (y>100)
        {
            doThing(x + 50, y);
        }
        else
        {
            doThing(x + 50, 100 - y);
        }
    }
    else
    {
        doThing(50 - x, y);
    }
}
```

Bracket Placement

Criteria:

- Brackets should follow a convention
- This convention should be consistently followed
- Some examples of conventions can be found on Wikipedia (https://en.wikipedia.org/wiki/Indent_style)

See example below:

BAD EXAMPLE

```
function doAThing(maxI,maxJ)
{ // PICO style
    for (var i=0;i<maxI;i++){ // K&R style
        for (var j=0;j<maxJ;j++)
        { somethingHappensHere(); // Allman style
        }
    }}
```

GOOD EXAMPLE (using the Allman style)

```
function doAThing(maxI,maxJ)
{
    for (var i=0;i<maxI;i++)
    {
        for (var j=0;j<maxJ;j++)
        {
            somethingHappensHere();
        }
    }
}
```

GOOD EXAMPLE (using the K&R style)

```
function doAThing(maxI,maxJ)
{
    for (var i=0;i<maxI;i++) {
        for (var j=0;j<maxJ;j++) {
            somethingHappensHere();
        }
    }
}
```

Line Length

Criteria:

- One line should have no more than one JavaScript statement
- If you have very long line comments or code you may choose how to handle it, for instance by
  - Allowing them to extend as far as needed, or
  - Artificially limiting their length and manually wrapping them at a particular length, e.g., 80 characters
- You should be consistent in how you choose to handle these case

See examples below:

BAD EXAMPLE

```
// [...]
function upperTriangularForm(coefficientMatrix,resultantVector)
{
    if (isNumeric(coefficientMatrix))
    {
        // We set up a specific function here, to ensure that all parts
        // of the matrix were numeric, as Javascript isn't strict with
        // types.

        [...]

        // Note that we update the final elimination matrix as we go to
save time, considering we'll need this to perform LU decomposition. This
way we don't need to store each elimination matrix used, we just need to
know the most recent one.
        matrixMultiplication(coefficientMatrix,eliminationMatrix);
        finalEliminationMatrix = matrixMultiplication(
                finalEliminationMatrix, eliminationMatrix);

        [...]
    }
}
```

GOOD EXAMPLE (using fixed line length)

```
// [...]
function upperTriangularForm(coefficientMatrix,resultantVector)
{
    if (isNumeric(coefficientMatrix))
    {
        // We set up a specific function here, to ensure that all parts
        // of the matrix were numeric, as Javascript isn't strict with
        // types.

        [...]
```

```
        // Note that we update the final elimination matrix as we go to
        // save time, considering we'll need this to perform LU
        // decomposition. This way we don't need to store each
        // elimination matrix used, we just need to know the most
        // recent one.
        matrixMultiplication(coefficientMatrix,eliminationMatrix);
        finalEliminationMatrix = matrixMultiplication(
                finalEliminationMatrix, eliminationMatrix);

        [...]
    }
}
```

Naming Conventions

Criteria:

- Function and variable names (including function parameters) must be **meaningful**, and be in **lowerCamelCase.**
- The name of each function and variable should to be **indicative** of its **purpose**.
- Class names should be **UpperCamelCase**.

See example below.

BAD EXAMPLE

```
// This function moves the image c to position represented by a,b
function A(a, b, c)
{
}
```

GOOD EXAMPLE

```
// This function moves the image 'imageToMove' to a position
//    represented by coordinates xCoordinate, yCoordinate
function moveToLocation(xCoordinate, yCoordinate, imageToMove)
{

}
```

Magic Numbers:

Criteria:

- Unnamed **constants** (i.e., values that don't change) should be **assigned to named variables**
- This is *particularly* important for regularly used values
- As with any variable, the name of this should indicate its purpose
- The convention for constants is that their names are in **ALL_CAPS** (with words separated by underscores) to indicate that they are fixed

See example below.

BAD EXAMPLE

```
var area = radius * radius * (3.14159);
var circumference = 2 * radius * (3.14159);
var kilometers = 1.60934 * miles;
```

GOOD EXAMPLE

```
var PI = 3.14159;
var MILES_TO_KM = 1.60934;

var area = radius * radius * PI;
var circumference = 2 * radius * PI;
var kilometers = MILES_TO_KM * miles;
```

Variable Scope

Criteria:

- Variables should be defined with the **most restrictive** (smallest) scope possible
- Global variables should only be used when **absolutely necessary**
- Global variables should be defined at the **top of file**, above function definitions and after any file documentation

See examples of scoping within functions below.

BAD EXAMPLE

```
function checkValidMove(attemptedMove)
{
	cellsJumped = 0;
	for (i = currentPiece[0];  i < attemptedMove; i++)
	{
		for (j = currentPiece[1]; j < attemptedMove[1]; j++)
		{
			if (isEnemy(i,j))
			{
				cellsJumped += 1;
			}
		}
	}
	return ((cellsJumped % 2) === 0);
}
// i, j and cellsJumped accessible outside of function (due to no 'var')
```

GOOD EXAMPLE

```
function checkValidMove(attemptedMove)
{
	var cellsJumped = 0;
	for (var i = currentPiece[0];  i < attemptedMove; i++)
	{
		for (var j = currentPiece[1]; j < attemptedMove[1]; j++)
		{
			if (isEnemy(i,j))
			{
				cellsJumped += 1;
			}
		}
	}
	return ((cellsJumped % 2) === 0);
}
```

See examples of global variables below.

BAD EXAMPLE

```
var result;

function sum(a, b)
{
      result = a + b;  // result only used here, but available everywhere

      return result;
}
```

GOOD EXAMPLE

```
function sum(a, b)
{
      var result = a + b;

      return result;
}
```

Function Declarations

Criteria:

- Like with magic numbers, functions should always be **given a name** (not anonymous)
- Functions should be defined at the **top** of a file, just below any global variables or constants, rather than immediately before they are used
- Functions should be defined in the order of dependency, with the functions up the top having no dependencies and those lower down requiring some of the functions above
- Functions should **not** be defined inside other functions.

See examples below.

BAD EXAMPLE

```
// This function is designed to find the maximum value in an
// array and change all elements to be that maximum value
function maximise(anArray)
{
	// Do not declare functions inside another function.
	function getMaxValue(theArray)
	{
		var theMax = theArray[0];
		for (var i = 1; i < theArray.length; i++)
		{
			if (theArray[i] > theMax)
			{
				theMax = theArray[i];
			}
		}
		return theMax;
	}

	var arrayMax = getMaxValue(anArray);
	for (var i = 0; i < anArray.length; i++)
	{
		anArray[i] = arrayMax;
	}
	return anArray;
}
```

GOOD EXAMPLE

```
// The following function can now be reused by other code.
function getMaxValue(anArray)
{
	var theMax = anArray[0];
	for (var i = 1; i < anArray.length; i++)
	{
		if (anArray[i] > theMax)
		{
			theMax = anArray[i];
		}
	}
	return theMax;
}

// This function is designed to find the maximum value in an
// array and change all elements to be that maximum value
function maximise(anArray)
{
	var arrayMax = getMaxValue(anArray);
	for (var i = 0; i < anArray.length; i++)
	{
		anArray[i] = arrayMax;
	}
	return anArray;
}
```

Class Methods

Criteria:

- **Private** methods should be defined using **var**
- **Public** methods should be defined using **this**
- As with function declaration, generally those with the most dependencies should be towards the bottom and those with the least towards the top

See example below.

```
function House(address, owner, alarmCode, contents)
{
	var alarmCode = alarmCode;
	var owner = owner;
	var contents = contents;
	var address = address;
	var alarmState = true;
	var attempts = 0;

	// Private: Returns true if the attempted code is valid.
	// Don't want this to be public or someone could just try many codes.
	var testAlarmCode = function(attemptedCode)
	{
		return alarmCode === attemptedCode;
	};

	// Public: Allows someone to try a code.
	this.disableAlarm = function(trialCode)
	{
		if (testAlarmCode(trialCode))
		{
			alarmState = false;
			attempts = 0;
		}
		else
		{
			attempts++;
			if (attempts >= 3)
			{
				reportIntrusionAttempt(address);
			}
		}
	};
}
```

Cohesion and Coupling - Writing Modular Code

Criteria:

- Always ensure that any function you create performs only one logical task.

See examples below.

BAD EXAMPLE

```
// This function is set up to try and race a series of cars until one
// wins.  It returns the top 5 cars
function raceAllTheCars(theCars)
{
    var win = 100;
    var topFive = [];
    for (var i = 0; i < theCars.length; i++)
    {
        var xChange = Math.random() * 5;
        theCars[i].fuel -= xChange / theCars[i].fuelEconomy;
        theCars[i].xPosition += xChange;
        var worstTopFive = -1;
        var worstTopFiveDist = -1;
        for  (var j = 0; j < topFive.length; j++)
        {
            if (topFive[j].xPosition > worstTopFiveDist)
            {
                worstTopFive = j;
                worstTopFiveDist = topFive[j].xPosition;
            }
        }
        if (topFive.length < 5)
        {
            topFive.push(theCars[i]);
        }
        else
        {
            if (worstTopFiveDist < theCars[i].xPosition)
            {
                topFive[worstTopFive] = theCars[i];
            }
        }
        if (theCars[i].xPosition > win)
        {
            return topFive;
        }
    }
}
```

GOOD EXAMPLE

```
// This function is set up to try and race a series of cars until one
// wins.  It returns the top 5 cars
function raceAllTheCars(theCars)
{
    var win = 100;
    var topFive = [];
    for (var i = 0; i < theCars.length; i++)
    {
        var xChange = Math.random() * 5;
        theCars[i].move(xChange);
        if (topFive.length < 5)
        {
            topFive.push(theCars[i]);
        }
        else
        {
            replaceSmallestWith(topFive, theCars[i]);
        }
        if (theCars[i].xPosition > win)
        {
            return topFive;
        }
    }
}

function replaceSmallestWith(anArray, replacement)
{
    var smallestIndex = indexOfMinimumValue(topFive);
    if (anArray(smallestIndex) < replacement.xPosition)
    {
        anArray[smallestIndex] = replacement;
    }
}

/* There would also be
 *   +  a move function
 *   +  an indexOfMinimumValue function
 * which have been omitted for conciseness.
*/
```

Comments and Documentation

Documenting Files

Each of these must:

- Explain the purpose of the file
- List the authors and team responsible for the code/content
- Include a last modified date and version number, if applicable
- List any other relevant details

See example below.

```
/*
 * Purpose: This file is designed to allow for easy changes to vision
 *          impaired modes for CSS-based websites
 * Organization/Team: Synergistic Site Solutions
 * Author: Preity Webb
 * Last Modified: 29 November 2014
 * Version: 3.0.4
 * Licence: Creative Commons Attribution; Non-Commercial (BY-NC)
*/
```

Function Comments

Each of these must:

- Explain the purpose of the function,
- Describe the parameters as well as the return value (if any), and
- Be completely clear to someone unfamiliar with the code

See example below.

```
// moveToLocation()
//
// This function moves the image imageToMove to position represented by
// xCoordinate, yCoordinate. It does this treating 0,0 as the origin at
// the centre of the screen; for images more than 1 pixel by 1 pixel, this
// function places the centre of the image at the location specified. The
// centre is defined as the point halfway between the top and bottom and
// left and right sides, where there is any doubt between several pixels,
// the upper right one is chosen.
//
// argument: xCoordinate: this represents the final location along the
//      x-axis for the object to be placed at.
//
// argument: yCoordinate: this represents the final location along the
//      y-axis for the object to be placed at.
//
// argument: imageToMove: this is the image, passed by reference, that
//      will be placed in the location specified by xCoordinate and
//      yCoordinate. Note that this image cannot exceed a size of 40
//      pixels by 100 pixels
//
// preconditions:
//      image must exist and be at least 1 pixel by 1 pixel
//      image cannot be larger than 40x100 pixels
//      coordinates must be integers as they represent individual pixels
//
// postconditions:
//      image will be at the location specified by the two coordinates
//
// returns:
//      This function does not return anything
//
function moveToLocation(xCoordinate, yCoordinate, imageToMove)
{
    [...]
```

Inline Comments

Each of these must:

- Explain why particular choices were made, and
- Explain any complex code that another software engineer may find confusing
- Should match the indentation of code within that block

See example below.

```
// [...]
function upperTriangularForm(coefficientMatrix,resultantVector)
{
    if (isNumeric(coefficientMatrix))
    {
        // We set up a specific function here, to ensure that all parts
        // of the matrix were numeric, as Javascript isn't strict with
        // types.

        [...]

        // Note that we update the final elimination matrix as we go to
        // save time, considering we'll need this to perform LU
        // decomposition. This way we don't need to store each
        // elimination matrix used, we just need to know the most
        // recent one.
        matrixMultiplication(coefficientMatrix,eliminationMatrix);
        finalEliminationMatrix = matrixMultiplication(
                finalEliminationMatrix, eliminationMatrix);

        [...]
    }
}
```

~ End of Document ~