HTML Coding Standards

1.Always Declare Document Type

Always declare the document type as the first line in your document.

The correct document type for HTML is:

```
<!DOCTYPE html>
```

2.Use Lowercase Element Names

Mixing uppercase and lowercase names looks bad

- Developers normally use lowercase names
- Lowercase looks cleaner
- Lowercase is easier to write

```
<body>
<p>This is a paragraph.</p>
</body>
```

3.Close All HTML Elements

In HTML, you do not have to close all elements (for example the `<p>` element).

However, we strongly recommend closing all HTML elements, like this:

```
<section>
 <p>This is a paragraph.</p>
 <p>This is a paragraph.</p>
</section>
```

4.Use Lowercase Attribute Names

Mixing uppercase and lowercase names looks bad

- Developers normally use lowercase names
- Lowercase looks cleaner
- Lowercase is easier to write

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

5.Always Quote Attribute Values

- Developers normally quote attribute values
- Quoted values are easier to read
- You MUST use quotes if the value contains spaces

```
<table class="striped">
```

6.Always Specify alt, width, and height for Images

Always specify the alt attribute for images. This attribute is important if the image for some reason cannot be displayed.

Also, always define the width and height of images. This reduces flickering, because the browser can reserve space for the image before loading.

```
<img src="html5.gif" alt="HTML5" style="width:128px;height:128px">
```

7.Spaces and Equal Signs

HTML allows spaces around equal signs. But space-less is easier to read and groups entities better together.

```
<link rel="stylesheet" href="styles.css">
```

8.Avoid Long Code Lines

When using an HTML editor, it is NOT convenient to scroll right and left to read the HTML code.

Try to avoid too long code lines.

9.Never Skip the <title> Element

The <title> element is required in HTML.

The contents of a page title is very important for search engine optimization (SEO)! The page title is used by search engine algorithms to decide the order when listing pages in search results.

The <title> element:

- defines a title in the browser toolbar
- provides a title for the page when it is added to favorites
- displays a title for the page in search-engine results

So, try to make the title as accurate and meaningful as possible:

10.Add the lang Attribute

You should always include the lang attribute inside the <html> tag, to declare the language of the Web page. This is meant to assist search engines and browsers.

11.Meta Data

To ensure proper interpretation and correct search engine indexing, both the language and the character encoding <meta charset="charset"> should be defined as early as possible in an HTML document:

12.Setting The Viewport

The viewport is the user's visible area of a web page. It varies with the device - it will be smaller on a mobile phone than on a computer screen.

You should include the following <meta> element in all your web pages:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

13.Accessing HTML Elements with JavaScript

Using "untidy" HTML code can result in JavaScript errors.

These two JavaScript statements will produce different results:

```
getElementById("Demo").innerHTML = "Hello";

getElementById("demo").innerHTML = "Hello";
```

14.Use Lower Case File Names

Some web servers (Apache, Unix) are case sensitive about file names: "london.jpg" cannot be accessed as "London.jpg".

Other web servers (Microsoft, IIS) are not case sensitive: "london.jpg" can be accessed as "London.jpg".

If you use a mix of uppercase and lowercase, you have to be aware of this.

If you move from a case-insensitive to a case-sensitive server, even small errors will break your web!

To avoid these problems, always use lowercase file names!

CSS Coding Standards

1.Formatting

All CSS documents must use **two spaces** for indentation and files should have no trailing whitespace. Other formatting rules:

•Use soft-tabs with a two space indent.

•Use double quotes.

•Use shorthand notation where possible.

•Put spaces after : in property declarations.

•Put spaces before { in rule declarations.

•Use hex color codes #000 unless using rgba().

•Always provide fallback properties for older browsers.

•Use one line per property declaration.

•Always follow a rule with one line of whitespace.

•Always quote url() and @import() contents.

•Do not indent blocks.

2.Naming

All ids, classes and attributes must be lowercase with hyphens used for separation.

```
/* GOOD */
.dataset-list {}

/* BAD */
.datasetlist {}
.datasetList {}
.dataset_list {}
```

3.Modularity and specificity

*Try to keep all selectors loosely grouped into modules where possible and avoid having too many selectors in one declaration to make them easy to override. This helps to keep your code modular and maintainable.

*Code Re-usability: Code which shares a very similar or identical structure should be written in such a way that it can be used further.

JavaScript Coding Standards

1.Variable Names

Use **camelCase** for identifier names (variables and functions).

All names start with a **letter**.

```
firstName = "John";
lastName = "Doe";
```

2.Spaces Around Operators

Always put spaces around operators (= + - * /), and after commas:

Examples:

```
let x = y + z;
const myArray = ["Volvo", "Saab", "Fiat"];
```

3.Code Indentation

Always use 2 spaces for indentation of code blocks:

Functions:

```
function toCelsius(fahrenheit) {
  return (5 / 9) * (fahrenheit - 32);
}
```

4.Statement Rules

General rules for simple statements:

- Always end a simple statement with a semicolon.

```
const cars = ["Volvo", "Saab", "Fiat"];

const person = {
 firstName: "John",
 lastName: "Doe",
 age: 50,
 eyeColor: "blue"
};
```

5.General rules for complex (compound) statements:

- Put the opening bracket at the end of the first line.
- Use one space before the opening bracket.
- Put the closing bracket on a new line, without leading spaces.
- Do not end a complex statement with a semicolon.

```
function toCelsius(fahrenheit) {
  return (5 / 9) * (fahrenheit - 32);
}
```

6.Object Rules

General rules for object definitions:

- Place the opening bracket on the same line as the object name.
- Use colon plus one space between each property and its value.
- Use quotes around string values, not around numeric values.
- Do not add a comma after the last property-value pair.
- Place the closing bracket on a new line, without leading spaces.
- Always end an object definition with a semicolon.

```
const  person = {

   firstName: "John",

   lastName: "Doe",

   age: 50,

   eyeColor: "blue"

};
```

7.Line Length < 80

For readability, avoid lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it, is after an operator or a comma.

```
document.getElementById("demo").innerHTML =
"Hello Dolly.";
```

8.Naming Conventions

Always use the same naming convention for all your code. For example:

- •Variable and function names written as **camelCase**
- •Global variables written in **UPPERCASE** (We don't, but it's quite common)
- •Constants (like PI) written in **UPPERCASE**

9.Loading JavaScript in HTML

Use simple syntax for loading external scripts

```
<script src="myscript.js"></script>
```

10.Accessing HTML Elements

A consequence of using "untidy" HTML styles, might result in JavaScript errors.

These two JavaScript statements will produce different results:

```
const obj = getElementById("Demo")

const obj = getElementById("demo")
```

11.Use Lower Case File Names

Most web servers (Apache, Unix) are case sensitive about file names:

london.jpg cannot be accessed as London.jpg.

Other web servers (Microsoft, IIS) are not case sensitive:

london.jpg can be accessed as London.jpg or london.jpg.

If you use a mix of upper and lower case, you have to be extremely consistent.

If you move from a case insensitive, to a case sensitive server, even small errors can break your web site.

To avoid these problems, always use lower case file names (if possible).

C# Coding Standards

1.Class Names

use **PascalCasing** for class names and method names.

```
public class ClientActivity
{
public void ClearStatistics()
   {
//...
    }
public void CalculateStatistics()
    {
//...
    }
}
```

2.Variable Names

use **camelCasing** for local variables and method arguments.

```
public class UserLog
{
public void Add(LogEvent logEvent)
{
int itemCount = logEvent.Items.Count;
// ...
}
}
```

3.Identifiers

Do not use **Hungarian** notation or any other type identification in identifiers

```
// Correct
int counter;
string name;
// Avoid
int iCounter;
string strName;
```

4.Constants

Dont use **Screaming Caps** for constants or readonly variables

```
// Correct
public static const string ShippingType = "DropShip";
// Avoid
public static const string SHIPPINGTYPE = "DropShip";
```

5.Abbreviations

Avoid using **Abbreviations**.

Exceptions: abbreviations commonly used as names, such as **Id, Xml, Ftp, Uri**

```
// Correct
UserGroup userGroup;
Assignment employeeAssignment;
// Avoid
UserGroup usrGrp;
Assignment empAssignment;
// Exceptions
CustomerId customerId;
XmlDocument xmlDocument;
FtpHelper ftpHelper;
UriPart uriPart;
```

6.Abbreviation Casing

use **PascalCasing** for abbreviations 3 characters or more (2 chars are both uppercase)

```
HtmlHelper htmlHelper;
FtpTransfer ftpTransfer;
UIControl uiControl;
```

7.No Underscores

Dont use **Underscores** in identifiers.
Exception: you can prefix private static variables with an underscore.

```
// Correct
public DateTime clientAppointment;
public TimeSpan timeLeft;
// Avoid
public DateTime client_Appointment;
public TimeSpan time_Left;
// Exception
private DateTime _registrationDate;
```

8.Type Names

use **predefined type names** instead of system type names like Int16, Single, UInt64, etc

```
// Correct
string firstName;
int lastIndex;
bool isSaved;
// Avoid
String firstName;
Int32 lastIndex;
Boolean isSaved;
```

9.Implicit Types

use implicit type **var** for local variable declarations.

Exception: primitive types (int, string, double, etc) use predefined names.

```
var stream = File.Create(path);
var customers = new Dictionary();
// Exceptions
int index = 100;
string timeSheet;
bool isCompleted;
```

10.Noun Class Names

use noun or noun phrases to name a class.

```
public class Employee
{
}
public class BusinessLocation
{
}
public class DocumentCollection
{
}
```

11.Interfaces

prefix interfaces with the letter I. Interface names are noun (phrases) or adjectives.

```
public interface IShape
```

```
{
}
public interface IShapeCollection
{
}
public interface IGroupable
{
}
```

12.File Names

Name source files according to their main classes. Exception: file names with partial classes reflect their source or purpose, e.g. designer, generated, etc.

```
// Located in Task.cs
public partial class Task
{
//...
}

// Located in Task.generated.cs
public partial class Task
{
//...
}
```

13.Namespaces

organize namespaces with a clearly defined structure

```
// Examples
namespace Company.Product.Module.SubModule
namespace Product.Module.Component
namespace Product.Layer.Module.Group
```

14.Curly Brackets

vertically align curly brackets.

```
// Correct
class Program
{
static void Main(string[] args)
{
}
}
```

15.Member Variables

declare all member variables at the top of a class, with static variables at the very top.

```
// Correct
public class Account
{
public static string BankName;
public static decimal Reserves;
public string Number {get; set;}
public DateTime DateOpened {get; set;}
public DateTime DateClosed {get; set;}
public decimal Balance {get; set;}
// Constructor
public Account()
{
// ...
}
}
```

16.Enums

use singular names for enums. Exception: bit field enums.

```
// Correct
public enum Color
{
Red,
Green,
Blue,
Yellow,
Magenta,
Cyan
}
// Exception
[Flags]
public enum Dockings
{
None = 0,
Top = 1,
Right = 2,
Bottom = 4,
Left = 8
}
```

17.Enum Types

Do not explicitly specify a type of an enum or values of enums (except bit fields)

```
// Don't
public enum Direction : long
{
North = 1,
East = 2,
South = 3,
West = 4
}
// Correct
public enum Direction
{
North,
East,
South,
West
}
```

18.Enum Suffix

Do not use suffix enum names with Enum

```
// Don't
public enum CoinEnum
{
Penny,
Nickel,
Dime,
Quarter,
Dollar
}
// Correct
public enum Coin
{
    Penny,
    Nickel,
    Dime,
    Quarter,
    Dollar
    }
```

SQL Coding Standards

1.Naming

-Tables:

Rules: Pascal notation; end with an ‘s’
Examples: Products, Customers Group related table names1

Stored Procs:

Rules: sp_[_] Examples: spOrders_GetNewOrders, spProducts_UpdateProduct

Triggers:

Rules: TR__ Examples: TR_Orders_UpdateProducts Notes: The use of triggers is discouraged

Indexes:

Rules: IX__ Examples: IX_Products_ProductID

Primary Keys:

Rules: PK_ Examples: PK_Products

Foreign Keys:

Rules: FK__ Example: FK_Products_Orderss

Defaults:

Rules: DF__ Example: DF_Products_Quantity

Columns:

If a column references another table’s column, name it ID Example: The Customers table has an ID column The Orders table should have a CustomerID column

General Rules:

Do not use spaces in the name of database objects Do not use SQL keywords as the name of database objects In cases where this is necessary, surround the object name with brackets, such as [Year]

2.Structure

• Each table must have a primary key
o In most cases it should be an IDENTITY column named ID
• Normalize data to third normal form
o Do not compromise on performance to reach third normal form. Sometimes, a little denormalization results in better performance.
• Do not use TEXT as a data type; use the maximum allowed characters of VARCHAR instead
• In VARCHAR data columns, do not default to NULL; use an empty string instead
• Columns with default values should not allow NULLs
• As much as possible, create stored procedures on the same database as the main tables they will be accessing

2.Formatting

• Use upper case for all SQL keywords
o SELECT, INSERT, UPDATE, WHERE, AND, OR, LIKE, etc.
• Indent code to improve readability
• Comment code blocks that are not easily understandable
o Use single-line comment markers(--)
o Reserve multi-line comments (/*.. ..*/) for blocking out sections of code
• Use single quote characters to delimit strings.
o Nest single quotes to express a single quote or apostrophe within a string For example, SET @sExample = 'SQL''s Authority'

• Use parentheses to increase readability o WHERE (color=’red’ AND (size = 1 OR size = 2))
• Use BEGIN..END blocks only when multiple statements are present within a conditional code segment.
• Use one blank line to separate code sections.
• Use spaces so that expressions read like sentences.
o fillfactor = 25, not fillfactor=25
• Format JOIN operations using indents o Also, use ANSI Joins instead of old style joins4
• Place SET statements before any executing code in the procedure.