**CODING GUIDELINES FOR HTML**

* Always declare document type

The correct document type for HTML is: <!DOCTYPE html>

* Use lowercase Element names

Lowecase looks cleaner and easy to style

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

* Close all the HTML elements
* Use lowercase attribute names
* Always quote attribute values.They are easier to read.
* Always specify alt,width and height for images.The alt attribute is important and reduces flickering,because the browser can reserve space for the image before loading.
* Spaces and equal signs: HTML allows spaces around equal signs to look better.
* 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.
* Blank lines and Indendation:

Do not add blank lines, spaces without a reason

For readability ,add blank lines to separate large or logical block of code

* Never skip the <title> Element

The page title is used by search engine alogorithm to decide the order when listing pages in search results

The <title> element:

Defines a title in the browser toolbar.

Display a title for the page in search-engine results.

Make sure that the title as accurate and meaningful.

* Omitting <html> and <body>

Always add <html> and <body> tags.

Omitting <body> can produce errors in older browsers.

Omitting <body> and <html> can also crash DOM and XML software.

* Omitting <head>

The HTML <head> tag can also be omitted. But the <head> tag is recommended.

* Add the lang attribute

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.

* 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.

Eg: <!DOCTYPE html>  
 <html lang="en-us">  
 <head>  
   <meta charset="UTF-8">  
   <title>Page Title</title>  
 </head>

* Setting the Viewport

The viewport is the user’s visible area of the web page. This gives the browser instructions on how to control the page’s dimension and scaling.

The width=device-width part sets the width of the page to follow the screen-width of the device.

The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

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

* HTML comments

Short comments should be written on one line

<!-- This is a comment -->

Long comments are easier to observe if they are indented with two spaces

<!--  
  This is a long comment example. This is a long comment example.  
  This is a long comment example. This is a long comment example.  
-->

* Using Style sheets

The style sheets is inserted in the head section

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

* Loading Javascript in HTML

<script src="myscript.js">

* File extensions

HTML files should have a **.html** extension (**.htm** is allowed).

CSS files should have a **.css** extension.

JavaScript files should have a **.js** extension.

* **Whitespace**: Use whitespace to improve readability. For example, add spaces around attribute values and before self-closing slashes (/>).

Example: <img src="image.jpg" alt="Description" />

These guidelines ensure the HTML code is well-structured, maintainable, and accessible, which is crucial for building robust and user-friendly web applications.

**CODING GUIDELINES FOR CSS**

CSS coding guidelines help maintain consistency, readability, and scalability in your stylesheets. Here are some best practices and guidelines to follow:

* **Indentation**: Use consistent indentation with 2 or 4 spaces per level to make the structure of your CSS clear.
* **Whitespace**: Use whitespace to improve readability. For example, add spaces around selectors, property names, and property values.
* **Braces**: Use the "end-of-line" style for braces (opening brace { on the same line as the selector), unless you have a specific reason to use another style.

css

Copy code

.selector {

property: value;

}

* **Semicolons**: End each property declaration with a semicolon (;). This is necessary to separate property declarations, even though the last one in a block doesn't strictly require it.
* **Comments**: Use comments to explain sections of your CSS or to provide context for future developers working on the code.

css

Copy code

/\* Header Styles \*/

.header {

background-color: #333;

color: #fff; /\* Text color \*/

}

* **Class and ID Names**: Use meaningful and descriptive names for classes and IDs. Prefer hyphens (-) for multi-word names.

css

Copy code

.main-content {

/\* styles \*/

}

#header-nav {

/\* styles \*/

}

* **Avoid Generic Names**: Avoid using overly generic class names like .box or .container unless they are widely understood patterns.
* **BEM (Block Element Modifier)**: Consider using BEM methodology for naming classes in larger projects to maintain clarity and modularity.

css

Copy code

.block {

/\* Block styles \*/

}

.block\_\_element {

/\* Element styles \*/

}

.block\_\_element--modifier {

/\* Modifier styles \*/

}

* **Modular Approach**: Organize CSS rules by component or module to improve maintainability.

css

Copy code

/\* Header Styles \*/

.header {

/\* styles \*/

}

.header\_\_logo {

/\* styles \*/

}

.header\_\_nav {

/\* styles \*/

}

* **Separation of Concerns**: Keep layout (structure), presentation (style), and behavior (JavaScript) separate. Use external stylesheets (<link rel="stylesheet" href="styles.css">) whenever possible.
* **CSS Preprocessors**: If using CSS preprocessors like Sass or Less, leverage their features (variables, mixins, nesting) to keep your stylesheets more maintainable and reusable.
* **Selectors**: Use efficient selectors to minimize browser rendering overhead. Avoid overly complex selectors (#header .nav ul li a) that can negatively impact performance.
* **Responsive Design**: Use media queries (@media) for responsive design rather than separate stylesheets for different screen sizes.

**CODING GUIDELINES FOR JAVASCRIPT**

* Indentation

Two space indentation is used in the javscript.

* Variable declaration

All variables must be declared with var before they are used.

Variables are declared at the beginning of the function.

All names start with the letter.

* Loops and control structures

Always follow whitespace after an identifier of control statements.

Always provide whitespace and linebreak after the semicolon

Always provide a whitespace after every coma.

* Statement guidelines

Simple statement:

The statement which only consists of a single line which end with a semicolon.

Compound statement

A list of statements with a closing bracket.

Usually do not put the semicolon at the end of these type of statements.

Multiline statements

Whenever a statement is not small enough to fit in one line, line breaks must occur after an operator.To improves readability lines should break into multiple lines.

* Comments

Use line comments, not block comments. The comments should start at the left margin. Use ‘//’ in start of comments.

* Use single quotes for strings except to avoid escaping
* No unused variables
* Add a space after keywords.
* Add a space before a function declaration’s parentheses.
* Always use === instead of ==

Eg: if (name === ‘John’)

* Keep else statements on the same line as their curly braces

Eg: if(condition){

//

}else{

//

}

* Always handle the err function parameter
* Declare browser globals with a /\* global \*/ comment.

Exceptions are: window, document and navigator.

* Multiple blank lines are not allowed.
* For the ternary operator in a multi-line setting,place ? and : on their own lines.
* For var declartions, write each declaration in its own statement.

Eg: var silent=true

Var verbose=true

Avoid declarations like:

Var silent=true, verbose=true

* Use camelcase when naming variables and functions.
* Commas must be placed at the end of the current line.
* Constructor names must begin with a capital letter.

Eg:

Function Animal(){

//

}

* Objects must contain a getter when a setter is defined.
* Avoid modifying variables declared using const
* Avoid using constant expressions in conditions.
* No debugger statements
* Immediately invoked function expressions must be wrapped.
* Arrays should have one space separating each element and the assignment operators. Use trailing commas in multi-line arrays.

Eg: let fruits = [

‘apple’,

‘banana’,

]

**CODING GUIDELINES FOR jQuery**

* When writing a jQuery plugin,we use jQuery $ variable as:

(function ($) {

//

})(jQuery)

* Always use a CDN to include jQuery on the page.

CDN deliver content faster then regular hosting provider because their server are globally distributed.

Increase scalability.

* Do not load multiple jQuery version
* jQuery variables: Always cache the jQuery selector returned objects in variables for reuse.
* Do nt use anonymous functions to attach events. Anonymous functions are difficult to debug,maintain,test ,or reuse.
* Whitespaces:

Use spaces around operators and after commas for better readability.

Use blank lines to separate logical sections of code.

* Naming Conventions

Use meaningful and descriptive names for variables and functions.

Use consistent naming conventions throughout the codebase.

* Comments and Documentation

Use comments to explain complex logic, assumptions

Keep comments concise and updated to reflect changes in code

* Performance

Cache jQuery objects to minimize DOM traversal.

* Event Handling

Avoid excessive use of `$(document).ready()`; consider using the shorthand `$(function() {})` instead.

* DOM Manipulation

Use chaning to streamline jQuery code and improve performance.

Minimize DOM manipulation inside loops.

* Selectors

Optimize selectors for performance. Prefer ID (‘#elementID`) and class (`.className`) selectors over tag selectors for better performance.

* Error-handling

Use `try-catch` blocks for error handling around jQuery operations that may fail.

**CODING GUIDELINES FOR C#**

* Class and method names should always be in Pascal Case

public class **Employee**

{

public Employee **GetDetails()**

{

//...

}

public double **GetBonus()**

{

//...

}

}

* Method argument and local variables should always be in Camel Case

public class Employee

{

public void PrintDetails(int **employeeId**, String **firstName**)

{

int **totalSalary** = 2000;

// ...

}

}

* Avoid the use of underscore while naming identifiers

// Correct

public DateTime fromDate;

public String firstName;

// Avoid

public DateTime from\_Date;

public String first\_Name;

* Avoid the use of System data types and prefer using the predefined data types

// Correct

int employeeId;

string employeeName;

bool isActive;

// Avoid

Int32 employeeId;

String employeeName;

Boolean isActive;

* Always prefix an interface with letter I

// Correct

public interface **IEmployee**

{

}

public interface **IShape**

{

}

public interface **IAnimal**

{

}

// Avoid

public interface **Employee**

{

}

public interface **Shape**

{

}

public interface **Animal**

{

}

* For better code indentation and readability always align the curly braces vertically

// Correct

class Employee

{

static void PrintDetails()

{

}

}

// Avoid

class Employee

{

static void PrintDetails()

{

}

}

* Always use the using keyword when working with disposable types. It automatically disposes the object when program flow leaves the scope.

using(var conn = new SqlConnection(connectionString))

{

// use the connection and the stream

using (var dr = cmd.ExecuteReader())

{

//

}

}

* Always declare the variables as close as possible to their use.

// Correct

String firstName = "Shubham";

Console.WriteLine(firstName);

//--------------------------

// Avoid

String firstName = "Shubham";

//--------------------------

//--------------------------

//--------------------------

Console.WriteLine(firstName);

* Always declare the properties as private so as to achieve Encapsulation and ensure data hiding.
* Always separate the methods, different sections of program by one space.
* Constants should always be declared in uppercase.
* Avoid magic numbers or hard-coded strings.Always use constants or enum instead.
* Method should have a single responsibility.
* Keep method lengths manageable
* Class structure: Place fields at the top of the class, followes by constructors, properties, methods, and finally nested types.