SUTRIX MEDIA

FRONTEND
(HTML / CSS / JAVASCRIPT)

Code Conventions

BY Len Nguyen

Revision History

- Version 1.0 – December 1st, 2011
  - Initial by Len Nguyen
- Version 1.1 – June 10th, 2012
  - Added CSS Sprites by Phuong Tran
  - Added JS Performance Tips by Anh Le
- Version 1.2 – December 1st, 2013
  - Added Build tools by Len Nguyen

Table of Contents

Overview

- The objective of this document is to define the coding standard
- It is important to follow all the rules defined in this document
- Every line of code should appear to be written by a single person, no matter the number of contributors
- Why code conventions ?
  - Consistency – like single person typed it
  - Readability – easier to read the source code
  - Maintainability – easier to modify the source code

HTML

CONTENT

HTML

- HTML stands for Hyper Text Markup Language
  - A markup language is a set of markup tags
  - HTML uses markup tags to describe the web pages
- File Naming
  - HTML files should be stored in and delivered as **file-name.html** files
  - Avoid naming a file with a period "."
- Indentation
  - Two spaces should be used as the unit of indentation

HTML

- Use well-formed HTML
  - All tags should be in lowercase
  - Closing tags
  - Nested elements
- Use well-structured HTML
  - Support dynamic content
- Use semantic HTML
  - Accessibility
  - SEO

HTML

- Naming Conventions
  - Identifiers (names, ids and classes) can contain only the characters [a-z0-9] and the hyphen (-)
  - Separate words in ID and CLASS names by a hyphen (-)
  - Avoid unnecessary long names
  - Choose semantic names based on functionality, not on appearance or position
  - Every line of code should appear to be written by a single person, no matter the number of contributors

HTML

- Rules
  - Use **p** tags for paragraph delimiters instead of multiple **br** tags
  - Use **div** tags to wrap labels and controls in a form
  - Use **fieldset** tags to group related elements in a form
  - Use **label** fields to label each form field, the **for** attribute should associate itself with the input field, so users can click the labels
  - Use **h1** for page title, **h2** for block title, **h3-h6** for smaller heading in content
  - The form element should has the same values of **name** and **id** property
  - The **id** attribute must be unique within the document
  - Do not use the **size** attribute on your input fields. Use CSS width instead
  - Add comments on some closing tags to indicate what element you're closing
  - Tables should not be used for page layout, they should be used for tabular data only
  - Make use of **thead**, **tbody** and **th** tags (and **scope** attribute) when appropriate
  - Use **microformats**, **microdata** when appropriate

HTML

- Rules
  - Do not use all caps or all lowercase titles in markup, instead apply the CSS property text-transform: uppercase/lowercase
  - The layer should be placed before closing of body tag. Prefer to get the content of layer from Ajax or generate from JavaScript
  - Avoid to use http and https protocols in the same page, use // instead
  - Use HTML encoded characters (&copy; instead of ©)
  - Use HTML5 custom data attributes (data-*)
  - Always use double quotes, never single quotes, on attributes
  - Nested elements should be indented once (two spaces)
  - Don't include a trailing slash in self-closing elements (HTML5 doctype)
  - Attribute order: class, id, name, data-*, src, for, type, href, title, alt, aria-*, role

HTML

- W3C validation
  - Make sure all tags are nested properly
  - Do not put a block element inside an inline element
  - Do not nest a **p** tag in a heading **h1-h6** tag, and vice versa
  - A link should have its **title** attribute
  - An image should have its **alt** attribute
  - A **form** tag should has its **action, method** attributes
  - Some tags in pairs (ul – li, ol – li, dl – dt & dd)
  - Form elements (input, textarea, select) must have **name** attribute

HTML

- General structure
  - [.wrapper]
    - .container
      - header
      - .inner
        - .sidebar
        - main
          - .block-1
          - .block-2
      - footer

HTML

- General block structure
  - .block-1
    - [.outer]
      - [.inner]
        - [.content]
          - p Hello World
        - [.group]
          - .block-2
          - .block-3

HTML

- Specific block structure
  - .block-1.product-block
    - .item
    - .item
    - .item
    - .item

  - .block-2.price-block
    - .row
      - .col
      - .col
      - .col
    - .row
      - .col
      - .col
      - .col

HTML

- Specific block structure
  - .slideshow(data-slideshow, data-effect="fade")
    - .preview
      - .wrap
        - ul
          - li
            - a
    - .controls
      - ul
        - li
          - a

HTML

- Newsletter
  - HTML3/CSS1 will be used
  - Need to test on:
    - Web: Google Mail, Yahoo! mail, Windows Live Email, Hotmail, etc
    - Software: MS Outlook, Mozilla Thunderbird, Apple Mail, etc
  - Your code should be inside BODY tag
  - Use TABLE structure
  - TABLE tags must have the following attributes: <table **cellspacing**="0" **cellpadding**="0" **border**="0" **width**="800" style="**width**: 800px">
  - Use thead, tbody and th tags in table
  - TD tags must have the valign attributes and must contain explicit width and height: <td **valign**="top" **width**="30" **height**="50" style="**width**: 30px; **height**: 50px;"></td>
  - Do not use **colspan** or **rowspan**. Use inline tables instead
  - IMG tags must contain **width**, **height**, **alt**, **border** attributes and inline CSS display: block: <img alt="" width="150" height="15" style="display: block;">

HTML

- Newsletter
  - Links with specific color: <a href="#" style="text-decoration: none;"><span style="color: #ffffff;">Example.com</span></a>
  - TD tags do not contain text must have font-size: 1. eg. <td width="50" height="50" style="width: 50px; height: 50px; **font-size**: 1px;">
  - TD tags contain text must have the font-size of the text: <td width="250" height="50" style="width: 250px; height: 50px; **font-size**: 12px;">Text</td>
  - All unnecessary spaces, tabs and line breaks inside TD tags must be removed. eg. <td valign="bottom" width="250" height="50" style="width: 250px; height: 50px; font-size: 12px;">Dear <b>Name</b>,<br><span style="font-size: 14px;">Text</span></td>
  - When text over background image or gradient background, cut it as image

CSS

PRESENTATION

CSS

- CSS stands for Cascading Style Sheets
  - Styles define how to display HTML elements
  - Is used to describe the presentation of HTML document

- File Naming
  - CSS files should be stored in and delivered as **file-name.css** files
  - Avoid naming a file with a period "."

- Indentation
  - Two spaces should be used as the unit of indentation

CSS

- Selector
  - A selector is the element that is linked to a particular style
  - Syntax: **selector** {property: value; property: value;}

- ID selector
  - The ID selector is used to specify a style for a single, unique element
  - Example: **#selector-long-name** {property: value; property: value;}

- CLASS selector
  - The CLASS selector is used to specify a style for a group of elements, several elements.
  - Example: **.selector-long-name** {property: value; property: value;}

CSS

- TAG selector
  - Example: **tag** {property: value; property: value;}

- UNIVERSAL selector
  - Example: * {property: value; property: value;}

- Contextual selector
  - Descendant selector: the 2nd element is nested within the 1st one
    p a {color: red;}
  - Adjacent selector: the 2nd element is immediately following by the 1st one
    h1 + p {font-weight: bold;}
  - Child selector: The 2nd element is the immediate child of 1st one
    ul > li {font-weight: bold;}

CSS

- Multiple selectors
  - Multiple selectors should be in order, HTML tags selector, HTML tags with class/id selector, .CLASS selector, #ID selector
  - Syntax:

    HTML tags selector, HTML tags with class/id selector, .CLASS selector, #ID selector {property: value; property: value;}

    Or

    HTML tags selector,

    HTML tags with class/id selector,

    .CLASS selector,

    #ID selector {

    property: value;

    property: value;

    }

CSS

- Syntax
  - The CSS syntax is made up of three parts: a selector, a property and a value: selector {property: value;}

1. selector1,[space]selector2[space]{property:[space]value;}

2. selector1,
   selector2[space]{property:[space]value;[space]property:[space]value;}

3.
   ```
   selector1,
   selector2[space]{
     property:[space]value;
     property:[space]value;
     property:[space]value;
   }
   ```

CSS

- Naming Conventions
  - Identifiers (names, ids and classes) can contain only the characters [a-z0-9] and the hyphen (-)
  - Selector long names using hyphen "-" separator
  - Avoid unnecessary long names
  - Choose semantic names based on functionality, not on appearance or position
  - Every line of code should appear to be written by a single person, no matter the number of contributors

CSS

- Naming Conventions
  - Naming:
    - ID: Don't use ID selectors in CSS
    - CLASS:
      - Use suffixes: -block, -list, -item, -form, -btn, -group for specific case
      - Use prefixes: block-, list-, item-, form-, btn-, group- for general case
      - Use prefixes for elements: select-, input-, width-, color-, editor-
      - Use classes: outer, inner, content, group, wrap for wrapper
      - Use general block naming: slideshow, slider, carousel, gallery, banner, accordion, calendar, datepicker
      - Use common classes: wrapper, container, main, primary, secondary, sidebar, header, footer, overlay, nav, slogan, loading, thumb, preview, highlight, featured, related, panel, module, box, layer, tab, rating, caption, description, breadcrumb, paging, social, toolbar, toolbox, tooltip, active, inactive, current, focus, warning, error, success

CSS

- Naming Conventions
  - Naming:
    - CLASS:
      - Use sprite classes: wi-general, wi-text, wi-icon, wi-button, wi-box, wi-layer, wi-form, wi-form-elements, wi-corner, wi-frame
      - Avoid naming: id, name, class, submit, reset

CSS

- Naming Conventions
  - Abbreviation images:
    - Particular elements
      - Background: **bgd-**
      - Photo / Picture: **photo-**
      - Button: **btn-**
      - Logo: **logo-**
      - Icon: **icon-**

    - Sprites elements: **their filenames**

CSS

- Image Optimization
  - The importance of reducing images sizes is a way to increase the overall speed of a webpage
  - Images must be optimized and highly compressed
  - Choose the same format for one type of images
  - Keep the same size for the same collection images
  - Image file formats:
    - JPG: photographic images with quality 65
    - GIF: animated images
    - PNG 8: images with simple colors
    - PNG 24: images with alpha transparency

CSS

- KITs
  - Prepare a modification kits for special text fonts, images, banner, frame and background
  - Keep the guides, layers, effects and unmerged texts in PSDs

CSS

- CSS Prints
  - Only show the necessary text

CSS

- CSS Sprites
  - A sprite combines multiple images into a one large image and using CSS background-position to only display parts of it
  - This is a technique for making webpage faster because it reduces the number of HTTP requests in the page
  - Do sprite for all backgrounds, icons, bullets, buttons, special text fonts, custom form elements, frames, boxes, layers, etc
  - Do not sprite for logos, particular images and photos
  - Do not make sprites too large to avoid memory usage problem

CSS

- CSS Sprites
  - Sprite Images:
    - Background: **wi-bgd-x, wi-bgd-y, wi-grd-x, wi-grd-y**
      - **wi-bgd-x:** background repeat x
      - **wi-bgd-y:** background repeat y
      - **wi-grd-x:** background gradient repeat x
      - **wi-grd-y:** background gradient repeat y
    - General: **wi-general**
      - **wi-general:** background fixed width and height
    - Text: **wi-text**
      - **wi-text:** special text fonts
      - use **wi-text-n** class selector for position
    - Icon: **wi-icon**
      - **wi-icon:** icons
      - use **wi-icon-n** class selector for position

CSS

- CSS Sprites
  - Sprite Images:
    - Button: **wi-button**
      - **wi-button:** button fixed width and height
      - use **wi-button-n** class selector for position
      - can inject into **wi-form**
    - Form: **wi-form**
      - **wi-form:** form elements fixed width and height, checkbox icons
      - can inject into **wi-general**
    - Form elements: **wi-form-elements**
      - **wi-form-elements:** form elements fluid-width, cut 3 pieces

CSS

- CSS Sprites
  - Sprite Images:
    - Frame: **wi-frame**
      - can inject into **wi-general**
      - use **wi-frame-n** class selector for position
    - Corner: **wi-corner**
      - **wi-corner:** cut 4 corners of box
    - Others: **wi-nav, wi-slider, wi-rating, wi-tab, wi-number**
    - Background repeat two ways, background gradient filled color, background shadow: cut separate background images
    - Use **wi-?** for normal, **wi-?-x** for repeat x and **wi-?-y** for repeat y

CSS

- CSS Specificity
  - Specificity determines which CSS rule is applied by the browsers
  - If two selectors apply to the same element, the one with higher specificity wins
  - Using !important overrides all specificity no matter how high it is. Avoid using it if possible
  - Understand cascading and selector specificity so you can write very terse and effective code

CSS

- CSS Specificity
  - Every selector has its place in the specificity hierarchy. There are four distinct categories which define the specificity level of a given selector:
    - Inline styles
      - <p style="padding: 5px;">example</p>
    - IDs
      - #content
    - Classes, attributes and pseudo-classes
      - .classes, [attributes], :hover, :focus, :active, :link, :visited, :lang, :first-child, :last-child, :nth-child, :nth-last-child, :only-child, :only-of-type, :empty, :target, :root, :not, :enabled, :disabled, :checked
    - Elements and pseudo-elements
      - p, :before, :after, :first-line, :first-letter, ::selection

CSS

- CSS Specificity
  - Memorize how to measure specificity:
    - Start at 0
    - Add 1000 for style attribute
    - Add 100 for each ID
    - Add 10 for each attribute, class or pseudo-class
    - Add 1 for each element name or pseudo-element

CSS

- CSS Specificity
  - Specificity Examples

* { }	0 (universal and inherited selectors)
li { }	1 (one element)
ul li { }	2 (two elements)
p:first-line { }	2 (one element, one pseudo-element)
p[title] { }	11 (one attribute, one element)
ul li.level { }	12 (one class, two elements)
li.level.odd { }	21 (two classes, one element)
style=""	1000 (one inline)
.level	10 (one class)
#sidebar	100 (one id)
body #sidebar .box p { }	112 (two elements, one id, one class)

CSS

- CSS Values
  - Colors
    - All color values are written in the hexadecimal format and using lowercase
  - Units
    - Use pixel (px) unit – corresponds to actual pixels on the screen – for margin, padding
    - Use em (em) unit – corresponds to the specified point size of the font – for text
    - Use percentage (%) unit for fluid width/height content
  - Fonts
    - Always specify a fallback generic font
    - Font names with spaces must surrounded by double-quotes

CSS

- CSS Shorthand
  - CSS shorthand is preferred because of its terseness
  - Follow the TRBL acronym
  - Common shorthand properties:
    - margin
    - padding
    - font
    - background
    - border
    - border-radius
    - transition
    - transform
    - list-style

CSS

- CSS Box Model
  - Margin
  - Border
  - Padding
  - Content

CSS

- Rules
  - Follow the CSS files in template structure
  - Use a reset CSS file to avoid browser inconsistencies
  - Use a minimal number of style sheets
  - Properties should be listed in group similar:
    - Display & Flow (display, visibility, float, clear)
    - Positioning & Floats (position, top, right, bottom, left, z-index)
    - Dimensions (width, *-width, height, *-height, overflow)
    - Margins & Paddings (margin, padding)
    - Borders & Outline (border, outline)
    - Background (background)
    - Typography (font-*, line-height, text-*, *-spacing, white-space, vertical-align, color, list-style)
    - Opacity & Cursors (opacity, cursor)

CSS

- Rules
  - Avoid using **!important** if possible
  - Use the **link** tag to include, never use **@import**
  - Quote attribute values in selectors, e.g. input[type="text"]
  - Avoid specifying units for zero values
  - Avoid using style inline in the HTML file
  - Avoid using single line CSS because it can cause issues with version control
  - Single declarations on one line
  - Multiple declarations, one per line
  - Elements occur only once inside a document should use ID selector, otherwise, use CLASS selector
  - Prefer CLASS selector than ID selector
  - Write selectors that are optimized for speed
  - For mobile version, should cut as separate images, do not use sprite images

JAVASCRIPT

BEHAVIOR

JAVASCRIPT

- JavaScript is the scripting language of the web
  - JavaScript was designed to add interactivity to HTML pages

- File Naming
  - JavaScript programs should be stored in and delivered as **file-name.js** files
  - Avoid naming a file with a period "."

- Indentation
  - Two spaces should be used as the unit of indentation
  - Avoid lines longer than 80 characters
  - When an expression will not fit on a single line, break it according to these general principles:
    - Break after a comma
    - Break before an operator
    - Align the new line with the beginning of the expression at the same level on the previous line

JAVASCRIPT

- Indentation
  - When an expression will not fit on a single line, break it according to these general principles:
    - Examples

```
var result = 0,
    longName = ‘’;

var result = longExp1 + longExp2
             + longExp3;

var result = (longExp) ? longResult1
                       : longResult2;

if((longCondition1 && longCondition2)
  || longCondition3){
}
```

JAVASCRIPT

- Naming Conventions
  - Identifiers (variables, methods) can contain only the characters [a-z0-9]
  - Capitalization:
    - functionNamesLikeThis
    - methodNamesLikeThis
    - variableNamesLikeThis
    - EnumNamesLikeThis
    - ClassNamesLikeThis
    - CONSTANTS_LIKE_THIS
  - Be descriptive
  - All code in any code-base should look like a single person typed it, no matter how many people contributed.

JAVASCRIPT

- Comments
  - Be generous with comments. It is useful to leave information that will be read at a later time by people (possibly yourself) who will need to understand what you have done
  - Make comments meaningful. Focus on what is not immediately visible
  - Comments should not be enclosed in large boxes drawn with asterisks or other characters
  - Comments should never include special characters such as form-feed and backspace
  - Multiline comments should be

    ```
    /*
    comment here
    */
    ```
  - Inline comments should be

    ```
    // comment here
    ```

JAVASCRIPT

- Declarations
  - One declaration per line is recommended since it encourages commenting

```
var result = 0, // comment here
       longName1 = '',
       longName2 = '', longName3 = '';
```

  - Do not put different types on the same line
  - Try to initialize local variables where they're declared
  - No space between a method name and the parenthesis "()"

```
function methodName(){
  // to do
}
```

JAVASCRIPT

- Declarations
  - Open brace “{“ appears at the end of the same line as the declaration statement

```
function methodName(){
  // do to
}
```

  - Closing brace “}” starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the “}” should appear immediately after the “{“

```
function methodName(){
  // to do
}

function empty(){}
```

  - Methods are separated by a blank line

JAVASCRIPT

- Declarations
  - JavaScript does not have block scope, so defining variables in blocks can confuse programmers who are experienced with other C family languages. Put declarations only at the beginning of blocks
  - Use of global variables should be minimized

JAVASCRIPT

- Statements
  - Simple Statements:
    - Each line should contain at most one statement

  - Compound Statements: Compound statements are statements that contain lists of statements enclosed in “{ }” (curly braces):
    - The “{“ should be at the end of the line that begins the compound statement
    - The “}” should begin a line and be indented to align with the beginning of the line containing the matching “{“

JAVASCRIPT

- Statements
  - Conditional statements
    - if statement

```
if(condition1){
  // to do if condition1 is true
}
else if(condition2){
  // to do if condition2 is true
}
else{
  // to do if neither condition1 nor condition2 is true
}
```

JAVASCRIPT

- Statements
  - Conditional statements
    - switch statement

```
switch(expression){
  case expression1:
    // to do 1
    break;
  case expression2:
    // to do 2
    break;
  default:
    // to do if expression is different from expression1 and expression2
}
```

JAVASCRIPT

- Statements
  - Loop statements
    - for statement

```
for(initialization; condition; update){
  // to do
}

for(variable in object){
  // to do
}
```

JAVASCRIPT

- Statements
  - Loop statements
    - while statement

```
while(condition){
  // to do
}
```

    - do statement

```
do{
  // to do
}
while(condition);
```

JAVASCRIPT

- Statements
  - Loop statements
    - try…catch statement

```
try{
  // to do
}
catch(variable){
}
finally{
  // to do
}
```

JAVASCRIPT

- White Space
  - Blank spaces should be used in the following circumstances:
    - A blank space should not be used between a function value and its "("
    - All binary operators except "." (period), "(" (left parenthesis) and "[" (left bracket) should be separated from their operands by a space

```
var result = longExp1 + longExp2;
var result = (longExp1 + longExp2);
var result = jsonData['result'];
```

    - No space should separate a unary operator and its operand except when the operator is a word such as typeof

```
var x = 0;
x++
--x;
var y = -x;
typeof x;
```

JAVASCRIPT

- White Space
  - Blank spaces should be used in the following circumstances:
    - Each ";" (semicolon) in the control part of a for statement should be followed with a space

```
for(var i = 0, j = 10; i < j; i++){
  // to do
}
```

    - Whitespace should follow every "," (comma)

```
var x = 10, y = 15;

function methodName(param1, param2, param3){
  // to do
}
```

JAVASCRIPT

- Rules
  - Variable declarations must start with **var** keyword
  - Variables and functions should be declared before used
  - Constants or configuration variables should be at the top of the file
  - JS expressions must end with a semi-colon
  - Don't rely on the user-agent string. Do proper feature detection
  - Don't use document.write function
  - Avoid using inline script in the HTML file
  - Avoid using eval function
  - All Boolean variables should start with “is”, “has”
  - Create functions which can be generalized, take parameters, and return values
  - Do not send too many function parameters
  - Minimizing repaints & reflows

JAVASCRIPT

- Rules
  - Do not compare x == true, use (x) instead
  - Use [value1, value2] to create an array
  - Use {member: value} to create an object
  - Comment your code! It helps reduce time spent troubleshooting JavaScript functions
  - End of file with a newline
  - Avoid:
    - Too much happening in a loop
    - Too much happening in a function
    - Too much recursion
    - Too much DOM interaction

JAVASCRIPT

- Performance Tips
  - Define local variables
    - Don’t

```
var user = document.getElementById(‘user’),
    pass = document.getElementById(‘pass’);
```

    - Do

```
var doc = document,
    user = doc.getElementById(‘user’),
    pass = doc.getElementById(‘pass’);
```

JAVASCRIPT

- Performance Tips
  - Avoid using eval or the Function constructor
    - Don't

```
function addMethod(object, property, code) {
    object[property] = new Function(code);
}
addMethod(myObj, 'methodName', 'this.localVar = 1');
```

    - Do

```
function addMethod(object, property, fn) {
    object[property] = fn;
}
addMethod(myObj, 'methodName', function () {
    this.localVar = 1;
});
```

JAVASCRIPT

- Performance Tips
  - Don't use try-catch-finally inside loop statements
    - Don’t

```
for(var i = 0; i < 10; i++){
  try{
    // to do
  }
  catch(){}
}
```

    - Do

```
try{
  for(var i = 0; i < 10; i++){
    // to do
  }
}
catch(){}
```

JAVASCRIPT

- Performance Tips
  - Pass functions, not strings to setTimeout and setInterval
    - Don't

```
setTimeout('doSomething()', 100);
setInterval('doSomething()', 100);
```

    - Do

```
setTimeout(doSomething, 100);
setInterval(doSomething, 100);
```

JAVASCRIPT

- Performance Tips
  - Better loop
    - Don't

```
for(var i = 0; i < results.length; i++){
  // to do
}
```

    - Do

```
for(var i = 0, len = results.length; i < len; i++){
  // to do
}
```

JAVASCRIPT

- Performance Tips
  - Better conditional
    - Don’t

```
if(type == ‘js’ | | type == ‘css’){
  // to do
}
```

    - Do

```
if(/^(js|css)$/.test(type)){
  // to do
}

if(({css: 1, js: 1})[type]){
  // to do
}
```

JAVASCRIPT

- Performance Tips
  - Chaining
    - Don't

```
$('#notification').fadeIn('slow');
$('#notification').addClass('active');
$('#notification').css('marginLeft', '50px');
```

    - Do

```
$('# notification')
  .fadeIn('slow')
  .addClass('active')
  .css('marginLeft', '50px');
```

JAVASCRIPT

- Performance Tips
  - Use jQuery.data method to store data
    - Don’t

```
var el = $(‘#element’)[0];
el.user = ‘user’;
el.pass = ‘pass’;
```

    - Do

```
$(‘#element’).data(‘login’, {
  user: ‘user’,
  pass: ‘pass’
});
```

JAVASCRIPT

- Performance Tips
  - Use CSS class to change style
    - Don't

```
$('#element').css({
  color: 'white',
  background: 'red'
});
```

    - Do

```
<style>
  .alert {color: white; background: red;}
</style>

<script>
  $('#element').addClass('alert');
</script>
```

JAVASCRIPT

- Performance Tips
  - https://developer.mozilla.org/en-US/docs/Developer_Guide/Coding_Style
  - https://github.com/rwaldron/idiomatic.js
  - http://www.jslint.com/lint.html
  - http://jshint.com/docs/options
  - http://coding.smashingmagazine.com/2012/11/05/writing-fast-memory-efficient-javascript
  - http://www.codeproject.com/Tips/623082/JavaScript-Performance-Tips
  - http://moduscreate.com/efficient-dom-and-css
  - http://addyosmani.com/jqprovenperformance
  - http://tutorialzine.com/2011/06/15-powerful-jquery-tips-and-tricks-for-developers
  - http://learn.jquery.com/performance/optimize-selectors
  - http://jonraasch.com/blog/10-advanced-jquery-performance-tuning-tips-from-paul-irish

BUILD TOOLS

GRUNT, JADE, LESS

BUILD TOOLS

- Grunt
  - JavaScript task runner
  - Reference @ http://gruntjs.com

- Jade
  - Node template engine
  - Reference @ http://jade-lang.com

- Less
  - CSS pre-processor
  - Reference @ http://lesscss.org

References

- W3C:
  - http://www.w3.org/TR/WCAG10-CORE-TECHS
  - http://www.w3.org/TR/WCAG10-HTML-TECHS
  - http://www.w3.org/TR/WCAG10-CSS-TECHS
  - http://www.w3.org/TR/WAI-WEBCONTENT-TECHS
  - http://www.w3.org/TR/UAAG10
  - http://www.w3.org/TR/1999/REC-html401-19991224
  - http://www.w3.org/TR/1998/REC-CSS2-19980512
  - http://www.w3.org/TR/CSS21
  - http://www.w3.org/TR/CSS1
  - http://www.whatwg.org/specs/web-apps/current-work/multipage/index.html
  - http://www.w3.org/QA/Tools
  - http://www.w3.org/wiki/HTML_structural_elements
  - http://www.w3.org/TR/WCAG10/full-checklist.html
  - https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

References

- Web Accessibility tool:
  - http://wave.webaim.org/report
  - http://achecker.ca/checker/index.php

References

- HTML:
  - http://www.w3schools.com/html/default.asp
  - http://reference.sitepoint.com/html
  - http://www.htmlhelp.com
  - http://www.htmldog.com
  - https://developer.mozilla.org/en/HTML
  - http://microformats.org/wiki/hcard
  - http://microformats.org/code/hcard/creator
  - http://www.w3.org/TR/microdata
  - http://schema.org

References

- CSS:
  - http://www.w3schools.com/css/default.asp
  - http://reference.sitepoint.com/css
  - http://www.cssbasics.com
  - https://developer.mozilla.org/en/CSS
  - https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Writing_efficient_CSS
  - http://websitetips.com/articles/css/sprites
  - http://www.impressivewebs.com/difference-block-inline-css
  - http://blog.themeforest.net/tutorials/vertical-centering-with-css
  - http://lesliefranke.com/files/reference/csscheatsheet.html
  - http://alistapart.com/article/responsive-web-design
  - http://hicksdesign.co.uk/boxmodel
  - http://dustindiaz.com/css-shorthand
  - http://lesscss.org
  - http://sass-lang.com

References

- JavaScript:
  - http://www.w3schools.com/js/default.asp
  - https://developer.mozilla.org/en/JavaScript
  - http://reference.sitepoint.com/javascript
  - http://jquery.com
  - http://dev.opera.com/articles/view/efficient-javascript
  - http://developer.yahoo.com/performance/rules.html
  - http://contribute.jquery.org/style-guide/js
  - http://coffeescript.org
  - http://nodejs.org

References

- HTML5 / CSS3:
  - http://html5.org
  - http://diveintohtml5.info/canvas.html
  - http://introducinghtml5.com
  - http://html5readiness.com
  - http://slides.html5rocks.com
  - http://html5demos.com
  - http://html5doctor.com
  - http://jade-lang.com
  - http://www.initializr.com
  - http://html5boilerplate.com

References

- Newsletter:
  - http://www.sitepoint.com/code-html-email-newsletters
  - http://kb.mailchimp.com/article/how-to-code-html-emails
  - http://www.campaignmonitor.com/resources/will-it-work/email-clients
  - http://www.campaignmonitor.com/css
  - http://www.greatsites.com.au/html_specifications

References

- Resource:
  - http://quirksmode.org
  - http://www.smashingmagazine.com
  - http://ajaxian.com
  - http://mashable.com
  - http://www.alistapart.com
  - http://jsbeautifier.org
  - http://dean.edwards.name/packer
  - http://www.html5canvastutorials.com
  - http://html2jade.aaron-powell.com
  - http://isobar-idev.github.io/code-standards
  - http://codeguide.co
  - https://developer.mozilla.org/en-US/docs/Web/Tutorials
  - http://addyosmani.com/resources/essentialjsdesignpatterns/book

SUTRIX MEDIA

SUTRIX MEDIA
15th Floor, Blue Sky Tower, 1 Bach Dang Street,
Ward 2, Tan Binh Dist., Ho Chi Minh City, Vietnam
+848 3997 5901
inquiry@sutrixmedia.com