04/14/2016

Lecture 1 : HTML Introduction.

- HTML (Hypertext Markup Language) ⇒ Structure
  i.e., components that the html document has.
  NOT tell anything how they are visually layed out.

CSS ⇒ Style

Javascript ⇒ Behavior, functionality.

- History of HTML

1997 —— 2000 —— 2004 —— 2007 —— 2011

HTML 4.0 & 4.01 ↓ by www consortium

XHTML 1.0 ↙

WHATWG ↑ by Browser vendor

W3C & WHATWG work together

HTML5

(W3C ~ Standard HTML5
| WHATWG ~ evolving HTML

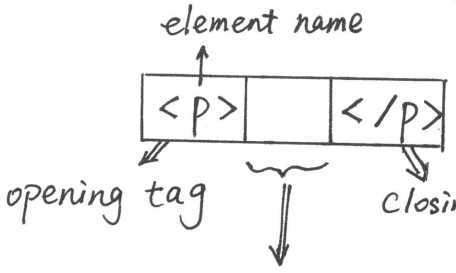
Track changes and what browser supports what :

www.w3.org/TR/html5/ , caniuse.com ,

validator.w3.org/# check a page can work well in browsers.

www.w3schools.com/browsers/browsers_stats.asp
see how the browsers share the market.

- Anatomy of an HTML Tag



exceptions :
 // Line break
<hr> // horizontal rule
they only have opening tag.

Content : Attribute name + Attribute value

must have space ↑ no space allowed / no attributes.

  is valid in HTML5.

no space allowed

values in quotes (~~sig~~ single or double)
Alternate single / double quotes when nested.

■ Basic HTML Document Structure.

- Declare. If not added, it means that you are not compliant to the HTML standard. And it may create problems. "Quirks Mode"
- "meta" is a stand-alone tag. Here it's always good to specify charset.
- Nested tags have to be closed before their parent tags.
- The intepreter always render the code sequentially (top to bottem)
- <title> appears on the tags of a webpage, <body> is shown in the main content.

no space can be lower case or upper case

```
<!doctype_html>
<html>          can have one or more spaces before/after "html".
<head>
    <meta charset="utf-8">      convey info to the browser.
    <title> Coursera </title>
</head>
<body>
    I'm learning so much.
</body>
</html>
```

■ HTML Content Models.

↓

nesting rules.

- Tradition 2 Models:

* Block-Level Elements

- Rendor to begin on a new line (by default)
- May contain inline or other block-level elements.

* Inline Elements

- Rendor on the same line (by default)
- May contain only other inline elements.

- HTML5 have more complex set of content categories.

Roughly { Block-level ⇒ flow content
| Inline ⇒ Phrasing content

Details in www.w3.org/TR/html5/dom.html#kinds-of-content

Unless defined, block-level element will take up as much horizontal space as allowed

.. For example :

```
<body>
Line1   <div> ** DIV1 </div>
Line2   <div> ** DIV2 : Follow DIV1 ** </div>
Line3   <span>** SPAN1 : Inline follow DIV2 ** </span>
Line4   <div>
            ** DIV3 : Follow SPAN1
            <span> ** SPAN2 : Inline inside DIV3. ** </span>
            Continue content of DIV3 **
        </div>
</body>
```

Output format :

```
Line 1 :  ** DIV1
Line 2 :  ** DIV2 : Follow DIV1 **
Line 3 :  ** SPAN1 : Inline follow DIV2 **
Line 4 :  ** DIV3 : Follow SPAN1 ** SPAN2 : Inline inside DIV3 ** Con
          tinue content of DIV3 **
```

Although Span is an inline element, but DIV is always rendered to a new line.

Although SPAN2 is an individual line in the code, but it will be rendered on the same line with other contents.

i.e., * whether \n is added to the code won't change the rendering of the elements.

Inline

* ✓ :
* X :

block-level

* : All spaces, enters are rendered as a single space.

04/15/2016

■ Heading Elements – semantic elements.

- sematic HTML element : implies some meaning to the content.

  i.e., tell sth. about the content (importants. comments, etc.) to humans and machines, rather than just a tag or division.

  May help SEO (search engine ranking)

- Headings

  e.g., <h1>, <h2>, ..., <hn>

MOST important	LEAST important.

- NEW HTML5 sematic elements (block-level) :

  1. <nav> : contents that used ~~to~~ for navigation. i.e. some links.

     usually nested in <header>

  2. <header> : header info about the page.

     e.g., company logos, navigations, etc.

  3. <section> : for categorizing contents.

     usually <article> is nested in it.

  4. <aside> : not directly related to the main content, but some other info related.

  5. <footer>

  These provide no more functionality, but for clarity.

■ Lists.

- All the list elements are nested in
- each element is inside
- can be nested in another <ul> or
- For ordered list, change <ul> into

Lists are very often used for structuring navigation portions.

For example:

Attribute:
<ol start="3" reversed>
<ol type="A">

```
<div>
Oreo cookie eating procedure:
<ol>
    <li> Open box </li>
    <li> Take out cookies </li>
    <li> Make a double oreo
        <ul>
            <li> peel off the top part </li>
            <li> place another in the middle </li>
            <li> put back the top part </li>
        </ul>
    </li>
    <li> Enjoy! </li>
</ol>
</div>
```

Output:

Oreo cookie eating procedure

1. Open box
2. Take out cookies
3. Make a doube Oreo
    - ...
    - ...
    - ...
4. Enjoy!

■ HTML Character Entity References

- To avoid ~~interpret~~ character being interpretted, but to treat them as regular content, remember to:

  Avoid: < > &

  Use: < > &

- Other commonly use ones:

  © --> ©

  space that avoids wrapping -->   similarly with mbox[ ]

  Note: not use it for extra spaces. Instead, put a around related contents and set margins.

  quotes --> "

  especially for some character sets that don't support " ".

■ Creating Links: .....

1. Internal Links: if address only contains a file name, it assumes that the page is in the same directory.
2. External Links: put full "http://..." as address.

   target="_blank" attribute is always added in the <a ...> tag, in order to open the link in a new page.

3. Same page but other position: (put as an attribute.)
   - label with an identifier: id="ID", or name="ID" in heading.
   - reference the position: #ID as the address.

   ↳ display: block / inline ↰

Note: <a> tag is both a block-level element and an inline element. ~~#~~ Take advantage of the former, to put in an image, a block, etc that the whole area is linkable.

■ Display Images:

Tag: <img attributes>, inline element

Attributes: src = " address " ; can be local or from webs.

width = " ", height = " " : always include them to insure proper layout. Otherwise, if the image takes time or fails to load, the space will ~~be~~ NOT be researved for the image.

alt = " alternate name "

*

Other formatting text:

1. , Italic.
2. , Bold.
3. <big></big>. <small></small>. not part of Latest HTML statement
4. <mark></mark>
5. <slub>, <sup> 下标,上标. MathML to type maths.

* Audio:

Tag: <audio src=" " (autoplay ><~~/audio~~>
controls

loop: repeating

* Video:

Attribute: src = " "

autoplay / controls.

loop

alt = " "

* Handle older ~~tags~~ browsers with the additional .

<new_tag>

 Sorry! Your browser does not support the new_tag tag

</new_tag>

04/18/2016

Lecture 2 CSS

■ Anatomy of a CSS Rule

```
selector ← p {        property ↑   value ↑
    color : blue ;
}
```

(can be neglected)

declarations

browsers have their own default css rules.

style sheet : a collection of css rules.

■ Element, Class and ID Selectors

1. Element selector : the name of the elements. Nothing to change on html.

   e.g., p, div, etc.

2. Class selector : declare + use :

```
.classname {
```

define with dot.

no space }

```
<p class="classname"> ... </p>
<p> ... </p>
<div class="classname"> ... </div>
```

3. id selector : can be only used in HTML ONCE.

```
#idname {
```

define with #

no space }

```
<p> ... </p>
<div id="idname" > ... </div>
```

Grouping Selectors : define selectors that have the same style.

```
div, .classname {
```

seperate by comma

}

```
<p class="classname"> ... </p>
<p> ... </p>
<div> ... </div>
```

same style as defined.

■ Combining Selectors.

1. Element with class selector.

```
p.classname {
```

no space

}

```
<p class="classname"> ... </p>
<p> ... </p>
<div class="classname"> ... </div>
```

effected by p.classname.

2. Child selector : only apply to DIRECT child

```
article > p { ... }
```

direct child

```
<article>   affected.
    <p>...</p>
</article>
<p>...</p>
<article>
    <div><p>...</p></div>
</article>
```

3. Descendant Selector : apply to all children.

```
article p { ... }
```

any level inside article.

```
<article>
    <p>...</p>
</article>
<p>...</p>
<article>          affected
    <div><p>...</p></div>
</article>
```

Note: child & descendant selectors can be used for class and id selectors as well.

4. Adjacent sibling selector

selector + selector

5. General sibling selector

selector ~ selector

These combining patterns can be combined.

e.g., div.classname element , div > div > p.

- * : all selectors (universal selector)
  often used to define styles that will be applied to all the contents, especially to overwrite browser-defined styles (margins, padding, etc.)

■ Pseudo-class selectors : ① can't be targeted with combinations of regular selectors
② based on users' actions.

*. Declaration : selector : pseudo-class selector name.

1. : link & : vistited

for element ... : a: link, a : visited.

The display style before / after the link is ~~clicked~~ visited.

2. : hover & : active

for element <a>, etc.

hover : when the mouse is hovering on an element.

active : the user clicked but not yet released the mouse.

3. : nth-child (k)

for the n^{th} child inside the selector.

k can be a number, or can be "even"/"odd".

They can be used combinedly. e.g., header li : nth-child (4) : hover

Tip : make sure the selectors are readable so that you'll understand what your targets are after a long time.

Example :

```
<style>
    a:hover div:nth-child(2) {
        font-size : 24px;
    }
</style>
<a>
    <div> Div 1 </div>
    <div> Div 2 </div>
    <div> Div 3 </div>
</a>
```

Div 1
Div 2 → when the user hovers the mouse over ANY area inside <a>
Div 3

↓

Div 1
Div 2
Div 3

CSS Rules Conflict Resolutions and Text Styling.

- Style placement

1. Inline : `<p style="declarations;">...</p>`
   NOT recommended. The least reusable one.

2. Tag : `<style>` css rules `</style>` inside `<head></head>`

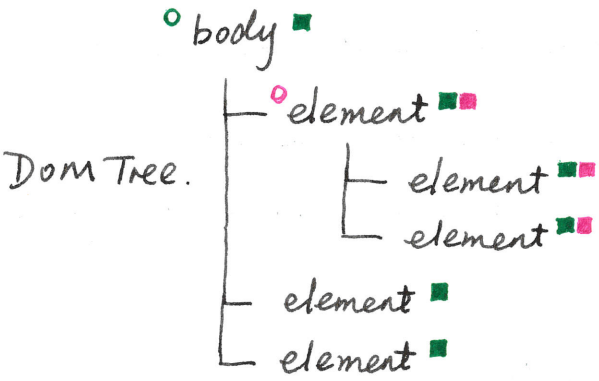
3. External stylesheets.
   Recommended.

- Conflict resolution : cascading algorithms.

1. Some concepts :

   - Origin precedence : ① Last declararation wins, when conflict.
     remembered that HTML renders top to bottom.
     ② Merge declaration, when no conflict.

   - Inheritance :
     all children inherit the properties of the parents.



DOM Tree.

   - Specificity :
     Most specific selector combination wins when conflicts.

style="" → ID → Class / pseudo-class / attribute → # of Elements

e.g., `header.navigation p { color: blue; }`
`p.blurb { color: red; }`

```
<header class="navigation">
    <p class="blurb">
        texts will be blue.
    </p>
</header>
```

■ Styling text

- font-family: ..., ..., ... * refer to "www.w3schools.com/cssref/css_web safe_fonts.asp"

  * generally Arial, sans-serif, Helvetica, ... are quite safe. It relies on the user's installed fonts.

- color: #0000ff * based on RGB, each assigned two digits

  * There're some specific color names available.

- font-style: italic / normal / obaque, ...
- font-weight: { bold (default 900) / 100~900
- font-size: { 24px; Absolute units of measurements. / 120%; .2em. Details below. Relative size.
- text-transform: capitalized / uppercase / lowercase;
- text-align: center / right / left.

■ Relative font size:

1. font-size: 120%;
2. font-size: 2em; # 2 times larger = 200%

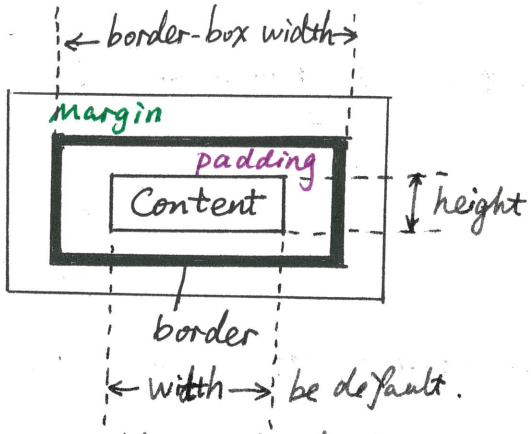
   # cummulative effect on previous font-size declaration, not overwrite.

   0.5em: # reduced by 2.

   Good thing about it is that the relative font sizes of different tags/elements remain the same, even if the user zooms in/out the page.

3. width: 120%, etc.
4. font-size: 5vw; # 5% of the ~~view~~ the screen width (viewport width)

The Box Model and Layouts.



■ The Box Model.

- padding : top right bottom left ;
  (default : all)

- For CSS3 :

box-sizing : border-box (content + padding + border)
content-box (default)

note : box-sizing is not inherited. So put it in *{ } selector.

■ • Cumulative margins v.s. collapsing margins :

* horizontal boxed : total_margin = left + right.
* vertical aligned : largest margin wins.

- height : if it's set to a number that is not enough to hold all the content, the contents will overflow.
  * overflow : hidden # hide the overflowed contents.
  * overflow : auto # add a scroll bar if overflows.
  * overflow : scroll # even the contents do not overflow, the scroll bar appears.

■ The background property.

1. Using an image as the background.
   - background-image : url("pwd");
   - background-repeat : no-repeat ; # default : repeat
   - background-position

↳ background : property1 property2 property3 ...

Note that any "-subproperty" surrounders to "background :" settings.

■ Positioning Elements by Floating.

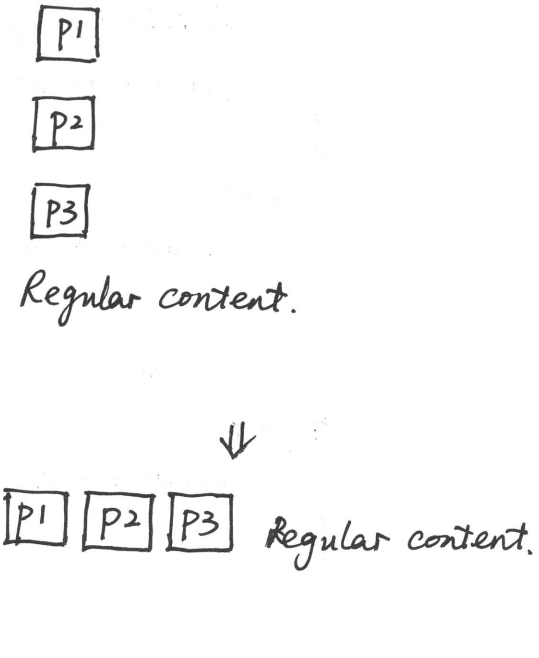
1. "float:" attribute.

- The browser takes it out of the regular document flow.
- Margins never collapse.

e.g.,
```
<div>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <section> Regular
        content.</section>
</div>
```

⇓

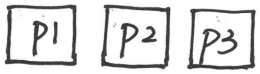
```
p {
    float: left;
}
```



2. "clear:" attribute.

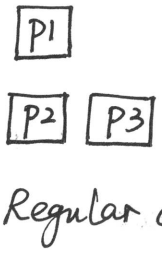
- Tell the browser to treat the non-floated elements as nothing being floated to its left or right, etc.

e.g.,
```
section {
    clear: left;
}
```



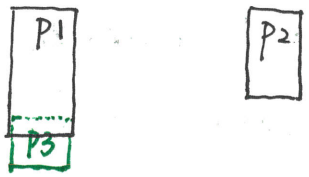
- Or resume the regular flow of the floated elements, and then let them float again

e.g.,
```
# p2 {
    clear: left;
}
```



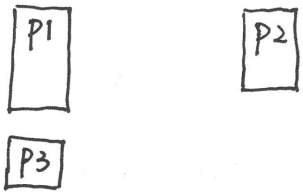
- "clear : both"

e.g.,
```
#p1 {
    height : 150px;
    float : left;
}
#p2 {
    height : 100px;
    float : right;
}
#p3 {
    clear : right;
}
```



⇓

```
#p3 {
    clear : both;
}
```



3. Two-column positioning : `{ width : 50%; float : left; }`

Remember to set `* { box-size : border-box }`

■ Relative and Absolute Element Positioning.

1. Concepts about " position : relative".

- Static positioning: normal document flow

  default. position offsets are ignored.

- relative positioning: element is positioned relative to its position in normal document flow.

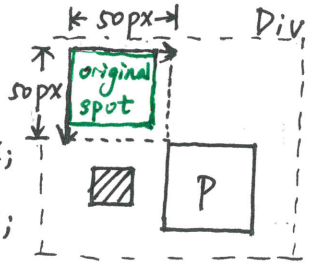
Note: * Positioning CSS properties: top, bottom, left, right.

* Element is not taken out of the normal document flow.

Its original spot is preserved even if removed visually.

e.g., relative positioning

```
p {
    position: relative;
    top: 50 px;   # = bottom: -50px;
    left: 50px;   # = right: -50px;
}
```



It means to move p FROM top & left by 50px.

The original spot is perserved. Often used for containers.

2. "position: absolute"

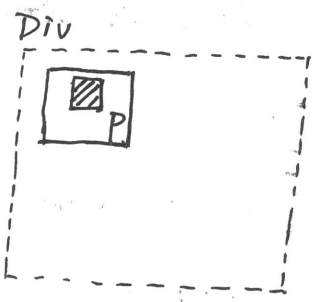
element is positioned relative to the position of the nearest ancestor which has positioning set on it.

Note: * By default, <html> is the only element that has non-static positioning set. Others are set to static.

* Element is taken out of the normal document flow.

e.g.,
     ▨
     P

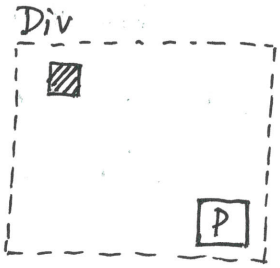
```
p {
    position: absolute;
}
```



The original spot for P is not preserved, so ▨ jumped up.

⇓

```
p {
    position: absolute;
    bottom: 10px;
    right: 10px;
}
```



⇓

```
div {   container
    position: relative;
    top: 50px;
    left: 50px;
}
```

⇓

The same. If container element is offset, everything inside is offset withit.

Introduction to Responsive Design

■ Media Queries.

To group styles together and target them to devices based on some criteria.

1. Syntex:

media feature

```
@media [True] { ... }
```

where the CSS styles will be applied if the condition [ ] is true.

2. Some common features:
   - (max-width : 800px) , (min-width : 800px)
     commonly used to target devices.
   - (orientation : portrait)
   - sreen , print, etc.

3. Media features can be combined with Logical Operators.
   - "and"
     e.g., @media (min-width : ~~78~~768px) and (max-width : 991px)
   - " , " = OR.

4. Common Approach: base styles + parallel media queries. to add/change.

e.g.,
```
p { color: blue }   /* base style */
...
@media ( min-width : 1200px ) { }
@media ( min-width : 992px ) and (max-width : 1199px)
{ }   /* add new feature */
```

Not to overlap boundaries.

The boundary is called "break point".

■ Responsive Design.

To adapt the page's layout to the viewing environment.

Alternative: Let the server detect user-agent (browser), and then serves up either a mobile version or a desktop one.
(But mobile versions are too various; not as popular).

1. 12-Column Grid Responsive Layout (proportion-based grids)

↳ factors: 1, 2, 3, 4, 6. (100%, 50%, 33.3%, ...). Each grid = 8.33%

Grids can be nested.

Realized by define width:

```
.col-1 { width: 8.33%; }
.col-2 { width: 16.66%; }
.col-3 { width: 25%; }
   ⋮
.col-12 { width: 100% }
```

Usually they're defined by how many grids they occupied.

2. Combine with CSS3 media queries

e.g.,

```
p { ... } /* base styles */
@media (min-width: 1200px) {
    .col-lg-1 { width: 8.33% }, ..., .col-lg-12 { width: 100% }
    + base styles { float: left; ... }
}
@media (min-width: 992px) and (max-width: 1199px) {
    .col-md-1 { width: 8.33% }, ...
    all { float: left; ... }
}
   ⋮
.row { width: 100% }
```

```
<div class="row">
    <div class="col-lg-3 col-md-6"><p>...</p></div>
       ⋮
</div>
```

Use 4-column in large screen & 2-column in small screen

3. fluid, flexible images, etc.

Note:

* Browsers on mobile devices usually zoom out the web pages, and make the width of the screen much larger than reality. In order to tell these browsers to stop auto-zoom, we can add meta tag:

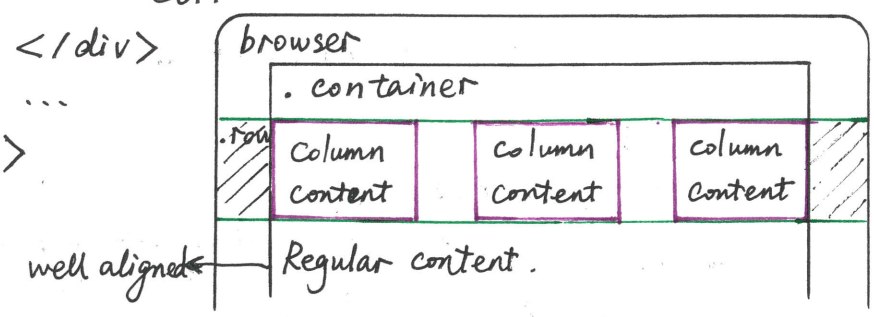
```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Introduction to Twitter Bootstrap.

- Twitter Bootstrap.
  - Bootstrap is the most popular framework for responsive, mobile-first projects. Mostly CSS classes.
  - Disadvantages: too big, but selectively download.

- The Bootstrap Grid System.

```
→ <div class="container">   /* must inside a container */
                            (fixed breakpoints)
                            or "container-fluid": full width with consistent paddings
    → <div class="row">     /* create horizontal groups of columns */
                            /* negative margin for compensation */
      → <div class="col-md-4">
          Col1
        </div>
        ...
    </div>
</div>
```



- col - SIZE - SPAN → # of columns than an element should span
  - * identify screen width rage.
  - * If the screen size is outside of that range, i.e., below that width, the columns will collapse and stack vertically, if no other rule is specified
- go to "getbootstrap.com/css/#grid" for details.
- specify "col-xs-SPAN" to force the horizontal layout of columns

- Void Element
  - * <meta>
  - * Handling multimedia : <img>
  - * Handling forms : <input>
  - * Handling breaks :
<wbr><hr>
    -
 = /n
    - <wbr> : auto break / wrap.
    - <hr> : horizontal rule.

- Table :

```
<table>
    <thead>
        <tr> <th>...</th> <th>...</th> </tr>
    </thead>
    <tbody>
        <tr> <td>...</td><td>...</td></tr>
        <tr> <td>...</td><td>...</td></tr>
    </tbody>
</table>
```

■ HTML Forms

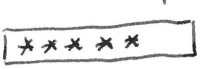
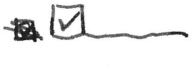
- Basic Structure

```
<form action="destination" method="get or post">
    ... form element.
    <input type="submit" value="Send">
</form>
```

1. action="destination" : which program to send the form to.
   it can be a http://... address, or an the same server (subdir/..)
2. method
   ① "get" → http://.... ? q= ... (name="q" in the <input>)
   - can see the form data in the URL
   - can only handle a small ~~text~~ transmission.

   ② "post"
   - cannot see the form data
   - can handle a big transmission, e.g., files.
3. <textarea> ... </textarea>
   attribute : rows, cols, name

- Form input elements. Each input should be given a name.
  <input type="submit, text, checkbox, radio, password, file">

1. text 
   password 
   remember to use "post" method.
   checkbox 
   radio 
2. They're used with a "submit" ~~to send~~ at the end of the <form> tag.

3. selecting from a list :

```
<select name="">
        <option value=""> ... </option>
                ⋮
</select>
```

- Useful Attributes

1. "value" = " " : fix what's shown at the start.
2. placeholder = " " : show useful text which disappears when the user enters something.
3. autofocus
4. required.

- `<label for="fieldname">` some text : `</label>`
  `<input type="" id="fieldname" name="">`
  `<br>`

Some text: ☐

- Handling file Upload.

1. add enctype = "multipart/form-data" and method = "post" to <form> tag.
2. `<input type="file">`.

- New HTML5 Input Element

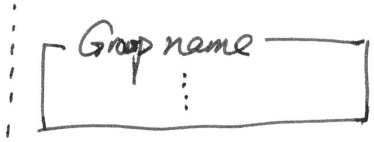
`<input type="number, date, time, color, range"`

↓

min, max, step.

■ Element Grouping.

```
<fieldset>
        <legend> Group name </legend>
                ⋮
</fieldset>
```



Lecture 4 Introduction to Javascript

where to put js:

① <head> or <body> ..., anywhere in the html page.
<script src=" "></script>, in source file.

② inside <script> ... </script> tag.

And by using: console.log( ); the content in ( ) is shown in the browser's console. (use it as "print")

■ Javascript Basics.

• Defining variables, function, and scope.

1. Variables: var a = ...

* No types are declared { Js: dynamically typed.
Same variable can holds different types

2. functions: function a ( ) { }

var a = function ( ) { }

* Here, a is NOT the return value, but the function itself

* Pass variables:

function a (x, y) { }

NOT add var.

* Retune values (call a function):

var value = a (...)

* Either including passing variables or not is legal when calling the function. (although it may not make sense)

3. Scope: Global | Function

* Everything is executed in an Execution Context

* Function invocation creates a new execution Context.

↓

Functions can be defined inside a function.

* Each Execution Context has:
  - Its own Variable Environment
  - Special 'this' object
  - Reference to its Outer Environment

* Global scope does not have outer environment.

* Scope Chain:
  Referenced variable will be searched for its current scope first.
  If not found, the Outer Reference will be searched.
  This will keep going until the Global scope.
  If not found in the Global scope, it will be "undefined".

Note: it doesn't matter where a function is called, but it does matter where it is defined.

For example:

```
function B ( ) { console.log (x); }
function A ( ) { var x=5; B( ); }
var x=2;
var y
A( );
```

The result is x=2.
Here B( ) is defined in the Global scope. So B( )'s outer reference is the Global scope, not A scope.

■ Types

1. Object : a collection of name:value pairs. can be nested.

e.g., Person object

```
first : "John",
last : "White",
social : {
            linkedin : "here",
            facebook : "there"
         }
```

2. Primitive : a single, immutable value

① Boolean : true ; false.

② Undefined : never set a varible to 'undefined'

③ Null : null.

④ Number : double-precision 64-bit floating point.
The ONLY numeric type in JS. (i.e., no int type. etc.)

⑤ String

⑥ Symbol :
* New to ES6 (released 2015)

Not : 'undefine' = has been declared, but no value is assigned.
≠ varible is not defined, an error.

■ Common language Construct.

- String concatination : var string = "Hello" + "world";
- Regular math operators : +, −, *, /
  'NaN' means not a number, it comes out in console if a "undefine" varible appears in ~~an of~~ math operator.
- (Equality (==) : x = 4; y = '4'; x == y : true
  | strict equality (===) : x === y : false. ⇑ type coercion
  Here when '==', the variable type can be automatically converted.

- true / false :

```
if ( false || null || undefined || "" || 0 || NaN) {
        console.log ("Never executed");
}
else {
        console.log ("All false");
}

if (true && "hello" && 1 && -1 && "false") {
        console.log ("All true");
}
```

One can check whether an expression / value is true or false by calling : Boolean (***);

- Put { on the same line : Otherwise, sometimes JS would automaticly put a ; after a line which ~~you~~ meant to have more contents (i.e., { }) on the next line.

For example,

```
function a() {
        return
        {
          name : "Ellie"
        };
}
console.log(a()); ⇒ undefined.
```

Because the compiler would make it into : return; and stop.

- for (var i=0 ; i<=10; i++) { }, for (var prop in object) { }.

■ Handling Default Values : make use of type coercion and true/false.

> true || false
<- true

> "hello" || "undefined
<- "hello"

> "hello" || "backup"
<- "hello".

Therefore, use : variable = variable || "backup";
to get rid of the circumstances where variable = "undefined.

■ Object.

- creation : `var obj = new Object();`

  * note that before setting name : value to an object (no matter it's nested or not) , the object MUST be created at first.

- Set / get the value by . name or [name] notation :

```
obj.firstname = "value1";
obj["second name"] = "value2"   // [ ] is used where the name cannot be referred by . notation.
```

- Nested object :

```
obj.thirdname = new Object();
obj.thirdname.nestedfirst = "nested value 1";
obj.thirdname.nestedsecond = "nested value 2";
concole.log(obj);  ⇒  Object { firstname: "value1", second name: "value2", thirdname: { nestedfirst: "nested 1", ... } }
```

- Object Literal :

```
var obj = {
        firstname : "value 1",
        secondname : {
                $stock : 110,
                "fav of me" : "red"
        }
}
```

■ Function : { * First-class data types. * Objects. ⇐ The most powerful feature of JS.

- That said, ~~calling~~ a function is called only by: func( ), the variable itself is an object.

e.g., function plus (x, y) { return x+y; };

```
console.log (plus);
```

⇒ function plus (x, y) { return x+y; }

```
plus.version = "v.1.0";
console.log (plus.version);
```

→ v.1.0

- functions can be nested / passed around.

Example 1 : a function that generates new functions :

```
function makeFunc (multiplier) {
    var newFunc = function (x) {
        return multiplier * x;
    };
    return newFunc;
}

var double = makeFunc (2);
    // here double is a function, equvillant to :
    var double = function (x) {
        return 2 * x;
    };
```

Example : passing functions as arguments :

```
function doOperationOn (x, operation) {
    return operation (x);
}
var result = doOperationOn (5, double);
    // Equavilent to :
    var result = double (5);
Concole.log (result);
```

→ 10

■ Passing variables by values v.s. by reference.

{ Objects : passed by reference : object ≈ pointer
primitives : passed by value.

```
var a=7;
var b=a;
b=5;
console.log(a);
  -> 7
```

```
var a={x:7};
var b=a;
b.x=5;
console.log(a.x);
  -> 5
```

■ "this"

- When a new function is created, the invocation of that function creates a new execution context, in which "this" object is created.
- this = windows, window = { external : Object; chrome : Object. .... }
- If a function is ~~declared~~ called by "new" keyword", i.e.,

```
var newObj = new existingFunc( );
```

then Object this is pointing to the created object itself.

~~i.e., this (inside of existingFunc) = new Ob existi~~

i.e., this = existingFunc { }

newObj = existingFunc { }

Remember function is an object.

- Creating an object from function constructors :

```
function Circle (radius) {
    this.radius = radius; // this ==circle
    this.getArea = function(){return Math.PI* Math.pow(this.radius, 2);};
}
var myCircle = new Circle(10); ⇒ Circle { radius : 10,
                                          getArea : function() }
```

- Construct objects with _proto_ :

```
// Object / Function Constructor
function Circle (radius) {
        this.radius = radius;
}
Circle.prototype.getArea = function () {
        return Math.PI * Math.pow (this.radius, 2);
        }
var myFirstCircle = new Circle (10);
var mySecondCircle = new Circle (20);
console.log (myFirstCircle.getArea());
        -> 314.1592...
consde.log (mySecondCircle);
        -> Circle { radius: 20 }
```

* Here myFirstCircle and mySecondCircle share the same _proto_, where getArea() is one of the ~~fun~~ methods.
* This is very useful to create objects that have the same structure and methods, but different values.
* Now you should have a better idea of objects and methods.

- In an object literal, "this" points to the object itself, rather than points to "window". Why?

— Because creating a new object we've use the "new" keyword: var obj = new Object(). You can think of Object() is a built-in function constructor with no name: value declared.

this = { window : in a regular function
| the object / function itself : when "new" keyword is used.

Note : if functions are nested in another function / object,
"this" object inside the nested function is windows.
So be careful when handling "this" in nested structures.

■ Arrays :

- create an new array by : var array = new Array();
- Elements in an array can be different data types. even functions
- Alternatively, initialize an array by : var array = [...];
- methods : .length()
- ~~Although~~ arrays are just objects, so elements can be added by array.new = "hi". It doesn't need to be limited to array[1]=...
- Loop over arrays : for (index in array) { array[index] ... }

■ Closures

when passing a function through returning, the memory preserves lexial environment for it. so that when the function is actually invoked, it knows where to find the ~~outer~~ passed variables.

■ Fake Name Spaces.

In order to avoid the overwritting of ~~same~~ variables that have the same name but defined in different script, we can wrap the name inside an object, as well as methods.

For example :

In script 1 :

```
var a. = { };
a.name = "Ellie";
a.func = function(){};
```

In script 2 :

```
var b = { };
b.name = "John";
b.func = function(){};
```

In html,

```
a.func(a.name); // this won't mix up with b.
```

■ IIFEs (immediately invoked function expressions)

- function declaration is wrapped up in ( )
- call the function inline.
- we can put a branch of codes in the function
- parameters inside ( ) can be exposed to Global by passing it to ~~wid~~ "window" object.

For example:

```
( function ( window) {
        var scripta = { };
        Scripta. name = "Ellie";
        var greeting = "Hi"; // A varicable that you don't know what to
                                store in an object
        Scripta. sayHi = function ( ){
            console. log ( greeting + scripta. name);
        }
        window. scripta = scripta;
} ) ( window);
// In HTML
~~scripta. sayHi~~  greeting = "Hello"; // it doesn't matter "greeting" is changed
~~gree~~            scripta. sayHi ( ); // scripta is visible.
                        → Hi Ellie
```

Lecture 5 Using JS to Build Web Applications

- DOM Manipulation
  - ↳ document object model
  - Document Object (window.document).
    1. document.getElementById(" ")
       returns the whole element. ~.value returns
    2. document instanceof HTMLDocument. == True
  - functions defined by JS can be called in HTML.
    On the other hand, methods of the "document" object can read in the HTML elements and change/edit HTML contents.

Example:

In JS script:

```
function SayHello() {
    var name = document.getElementById("name").value;
    document.getElementById("content").innerHTML =
        "<h2> Hello " + name + "! <h2>";
    # get the input & write in the HTML the whole " " contents.
    if (name == "student") {
        var title = document.querySelector("#title").textContent
        # the syntax of querySelector is the same as in CSS.
        title += " & Lovin' it!";
        document.querySelector("#h1").textcontent = title;
        # Modify the HTML Contents.
}
```

In HTML: `<body> <h1 id="title"> Lecture 53 </h1>`
Say hello to `<input id="name" type="text">`
`<button onclick="sayHello();"> Say it! </button>`
`<div id="content"></div>`
`<script scr="js/script.js"></script> </body>`

■ Handling events

- Event handlers : functions, bind with other methods, to certain events happening in the browser.
- Events : life cycle (e.g., page loaded)
  user interaction (e.g., type, click)
- Assign event handlers to particular events.

1. use "on~" attribute. in HTML elements.

e.g., onblur = "func()"
execute the function when the field loses focus.
onclick = "func()", etc.

* side effect 1 : HTML is just for content, not for function.
* side effect 2 : "this" object in the function points to the global context. (window)
  The function is not actually assigned to the event.

2. Unobstrusive event binding.

① document.querySelector("button").addEventListener("click", sayHello)
now "this" points to its container – <button> object.

② document.querySelector("button").onclick = SayHello;

* Make use of "this" object.

For example, if you want to change the text on the button, just simply add:

```
this.textContent = "said it";
```

to the sayHello function. That equals to:

```
document.querySelector("button").textContent = "said it".
```

- Where to put js script:

1. Add <script scr=" "></script> to the very end of HTML
2. Place <script scr=" "></script> to the <head> tag, and put all the js functions / contents in:

```
document.addEventListener("DOMContentLoaded",
    function (event) {
        ↳ object for event handler.
        function sayHello (event) {
            ⋮
        }
        document.....
        ;
    }
);
```

In this way, the js is loaded before ~~anything else is loaded~~ images, CSS and scripts are loaded.
(external sources)

■ "event" argument.

- "event" is an object, which is passed to every event handler that tells the type of event, and the properties of that event.
- details see http://developer.mozilla.org/en-US/docs/Web/Events
- "event" enables JS functions to obtain infos of events and do some actions based on users' actions.

For example: show in console the coordinates of cursors when "shift" is pressed

```
document.querySelector("body").addEventListener("mousemove",
    function (event) {
        if (event.shiftKey === true) {
            console.log("x: " + event.clientX);
            console.log("y: " + event.clientY);
        }
});
```

■ HTTP Basics (HyperText Transfer Protocol)

- Based on request/response Stateless protocol.

  client open connection to server
  ↓
  client sends HTTP request for a source
  ↓
  server sends HTTP response to the client
  ↓
  client closes connection to server.

- URN (uniform Resource Name)
  * uniquely identify resource / name of resource
  * do not tell how to get it
- URI (Uniform Resource Identifier)
  * uniquely identify resource / location of resource
  * do not necessarily tell how to get it.

  e.g., /official_web-site/index.html.
- URL (Uniform Resouree Locator)
  * Form of URI that provides info on how to get resource.

  e.g., http://www.mysite.com/official_web-site/index.html.

- HTTP Request Structure ~~(GET)~~

  ↳ Actual language that browsers communicate with the server.

  Example : GET request.

  GET /index.html?firstname=Ellie HTTP/1.1

  ⇓ Method (command)

  ⇓ URI string

  ⇓ Query String ? name1=value1 & name2=value2

  ⇓ HTTP version.

  the server is already connected,
  and the URI is asked within the content of the server.

- HTTP Methods
  - * GET : Retrive the resource
    - * Data is passed to server as part of the URI (query string)
  - * POST : Sends data to server in order to be processed
    - * Data is sent in the message body (can handle a file)

e.g., POST / index.http HTTP/1.1 # no query string.
Host : coursera.org
Accept-Charset : utf-8] request headers.
firstname = Ellie
...] message body.

- HTTP Response Structure

HTTP/1.1 200 OK # HTTP version ␣ Response Status Code ␣ English phrase describing Status Code
Date : Thu, 12 May 2016 19:34:01 GMT
Content-Type : text/HTML } header
<html>
<body>
... } message body content
</body> can be plain-text, xml, JSON, etc.
</html>

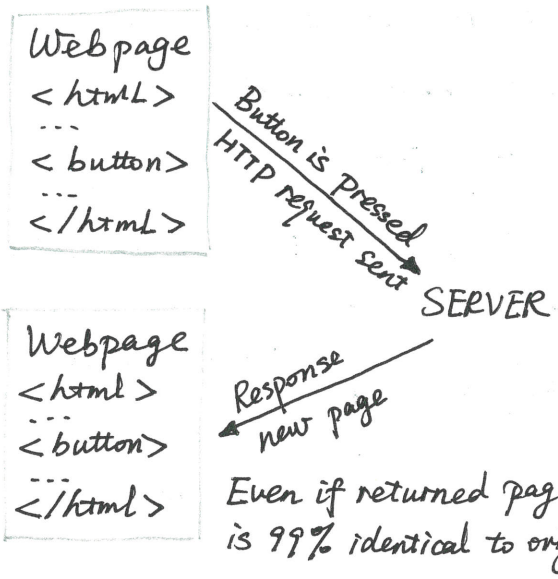
Common Response Status Code :

- * 200 OK # OK, here's the content you requested
- * 404 Not Found # Server can't find the resource
- * 403 Forbidden # Unauthenticated client tried to access a secure resource
- * 500 Internal Server Error # Some unhandled error was raised on the server

Ajax Basis

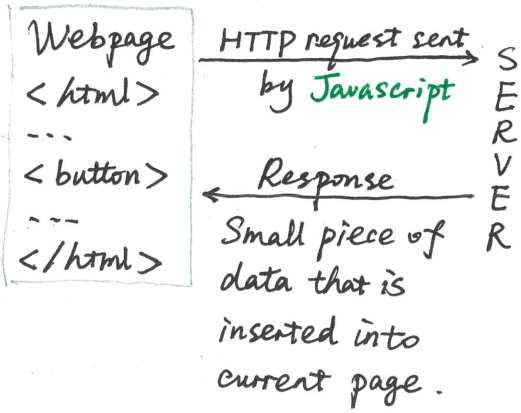
↳ Asynchronous Javascript And XML

Traditiond Web App How



Even if returned page is 99% identical to original.

Ajax Web App How



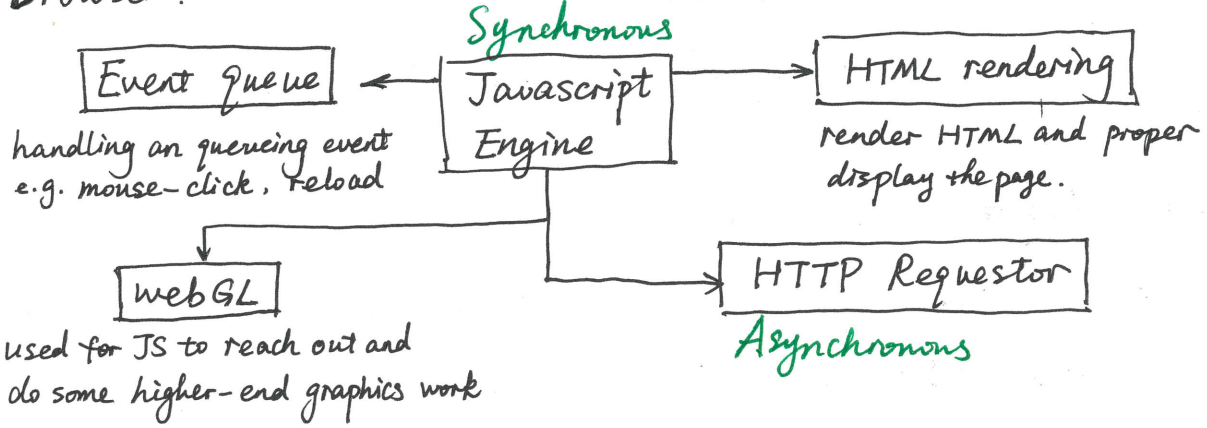
* Faster response.

Synchronous execution: Execution of one instruction at a time.

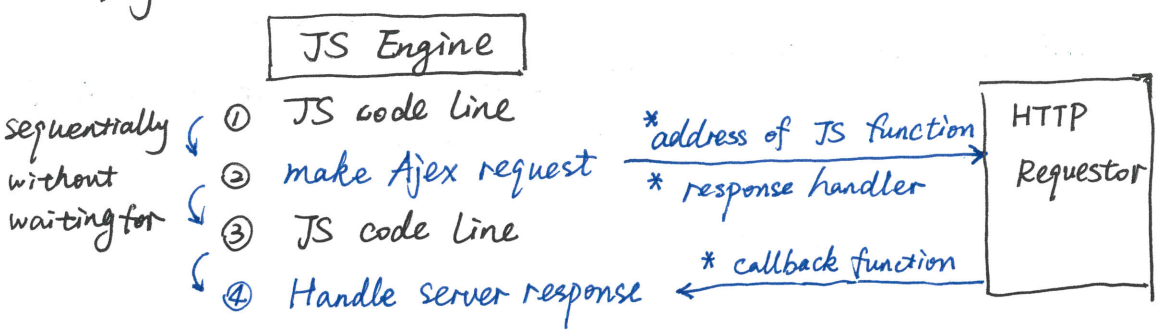
Asynchronous execution: Execution of more than one instruction at a time.

∴ Asyn-exec instruction returns rightaway, and the actual execution is done in a seperate thread or process.

Browser:



Ajax Process:



Code:

```
( function ( global ) {

// Set up a name space for the utility
var ajaxUtils = { };

// returns an HTTP request Object
function getRequestObject ( ) {
        if ( window. XMLHttpRequest) {
                return ( new XMLHttpRequest());
        }
        else if ( window. ActiveXobject) {   // For very old IE only.
                return ( new ActiveXObject ("Microsoft. XMLHTTP");
        }
        else {
                global. alert ("Ajax is not Supported!");
                return (null);
        }
}

// Makes an ajax GET request to 'requestURL'
ajaxUtils. sendGetRequest = function (requestURL, responseHandler){
        var request = getRequestObject();  // Make it local, not global,
                                           to avoid conflicts when the
                                           function is called by many
                                           process at the same time.
        // Define the function to be called every time there's a change
           in the communication stage.
        request. onreadystatechange = function ( ) {
                handleResponse ( request. responseHandler);
        };
        // use "open" command to set info of the request.
        request. open ("GET", requestURL, true );
                                                  ↓
        // send the request to the server        Asynchronous on.
        request. send ( null ); // for POST only.
}
```

```
// Only call responseHandler (pass the request) only if response is ready
   and no error.
function handleResponse (request, responseHandler) {
    if ((request.readyState == 4) &&  // The last state and is ready to go.
        (request.status == 200)) {    // no error.
        responseHandler (request);
    }
};

// Expose utility to the Glible object.
global.$ajaxUtils = ajaxUtils;
}) (window);


// Javascript, event handling:
document.addEventListener ("DOMContentLoaded",
    function (event) {
        document.querySelector ("button").addEventListener
        ("click", function () {
            // Call server to get the name.
            $ajaxUtils.sendGetRequest ("/data/name.txt",
The actual func that's executed ← // function (request) {
put it inside $ajaxUtils,          ← (var name = request.responseText;
not as a seperate function                document.querySelector ("#content")
~~or out s~~ outside, because                  .innerHTML = "<h2>Hello" + name;
Ajax is asynchronous, and          });
these lines will be executed
without completion) ;of the request if they
    }          are put outside next.
);
```

■ Processing JSON (JavaScript Object Notation)

- St JSON = simple textual representation of data, and is completely independent of any language. Good format for passing data server ↕ client.
- Syntax rules:
  1) Subset of JS object literal syntax, but
     ① property names must be in double quotes.
     ② String values must be in double quotes.
  2) Others are the same.

Example:
```
var jsonString =
'{
    "firstname" : "Ellie",
    "likeprogramming" : true,
    "Year" : 1991
}';
```

- Common Misconception
  1) JSON is NOT a JS object literal.
  2) JSON is just a string.
  3) Need to convert JSON into a JS object, and vice visa.
- JSON ↔ Object:

JSON ⇒ Object : `var obj = JSON.parse(jsonString);`

Object ⇒ JSON : `var str = JSON.stringify(obj);`

Example: implement into JS:
```
$ajaxUtils.sendGetRequest("/data/j name.json/",
    function(res) { // passing responses from server
        document.querySelector("#content").innerHTML
            = "<h2>" + res.property1 + res.property2 + "</h2>";
    });
```
in function handleResponse:
```
... responseHandler(JSON.parse(request.ResponseTex));
```

Real Site.

- Nav menu Automatic collapse.

*. Using Jquery style to be compatible with Bootstrap functions.

```
$( function( ) {  // == document.addEventListener("DOMContentLoaded",...
                     That is loaded before any external resources are
                     loaded when refreshing the page.

    $("#navbarToggle").blur(
    // $(" ") is a selector. == document.querySelector("#...").
       addEventListener("blur", ...)
        function (event) {
            var screenwidth = window.innerWidth;
            if (screenwidth < 768) {
                $("#collapsable-nav").collapse('hide');
            }
        });
});
```

- Dynamically Loading Home View Content. using JS/Ajax

```
(function (global) {
    var dc = { };
    // define marcros
    var homeHTML = "snippets/home-snippets.html";
    // Convenience function for inserting innerHTML for "selector"
    var insertHTML = function (selector, HTML) {
        var targetElem = document.querySelector(selector);
        targetElem.innerHTML = html;
    };
    // Showing Loading icon (from www.ajaxload.info)
    var showLoading = function (selector) {
        var html = "<div class='text-center'>";
        html += "<img src='images/ajax-loader.gif'></div>";
```

```
    insertHTML(selector, html);
  };
  // On page load.
  document.addEventListener("DOMContentLoaded", function(event) {
      showLoading("#main-content");
      $ajaxUtils.sendGetRequest( // Defined previously.
          homeHTML,
          function(respText) {
              document.querySelector("#main_content").innerHTML
                  = respText;
          },
          false); // can turn asynchrous off don't take as JSON
  });

  global.$dc = dc;
})(window);
```

In HTML. `<div id="main-content class="container"></div>`

The main contents are in "snippets/home-snippets.html";

- Dynamically Loading Menu Categories View using JSON.

In html snippet (template for each item)

```
<div class="col-md-3 col-sm-4 col-xsm-6, col-xxs-12">
    <a href="#" onclick="$dc.loadMenuItems('{{short_name}}');">
        <div class="category_title">
            <img width="200" height="200" src="images/menu/
            /{{short_name}}/{{short_name}}.jpg" alt="{{name}}">
            <span>{{name}}</span>
        </div>
    </a>
</div>
```

single-page loading, not a href to another page.

get the value of each property, defined as follows (next page)

In js:

```
// Return substitute of '{{propName}}' with propValue
var insertProperty = function (string, propName, propValue) {
    var propToReplace = "{{" + propName + "}}";
    string = string.replace(new RegExp(propToReplace, "g"), propValue);
    return string;
}

// Load the menu categories view, which is called at clicking the menu tile
                                                    at the home page
dc.loadMenuCategories = function () {
    showLoading("#main-content");
    $ajaxUtils.sendGetRequest(
        allCategoriesUrl,   //http://.../categories.json
        buildAndShowCategoriesHTML);
}

// Builds HTML for the categories page based on the data from the server
function buildAndShowCategoriesHTML(categories) {
    // Load title snippet
    $ajaxUtils.sendGetRequest(                    "snippets/categories-title-snippet.html"
        categoriesTitleHTML, function (categoriesTitleHTML) {
        // retrieve single category snippet
            $ajaxUtils.sendGetRequest("snippets/categories-snippet.html"
                categoriesHTML, function (categoriesHTML) {
                    var categoriesViewHTML =
                        buildCategoriesViewHTML(categories,
                                                categoriesTitleHTML,
                                                categoriesHTML);
                    insertHTML("#main-content",
                               categoriesViewHTML);
                }, false);
        }, false);
}
```

don't take it with JSON

```
// Using categories data and snippet HTML to build categories view
   HTML to be inserted into page.
function buildCategoriesViewHTML(categories, categoriesTitleHTML,
                                 categoriesHTML){

    var finalHTML = categoriesTitleHTML + "<section class='row'>";

    // Loop over categories object. defined in ~~JSON~~ Object.
    for( var i=0; i< categories.length ; i++){
        var html = categoriesHTML;
        var name = "" + categories[i].name;
        var short_name = categories[i].short_name;
        html = insertProperty(html, "name", name);
        html = insertProperty(html, "short_name", shortname);
        // add each item in the rendered page
        finalHTML += html;
    }

    finalHTML += "</section>";
    return finalHTML;
}
```

- Dynamically Loading Single category view.

1. menu-items-title.html:

```
<h2 id="menu-categories-title" class="text-center">
    {{name}} Menu </h2>
<div class="text-center">{{special_instructions}}</div>
// Note that in categories.json, each item has:
    "name": "...", "id"="...", "short_name"="...",
    "special-instructions": "..."
```

for each category.

2. in JS: taken from Ruby on the Rail App

```
var menuItemsUrl = "http://.../menu_items.json?category=";
:
```

```
dc.loadMenuItems = function (categoryShort) {
    showLoading ("#main-content");
    $ajaxUtils.sendGetRequest (menuItemsUrl + categoryShort,
                               builtAndShowMenuItemsHTML);
}; // default : url is JSON

function builtAndShowMenuItemsHTML (categoryMenuItems) {
    // analogue of the category one
}

function buildMenuItemsViewHTML (..., ..., ...) {
    // Generate different categories pages
    menuItemsTitleHTML = insertProperty (menuItemsTitleHTML,
                                         "name", categoryMenuItems.category.name);
    ..... ("special_instructions", ...);
    var finalHTML = menuItemsTitleHTML;
    for (var i=0; i<menuItems.length; i++) {
        :
        html = insertPrice (html, "price_small", menuItems[i].price_small);
    }
    // Add clearfix after every second menu item
    if (i%2 != 0) { html += "<div class="clearfix visible-lg-block..."></div>"; }
}
function insertPrice (html, pricePropName, priceValue) {
    // Check if the price exists. If not, return empty string
    if (!priceValue) {
        return insertProperty (html, pricePropName, "");
    }
    // Format price
    priceValue = "$"+ priceValue.toFixed(2);
    return insertProperty (html+pricePropName, priceValue);
}
```

- Changing 'active' Button Style Through JS.

```
var switchMenuToActive = function() {
    // Remove 'active' from Home Button. (Because all the menu
       and single category pages are built upon Home page)
    var classes = document.querySelector("#navHomeButton").
                  .className;
    classes = classes.replace(new RegExp("active", "g"), "");
    document.querySelector("#navHomeButton").classname = classes;
    // Add "active" to menu button if not already there
    classes = document.querySelector("#naveMenuButton").classname;
    if (classes.indexOf("active") == -1) { // check if "active" is
        classes += " active";                  in the classes string.
        document.querySelector("#navMenuButton").classname=classes;
    }
};
```

then add "switchMenuToActive();" into builtAndShow... functions.