# MODULE - II ①

by Sheriy Noel CMR

## HTML Tables and forms

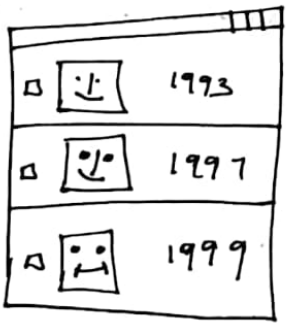
### Introducing Tables :-

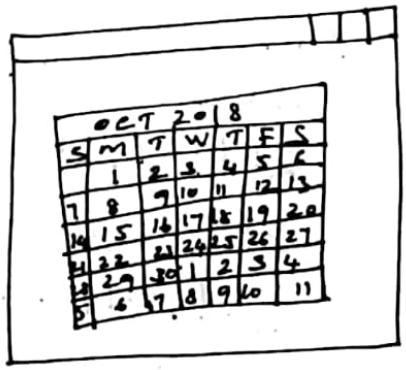
→ A table in HTML is created using the <table> element and can be used to represent information that exists in a two-dimensional grid.

→ Tables can be used to display calendars, financial data, pricing tables and many other types of data.

→ Tables can contain any type of data; numbers, text, images, forms, even other tables.

eg :




### * Basic Table Structure :-

To implement the following table,

| | | | |
|---|---|---|---|
| The Death of Marat | Jacques | 1793 | 162cm 128cm |
| Burial at Ornans | Gustave | 1849 | 314cm 663cm. |

<table> → Contain any number of rows.

<tr> → to add a row

<td> → any number of data cells.

Shirly Nael - CMRIT

CMR

<table>
<tr><td>The death of Marat</td><td>Jacques</td><td>1793</td><td>162 cm</td><td>128 cm</td></tr>
<tr><td>Burial at Ornans</td><td>Gustave</td><td>1849</td><td>314 cm</td><td>663 cm</td></tr>
</table>

```
<table>
  <tr>
    <td> The death of marat </td>
    <td> Jacques </td>
    <td> 1793 </td>
    <td> 162 cm </td>
    <td> 128cm </td>
  </tr>
  <tr>
    <td> Burial at Ornans </td>
    <td> Gustave </td>
    <td> 1849 </td>
    <td> 314 cm </td>
    <td> 663 cm </td>
  </tr>
</table>
```

fig : Basic table structure.

→ Many table will contain some types of heading in the first row.

<th> → indicate header data.
↳ browser make the th element in bold.

eg:

| Title | Artist | Year | Width | Height. |
|---|---|---|---|---|

```
<table>
    <tr>
        <th> Title </th>
        <th> Artist </th>
        <th> Year </th>
        <th> width </th>
        <th> Height </th>
    </tr>
</table>.
```

* Spanning Rows and Columns :-

→ To cover several rows or columns, the rowspan and colspan attributes are used.

eg : see next page :

* Additional Table elements :-

<caption> → provide a brief title or description of the table.

<thead> → defines a set of rows defining the head of the columns of the table.

<tbody> → comprise the body of the table.

<tfoot> → defines a set of rows summarizing the columns of the table.

CMR

<colgroup> → Columns can be kept within colgroup to use common styles.

<col> → specifies column properties for each columns within a <colgroup> element.

eg:

```
<table>
  <caption> 19th century French paintings </caption>
  <colgroup>
    <col span="2" style="background-color: red">
    <col style="background-color: yellow">
  </colgroup>
  <thead>
    <tr> <th> Title </th> <th> Artist </th>
         <th> year </th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td colspan="2"> Total no of paintings </td>
      <td> 2 </td>
    </tr>
  </tfoot>
  <tbody>
    <tr> <td> The death of marat </td>
         <td> Jacques </td>
         <td> 1793 </td>
    </tr>
  </tbody>
</table>
```

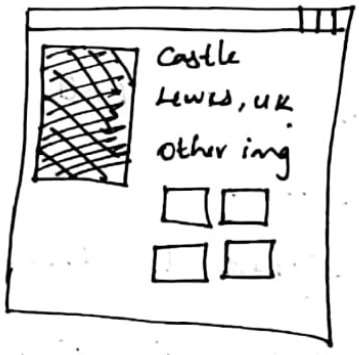
19th century french paintings

| Title | Artist | year |
|---|---|---|
| The death of marat | Jacques | 1793 |
| Total no of paintings | | 2 |

Shruthy Noel.
CMR

* Using tables for Layout :-

eg:




→ In the above example, tables are used to design layouts.

→ It was not uncommon for a complex layout to have dozens of embedded tables.

→ The practice of using tables for layout had some problems.

① This approach dramatically increase the size of the HTML document.

↳ These larger files take longer to download, and also difficult to maintain because of extra markup

② The resulting markup is not semantic.

↳ tables are meant to indicate tabular data

③ users who rely on software to voice the content table-based layouts can be extremely uncomfortable and confusing to understand.

# Styling Tables :-

## * Table Borders :-

Borders can be assigned to both the <table> and the <td> element

→ Borders cannot be assigned to the <tr>, <thead>, <tfoot> and <tbody> elements.

eg : ① table { border : solid 1Pt black; }

② td { border : solid 1pt black; }

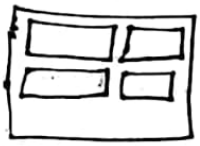
③ table { border : solid 1Pt black;
border-collapse : collapse;
}

④ table { border : solid 1Pt black;
border-spacing : 10pt;
}

td
{ border : solid .1Pt black;
}

→ border-collapse property selects the tables border model.

→ values are collapse, seperate.



<table>


→ To adjust the space between these adjacent borders via the border-spacing property.

*Boxes and zebras:

→ There are number of ways to style a table.

① Box format :-

eg: table
{
font-size : 0.8em; font-family Arial;
border-collapse : collapse;
border-top : 4px solid #DCA806;
border-bottom : 1px solid white;
text-align : left;
}
caption
{
font-weight : bold; padding : 0.25em 0 0 0;
text-align : left;
text-transform : uppercase;
border-top : 1px solid #DCA806;
}
thead tr
{ background-color : #CACACA;
}

```
th {
    padding : 0.75em;
}
tbody tr
{
    background-color : # f1f1f1;
    border-bottom : 1px solid white;
    color : # 6E6E6E;
}
tbody td
{
    padding : 0.75em;
}
tbody tr : hover {
    background-color : #9e9e9e;
    color : black;
}
tbody tr : nth-child (odd) {
    background-color : white;
}
```

} zebras.

## Introducing Forms :-

→ Forms provide the user with an alternative way to interact with a web server.

→ Using a form, the user can enter text, choose items from lists, and click buttons.

→ programs running on the server will take the input from HTML forms and save it in a database, interact with an external web server or customize subsequent HTML based on that input.

## * Form Structure :-

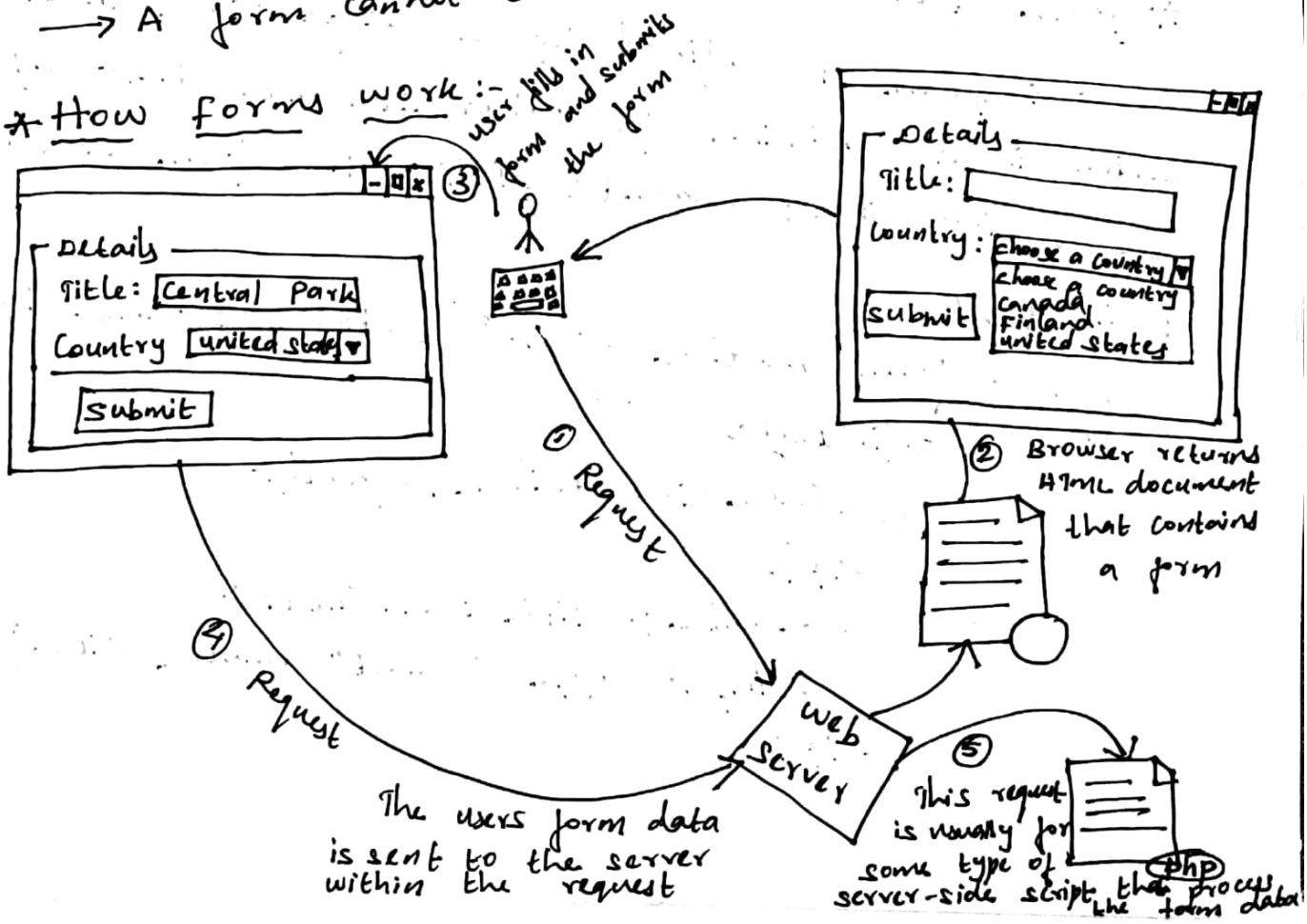
→ Form is defined by a <form> element, which is a container for other elements that represent the various input elements within the form as well as plain text.

eg:
```
<form method = "get" action = "process.php"
    <fieldset>
        <legend> Details </legend>
        <p>
            <label> Title : </label>
            <input type = "text" name = "title" />
        </p>
```

```
<p>
  <label> Country : </label>
  <select name = "where">
    <option> Choose a country </option>
    <option> Canada </option>
    <option> Finland </option>
    <option> United states </option>
  </select>
 </p>
 <input type = "submit" />
</fieldset>
</form>
```

→ A form cannot contain another <form> element.

*How forms work :-



Shirly Noel
CMR

→ While forms are constructed with HTML elements, a form also requires some type of server-side resource that processes the users form input as shown in the fig.

→ The process begins with a request for an HTML page that contains some type of form on it.

→ After the user fills out the form, there needs to be some mechanism for submitting the form data back to the server.

→ This is achieved via a submit button.

→ Interaction between the browser and the web server is govered by the HTTP protocol, the form data must be sent to the server via a standard HTTP request.

→ This request is typically some type of server-side program that will process the form data in some way; this could include checking it for validity, storing it in a database or sending it in an email.

* Query strings:-

→ The browser packages the users data input into something called a query string...

→ The query string is a series of name=value pairs seperated by ampersands (the & character).

→ names in the query string were defined by the HTML form. eg: title and where in previous example

→ values in the query string are the data entered by the user.

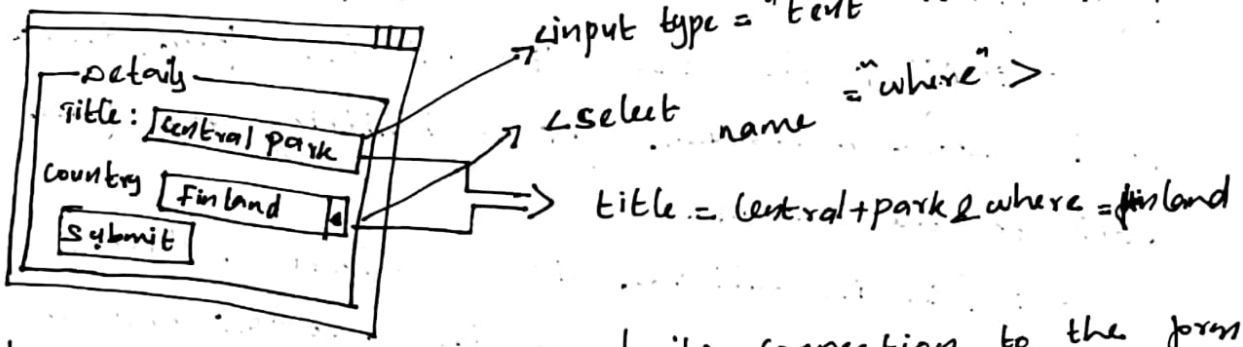



fig: Query string data and its connection to the form elements.

→ The above figure illustrates, how the form data is packaged into a query string.

→ Query strings have certain rules defined by the HTTP protocol.

→ characters such as spaces, punctuation symbols and foreign characters cannot be part of a query string.

→ Instead, such symbols must be URL encoded (percent encoded).

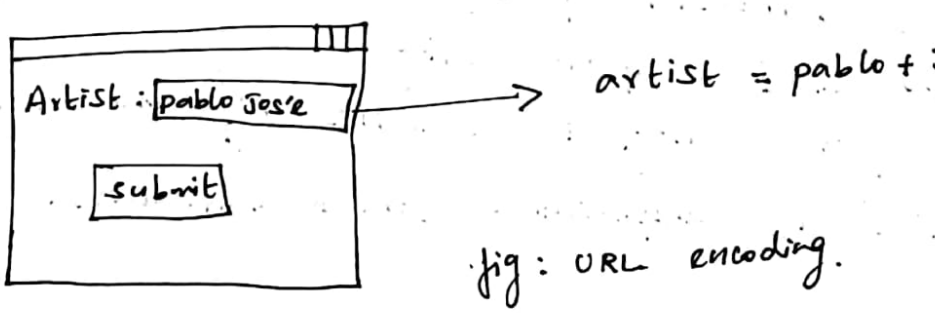



fig: URL encoding.

* The <form> Element :-

→ The form element contains two important attributes, namely the action and the method attributes.

action => This attributes specifies the URL of the server-side resource that will process the form data.

↳ This could be a resource on the same server as the form or a completely different server.

method => This attribute specifies how the query string data will be transmitted from the browser to the server.

↳ There are 2 possibilities : GET and POST.

Difference between GET and POST :

GET => The browser locates the data in the URL of the request.

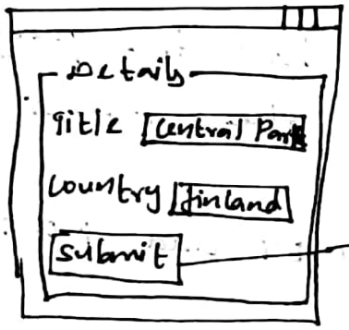
POST => The form data is located in the HTTP header after the HTTP variables.

| Type | Advantages | Disadvantages |
|---|---|---|
| GET | ① Data can be clearly seen in the address bar<br>② Data remains in browser history and cache.<br>③ Data can be bookmarked | → DisAdv in production<br>→ security risk on public computers<br>→ limits the number of characters in the form data returned. |

CMR

| Type | Advantages | DisAdvantages |
|---|---|---|
| POST | ① Data can contain binary data. ② | Data is hidden from user submitted data is not stored in cache, history or bookmarks. |

→ For using potentially security risk data, post method can be used.
↓
password.




```
<form method = "get" action = "process.php">
        ⇓
GET / process.php ? title = Central + park &
        where = finland  http/1.1
<form method = "post" action = "process.php">
```

HTTP header

```
POST / process.php  http/1.1
Date : sun, 20 may 2012 23:59:39 GMT
HOST : www.mysite.com
user - Agent : mozilla 9 / 4.0
Content - Length : 47

title = Central + park & where = finland
```

fig : GET versus post.

Ashly Noel
CMR

## Form Control Elements :-

→ The following table shows the form-related HTML elements.

| | Type | Description. |
|---|---|---|
| 1. | <button> | – Defines a clickable button |
| 2. | <datalist> | – An HTML5 element that defines list of pre-defined values to use with input fields. |
| 3. | <fieldset> | – Groups related elements in a form together |
| 4. | <form> | – Defines the form container. |
| 5. | <input> | – Defines an input field. HTML5 defines over 20 different types of input. |
| 6. | <label> | – Defines a label for a form input element. |
| 7. | <legend> | – Defines the label for a fieldset group. |
| 8. | <option> | – Defines an option in a multi-item list |
| 9. | <optgroup> | – Defines a group of related options in a multi-item list. |
| 10. | <select> | – Defines a multi-item list |
| 11. | <textarea> | – Defines a multiline text entry box. |

CMR

* Text Input Controls :-

→ Forms need to gather text information from the user.

→ The below table lists the different text input Controls.

→ some of the HTML5 text elements are not uniformly supported by all browsers, they still work as regular text boxes in older browsers.

| | Type | Description |
|---|---|---|
| 1. | text | – creates a single-line text entry box. <input type="text" name="title" /> |
| 2. | textarea | – creates a multiline text entry box. <textarea rows="3" ... /> |
| 3. | password | – creates a single-line text entry box for a password. <input type="password" ... /> |
| 4. | search | – creates a single-line text entry box suitable for a search string. <input type="search" ... /> |
| 5. | email | – creates a single-line text entry box suitable for entering an email address. <input type="email" ... /> |

6. tel – creates a single-line text entry box suitable for entering a telephone number.

```
<input type="tel" .... />
```

7. url – creates a single-line text entry box suitable for entering a URL.

```
<input type="url" .... />
```

## * Choice Controls :-

→ Forms often need the user to select an option from a group of choices.

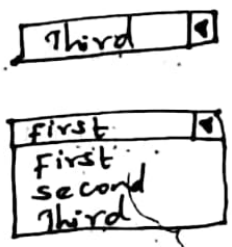
### 1) Select Lists :

→ The <select> element is used to create a multiline box for selecting one or more items.

→ Options for the select element are using <option> element.

eg :
```
<select name="choices">
    <option> first </option>
    <option> second </option>
    <option selected> third </option>
</select>
```

o/p :

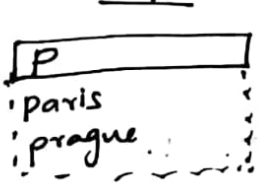



→ The selected attribute in the option makes it a default value.

eg: 2 `<input type="text" name="city" list="cities" />`

```
<datalist id="cities">
    <option> calcutta </option>
    <option> London </option>
    <option> paris </option>
    <option> prague </option>
</datalist>
```

o/p




eg: 3

```
<select name="choices">
    <optgroup label="North America">
        <option> calgary </option>
        <option> Los Angels </option>
    </optgroup>
    <optgroup label="Europe">
        <option value="1"> London </option>
        <option value="2"> paris </option>
    </optgroup>
</select>
```

cities:


choices = calgary

choices = 1

2) Radio Buttons :-

→ To select a single item from a small list of choices.

`<input type="radio">`

→ The buttons are made mutually exclusive ie) only one can be chosen, by sharing the same name

attribute.

→ The checked attribute is used to indicate the default choice.

eg: <input type = "radio" name = "where" value = "1">
North America <br/>
<input type = "radio" name = "where" value = "2">
South America <br/>
<input type = "radio" name = "where" value = "3">
Asia <br/>

O/P

○ North America
○ South America
⊙ Asia

3) Checkboxes:-

→ It is used for getting yes/no or on/off responses from the user.

<input type = "checkbox">

→ checkboxes can be grouped together by sharing the same name attribute.

→ Each checked checkbox will have its value sent to the server.

→ The checked attribute can be used to set the default value of a checkbox.

eg: <label> I accept the software license </label>
<input type = "checkbox" name = "accept">
<label> where would you like to visit? </label><br>
<input type = "checkbox" name = "visit" value = "canada"> Canada <br />
<input type = "checkbox" name = "visit" value = "france"> France <br />
<input type = "checkbox" name = "visit" value = "Germany"> Germany <br />

→ ? accept = on & visit = canada & visit = Germany

O/P

I accept the software licsence ☑
☑ Canada
☐ France
☑ Germany

## Button Controls

→ HTML defines several different types of buttons -

① <input type = "submit"> — creates a button that submits the form data to the server.

② <input type = "reset"> — creates a button that clears any of the users already entered form data.

③ <input type = "button"> — Create a custom button. This button may require Javascript for it to actually perform any action.

④ <input type = "image"> — creates a custom submit button that uses an image for its display.

⑤ <button> — creates a custom button.

<button type = "submit"><img src = "..."></button>

* Specialized controls:-

→ a special purpose form controls that are available in all browsers.

① <input type = "hidden" name = "cust" value = "123">

↳ The input field is hidden and is not shown to the user, but the data is sent when the form is submitted.

② <input type = "file">

eg: <form method = "post" enctype = "multipart/form-data" ...>

.....

<label> upload a travel photo </label>

<input type = "file" name = "photo" />

....

</form>

→ This input field is to upload a file from a client to the server.

→ form element must use the post method and it must include the enctype = "multipart / form-data" attributes as well.

→ In query strings, form data is URL encoded. i.e) enctype = "application / x-www-form-urlencoded")

→ files cannot be transferred to the server using normal URL encoding. Hence the need for the alternative enctype attribute.

* Number and Range:

→ HTML 5 introduced two new controls for the input of numeric values.

① → <label> Rate this photo </label>
<input type = "number" min = "1" max = "5" name = "rate" />

o/p Rate this photo: 2

②
Beginner
<input type = "range" min = "0" max = "10" step = "1" name = "survey" />
Expert.

o/p Beginner ——— Expert

→ When input via a standard text control, numbers typically require validation to ensure that the user has entered an actual numbers, because the range of numbers is infinite.

→ The number and range controls provide a way to input numeric values that eliminate the need for client-side numeric validation.

* Color :

→ HTML 5 Color control provides a convenient interface for the user.

eg : <label> Background Color : <br/>
<input type = "color" name = "bg" />

Date and Time controls :-

→ The new date and time controls in HTML try to make it easier for users to input the date and time values.

| | Type | Description |
|---|---|---|
| 1. | date | creates a general data input control. The format is "yyyy-mm-dd". |
| 2. | time | creates a time input control. format "HH:MM:SS" |

| | | |
|---|---|---|
| 3. | datetime | enter date and time |
| 4. | datetime-local | date and time without specifying timezone |
| 5. | month | enter month in a year format "yyyy-mm" |
| 6. | week | enter week in a year. format "yyyy-W##" |

eg: <input type="time" ... />
<input type="datetime" ... />
<input type="datetime-local" ... />
<input type="month" ... />
<input type="week" ... />

## Table and Form Accessibility:

→ Web accessibility refers to the assistive technologies, various features of HTML that work with those technologies, and different coding and design practices that make a site more useful for people with visual, mobility, auditory and cognitive disabilities.

→ To improve the accessibility of websites, the W3C created the web Accessibility Initiative (WAI) in 1997.

→ The most important guidelines in that document are,

* provide text alternatives for any nontext content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language.

* Create content that can be presented in different ways without losing information or structure.

* Make all functionality available from a keyboard.

* provide ways to help users navigate, find content, and determine where they are.

*Accessible Tables :-

→ Users who rely on visual readers can find pages with many tables especially difficult to use.

→ To improve the situation is to only use tables for tabular data, not for layout.

→ Using the following accessibility features for tables in HTML can improve the experience for those users.

① Describe the table's content using the <caption> element. This provides the user with the ability to discover what the table is about before having to listen to the content of each and every cell in the table.

→ For long description for the table, put the table within a <figure> element and use the <figcaption> element to provide the description.

② Connect the cells with a textual description in the header.

→ users rely on visual reader, this is not an easy task.

→ To listin to reader software, the contents of a table sounds like "row 3, cell 4 : 45.56; row 3; cell 5 : Canada".

*Accessible forms :-

→ Browser provide some unique formatting to the <fieldset> and <legend> elements.

→ Their main purpose is to logically group related form input elements together with the <legend> providing

a type of caption for those elements.

<label> has no special formatting. Each <label> element should be associated with a single input element.

→ To associate explicitly, use the for attribute.

→ If the user clicks on or taps the <label> text, that control will receive the form's focus.

eg: <label for="f-title">Title: </label>
<input type="text" name="title" id="f-title" />
<label for="f-country">Country: </label>
<select name="where" id="f-country">
<option>Choose a country </option>
<option>Canada </option>
<option>Finland </option>
</select>

## Microformats :-

Microformat is a small pattern of HTML markup and attributes to represent common blocks of information such as people, events and news stories.

→ This information can be extracted and indexed by software agents.

eg:

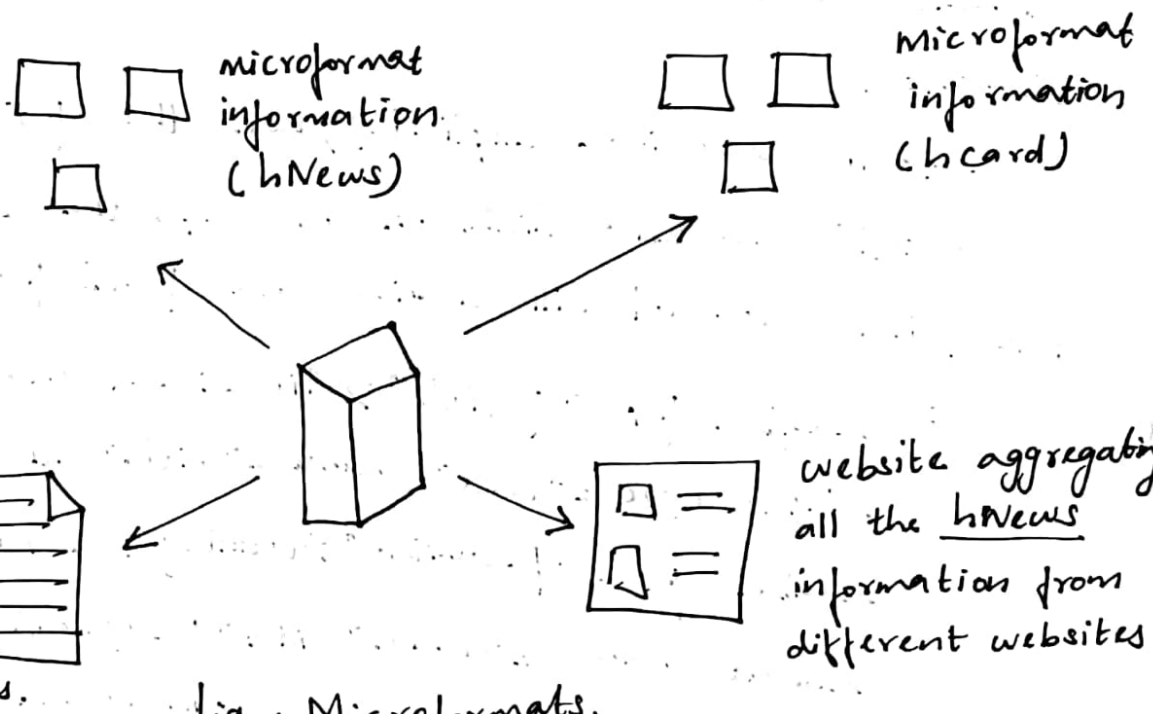



fig: Microformats.

→ The idea behind microformats is that if this type of common information were tagged in similar ways, then automated tools would be able to gather and transform it.

→ most common microformats is hCard; which is used to semantically markup contact information for a person.

Sherly Noel
CMR

→ Google Map search results now make use of the hCard microformat so that if you used the browser extension, you could save the information to your computer on phone's contact list.

eg: Example of an hcard.

```
<div class = "vcard">
    <span class = "fn"> Randy Connolly </span>
    <div class = "org"> Mount Royal univ </div>
    <div class = "adr">
        <div class = "street-address"> 4825 mount
            Royal Gate SW
        </div>
        <div>
            <span class = "locality"> Calgary </span>
            <abbr class = "region" title = "Alberta">
                AB
            </abbr>
            <span class = "postal-code"> T3E 6K6 </span>
        </div>
        <div class = "country-name"> Canada </div>
    </div>
    <div> phone: <span class = "tel">+1-403-440-
        6111 </span> </div>
</div>
```

# ADVANCED CSS — Layout

## * Normal flow

**Block Level Elements :-**

eg). <p>, , <h2>, <ul>, <table>.

→ Block level elements begin with a line break. ie) start on a new line.

→ without styling, two block-level elements cant exist on the same line.

→ Block level elements use the normal CSS box model, in that they have margins, padding, borders and background colors.

**Inline Elements :-**

→ Inline elements are displayed within lines.

eg). , <a>, <img> and .

→ Inline elements line up next to one another horizontally from left to right on the same line.

→ when there isn't enough space left on the line, the content moves to a newline.

2 types of Inline elements;

i) Replaced inline elements.

ii) Non replaced inline elements.

**Replaced Inline Elements :-**

→ The elements, whose content is defined by external resource and thus appearance eg) <img>

→ These elements have a width and height that are defined by the external resource.

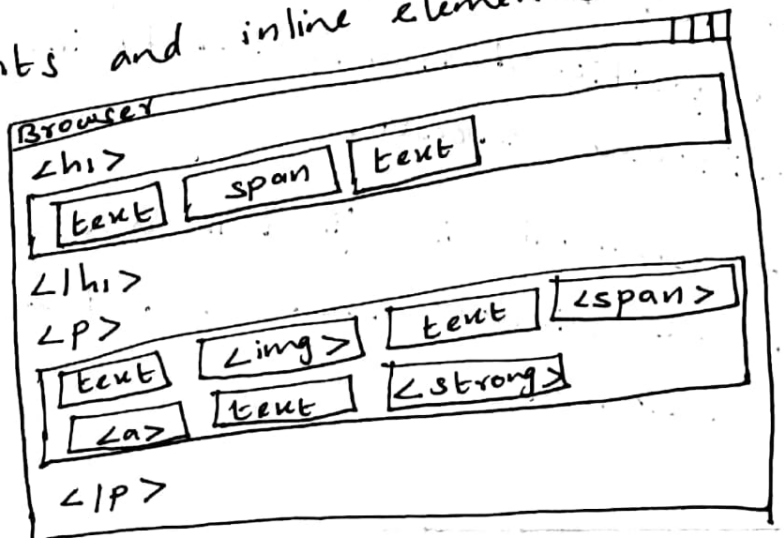
**Non Replaced inline elements :-**

→ The elements, whose content is defined within the document, which includes all the other inline elements.

→ It have a constrained box model, ie) width is defined by their content.

→ In a document with normal flow, block-level elements and inline elements work together.

eg).



→ Block level elements will flow from top to bottom, while inline elements flow from left to right within a block.

→ It is possible to change whether an element is block-level or inline via the CSS display property.

→ Consider the following two CSS rules,

span { display : block ; } → displays in new line

li { display : inline ; } → displays in same line.

## Positioning Elements :-

→ It is posible to move an item from its regular position in the normal flow.

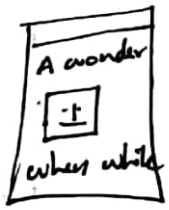
→ The position property is used to specify the type of positioning. The values are,

| | value | Description |
|---|---|---|
| 1. | absolute | The element is removed from normal flow and positioned in relation to its nearest positioned ancestor. |
| 2. | Fixed | The element is fixed in a specific position in the window even when the document is scrolled. |

| | | |
|---|---|---|
| 3. | relative | The element is moved relative to where it would be in the normal flow. |
| 4. | static | The element is positioned according to the normal flow. This is the default. |

eg: position : absolute;

```
<p> A wonderful serenity has taken ..... </p>
    <figure>
        <img src = "828.jpg" />
        <figcaption> museum </figcaption>
    </figure>
<p> when, while the lovely valley ... </p>
```




```
figure {
    border : 1pt solid #ABABAB;
    background-color : #EDEDED;
    padding : 5px; width : 150px;
    position : relative;
    top : 150px; left : 200px;
}
```


→ The top, left, bottom and right properties are used to indicate the distance the element will move.

→ The effect of these properties varies depending upon the position property.

① Relative Positioning :-

→ An element is displaced out of its normal flow position and moved relative to where it would have been placed.

→ The other content around the relatively positioned element remembers the elements's old position in the flow; thus the space the element would have occupied is preserved.

eg : see previous page.

② Absolute positioning :-

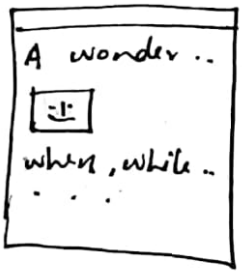
→ When an element is positioned absolutely, it is removed completely from normal flow.

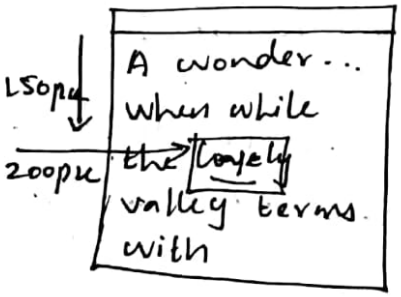
→ Thus unlike with relative positioning, space is not left for the moved element, as it is no longer in the normal flow.

→ Its position is moved in relation to its container block.

eg : position : absolute.

→ replaced position attribute value in the previous example.




③ 2-Index :-

→ Only positioned elements will make use of their z-index.

→ Each positioned element has a stacking order defined by the z-index property (z-axis).

→ Items closest to the viewer have a larger z-index value.

→ Setting the z-index value of elements will not move them on top or behind other items.

④ Fixed Position :-

→ In fixed positioning, elements do not move when the user scrolls up or down the page.

→ This is most commonly used to ensure that navigation elements or advertisements are always visible.

eg: figure {
position : fixed;
top : 0;
left : 0; }

## Floating Elements:-

→ This is to displace an element out of its position in the normal flow via the CSS float property.

→ A element can be floated to the left or floated to the right.

→ When an item is floated, it is moved all the way to the far left or far right of its containing block and the rest of the content is "re-flowed" around the floated element.

### * Floating within a Container:-

→ A floated item moves to the left or right of its container called as containing block.

→ The containing block is the HTML document itself so the figure moves to the left or right of the browser window.

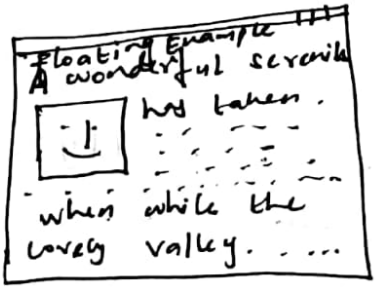
eg: <h1> Float Example </h1>
 A wonderful serenity has taken ..... 
<figure>
<img src = "J1.jpg" />
<figcaption> British museum </figcaption>

</figure>
 when, while the lovely valley 

CSS

```
figure {
    border : 1pt solid #A8A8A8;
    background-color : #EDEDED; float : left
    margin : 0; padding : 5px; width : 150px; }
```

O/P




* floating multiple items side by side :-

→ To float multiple items that are in proximity, each floated item in the container will be nestled up beside the previously floated item.

→ All other content in the containing block will flow around all the floated elements.

eg :

```
<article>
    <figure>
        <img src = "a1.jpg" />
        <figcaption> picture 1 </figcaption>
    </figure>
    <figure>
        <img src = "a2.jpg" />
        <figcaption> picture 2 </figcaption>
```

```
</figure>
.....
</article>
```

```
CSS
figure:{
.....
width : 150px;
float : left;
}
```

o/p


→ The above create some pretty messy layouts as the browser window increases or decreases in size.

→ To stop elements from flowing around a floated element by using the clear property.

→ The values are,

| | Value | Description |
|---|---|---|
| 1. | left | The left hand edge of the element cannot be adjusted to another element. |
| 2. | right | The right hand edge of the element cannot be adjusted to another element. |
| 3. | both | Both the left-hand and right-hand edges of the element cannot be adjacent to another element. |

4. none — The element can be adjacent to other elements

eg :
```
<article>
    <figure>
        <img src="a2.jpg" />
        <figcaption> picture 1 </figcaption>
    </figure>
    <figure>
        <img src="a3.jpg" />
        <figcaption> picture 2 </figcaption>
    </figure>
    ....
    <p class="first"> when, while the lovely.. </p>
</article>
```

CSS
```
.first {
clear: left;
}
```

* Containing floats :-

→ When an element is floated within a containing block that contains only floated content.

eg :
```
<article>
    <figure>
        <img src="aa.jpg">
        <figcaption> British museum </figcaption>
    </figure>
    <p class="first"> When, while the lovely.. </p>
</article>
```

CSS
```
figure img {
    width : 170px;
    float : left;
    margin : 0 5px;
}
```

```
figure figcaption {
    width : 100px;
    float : left;
}
```

```
figure
{
    border : 1pt solid #262626;
    background-color : #c1c1c1;
    padding : 5px;
    width : 400px;
    margin : 100px;
}
```

```
.first{
    clear : left;
}
```

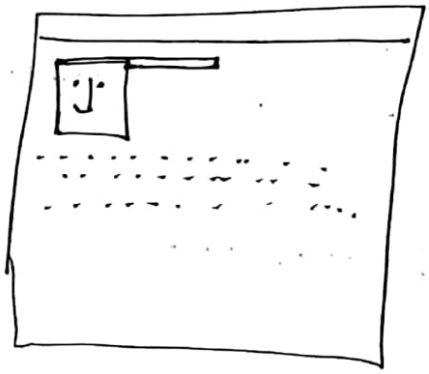


fig : Disappearing parent containers.

→ In the above example, both of these elements are floated to the left.

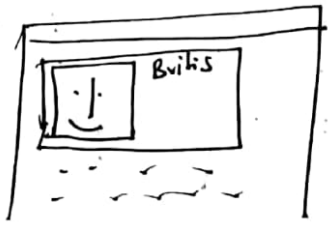
→ This means, both elements have been removed from the normal flow.

→ A solution is to float the container depending on the layout. The property is,

```
overflow : auto;
```

→ Add this line to the figure element in CSS

→ Later, the output looks as,



Sherly Noel
CMR

## * Overlaying and hiding Elements :-

→ One common design tasks with CSS is to place two elements on top of each other or to selectively hide and display elements.

→ positioning is important to both of these tasks. and used for smaller design changes such as moving items relative to other elements within a container.

→ Relative positioning is used to create the positioning content for a subsequent absolute positioning move.

→ Absolute positioning in positioning is relation to the closest positioned ancestor.

⟹ There are 2 different ways to hide elements in CSS.

① display property → values, auto, none

② visibility property → values, visible, hidden.

display => It takes an item out of the flow.

visibility => hides the element, but the space for that element remains.

eg:

```
<figure>
    <img src="a1.jpg" />
    <figcaption> British museum </figcaption>
    <img src="w.jpg" class="overlayed" />
</figure>.
```

CSS

o/p

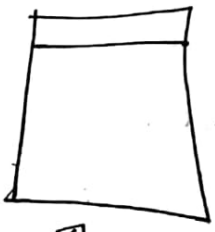
2 images overlayed

```
.overlayed { position : absolute;
             top : 10px ; left : 10px;
             display : none;
           }
```

=> 

---

```
figure {
  . . . .
  . . . .
  visibility : hidden;
}
```

=> 

no image

## Constructing Multicolumn Layouts

→ By using positioning and floats, create more complex layouts.

* Using floats to create columns :-

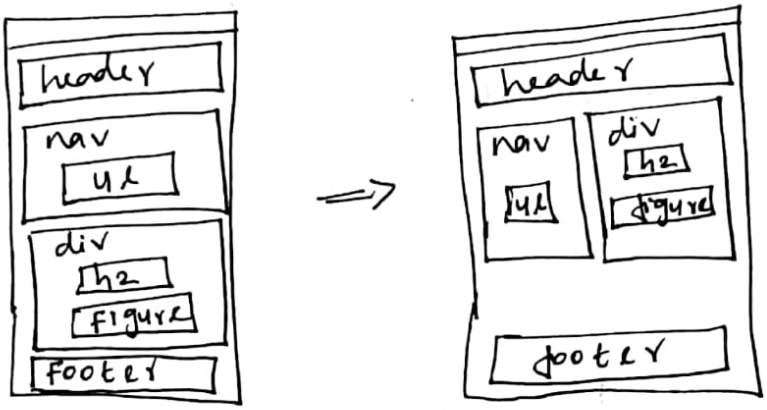
→ The common way to create columns of content is using float property.

→ The first step is to float the content container on the left-hand side or to the right.

→ The floated container need to have a width specified.

Sherly Noel
CMR

eg: ①




```
nav
{
  float : left;
  width : 12em;
}
```

fig: Creating two-column layout

eg: ③

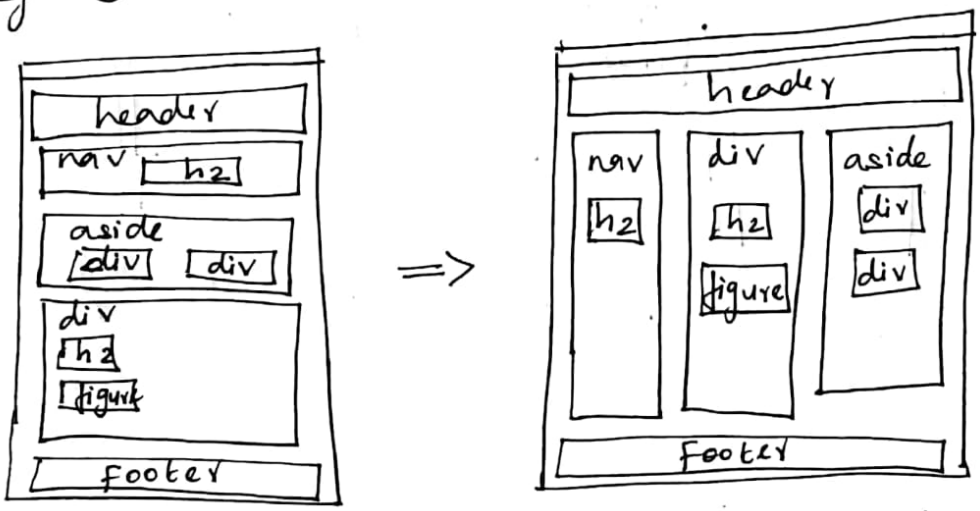



fig: Creating a three-column layout.

→ Another approach for creating a three-column layout is to float elements within a container element.

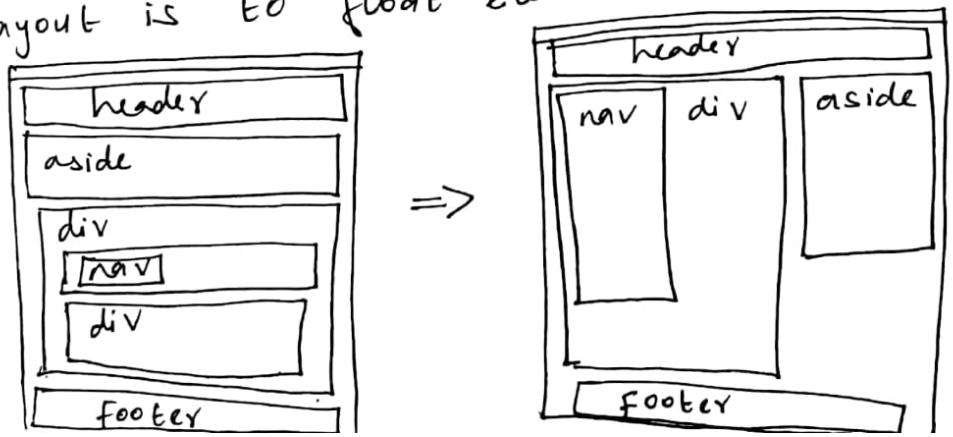



fig: creating a three-column layout with nested floats.

* Using positioning to create columns :-

→ Positioning is used to create a multicolumn layout.

eg :

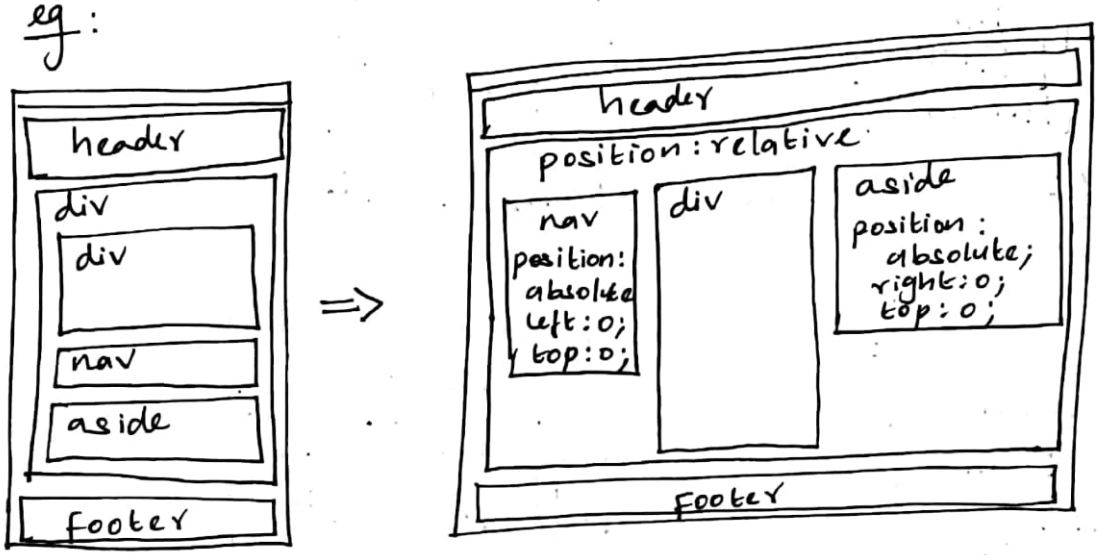



fig : Three - column layout with positioning.

→ Absolute positioning has its own problems. What would happen if one of the sidebars had a lot of content and was thus quite long?

→ In the floated layout, this would not be a problem, because when an item is floated, blank space is left behind.

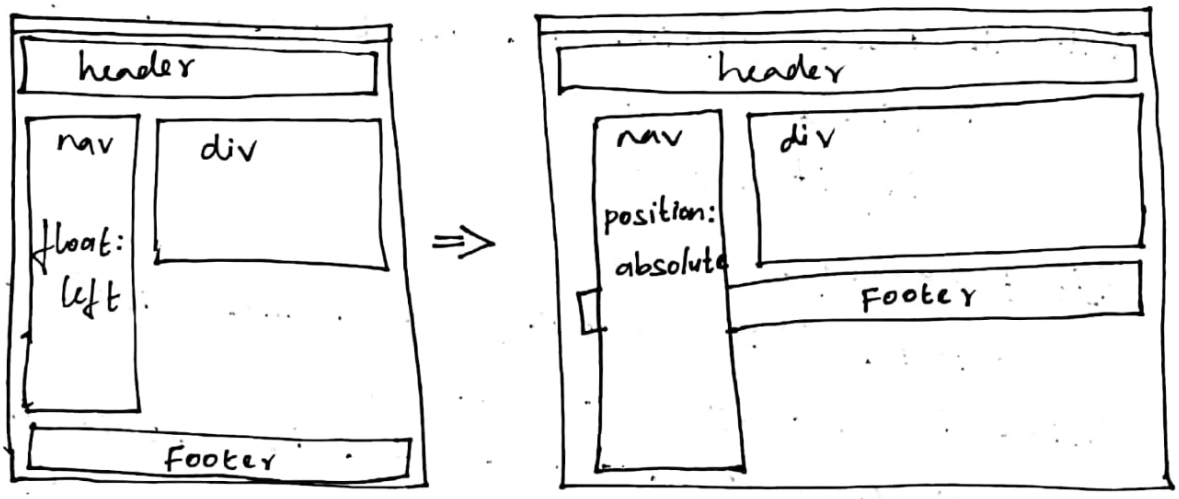
→ But when an item is positioned, it is removed entirely from normal flow, so subsequent items will have no knowledge of the positioned item.

Asherly Noel
CMR

→ One solution to the problem is to place the footer within the main container.

→ However, this has the problems of a footer that is not at the bottom of the page.

eg.




Elements that are floated leave behind space for them in the normal flow.

Absolute positioned elements are taken completely out of normal flow, meaning that the positioned element may overlap subsequent content.

## Approaches to CSS Layout :-

→ The main problem faced by web designers is that the size of the screen used to view the page can vary quite a bit.

→ Some users will visit a site on a 21-inch wide screen monitor that can display 1920 x 1080 pixel (px); other will visit it on an older iphone with a 3.5 screen and a resolution of 320 x 480 px.

→ Users with the large monitor might expect a site to scale to the smaller size and still be usable.

→ satisfying both users can be difficult.

→ Different approaches has different solutions, but all these solutions are on these basic models.

① Fixed Layout

② Liquid Layout.

Fixed Layout :-

→ In this layout, the basic width of the design is set by the designer.

→ A common width used is 960 to 1000 pixel range, which fits nicely in the common desktop monitor resolution (1024 x 768)

→ This content can then be positioned on the left or the center of the monitor.

→ Fixed layouts are created using pixel units, within a  container whose width property set to some width.

eg: <body>

<header> ..... </header>
 ..... 
<footer> ..... </footer>

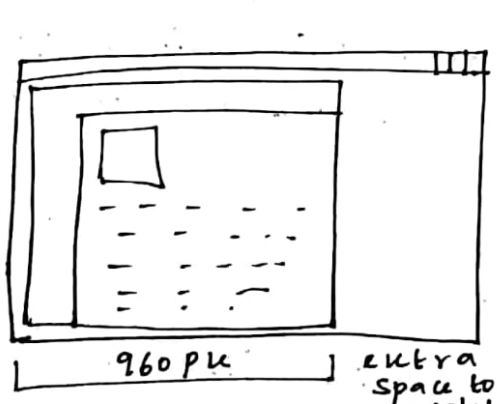
</body>

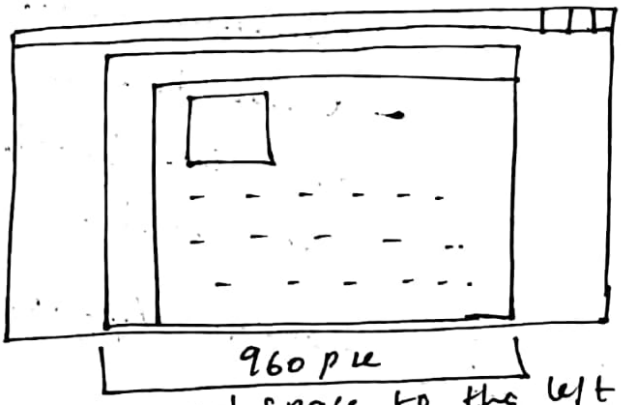
Shirly noel
CMR

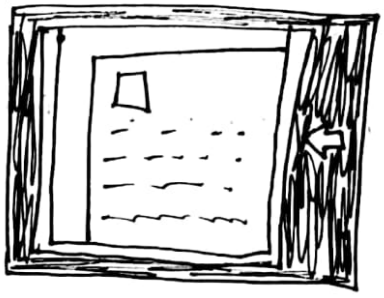
```
div #wrapper {
    width : 960px;
    margin-left : auto;
    margin-right : auto;
    background-color : tan;
}
```

Adv :- Easier to produce.
- Optimized for typical desktop monitors.

DisAdv :- For larger screens, there may be an excessive amount of blank space to the left and/or right of the content.
- When the browser window shinks below the fixed width; the user will have to horizontally scroll to see all the content.



The problem with fixed layouts is that they don't adapt to smaller viewports.

fig : problem with fixed layouts.

Liquid Layout :- (or) fluid layout.

→ This approach is to deal with the problems of multiple screen size.

→ In this approach, widths are not specified using pixels, but percentage values.

→ Its percentage of the current browser width, so a layout in which all widths are expressed as percentages should adapt to any browser size.

Adv :- It adapts to different browser sizes, so there is neither wasted white space nor any need for horizontal scrolling.

DisAdv :- It is difficult to create because some elements such as images have fixed pixel sizes.
- When the screen grows or shrinks dynamically the line length may become too long or too short.

③ Other Layout Approaches :-

→ Some approaches combine fixed and liquid layout, and called as hybrid layout.

→ It this approach, they combine pixel and percentage measurements.

## Responsive Design :-

→ The page responds to changes in the browser size that go beyond the width scaling of a liquid layout.

→ problem in liquid layout is that images and horizontal navigation elements tend to take up a fixed size, and when the browser window shrinks to the size of a mobile browser, liquid layouts can become unusable.

→ In a responsive layout images will be scaled down and navigation elements will be replaced as the browser shrinks.

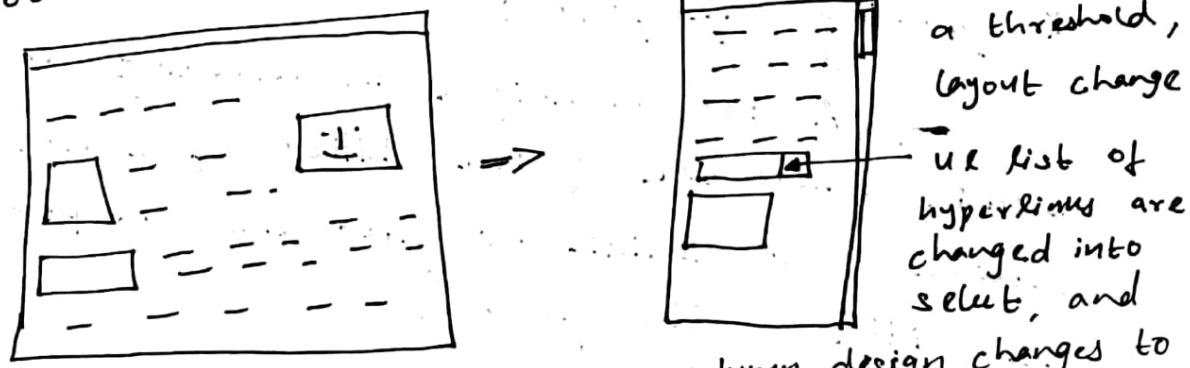



fig : Responsive Layouts.

The four key components that make responsive design work. They are,

1. Liquid Layouts
2. scaling images to the viewport size
3. setting viewports via the <meta> tag
4. customizing the css for different viewports using media queries.

→ Responsive design begins with a liquid layout. ie) most elements have their widths specified as percentages

eg)
```
img {
    max-width : 100%;
}
```

* setting viewports :-

→ A key technique in creating responsive layouts makes use of the ability of current mobile browsers to shrink or grow the web page to fit the width of the screen.

→ The way this works is the mobile browser renders the page on a canvas called the viewport.

eg: iphone viewport = 980 px.

Shirly Noel. CMR

```
<html>
  <head>
    <meta name = "viewport"
          content = " width = device-width" />
```

→ device-width sets the pixels wide as the device screen width.

→ ie) if the device has a screen that is 320px wide, the viewport width will be 320px.

* Media Queries :-

→ The other key component of responsive design is css media queries.

→ A media query is a way to apply style rule based on the medium that is displaying the file.

```
@media only screen and (max-width : 480px)
{ ..... }
```

- @media : defines this a media query
- only : use only this style if both conditions are true
- screen : device has to be a screen
- (max-width : 480px) : use this style if width of viewport is no wider than 480 pixels.

fig : Sample media query.

→ The above figure illustrates the syntax of a typical media query.

→ The queries are boolean expressions and can be added to your css files or to the <link> element

to conditionally use a different external css file based on the capabilities of the device.

| Feature | Description |
|---|---|
| 1. width | width of the viewport |
| 2. height | height of the viewport |
| 3. device-width | width of the device |
| 4. device-height | height of the device |
| 5. orientation | whether the device is portrait or landscape |
| 6. color | the number of bits per color. |

→ The above table list the browser features. most of the feature have min- and max- version

→ Contemporary responsive sites will typically provide css rules for phone displays first, then tablets, then desktop monitors, an approach called progressive enhancement, in which a design is adapted to progressively more advanced devices.

style.css

```
/* rules for phones */
@media only screen and (max-width: 480px)
{ #slider-image { max-width: 100px; }
  #flash-ad { display: none; }
}
```

```
/* css rules for tablets */
@media only screen and (min-width: 481px)
      and (max-width: 768px)
  {
  .....
  }
/* css rules for desktops */
@media only screen and (min-width: 769px)
{
  .....
}
```

→ Instead of having all the rules in a single file, put them in seperate files and add media queries to <link> elements.

eg: <link rel="stylesheet" href="mobile.css"
media="screen and (max-width: 480px)"/>
<link rel="stylesheet" href="tablet.css" media=
"screen and (min-width: 481px) and
(max-width: 768px)"/>
<link rel="stylesheet" href="desktop.css"
media="screen and (min-width: 769px)"/>

```
<!-- [if lt IE 9]>
  <link rel="stylesheet" media="all" href=
        "style-ie.css"/>
<![endif]-->
```

# CSS FrameWorks

A css framework is a precreated set of css classes or other software tools that mak it easier to use and work with css.

→ There are two main types of css framework.

① Grid systems.

② css preprocessors.

① Grid systems :-

→ This system is easier to create multi-column layouts.

eg :- Bootstrap (twitter.github.com/bootstrap),
Blueprint (www.blueprintcss.org)
960 (960.gs).

→ print designers typically use grids as a way to achieve visual uniformity in a design.

→ The first thing, designer construct is a 5- or 7- 12-column grid in a page layout program like InDesign or Quark Xpress.

Shirly Noel
CMR

→ CSS frameworks provide similar grid features.

→ The 960 framework uses either a 12- or 16-column grid.

→ The Bootstrap uses a 12-column grid.

→ Blueprint uses a 24-column grid.

→ The grid is constructed using  element with classes defined by the framework.

→ In bootstrap and 960, elements are laid out in rows; elements in a row will span from 1 to 12 columns.

→ In 960 system, a row is terminated with  .

→ In Bootstrap system, content must be placed within the  row container.

eg:
```
<head>
    <link rel = "stylesheet" href = "reset.css" />
    <link rel = "stylesheet" href = "text.css" />
    <link rel = "stylesheet" href = "960.css" />
</head>
<body>
    <div class = "container_12">
        <div class = "grid_2">
```

```
        left Column
    </div>
    <div class = "grid-7">
        main content
    </div>
    <div class = "grid-3">
        right column
    </div>
    <div class = "clear"></div>
</div>
</body>
```

fig : Using the 960 grid.

```
<head>
    <link href = "bootstrap.css" rel = "stylesheet"
</head>
<body>
    <div class = "container">
        <div class = "row">
            <div class = "col-md-2">
                left Column
            </div>
```

```
<div class = "col-md-7">
    main content
</div>
<div class = "col-md-3">
    right column
</div>
</div>
</div>
</body>
```

fig: using the Bootstrap grid.

→ Both of these frameworks allow columns to be nested, making it quite easy to construct the most complex of layouts.

→ Bootstrap provides more than just a grid system.

→ It also has a wide variety of very useful additional styling classes such as classes for drop-down menus, fancy buttons and form elements and integration with a variety of jQuery plug-ins.

(2) * CSS preprocessors :-

→ these are tools that allow the developer to write css that takes advantage of programming ideas such as variables, inheritance, calculations

and functions.

→ It is a tool that takes code written in some type of preprocessor language and then converts that code into normal css.

Adv :- It can provide additional functionalities that are not available in css.

eg: For color scheme, the background color of header, footer, fieldset, textarea is set to #796d6d.

→ If the client likes #e8cfcf more than #796d6d, then some type of copy and replace is necessary, which leaves the probability to made to the wrong elements.

→ In normal css, one has to copy and paste to create that uniformity.

→ In a programming language, a developer can use variables, nesting, functions or inheritance to handle duplication and avoid copy-and-pasting and search-and-replacing.

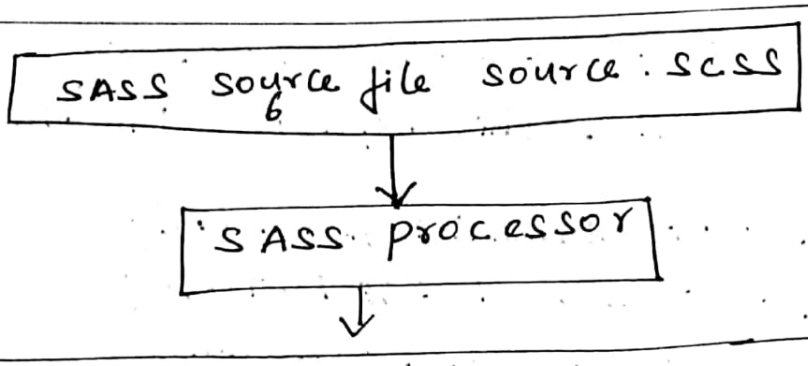
Shirly Noel
CMR

→ CSS preprocessors such as LESS, SASS and stylus provide this type of functionality.

eg:
```
$colorschemaA : #796d6d;
$colorschemaB : #9c9c9c;
$paddingCommon : 0.25em;

footer {
    background-color : $colorschemaA;
    padding : $paddingCommon * 2;
}
@mixin rectangle ($colorBack, $colorBorder)
{   border : solid 1Pt $colorBorder;
    margin : 3px;
    background-color : $colorBack;
}
fieldset {
    @include rectangle ($colorschemeB,
                        $colorschemeA);
}
.box {
    @include rectangle ($colorschemeA, $colorschemeB);
    padding : $paddingCommon;
}
```

fig : source.scss




```
footer {
    padding : 0.50em;
    background-color : #796d6d;
}
fieldset {
    border : solid 1pt #796d6d;
    margin : 3px;
    background-color : #9c9c9c;
}
.box {
    border : solid 1pt #9c9c9c;
    margin : 3px;
    background-color : #796d6d;
    padding : 0.25em;
}
```

Generated css file. style.css