CS2204 Fundamentals Of IAD

5. CSS - part II

This part of CSS covers mainly the layout and selected techniques of CSS3.

Layout is the positioning of blocks, written in HTML, within a Web page.

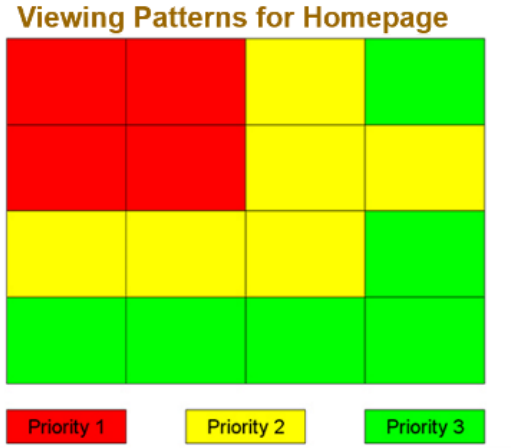
CSS3 was developed along with HTML5 to address limitations in CSS2 which could not handle a lot of basic requirements in common Web Apps.

5.1. Layout

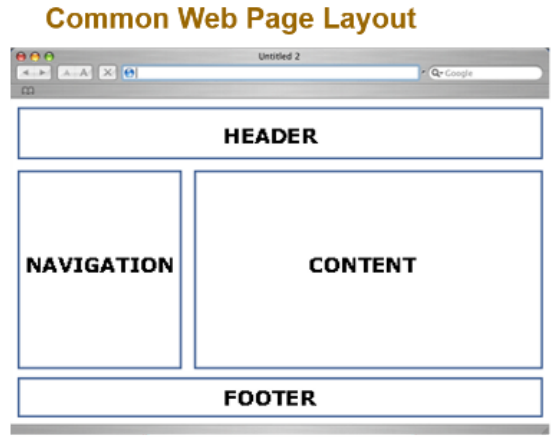
5.1. Layout

Layout refer to the arrangement/positioning of text and graphics. An appropriate layout enhances visitors' ability to read and find information on a page. It also creates visual interest to attract users visiting the Web site and generate traffic which is important nowadays.

Viewing pattern studies lead to some common, typical layouts. Note that modern Web sites usually change their layout frequently, may be in 6 months or even shorter periods. Ease of change is therefore an important consideration. Latest development not seen in the past also asks for support of multiple views/devices for Web sites - Responsive Design.



Ruel, L. and Outing, S. (2006). *Viewing Patterns for Homepages.*



ADC. (2006). *Web Page Development: Best Practices.*

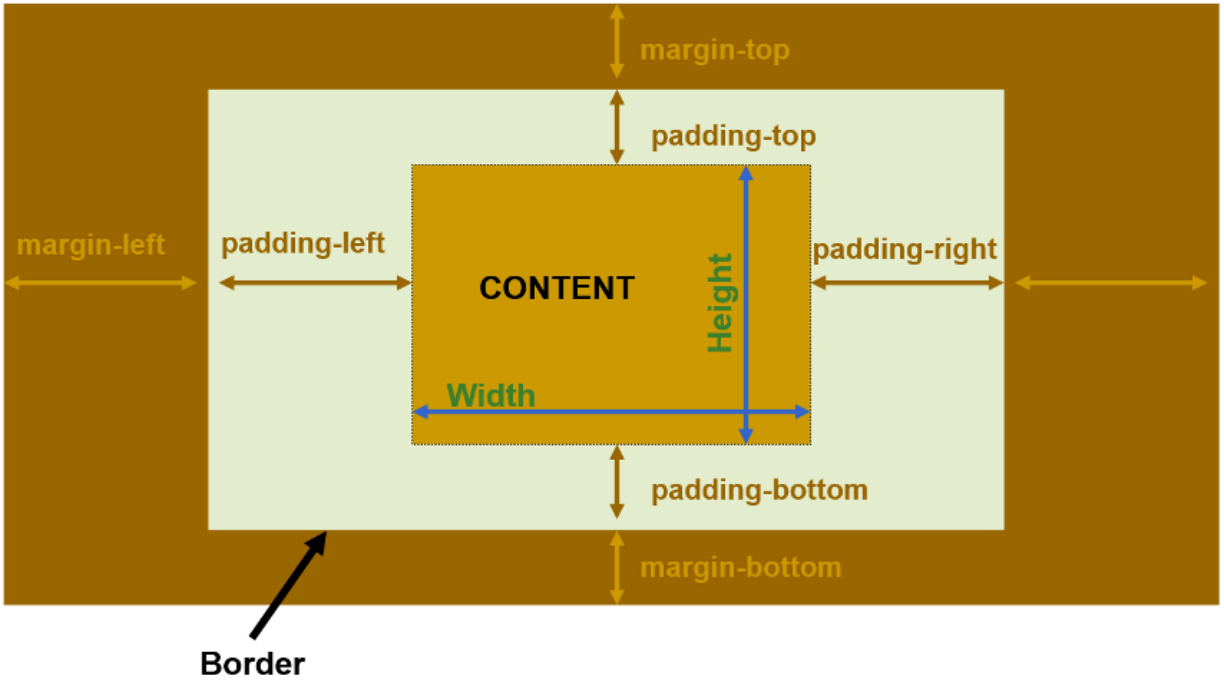
5.1.1. The box model

What is a block? Recalling in a Web page, a HTML element can contain other element(s). Therefore a block can be any HTML tag depending on at which level we are looking.

Each tag can be treated as a discrete element box on the screen and controlled by CSS, with the following properties:

- Content: at the center of the box; includes all decendant tags
- Width & Height: the dimensions of the content area
- Padding: the space between the border and the content of the element; background colors and images will also fill this space
- Border: a line that surrounds the element; invisible unless its color, width, and style are set
- Margin: the space between the border of the element and other elements in the window or container

The Box Model



5.1.2. Margins

5.1.2. Margins

Used to set the space between that element and other elements in the same container (could be the whole window if it is the outermost one). The property can be a single

```
<margin>
```

or margins of 4 sides:

```
<margin-left>, <margin-top>, ...
```

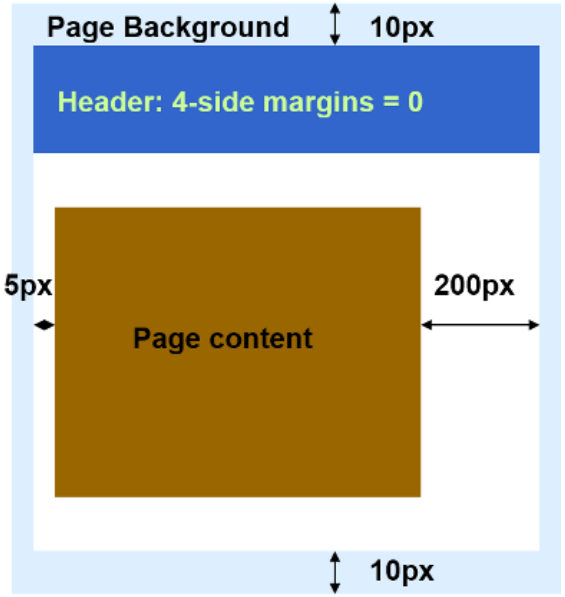
For the value, it can be set in different ways:

- Length - in an absolute unit, such as px
- Percentage - with reference to the parent element's width. What if the parent's width is also in percentage? The value will be calculated by going up one level to the grand parent, or until to the ultimate ancestor (Window) to get a value setting
- Auto - leave to the browser's calculation, usually used for centering, e.g.

  ```
  {magin: 5% auto;}
  ```

  the browser will base on the content's width which is not known when we are writing the page to calculate the left and right margins so that the block can be centered.

margin: <value for all sides>
margin: <top/bottom> <left/right>
margin: <top> <left/right> <bottom>
margin: <top> <right> <bottom> <left>



5.1.3. Border

5.1.3. Border

The border property is used to set a line around your box on all four sides of any width, color and style. Can also set individual properties : border-width, border-color and border-style. Values can be:

- border-width : | thin | medium | thick | inherit
- border-color : | transparent | inherit
- border-style : dotted | dashed | solid | double | groove | ridge | inset | outset | none | inherit

Different sides can be set as border-top, border-left, etc.

Border can be useful in knowing the exact position of a block when we work with complicate layout. Remember the universal selector?

```
* {border: 1px solid gold;}
```

this will give all elements a border so that we can check the positions. It can be removed easily too after debugging.


border: <value for all sides>
border : <top/bottom> <left/right>
border : <top> <left/right> <bottom>
border : <top> <right> <bottom> <left>

5.1.4. Padding

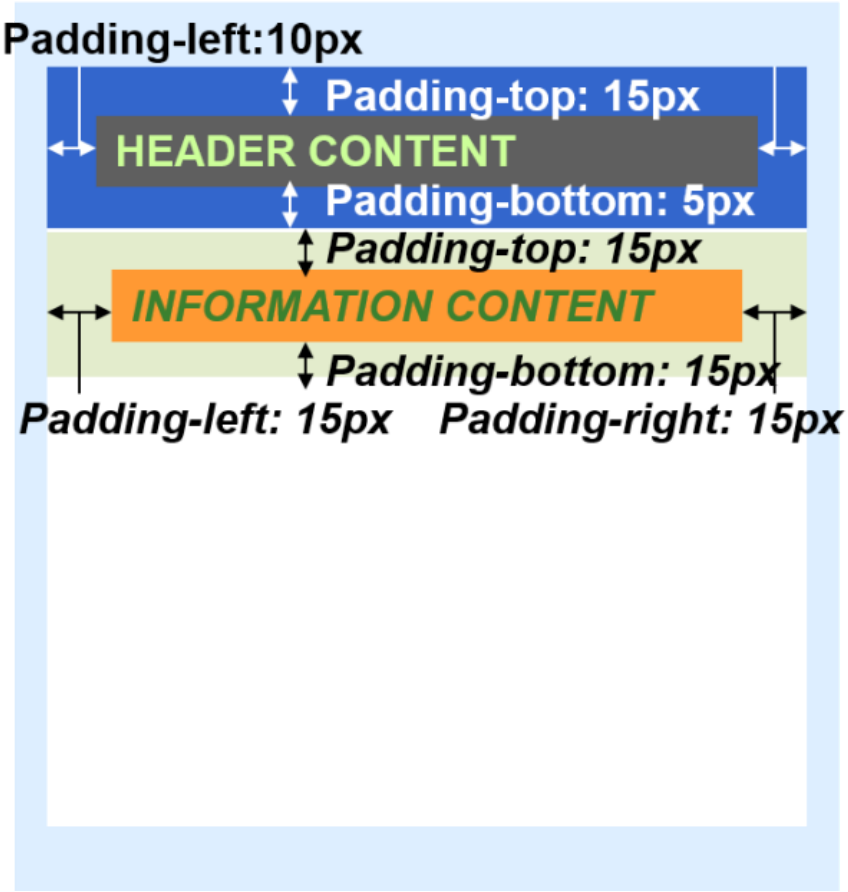
5.1.4. Padding

The padding property is used to add space between the border and the content. Useful if you do not want the content to be right next to the edges (border) which could make contents look very packed. It is often confused with the use of margin.

Setting and values are similar to margin or border:

- Length
- Percentage - set a padding proportional to the parent element's width

```
padding : <value for all sides>
padding : <top/bottom> <left/right>
padding : <top> <left/right> <bottom>
padding : <top> <right> <bottom> <left>
```



5.1.5. Width & Height

5.1.5. Width & Height

By default, the browser sets the width and height to auto and calculate the value on rendering. The effect is try to display all the content (for image or video, the original dimensions will be used). The width and height properties can be used to override the default setting. Note that a width setting has no effect if the element is of type inline, e.g. <img>, <a>, etc.

Value:

- Length
- Percentage - refer to the parent element's setting
- auto - the default, calculated by the browser on rendering

It is a bit tricky to use percentage for height which is usually not known. To make it work, a % setting has to be used up to the highest level, i.e. <html>, then <body>, etc.

```
html {height: 99%;}
body {height: 99%;}
```

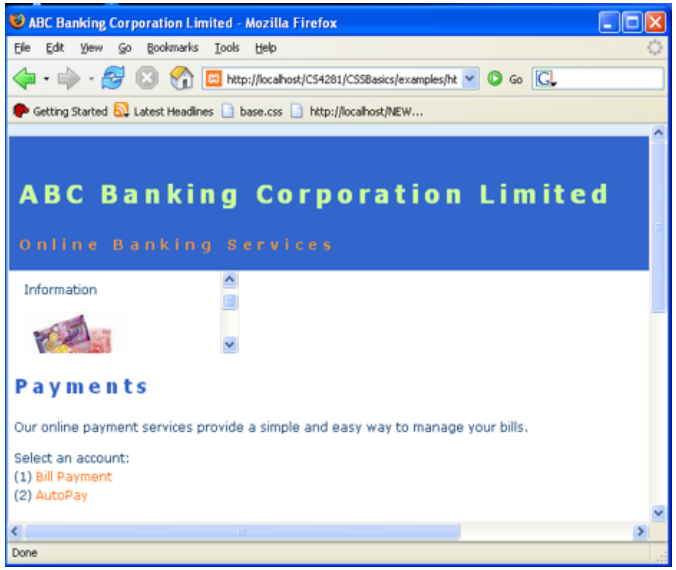
5.1.6. The overflow property

Once height/width of an element is set, the content might be larger than the space available (i.e. the size of the container). Different situations may happen, some content is not displayed, scroll bars appear or content lies partly or entirely outside of the box. This is known as overflow. The overflow property can be used to specify what should be done.

Value:

- Scroll, which sets scroll bars around the visible area
- Hidden, which hides the overflow
- Visible, which forces the cropped part of the element to show up
- Auto, which allows the browser to decide whether scroll bars need to be displayed and the content is shown inside the container

Container blocks should be set with auto to avoid complication.



5.2. Structuring your page for layout

5.2. Structuring your page for layout

Consider a typical layout with 4 sections:

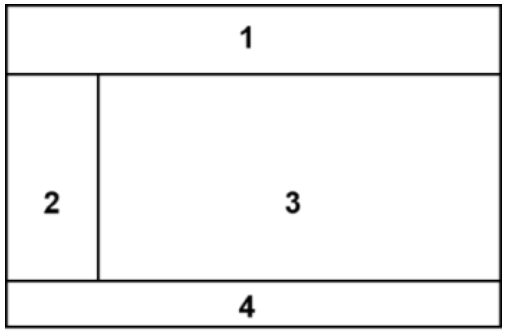
- Header
- Content
- Navigation
- Footer

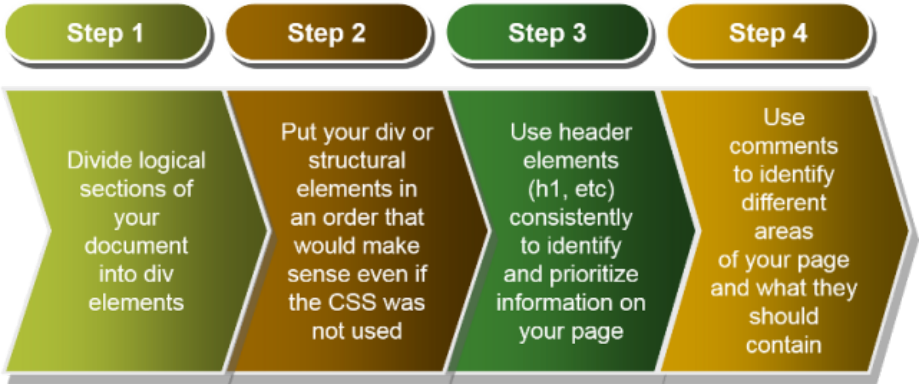
Recall the new structural tags in HTML5, may use them: <header>, <section>, <nav>, <footer>

Use the steps below in the diagram to come up with the HTML

```
<div id="container">
  <div id="header>…</div>
  <div id="content">…</div>
  <div id="navigation">…</div>
  <div id="footer">…</div>
</div>
```

what is an order that make sense?





5.3. Positioning

5.3. Positioning

Layout depends on how the browser puts elements into containers. This is called positioning and there are 4 positioning schemes.

Static:

- this is the default, also known as normal flow
- just lay out the content normally according to sequence and display property (block or inline)
- no effect for top/bottom/left/right property values
- width and height of container, the float, clear or overflow property can affect the flow
- in normal flow, there is no overlap

Other schemes include: fixed, absolute and relative. Overlap may occur

5.3.1. Static Positioning - Normal Flow

In this positioning, browser places elements (boxes) one by one into container(s) starting from the window. Positioning then depends the containers' dimension and overflow properties, the elements' width and height, display properties (block or in-line, ...). Let's look at the effect of the display property.

Display: block

- generates a block box
- goes to a new line
- can set dimension, i.e. width, height, etc.
- can set horizontal and vertical margins, vertical margins will be collapsed - max(margin-bottom, margin-top), i.e. take the larger one

Display: inline

- generates an inline box
- layout on the same line
- cannot set dimensions
- can only set horizontal margin

- in general, don't put block element inside inline element

Display: none

- make the element disappear (but the element still exists in the DOM)
- does not take up any space (no box generated)
- compare with the property visibility: hidden

Display: inline-block

- positions like inline but behaves like block, therefore width and height can be set
- useful to style label and input elements in forms

Others:

- e.g. table-cell, flex (CSS3) more advanced
- can explore as extension reading

5.3.1.1. Float property

The basic technique to set multi-column designs in normal flow.

```
{float: value;}
```

where value can be left, right or none. Note that this the value of float property, not to be confused with the properties left, right, top and bottom discussed later.

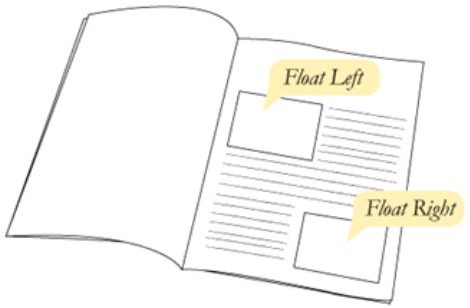
Definition in HTML standard : "A float is a box that is shifted to the left or right on the current line. The most interesting characteristic of a float (or “floated” or “floating” box) is that content may flow along its side (or be prohibited from doing so by the “clear” property). Content flows down the right side of a left-floated box and down the left side of a right floated box".

Easier to understand explanation:

- right - put the selected element to the right of the next following element horizontally on the same line (if the width is wide enough)
- left - put the selected element to the left of the next following element horizontally on the same line (if the width is wide enough)
- none - which overrides floating for this element, usually used in a sequence of float to stop a certain element floating

```
li {float: left;}
li:nth-child(3) {float: none;}
```

Remember the float of an element is relative to the element following it as written in HTML code.



https://css-tricks.com/all-about-floats/

5.3.1.2. Float - 2 columns design | **Example 1 - http://courses.cs.cityu.edu.hk/cs2204/example/html/14-FloatElement.html**

5.3.1.2. Float - 2 columns design

Consider the following HTML:

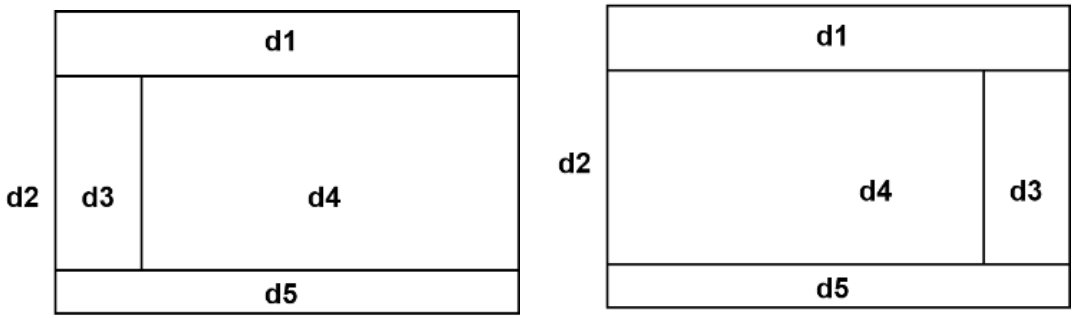
```
<div id="d1"></div>
  <div id="d2">
    <div id="d3"></div>
    <div id="d4"></div>
   </div>
<div id="d5"></div>
```

Float left for menu

- #d3 {float: left;}
- may need to set margin-left of #d4

Float right for menu

- #d3 {float: right;}
- d3 float right relative to the following element (i.e. d4)



5.3.1.3. Float - 3 columns design

5.3.1.3. Float - 3 columns design

Consider the following HTML:

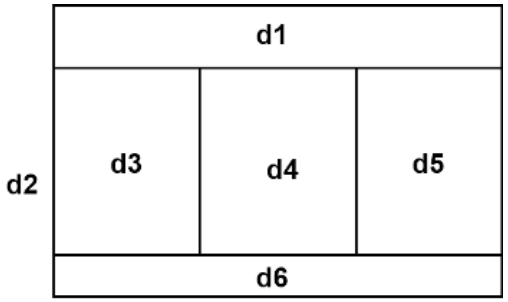
```
<div id="d1"></div>
  <div id="d2">
    <div id="d3"></div>
    <div id="d4"></div>
    <div id="d5"></div>
   </div>
  <div id="d6">
</div>
```

Float left:

- #d3, #d4 {float: left;}
- no need to use #d5 {float: right;}, may cause problem
- #d3, #d4, #d5 {float: left;} may work because all of them are contained inside d2

Alternate way to do the same:

- rearrange sequence d3, d5, d4
- #d3 {float: left;} #d5 {float: right;}
- d3 float left relative to following element, d5 float right relative to d4
- difference? d3 and d5 stick to the left and right hand sides, with d4 takes up all space in the middle



5.3.1.4. Clear

5.3.1.4. Clear

To stop floating sequence at some point, {float: none:} can be used. It is also possible to do this by the clear property.

```
{float: left|right|both}
```

In the diagram below, the images and text (h3) are in a sequence of float. Depending on the width of the container, the text may or may not be always starting at a new row. To make sure it starts always on a new row, can stop the float of text by

```
{clear: both;}
```

i.e. no element should appear on its left and right.

Another common use of clear is to make sure an element starts on a new line, e.g. a footer to appear always in new row regardless how other blocks are floating (second diagram below).

Image Gallery

Try resizing the window to see what happens when the images does not have enough room.

Second row

Source: https://css-tricks.com/all-about-floats/

5.3.1.5. Column related property | **5.3.1.3. Float - 3 columns design** | **Clear example - https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_clear**

5.3.1.5. Column related property

Use of float property to design mult-column layout is fairly complicated although flexible. There are other choices available in the fast developing CSS3 techniques

One of these features is the column-count, a property to be set in the container and its content will be split into the assigned columns

```
<div id="container">
  <p>content....

  .....

  </p>
</div>

#container {
  column-count: 3;
}
```

There are column related properties to further manage the columns

5.3.2. Fixed, Absolute, & Relative Positioning

Fixed:

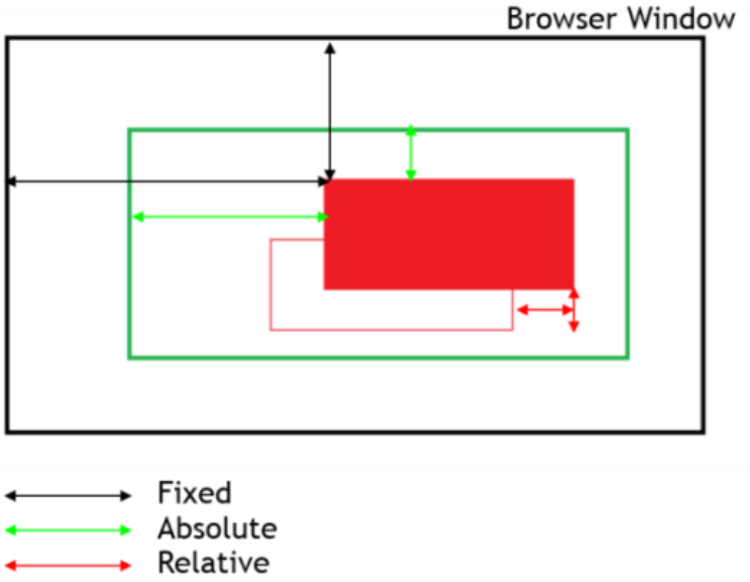
- controlled by top/bottom/left/right values relative to the initial containing block (browser window)
- fixed elements are removed from normal flow (as if they are not present in laying out other elements)
- can overlap with other elements, control by z-index

Absolute:

- controlled by top/bottom/left/right values relative to the first parent element with positioning, if not existing, go up, until up to browser window
- absolute elements are removed from normal flow
- used to control position within a container which can either be fixed or static in positioning
- can overlap with other elements, then control by z-index

Relative:

- first use normal flow and further adjusted by top/bottom/left/right values
- can overlap with other elements, then control by z-index



5.3.3. Z-index

5.3.3. Z-index

Consider the following HTML:

```
<div id="product">
 <img id="prod1" src=“ " alt=“ " />
 <img id=“prod2“ src=“ " alt=“ " />
</div>
```

with CSS rules:

```
#prod1 {position: relative; top: +15px; left: +15px; z-index: 1;}
#prod2 {position: relative; top: +100px; left: -15px; z-index: 0;}
```

The result? No z-index, the later elements will be on top. With z-index, the larger ones will be on top. Note that z-index value is evaluated absolutely, 99 > 0 > -3 > -99.

5.4. Fixed layout

5.4. Fixed layout

Not to be confused with Fixed Positioning. For normal flow (static positioning), layout will usually change if window (container) size changes, e.g. elements are forced into new line when the width is reduced. To prevent this layout change, fixed layout can be used.

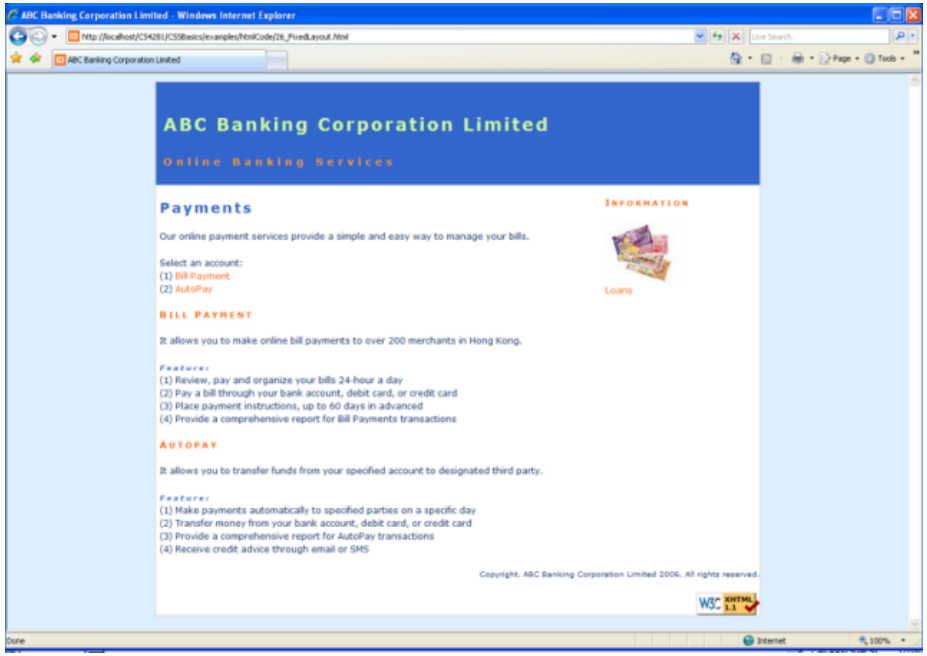
The technique is quite simple, use a big container (body or div) to hold the page and set its width (height) to a fixed, absolute value (e.g. 1024px).

Advantages:

- the pages look identical, no matter what window size (device) is used
- fixed width/height elements will not overpower text on smaller monitors

Limitations:

- cause horizontal/vertical scrolling in smaller browser windows
- result in large area of white space in larger screen
- cannot fit in small screen device


ABC Banking Corporation Limited - Windows Internet Explorer
ABC Banking Corporation Limited
Online Banking Services
Payments
Our online payment services provide a simple and easy way to manage your bills.
Select an account:
(1) Bill Payment
(2) AutoPay
BILL PAYMENT
It allows you to make online bill payments to over 200 merchants in Hong Kong.
Feature:
(1) Review, pay and organize your bills 24-hour a day
(2) Pay a bill through your bank account, debit card, or credit card
(3) Place payment instructions, up to 60 days in advanced
(4) Provide a comprehensive report for Bill Payments transactions
AUTOPAY
It allows you to transfer funds from your specified account to designated third party.
Feature:
(1) Make payments automatically to specified parties on a specific day
(2) Transfer money from your bank account, debit card, or credit card
(3) Provide a comprehensive report for AutoPay transactions
(4) Receive credit advice through email or SMS
INFORMATION
Loans
Copyright. ABC Banking Corporation Limited 2006. All rights reserved.

5.5. Liquid Layout

Also known as fluid layout. The size of elements will enlarge or shrink when the window (container) size changes. The technique is to set all dimensions in relative units, e.g. percentages. Since they are relative, dimensions enlarge or shrink according to the container dimension.

Advantages:

- pages expand and contract to fill the available space; all available screen space can be used
- provide consistency in relative width/height, allowing a page to handle requirement like larger font sizes compared to other content
- become an important concept in Responsive Design

Limitations:

- little precise control over the width/height of the various elements of the page
- result in columns of text that are too wide to be read comfortably, or too small for words to show up clearly on small screen
- have problem when a fixed width element is placed inside a liquid column, e.g. the column is rendered without enough space for an image, browsers will either

increase the column width, or cause overlap in text and image

5.6. Mobile friendly layout

What are the differences? Different versions: desktop, tablet, phones.

Navigation, Scroll & Tab:

- no mouse and no mouseover (hover)
- swipe to scroll - horizontal and vertical
- tab to click
- different drag & drop in different devices
- orientation may change - portrait & landscape

Smaller Screen and many different sizes: desktop > 1024 pixels, iPhone 5 - 1136 x 640, iPhone 4 - 960 x 640, iPad - 1024 x 768, Samsung Galaxy S4 - 1920 x 1080, ... and so much more

Text & Images Resizing:

- bigger font size in mobile looks better
- use relative size : desktop - 1, Android - 2, iPhone - 1.75 and iPad - 1.5 (by trial & error - suggested in "HTML5 for iOS & Android")

- images scale down better than scale up

5.6.1. Viewport

5.6.1. Viewport

Viewport is a rectangle (window) through which you see your document

Simple in desktop but become messy in mobile browsers

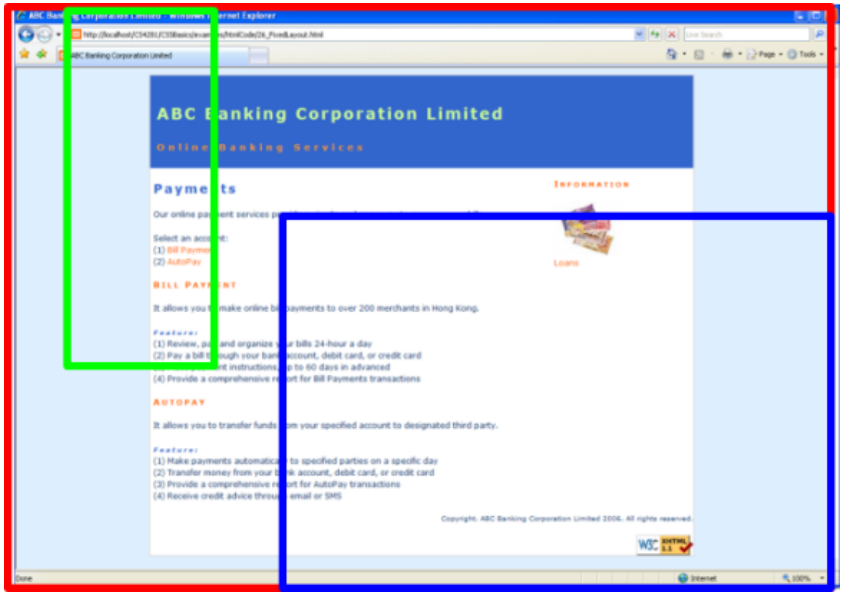
- equal to the window size if no zoom
- zoom in - viewport becomes smaller, can only see part of the document
- zoom out - larger

Most mobile browsers again by default use 100% zoom (of your original page)

- your page becomes real small to fit in the small screen
- can use the viewport meta tag (originally invented by Apple)

```
<meta name="viewport" content="width=320">
<meta name="viewport" content="width=device-width">
<meta name="“viewport” content="“width=device-width, initial-scale=1.0”> - this
is commonly used as default
```

very messy, different browser (in different platform) behaves differently, e.g. how to scale and using default viewport size



5.6.2. Responsive Layout Design

5.6.2. Responsive Layout Design

Responsive layout is one that can respond to different environment (device). Can use the following steps.

Liquid/fluid layout

- already learnt about fix and liquid layout
- we know elements with fix sizes will have problem, should always use relative sizes

Think in terms of a fluid grid (rows & columns but not using table) - also known as Adaptive Layout

- make the grid size relative
- put elements into the grids, make all sizes relative (percentage)
- let them flow - the browser puts the grids into the window

Use media query to make it fully responsive

- learnt about media type already

```
<link rel="stylesheet" type="text/css" href="core.css“
media="screen">
<link rel="stylesheet" type="text/css" href="print.css“
media="print">
```

- CSS3 introduces more media query

```
<link rel="stylesheet" type="text/css“ media="screen and (max-
device-width: 480px)" href="shetland.css">
media="screen and (max-device-width: 480px) and (resolution:
163dpi)“
@media screen and (max-device-width: 480px) { … }
@import url("shetland.css") screen and (max-device-width: 480px);
```

Write up the different style sheets for different media.

5.6.3. Responsive Layout Example

Non responsive design

- will not adapt to screen size change
- mobile browsers usually scale the whole page to the small screen

Responsive Design - detect orientation using media query
Landscape mode

- 2 columns design
- fit as many images along a row as possible

Portrait mode

- single column
- the menu links have more space, easier to tab
- one image in a row with caption at the right

all margins, sizes are relative

5.7. Screen layout partitioning - grids | **Non responsive - http://courses.cs.cityu.edu.hk/cs2204/example/html/18-NonResponsiveDesign.html** | **Responisve - http://courses.cs.cityu.edu.hk/cs2204/example/html/18-ResponsiveDesign.html**

5.7. Screen layout partitioning - grids

So far learnt two ways to partition a layout:

- table - obsolete way, should not be used because of maintenance problem
- structural tags: div, header, footer, section & article - good for most situations; cannot reuse external page (i.e. need to duplicate code in different pages)

The third way iframe - can reuse HTML codes. Some older Web pages use <frameset>, not supported in HTML5 anymore and is not compatible with the concept of CSS.

5.7.1. iframe

Allows

- insertion of a frame containing a block of external HTML and can be aligned with surrounding tags
- controlled by CSS

```
<iframe id="pageContent" name="PaymentFrame" src="21_IFramePayment.html">
<--! fall back code -->[Your user agent does not support frames or is
currently configured not to display iframes.]
</iframe>
```

Limitation

- dynamic content may not fit in well with the default height of an iframe, the actual height is not known when the page is written
- may create inconvenience for cross domain content - content of iframe not coming from your own Web site, e.g. local file testing iframe with other site content

5.8. Selected CSS3 techniques

Selected techniques which are not available in CSS2. A lot more CSS3 properties are now supported by browsers, should explore as extension reading.

- Round Corners
- Box Shadow
- Gradient
- Transform
- Transition
- Animation

5.8.1. Round border corner & Box shadow

Surprisingly difficult to do this before CSS3. Draw an arc for the round corner, specify the radius of the circle which the arc belongs.

- `border: 2px solid red; border-radius: 24px;`
- what is border-radius: 50%?
- can use 4 numbers:

  `border-radius: 24px 0 24px 0;`

Create a shadow for a box

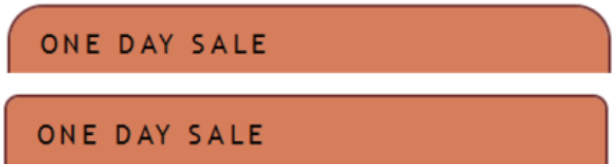
`box-shadow: x, y, width, color`

- x, y - position of the box relative to element
- width of shadow shown

You may see CSS3 properties in older Web pages as:

- border-radius: 10px; (official standard)
- -webkit-border-radius: 10px; (Safari, Chrome)
- -moz-border-radius: 10px; (Firefox)

this is known as vendor-specific prefixes used for a particular browser when the property is not supported by all as a standared property; other browsers would ignore the property.



5.8.2. Gradient color | **Round corner example - https://www.w3schools.com/cssref/css3_pr_border-radius.asp** | **Shadow example - http://www.w3schools.com/cssref/tryit.asp?filename=trycss3_box-shadow**

5.8.2. Gradient color

In CSS2, it is not easy to create a color composed of gradual change of two colors which is called gradient, e.g.

```
background: -webkit-linear-gradient(top, #FF8000, #FFFF00);
```

Specify 2 colors (start & end) and the browser would "interpolate" the colors in between. Note the result is a special type of image and therefore cannot be used as a value for any color related property. Claimed to be supported without vendor prefix but seems not the case(?) Need to verify.

Linear gradient

- top to bottom : linear-gradient(yellow, green);
- left to right : linear-gradient(left, yellow, green);
- angle : linear-gradient(top left, yellow, green);

Radial gradient:

- circle : radial-gradient(circle, yellow, green);
- eclipse : radial-gradient(eclipse, yellow, green);

5.8.3. Transform | **w3schools example - https://www.w3schools.com/css/css3_gradients.asp** | **CSS3 techniques - http://courses.cs.cityu.edu.hk/cs2204/example/html/16-CSS3AdvanceTech.html**

5.8.3. Transform

Transform and/or animation can create visually interesting effect for Web pages. As the property name implies, transform performs transformation on the orginal style (position) of an element to the target style (position). Only simple transform is introduced. More sophisticated effect should be explored as extension reading.

Transform is the property and the value is the result of different value functions:

- `transform: translate(10px, 10px);`
- `transform: scale(2, 1.5);`
- `transform: rotate(45deg);`
- `the default transform origin is at centre`

Can be applied in 3D:

- `transform: rotateZ(90deg);`
- `transform: scaleY(y);`

5.8.4. Transition

5.8.4. Transition

Best understood as 2 states of a property changing gradually over a period of time, usually associate with an event to cause the property change.

```
#trans {
 transition: background-color 3s linear;
 -webkit-transition: background-color 3s linear;
}
#trans:hover {
 background-color: #004080;
}
```

The property is background-color:

- initial state is the original background color
- final state is background color #004080
- triggering event is mouse over (:hover)
- different from normal :hover (i.e. without transition) is the gradual constant change (linear) over a period of 3 seconds rather than an instant change

The linear value is called the speed curve used to define the speed of change over the duration, can be:

- ease - specifies a transition effect with a slow start, then fast, then end slowly (this is default)
- linear - specifies a transition effect with the same speed from start to end
- ease-in - specifies a transition effect with a slow start
- ease-out - specifies a transition effect with a slow end
- ease-in-out - specifies a transition effect with a slow start and end
- cubic-bezier(n,n,n,n) - lets you define your own values in a cubic-bezier function

5.8.5. Animation

An extension of transition by having more complicated properties and styles; can have iteration count (to perform the animation for a number of times) and more than 2 states. Introduce the key frame concept:

```
/* The animation code */
@keyframes example {
    0%   {background-color:red; left:0px; top:0px;}
    25%  {background-color:yellow; left:200px; top:0px;}
    50%  {background-color:blue; left:200px; top:200px;}
    75%  {background-color:green; left:0px; top:200px;}
    100% {background-color:red; left:0px; top:0px;}
}
/* The element to apply the animation to */
div {
    width: 100px;
    height: 100px;
    position: relative;
    background-color: red;
    animation-name: example;
    animation-duration: 4s;
}
```

5.9. Conclusion

Critical thinking:

- CSS looks complex, why not use table?
- CSS may look different in different browsers, any way to control?
- Seems only a few techniques have been learnt, what is next?

Remember CSS is declarative? Quite a lot of redundant codings are required for large projects. It would be nice if CSS can be turned into a programming language with features like variable, function, if-then-else, loop and extend, etc. These can be done with pre-processors such as SASS, LESS or Stylus but we cannot teach all these in an introductory course.

Dynamic Drive CSS Layouts

W3C CSS Tips & Tricks