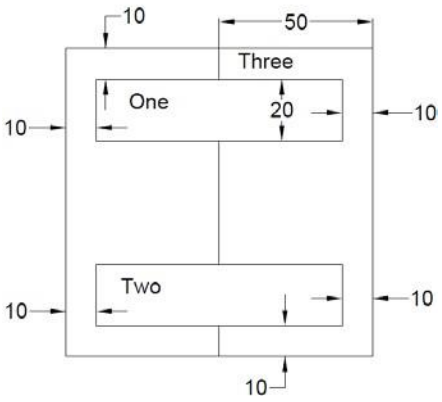
Sample exam question
Open-book & open-Internet
No discussion & collaboration between students, however.

Question 1

Consider the following piece of HTML:

```
<body>
<div id="one">One</div>
<div id="two">Two</div>
<div id="three">Three</div>
</body>
```



Suppose that you wanted the layout to look like the picture at right, where all dimensions are a percentage of the window size (i.e., 10 means 10%). Write a short set of CSS rules that would produce the desired result. (Do not worry about the coloring or the borders; just the dimensions and positioning.)

The border-radius property adds rounded corners to any element.

Question 2

In this problem, you will write CSS with the provided HTML to produce the expected page output below (appearance details are given to supplement the expected output image where needed).

Provided HTML `body`:

```
<body>
  <section>
    <header>
      <h1>Baby Sloth!</h1>
    </header>
    <div id="s">
      <div id="l">
        <span id="o"></span>
        <span id="t"></span>
        <span id="h"></span>
      </div>
    </div>
  </section>
</body>
```

Appearance Details:

- The `section` is 40% of the page's width. The text and sloth image are centered on the page.
- The `h1` text has a font family of Helvetica, falling back to Arial if Helvetica is not available on the system, falling back to the system's `sans-serif` default font if Arial is also not available.
- The background color of the outermost circle is `sienna` and the background color of the inner circle containing the "eyes" and "nose" (which each have a `black` background) of the sloth is `peru`.
- The two `div` circles have a 1px-width solid black border and a border radius of 50%.
- The outermost circle has a width and height of 300px. Its inner circle has a width and height of 110px.
- The `#t span` is positioned in the center of the face and has a height of 25px - the `#o` and `#h` spans both have a height of 20px and touch the very left and right borders of the parent `#l div`, respectively. All `span` elements have a width of 20px and a border radius of 50%.

Question 3:

https://eclass.yorku.ca/eclass/mod/lesson/view.php?id=307326

Question 4:

Explain what this script does:

```
#!/bin/bash
for  I   in *.mp3 ; do
   ffmpeg  -i   "$i"    $(basename "${i/.mp3}").ogg
done
```

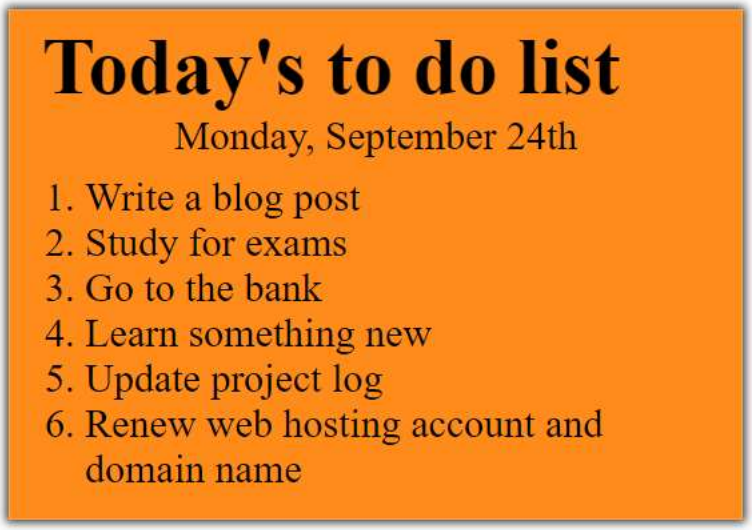
Question 5

https://eclass.yorku.ca/eclass/mod/lesson/view.php?id=307326

https://eclass.yorku.ca/eclass/mod/lesson/view.php?id=385415

Problem 6.

Create a Web Page like the following. Write your own to do list!



Create two files: "**to-do-list.html**" and "**to-do-list.css**"

Constraints

- ** ** with **class="my-list"** as container
  - Background color: **#ff8c1a;**
  - Padding: **8px 24px**;
  - Width: **500px**;
  - Text size: **30px;**
  - Border: **1px solid #ffa64d**;
  - box-shadow: **0 0 10px 2px #333333;**
- **<ol>**
  - Margin: **12px;**
- **<p>**
  - Center the text

Problem7.

Create a Web Page like the screenshot below.

Create two files: "**simple-article.html**" and **"simple-article.css".** Use **semantic HTML** to create HTML the page.

Constraints

- **`<body>`**
  - **`font-family:"Lato", sans-serif;`**
    "https://fonts.googleapis.com/css?family=Lato:300,400,900"
- Content
  - Width: **`550px;`**
  - Border radius: **`15px;`**
  - Center the **`content`**
  - Padding: **`25px;`**
  - Background color: **`#fafafa;`**
  - **`box-shadow: 10px 11px 35px -14px rgba(0,0,0,0.54);`**
- **`<h1>`** tag for heading
  - Margin bottom: **`0;`**
  - Padding: **`10px;`**
  - Text center
  - Letter spacing: **`5px;`**

Question 8: Short answer questions

1. Write a CSS style specification rule that would make all unordered lists (`<ul>` tags) have square bullets and a purple background.

2. Write a rule that would make this, and all other paragraphs appear in 12-point font.

3. Write a rule that would make this and other paragraphs in the same class as this example appear in italic type.

4. Write a rule that would make this paragraph only appear in bold type.

- **Image**
  - Width: **`200px;`**
  - Border radius: **`50%;`**
  - Margin: **`15px;`**
  - Border: **`5px dotted #8b603d;`**
  - Use "**left floating**" for the image: https://css-tricks.com/almanac/properties/f/float/
- **Paragraphs**
  - Text size: **`14px;`**
  - Use "**`text-align: justify`**" to align the text left and right (learn more at http://www.w3schools.com/css/css_text.asp).

Itec/MODR 2635 EN
Fall 2020

Homework 1: Submission LINK

What to submit: a PDF file with answer to questions, with LINKS to your results (web pages).

Moodle: https://eclass.yorku.ca

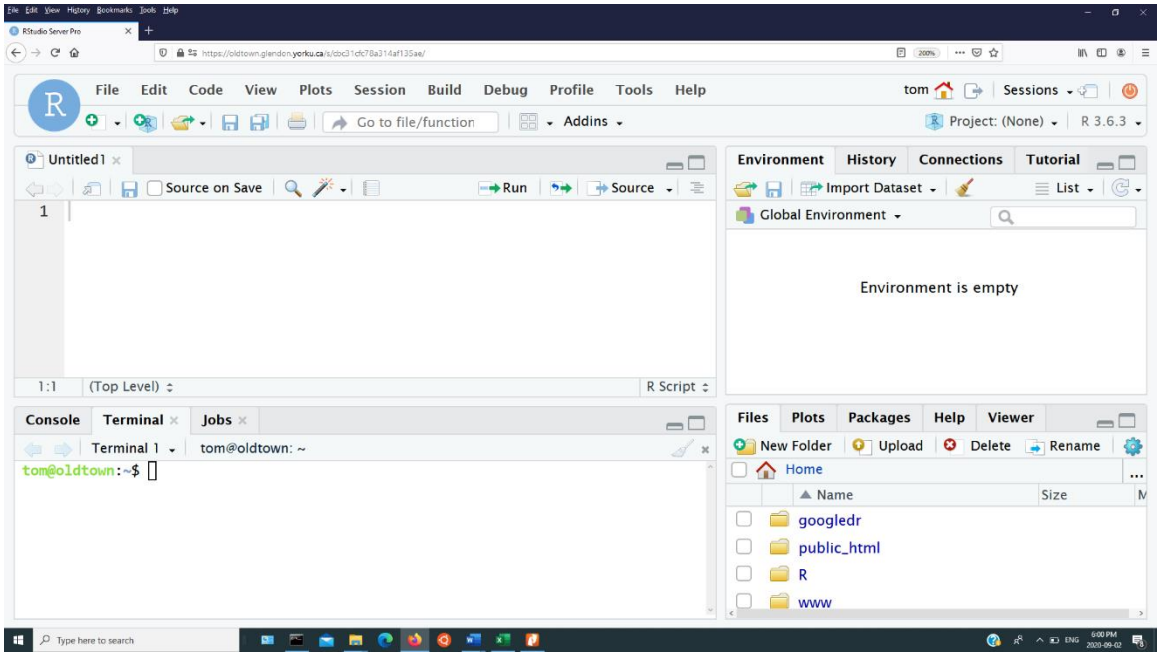
Rstudio-server: https://oldtown.glendon.yorku.ca/
Username: yourlastname (always in small letters)
Password: your_student_number
Your web pages are at: **http://oldtown.glendon.yorku.ca/~yourlastname**

Login our server at: https://oldtown.glendon.yorku.ca/
Username: yourlastname (always in small letters)
Password: your_student_number

- For example, if your full name is "Peter Toft", then you web site is http://oldtown.glendon.yorku.ca/~toft/
- If your last name is "Peter El Hare", then your web site is http://oldtown.glendon.yorku.ca/~elhare/ (last name is El Hare)
- There are 2 students D Chen and J Chen. So the web is for each student: http://oldtown.glendon.yorku.ca/~dchen/ http://oldtown.glendon.yorku.ca/~jchen/
- **Your web pages are in the directory www:** /home/yourlastname/www
- **Your home directory is /home/yourlastname**
- **~** is the abbreviation for **/home/lastname**

You will have next this graphic, or graphical user interface (GUI) below:

-



Click on "terminal" to activate that console in order to type command line (CLI), followed by "return" key. So we are giving a command to the computer (operating system) by typing a text (a keyword); more on Human–computer interaction.

Utility	Description
`ls`	"list" files and directories
`pwd`	"print working directory"
`cd`	"change (your working) directory"
`mkdir`	"make directory"
`rmdir`	"remove directory"
`cp`	"copy" a file or directory
`mv`	"move" a file or directory (i.e., rename it)
`rm`	"remove" a file (i.e., delete it)

Displaying Text Files

It is often convenient to look at the contents of a text file without having to open it in an editor. Previously in this lab, we saw that `cat` can be used for this purpose, but it is most useful for short files that can be viewed all on one screen.

GNU/Linux provides several other utilities that are useful for "paging" through text files (i.e., for viewing files one page at a time). Several of these commands are outlined in the following table.

Command	Description
`more`	move through a file screen by screen (hit space for the next page, return for one more line)

Exercise 1: 10 points

Definition (Wiki): A **command-line interface** (**CLI**) processes commands to a computer program in the form of lines of text. (Issue a command by typing text at a **terminal,** then press return.)

Try the following CLI:

(a) cd www
(b) mkdir 2021
(c) pwd
(d) cd 2021
(e) pwd
(f) mkdir images
(g) cd images
(h) convert -size 800x800 canvas:white white.png
(i) convert -size 800x800 canvas:blue blue.png
(j) Visit: http://oldtown.glendon.yorku.ca/~lastname/2021/images
(k) Download from http://oldtown.glendon.yorku.ca/~teaching/2915/l3/codes001.zip and display the content of this ZIP file at http://oldtown.glendon.yorku.ca/~lastname/2915/l3/

Exercise 2: 10 points

(a) In a terminal console, put yourself in Directory
/home/lastname/www/2021/images
using CLI (command line interface)

(b) In the GUI (graphical user interface) copy the song petit_navire.mp3 from
http://oldtown.glendon.yorku.ca/~teaching/music/mp3/petit_navire.mp3
to your directory:
/home/lastname/www/2021/images (i.e. Your PATH you made in (a)).

(c) Execute the CLI:

```
ffmpeg  -i blue.png -i petit_navire.mp3 -c:v libx264 -pix_fmt yuv420p
mymusic.mp4
```

Exercise 3: 10 points

#	Operation	What is the Command?
1	list files & directories.	
2	Put yourself at the HOME directory	
3	make a sub-directory called "**myfile**" in the home directory.	
4	Put yourself at the PATH /home/lastname/myfile	
5	Show your current PATH	
6	show the path for where you are in the directory.	
7	Make a subdirectory at /home/lastname/myfile/mp3	
8	list the contents of the home directory, no matter where you are.	
9	Put yourself at ~/myfile/mp3	
10	Go to ~ (the **HOME** directory which is /home/lastname)	
11		
12		
13		
14		
15		
16		
17		
18		
19		

20		

HTML Introduction

What is HTML?

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

---

A Simple HTML Document

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Example Explained

- The `<!DOCTYPE html>` declaration defines this document to be HTML5
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the document
- The `<title>` element specifies a title for the document
- The `<body>` element contains the visible page content
- The `<h1>` element defines a large heading
- The `<p>` element defines a paragraph

HTML Tags

HTML tags are element names surrounded by angle brackets:

<tagname>content goes here...</tagname>

- HTML tags normally come **in pairs** like and
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

Tip: The start tag is also called the **opening tag**, and the end tag the **closing tag**.

Make a web page with the html5 code given above on page 5, to be displayed at:

http://oldtown.glendon.yorku.ca/~yourlastname/examples/ex1

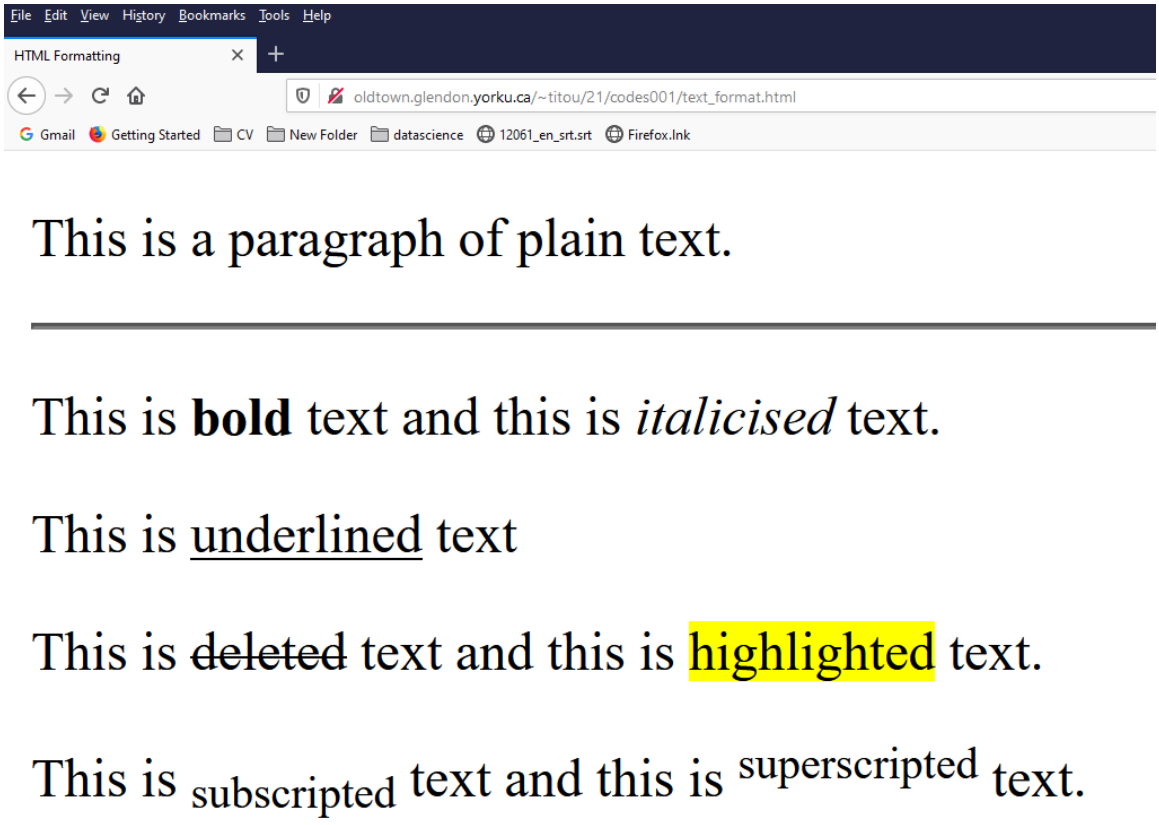
Exercise 4: 10 points

Complete the html code given below:

```
<!DOCTYPE html>
<html>
<head>
 <title>HTML Formatting</title>
 <meta charset="utf-8"/>
</head>

<body>
<p>This is a paragraph of plain text.</p>
<hr>
<p>This is strong text and this is italicised< text.</p>
<p>This is underlined text</p>
<p>This is deleted text and this is highlighted text.</p>
<p>This is subscriptedtext and this is superscripted text.</p>
<!-- this is a comment -->
</body>
</html>
```

So as to give this result below at your URL:

http://oldtown.glendon.yorku.ca/~lastname/2635/text/format/



Exercise 5: html5 10 points

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Ma passion pour les animaux</title>
  </head>

  <body>
    <h1>Titre de niveau 1</h1>
    <p>Voici mes animaux:</p>
    <ol>
      <li><strong>Dragon</strong></li>
      <li>chat</li>
      <li>giraffe</li>
    </ol>
  </body>
</html>
```

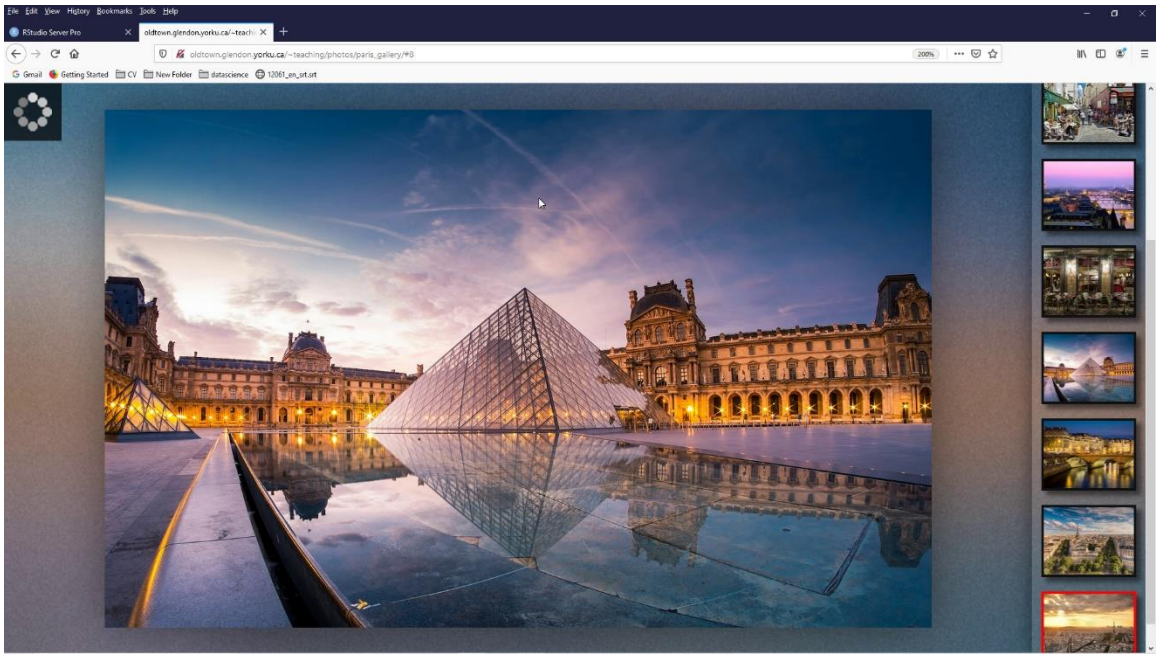
Make a web page with the html5 code given above to be displayed at:

http://oldtown.glendon.yorku.ca/~yourlastname/examples/ex212

Exercise 6: 10 points .Making a photo gallery with CLI.

(a) View, repeat and explain each of the CLI given in this video:

http://oldtown.glendon.yorku.ca/~teaching/2021/l1/ex004.mp4



(b)
The program is fgallery. The syntax (how to use it) is:

fgallery input_directory output_directory

I gave you the images paris.zip in your directory /home/lastname/www/examples/. Make a photo gallery with these images at your URL:

http://oldtown.glendon.yorku.ca/~lastname/vacances/paris

List all your steps.

Exercise 7: 10 points.

Your web page is at: **http://oldtown.glendon.yorku.ca/~yourlastname**

Visit the following links:

- http://oldtown.glendon.yorku.ca/~yourlastname/music/mp3

    - What should you do so that the URL link above be renamed as: http://oldtown.glendon.yorku.ca/~yourlastname/music/french2

    - What should you do so that that your Paris album is at this location: http://oldtown.glendon.yorku.ca/~yourlastname/paris/memory/french2

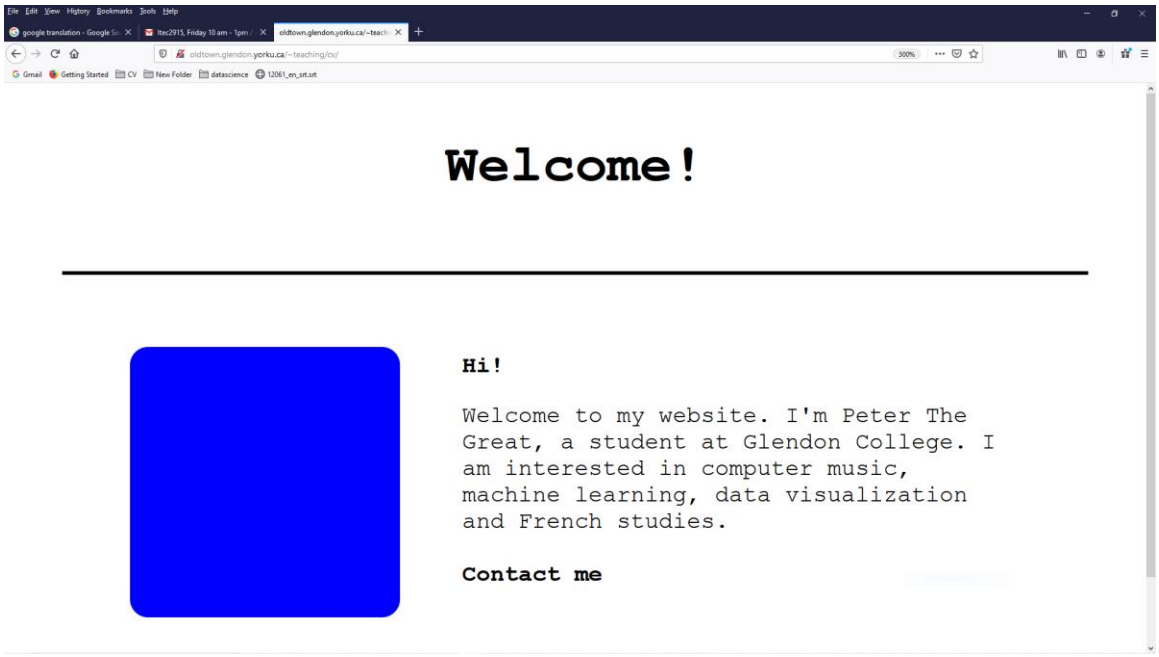
- http://oldtown.glendon.yorku.ca/~yourlastname/cv/index.html

    - How would you replace the ”blue box” with your own photo?
    - How would you replace “Welcome” by “Bienvenue!”
    - See: https://youtu.be/ZRfsCQPGiS4
    - or http://oldtown.glendon.yorku.ca/~teaching/2021/syllabus/oldtown_rstudio_server_title_cv.mp4

Hint: Edit the file index.html at /home/lastname/www/cv
Replace myphoto.png by your own image with the same name

Here is the html and css codes for the web page above:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="css/cv.css">
<!-- Make for my students in Itec at Glendon College -->
</head>
<body>

<div id="header">
<h1> Welcome! </h1>
</div>

<h2>  </h2>
<div id="section">

    <div id="left">
    <center>
    <img src="myphoto.png" width="150px", height = "150px">
    </center>
    </div>

    <div id = "right">
    <b>Hi!</b><p>

        Welcome to my website. I'm Peter The Great, a student at
        Glendon College. I am interested in computer music, machine
        learning, data visualization and French studies.
    <br><br>
    <a href="mailto:someone@yoursite.com">Contact me</a>
    <br>

    </div>

</div>


</p>

<p>

</body>
</html>
```



CSS Basics

A basic CSS rule has the following format:

```
selector {
  property: value;
}
```

Selector
: The selector is the element that the rule will affect

Property
: The property is the actual CSS rule

Value
: The value is the value we want for the given property

Declarations and Declaration Blocks

Each property and value set are called a declaration.

```
selector {
  property: value;
}
```

You can list as many declarations as you want. A group of declarations are called a declaration block.

```
selector {
```

```
  property: value;
  property: value;
}
```

Things to note:

- Declaration blocks are always surrounded by curly braces
- Properties and values are always separated by a colon
- declarations always end with semicolons
- Spacing does not matter

CSS Comments

You type comments in to your CSS by using (`/*`) and (`*/`). Comments can span multiple lines.

```
/* A CSS comment */

/* A
Comment in
multiple lines
*/
```

Element Type Selectors

The most basic CSS selectors are **Element Type Selectors**. These are really just HTML tags.

For example, if we wanted to make all paragraphs have red text, we would use the following CSS rule:

```
p {
  color: red;
}
```

Multiple Selectors

If we wanted to make all of the ordered lists and unordered lists red, in addition to the paragraphs, we could use three rules like this:

```
p {
  color: red;
}
ol {
  color: red;
}
ul {
  color: red;
```

```
}
```

Better yet, we can **combine** these three rules, by listing the selectors, separated by commas.

```
p, ol, ul {
  color: red;
}
```

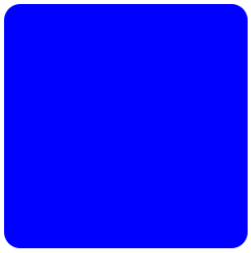
Exercise 8: 10 points

Given the code:

```
<!DOCTYPE html>
<html>
<head>
<!-- Make for my students in Itec at Glendon College -->
</head>
<body>

<div id="header">
<h1> Welcome......! </h1>
</div>

<h2>  </h2>
<div id="section">

   <div id="left">
   <center>
   <img src="myphoto.png" width="150px", height = "150px">
   </center>
   </div>

   <div id = "right">
   <b>Hi!</b><p>

      Welcome to my website. I'm Peter Toft, a student at Glendon College. I a
m interested in computer music,  machine learning,
      data visualization and French studies.
   <br><br>
   <a href="mailto:tu@gmail.com">Contact me</a>
   <br>

   </div>

</div>


</p>

<p>

</body>
</html>
```

Write the style.css so that your result will look like this:

Welcome.....!



Hi!

Welcome to my website. I'm Peter Toft, a student at Glendon College. I am interested in computer music, machine learning, data visualization and French studies.

Contact me

Exercise 9: 20 points .Web security

(a) Make the URL: http://oldtown.glendon.yorku.ca/~lastname/practice202/

(b) Protect the URL above with username "tintin" and password "glendon2020".

Here is your .htaccess file:

AuthType Basic
AuthName "restricted area"
AuthUserFile /home/teaching/secret/.htpasswd
Require valid-user

Solution:

1. mkdir /home/lastname/www/practice2
2. Create .htaccess at PATH above has the following 4 lines as content:

   AuthType Basic
   AuthName "restricted area"
   AuthUserFile /home/lastname/secret/.htpasswd
   Require valid-user

3. Create directory /home/lastname/secret/
   (use the command mkdir)
4. Create username and password:
   htpasswd -c /home/lastname/secret/htpasswd username

   - done

Links:

https://www.w3schools.com/

Notes & Misc. Commands:

```
ffmpeg -loop 1 -i image.png -c:v libx264 -pix_fmt yuv420p -vf
scale=320:240 out.mp4


ffmpeg -r 5 -i img%03d.jpg -vcodec h264 -pix_fmt yuv420p -crf 22 -s
1920x1080 MOVIE.mp4

Recording using your sound card (stereo mix) and Audacity.
```

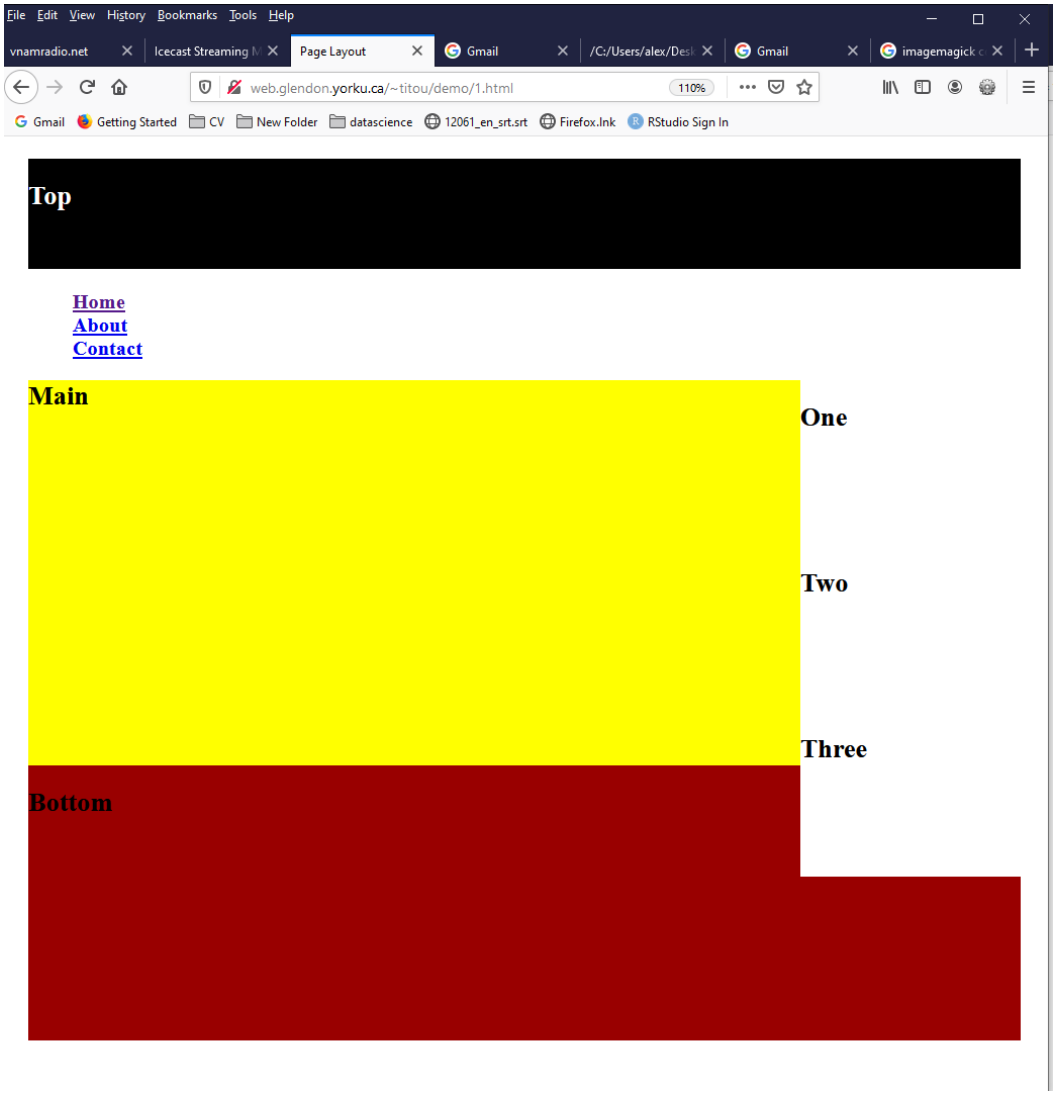
Test -- Itec/MODR 2635
Due: Midnight, Wednesday Nov 11, 2020 at this LINK.

Question 1:

(a) Create the page (see graphic below) at your URL
http://oldtown.glendon.yorku.ca/~lastname/2635/test
Your design must look as closely as possible to the graphic below.

(b) Protect the page in (a) with 3 usernames:
tintin1, tintin2 and tintin3.
and passwords: t001ab, t002ab, t003ab, respectively.

Please submit **a report** with your external style sheet (style.css), your html and link to your result page.



Layout

The layout process should take advantage of good semantic html, with the content of the page logically contained within various elements, such as **header, main, aside, section, article, figure, footer, p** and the **h1 – h6** headings. Properly applied, these elements will enable accessible content for all types of users, will optimize search engine results, and will create many of the selectors that CSS will apply to for styling and layout. In addition to these semantic elements, **div** is also available to create elements where there is a need for a container for visual layout purposes but no semantic element seems appropriate.

The display property determines whether the item appears as a new "block" (basically like a new paragraph) or whether it appears inline with the preceding element. Most html elements are, by default, block level (though some, like **em, cite, i** and others are inline). Use display to override or set elements to block, inline, inline-block (appear inline but can have block-type properties such as **width, height, margin** or **padding**) or none (removes the element from the page display entirely, leaving no space in its absence.)

So using this style rule for the normally block-level **p** element…

```
p {
    display: inline;
}
```

will produce a result in the browser like:

This is the first p element. This is the next p element.

Use the visibility: hidden property to make an element invisible, with a same-sized blank space in its stead. Use visibility: visible to reveal the element.

Set the height or width of individual elements, either in px for a fixed layout or in percentage (%) for a responsive layout. The percentage is based on the width of the element's parent.

```
main {
    width: 600px;
}

p {
    width: 75%;
}
```

Generally, you should avoid setting the **height** of elements, as dynamic content may overflow a set height in undesirable ways. If, for visual design purposes, there is a minimum height that you desire, use min-height instead.

Similarly, you can set a minimum or maximum width for an element with min-width or max-width (often used to limit the maximum width of the entire page to avoid ugly results on super-wide monitors.)

If you don't explicitly set the **width** or **height**, auto is used as the value. For elements that display as a block, the auto value for width is calculated from the width of the containing block minus the element's padding, borders, and margins. Elements like images have an auto width equal to the actual pixel dimensions of the original file.

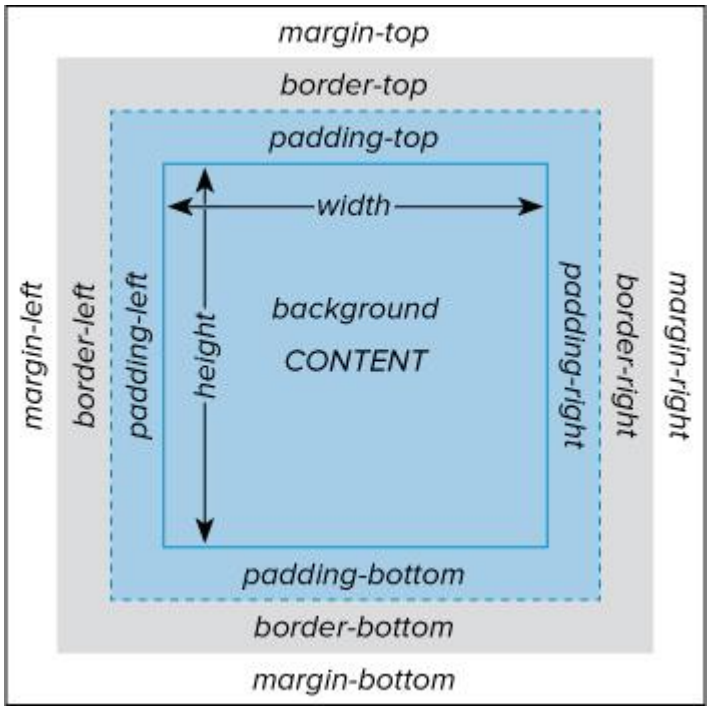
Use padding to add space between the content of an element and its border. The color or texture that shows in the padded area is the element's background, as set via **background**, **background-color** , or **background-image**. You can use pixels, percentages, ems or rems for padding values.

```
.about {
    background-color: #2b2b2b;
    padding: .3125em .625em .25em .5em;
}
```

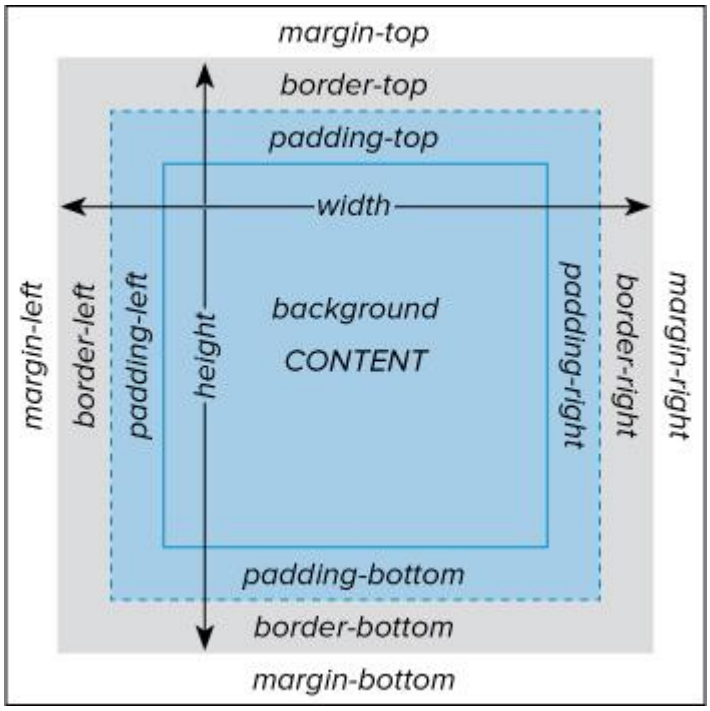
If you use one value for padding it sets all sides equally. Two values will set the top and bottom, and the left and right, in that order. Three values set the top, both sides, and bottom spacing. Finally, four values set, in order, the top, right, bottom and left values.

All of these different padding values can be set individually with padding-top, padding-right, padding-bottom and padding-left.

The following diagram shows the default relationship of width and height to the content, padding, border and margin properties.



The next diagram shows the relationship of width and height to those same properties when box-sizing: border-box is applied to the element. The dimensions of the element include the border (but not the margin) when box-sizing: border-box is applied.



The border of an element encloses its content, and its padding if any has been specified. The single border tag takes values for stroke width (in **px** or **em**), style (**solid**, **dashed**, **ridge**, **groove**, **inset** or **outset**) and color.

```
.about img {
    border: 5px solid #bebebe;
}
```

There are also separate tags for border-top, border-right, border-bottom, border-left, border-style, and border-width.

```
footer {
    border-bottom: 1px solid #dbdbdb;
    border-top: 3px solid #dbdbdb;
}

.ridge {
    border-style: ridge;
    border-color: orange;
}
```

Margins are added outside of the content and any padding and/or borders that have been applied to the element. The margin tag works very much like padding. It takes values in px, percentages, ems, rems or auto.

```
.map {
    margin: 1.4375em 0 .8125em;
}
```

If you use one value for margin it sets all sides equally. Two values will set the top and bottom, and the left and right, in that order. Three values set the top, both sides, and bottom spacing. Finally, four values set, in order, the top, right, bottom and left values.

All of these different margin values can be set individually with margin-top, margin-right, margin-bottom and margin-left.

A value of **auto** is often used to center elements within their container. Something like:

```
.page {
    margin: 0 auto;
    width: 960px;
}
```

will center the container (probably a **div** element with a "page" class) horizontally within the browser window (as the left and right values of **margin** are set to **auto**, while top and bottom are simply **0**).

You can create runaround or text-wrap effects by using the float property. This technique can also be used to create multi-column layouts.

The **float** property is added to the element (a **figure, img, div, article, section**, etc.) around which you want text or other elements to wrap.

```
#mainpic {
        float: left;
        width: 300px;
        margin: 6px;
}
```

the rule above will position the mainpic element to the left, and the text or other content after it in the html document will wrap around it to the right. Note that for **float** to work properly in this case, the element with the **float** property must have a **width** specified, and the elements that are going to be doing the wrapping must NOT have widths applied. If all of the elements DO have specific widths, then each one must be given a **float** property, as in the following.

```
#mainpic {
        float: left;
        width: 300px;
        margin: 6px;
}

#sidepic {
        float: right;
        width: 200px;
        margin: 10px;
}
```

Using **float** is a very elegant way to create flexible and sophisticated layouts, but it does create a few complications. First, everything below the floated element in the html will just keep scooting up to wrap around the content, whether you want it to or not. Add the clear: left, clear: right or clear: both property to the first element that you want to have ignore the **float**. This will end the float effect for it and all later elements.

The second complication is when a container element (like a **main, article, figure**, etc) has all of its children set to **float** (imagine a **figure** element where both the **img** and the **figcaption** elements are set to **float.**) In this case, the containing element will, as far as CSS is concerned, have no actual content, and no actual height. This will screw up the rendering of borders and backgrounds. The common fix here is a workaround class called *clearfix.* Add these style rules for the new class to your main stylesheet.

```
.clearfix:before,
.clearfix:after {
    content: " ";
    display: table;
}

.clearfix:after {
    clear: both;
}

.clearfix {
    *zoom: 1;
}
```

Then, in the html pages, add the *clearfix* class name to any container elements whose children are all floating. This will give those containers back their heights (the height of their tallest child) and will clear the float from that point on.

```
<div class="container clearfix">
```

Recall that elements may have multiple classes. In the example, the **div** element already belonged to a *container* class. We've just also made it a member of the *clearfix* class, so that those style rules will apply to it.

Relative and Absolute Positioning

Elements in an html page have a natural position on the page, with later block style elements appearing below earlier ones. This positioning may be modified by properties like width, padding, margin and float, but the flow from one element to the next is still reflected in the appearance of content down the screen in the browser. This natural flow may be adjusted or completely overridden by using relative or absolute positioning for elements.

Relative positioning is accomplished by adding a position: relative; rule to an element's CSS styling, and then applying a top, right, bottom or left offset value.

```
.example {
        position: relative;
        top: 35px;  /* 35 pixels below the original location */
        left: 100px;  /* 100 pixels to the right of the original location */
}
```

The offset value here is 35px below and 100px to the right of its normal location.

Relatively positioned elements leave a space in their original location.

Elements can have **absolute positioning** that specifies their position relative to the body element, or to their most immediate positioned ancestor element. **Top, right, bottom** and **left** offset values are set similarly to relative positioning.

```
header {
        position: absolute;
        top: 41px;
        left: 0px;
}
```

Assuming the header element here is not contained within some larger div, the positioning is relative to **body** (functionally, the entire browser screen.)

Absolutely positioned elements do NOT leave any space in their original location, and in fact they have been removed from the document flow entirely (so other elements cannot float around them, for example.)

Overlapping elements

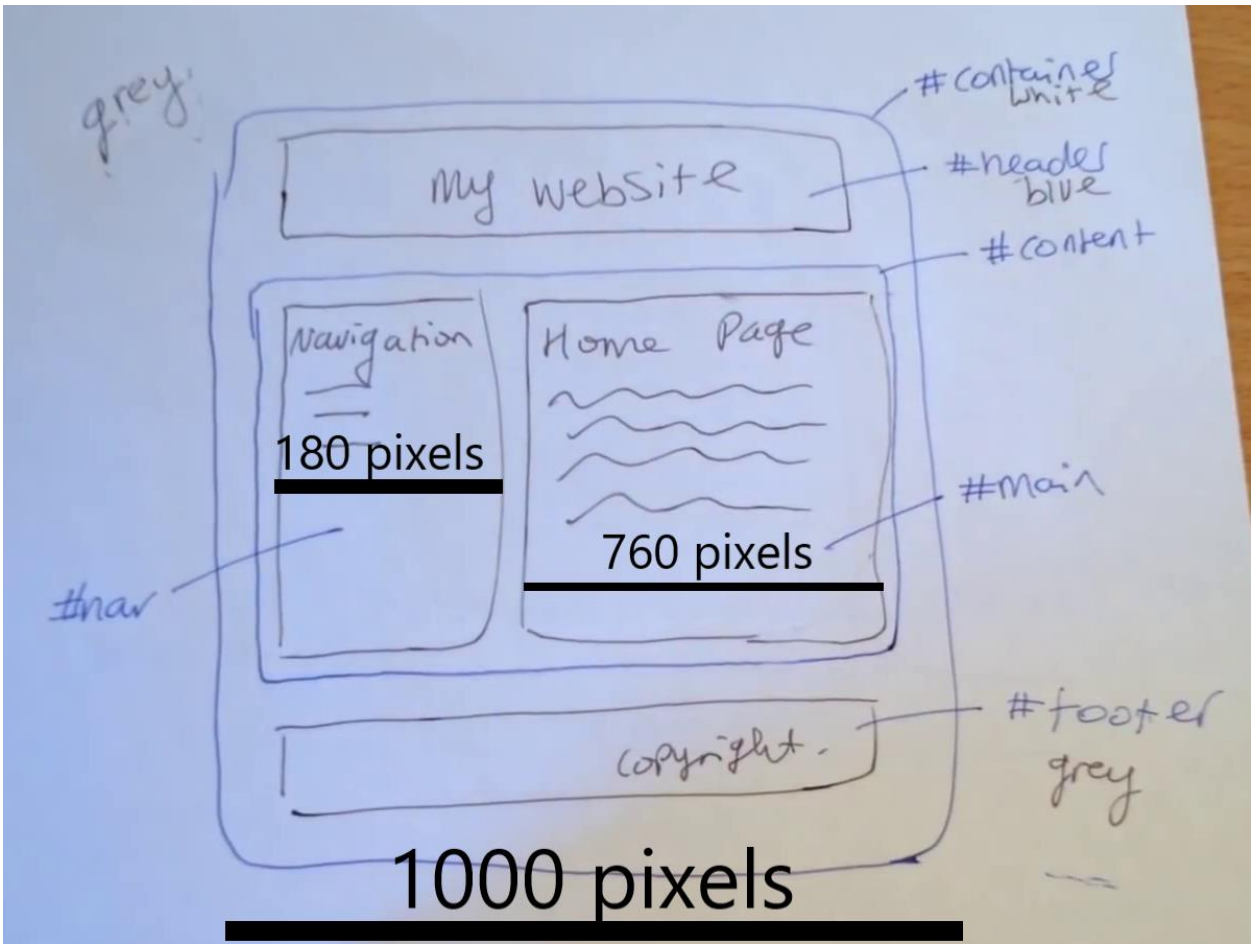
Because relative and absolute positioning disrupt the normal flow of how elements display on the screen (generally following one after the other in the order seen on the html page,) it is possible for these elements to overlap other content on the page. Use the z-index property to determine the stacking order of overlapping elements. The higher the z-index value, the closer the element is to the top of the stack.

```
#card1 {
        position: absolute;
        top: 20px;
        right: 50px;
        z-index: 2;
}

#card2 {
        position: relative;
        top: 120px;
        right: 50px;
        z-index: 8;
}

#card3 {
        position: absolute;
        top: 220px;
        right: 10px;
        z-index: 1;
}

#card4 {
        top: 220px;
        right: 10px;
        z-index: 50;
}
```

In the above example, #card2 will appear above @card1, which will appear above #card3. Although #card4 has the highest z-index value, it will be beneath all of the other elements, as it has no positioning. Setting a previously positioned element to position:static will also invalidate any z-index setting for that element.

Itec/MODR 2635: html5 and CSS
Homework 2: Submission LINK.

Question 1: (100 points)
Design the following web page using an external CSS style (style.css):



To be displayed at your URL:

http://oldtown.glendon.yorku.ca/~lastname/css/2635/

Submit a short report with code and links to your web page.
This homework has only one question.

Colours, CSS (in HTML), Images

Colour

1. In HTML a colour can be specified as a name (Red, Blue, Green, Yellow and etc..), in RGB values, or in Hexadecimal

CSS

1. CSS stands for Cascading Style Sheets.
   a. CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**.
   b. CSS **saves a lot of work**. It can control the layout of multiple web pages all at once.
   c. CSS can be added to HTML elements in 3 ways:
      i. **Inline** - by using the style attribute in HTML elements
      ii. **Internal** - by using a <style> element in the <head> section
      iii. **External** - by using an external CSS file

   The most common way to add CSS, is to keep the styles in separate CSS files. However, here we will use inline and internal styling, because this is easier to demonstrate

2. Inline CSS is used to apply a unique style to a single HTML element. Inline CSS uses the *style attribute* of an HTML element.

```
<h1 style="color:blue;">This is a Blue Heading</h1>
```

3. Internal CSS is used to define a style for a single HTML page. An internal CSS is *defined in the <head> section* of an HTML page, by using a <style> tag:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: powderblue;}
h1   {color: blue;}
p    {color: red;}
</style>
</head>
<body>

<h1>This is a heading</h1>
```

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

</body>
</html>
```

4. External CSS has an external style sheet that is used to define the style for many HTML pages. **With an external style sheet, you can change the look of an entire web site, by changing one file!** To use an external style sheet, add a link to it in the <head> section of the HTML page:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

An external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a .css extension.

Here is how the "styles.css" looks:

```
body {
    background-color: powderblue;
}
h1 {
    color: blue;
}
p {
    color: red;
}
```

5. Common CSS Fonts contain the following information
    a. Colour
    b. Font-Family
    c. Font-Size

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
   color: blue;
   font-family: verdana;
   font-size: 300%;
}
p  {
   color: red;
   font-family: courier;
   font-size: 160%;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

6. CSS Border - The CSS **border** property defines a border around an HTML element.

```
p {
   border: 1px solid powderblue;
}
```

7. CSS Padding - The CSS **padding** property defines a padding (space) between the text and the border:

```
p {
   border: 1px solid powderblue;
   padding: 30px;
}
```

8. CSS Margin - The CSS **margin** property defines a margin (space) outside the border.

```
p {
   border: 1px solid powderblue;
```

```
    margin: 50px;
}
```

Note: You can also declare all the margins and paddings of an element in a single property as follows:

```
margin: 10px 10px 10px 10px;
```

If you declare all 4 values as above, the order is as follows (clockwise from the top):

1. top
2. right
3. bottom
4. left

If only one value is declared, it sets the margin on all sides.

```
margin: 10px;
```

If you only declare two or three values, the undeclared values are taken from the opposing side.

```
margin: 10px 10px; /* 2 values * Top and Sides/
margin: 10px 10px 10px; /* 3 values * Top, Right, Bottom/
```

You can set the margin property to negative values. If you do not declare the margin value of an element, the margin is 0 (zero).

```
margin: -10px;
```

All the coding we have done so far with CSS has ensured that all <p> will be altered via the CSS. However, what if we only wanted a few specific <p> tags to be manipulated via CSS and we wanted the others to stay the same? Here is where we use the id and class attributes.

9. The id Attribute - To define a specific style for one special element, add an id attribute to the element. (the # code)

   In the HTML we would type:

```
<p id="p01">I am different</p>
```

and in the CSS we would type:

```
#p01 {
   color: blue;
}
```

It is important to note that an id attribute goes hand in hand with the hashtag (#) code from CSS.

10. The class attribute - To define a style for a special type of elements, add a class attribute to the element:

In the HTML we would type:

```
<p class="error">I am different</p>
```

In the CSS we would type:

```
.error {
   color: red;
}
```

It is important to note that the class attribute goes hand in hand with the dot (.) code in CSS.

Images

1. The <img> tag is empty, it contains attributes only, and does not have a closing tag. The <img> tag needs a source (can be a URL or a file from your folder, an alternate attribute (alt) which states what the image is just in case the browser cannot find the image and a style attribute which defines the width and height of the image.

   <img src="*url*" alt="*some_text*" style="width:*width*;height:*height*;">

2. It is important to use the style attribute so that it the width and height of an image does not conflict with CSS coding. For example:
   http://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_style

3. You can also use an image as a link.
   ```
   <a href="default.asp">
     <img src="smiley.gif" alt="HTML tutorial" style="width:42px;height:42px;border:0;">
   </a>
   ```

HTML 5

New HTML5 Elements
The most interesting new HTML5 elements are:

- New semantic elements like <header>, <footer>, <article>, and <section>.
- New attributes of form elements like number, date, time, calendar, and range.
- New graphic elements: <svg> and <canvas>.
- New multimedia elements: <audio> and <video>

New HTML5 API's (Application Programming Interfaces)
The most interesting new API's in HTML5 are:

- HTML Geolocation
- HTML Drag and Drop
- HTML Local Storage
- HTML Application Cache
- HTML Web Workers
- HTML SSE

Tip: HTML Local storage is a powerful replacement for cookies.

Removed Element	**Use Instead**
<acronym>	<abbr>
<applet>	<object>
<basefont>	CSS
<big>	CSS
	CSS
<dir>	
<font>	CSS
<frame>	
<frameset>	
<noframes>	
<strike>	CSS, <s>, or <del>
<tt>	CSS

HTML5 Browser Support
HTML5 is supported in all modern browsers.

- In addition, all browsers, old and new, automatically handle unrecognized elements as inline elements.
- Because of this, you can "teach" older browsers to handle "unknown" HTML elements.

Define Semantic Elements as Block Elements
HTML5 defines eight new semantic elements.

- All these are block-level elements.
- To secure correct behavior in older browsers, you can set the CSS display property for these HTML elements to block:

```
header, section, footer, aside, nav, main, article, figure {
  display: block;
}
```

New Elements in HTML5
Below is a list of the new HTML5 elements, and a description of what they are used for.

New Semantic/Structural Elements
HTML5 offers new elements for better document structure:

Tag	Description
<article>	Defines an article in a document
<aside>	Defines content aside from the page content
<bdi>	Isolates a part of text that might be formatted in a different direction from other text outside it
<details>	Defines additional details that the user can view or hide
<dialog>	Defines a dialog box or window
<figcaption>	Defines a caption for a <figure> element
<figure>	Defines self-contained content
<footer>	Defines a footer for a document or section
<header>	Defines a header for a document or section
<main>	Defines the main content of a document
<mark>	Defines marked/highlighted text
<meter>	Defines a scalar measurement within a known range (a gauge)
<nav>	Defines navigation links
<progress>	Represents the progress of a task
<rp>	Defines what to show in browsers that do not support ruby annotations
<rt>	Defines an explanation/pronunciation of characters (for East Asian typography)
<ruby>	Defines a ruby annotation (for East Asian typography)
<section>	Defines a section in a document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time
<wbr>	Defines a possible line-break

New Form Elements

Tag	Description

<datalist>	Specifies a list of pre-defined options for input controls
<output>	Defines the result of a calculation

New Input Types

New Input Types	New Input Attributes	
Color	autocomplete	placeholder
Date	autofocus	required
Datetime	form	step
Datetime-local	formaction	
Email	formenctype	
Month	formmethod	
Number	formnovalidate	
Range	formtarget	
Search	height and width	
Tel	list	
Time	min and max	
Url	multiple	
Week	pattern (regexp)	

HTML5 Graphics

Tag	Description
<canvas>	Draw graphics, on the fly, via scripting (usually JavaScript)
<svg>	Draw scalable vector graphics

New Media Elements

Tag	Description
<audio>	Defines sound content
<embed>	Defines a container for an external (non-HTML) application
<source>	Defines multiple media resources for media elements (<video> and <audio>)
<track>	Defines text tracks for media elements (<video> and <audio>)
<video>	Defines video or movie

HTML5 - New Attribute Syntax

HTML5 allows four different syntaxes for attributes.

This example demonstrates the different syntaxes used in an <input> tag:

Type	Example
Empty	<input type="text" value="John" disabled>
Unquoted	<input type="text" value=John>
Double-quoted	<input type="text" value="John Doe">
Single-quoted	<input type="text" value='John Doe'>

In HTML5, all four syntaxes may be used, depending on what is needed for the attribute.

HTML5 Semantic Elements
Semantics is the study of the meanings of words and phrases in a language.

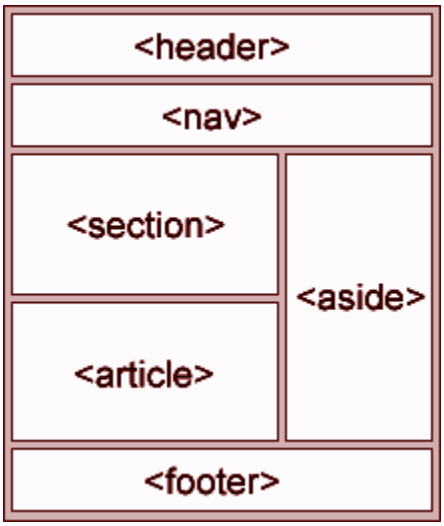
Semantic elements = elements with a meaning.

What are Semantic Elements?
A semantic element clearly describes its meaning to both the browser and the developer.

- Examples of non-semantic elements: and - Tells nothing about its content.
- Examples of semantic elements: <form>, <table>, and <article> - Clearly defines its content.

New Semantic Elements in HTML5
Many web sites contain HTML code like: to indicate navigation, header, and footer.

HTML5 offers new semantic elements to define different parts of a web page:



<article>
<aside>
<details>
<figcaption>
<figure>
<footer>
<header>
<main>
<mark>

<nav>
<section>
<summary>
<time>

HTML 4	**HTML 5**
	<header>
	<nav>
	<section>
	<article>
	<footer>

Use Lower Case Element Names
HTML5 allows mixing uppercase and lowercase letters in element names.

We recommend using lowercase element names because:

- Mixing uppercase and lowercase names is bad
- Developers normally use lowercase names (as in XHTML)
- Lowercase look cleaner
- Lowercase are easier to write

CSS: Introduction

The problem with HTML:

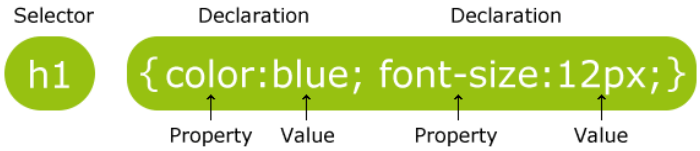
- HTML was originally intended to describe the content of a document
- Page authors didn't have to describe the layout--the browser would take care of that
- This is a good engineering approach, but it didn't satisfy advertisers and "artists"
- As a result, HTML acquired more and more tags to control appearance
    - Content and appearance became more intertwined
    - Different browsers displayed things differently, which is a real problem when appearance is important

Cascading Style Sheets

A Cascading Style Sheet (CSS) describes the appearance of an HTML page. It is usually a separate document included in the HTML but it can also be embedded inside the HTML.
Here is a demonstration of what can be done with CSS (click on the various stylesheet buttons to switch styles): https://www.w3schools.com/css/css_intro.asp
CSS has the following advantages:

- It lets you separate content from presentation
- It lets you define the appearance and layout of all the pages in your web site in a single place
- It can be used for both HTML and XML pages
- CSS has the following disadvantage:
- Most browsers don't support it very well

CSS syntax



CSS syntax is very simple--it's just a file containing a list of selectors (to choose tags) and descriptors (to tell what to do with them):
Example:
h1 {color: green; font-family: Verdana} says that everything included in h1 (HTML heading level 1) tags should be in the Verdana font and colored green

A CSS file is just a list of these selector/descriptor pairs. Selectors may be simple HTML tags or XML tags, but CSS also defines some ways to combine tags. Descriptors are defined in CSS itself, and there is quite a long list of them.
The general syntax is:

```
selector {property: value}
```

or

```
selector, ..., selector {
   property: value;
   . . .
   property: value
}
```

where *selector* is the tag to be affected (the selector is case-sensitive if and only if the document language is case-sensitive) *property* and *value* describe the appearance of that tag. Spaces after colons and semicolons are optional. A semicolon must be used *between* property:value pairs, but a semicolon after the last pair is optional.
Examples

```
/* This is a comment */
h1,h2,h3 {font-family: Arial, sans-serif;}   /* use 1st available font */
p, table, li, address {                /* apply to all these tags */
  font-family: "Courier New";      /* quote values containing spaces */
  margin-left: 15pt;                /* specify indentation */
}
p, li, th, td {font-size: 80%;}      /* 80% of size in containing element */
th {background-color:#FAEBD7}  /* colors can be specified in hex */
body { background-color: #ffffff;}
h1,h2,h3,hr {color:saddlebrown;} /* adds to what we said before */
a:link    {color:darkred}             /* an unvisited link */
a:visited {color:darkred}             /* a link that has been visited */
a:active  {color:red}                 /* a link now being visited */
a:hover   {color:red}                 /* when the mouse hovers over it */
```

CSS Units

Units appear in many CSS properties, for example font-size. It is always better to use percentages than absolute units such as pixels. For example a font-size of 200% will double the current size and will always be consistent with the display properties.

%	percentage
in	inch
cm	centimeter
mm	millimeter
em	1em is equal to the current font size. 2em means 2 times the size of the current font. E.g., if an element is displayed with a font of 12 pt, then '2em' is 24 pt. The 'em' is a very useful unit in CSS, since it can adapt automatically to the font that the reader uses
ex	one ex is the x-height of a font (x-height is usually about half the font-size)
pt	point (1 pt is the same as 1/72 inch)

pc	pica (1 pc is the same as 12 points)
px	pixels (a dot on the computer screen)

CSS Colors

Colors can be applied to many properties such as background-color. Color can be specified using names (“red”, “blue”) but for a full range of colors it is recommended to use HX or RGB values:

	Color HEX	Color RGB
	#000000	rgb(0,0,0)
	#FF0000	rgb(255,0,0)
	#00FF00	rgb(0,255,0)
	#0000FF	rgb(0,0,255)
	#FFFF00	rgb(255,255,0)
	#00FFFF	rgb(0,255,255)
	#FF00FF	rgb(255,0,255)
	#C0C0C0	rgb(192,192,192)
	#FFFFFF	rgb(255,255,255)

CSS Selectors

In order to apply a style to an element, you need to defined a selector for that element. An HTML tag can be used as a simple element selector:

body { background-color: #ffffff }

You can use multiple selectors:

em, i {color: red}

You can repeat selectors:

h1, h2, h3 {font-family: Verdana; color: red}

h1, h3 {font-weight: bold; color: pink}

When values disagree, the last one overrides any earlier ones

The universal selector * applies to any and all elements:

* {color: blue}

When values disagree, more specific selectors override general ones (so em elements would still be red)

We have used the following CSS, in this order:

1. *em {color: red}*
2. * {color: blue}
3. **b {color: pink; color: black}**

A descendent selector chooses a tag with a specific ancestor:

p code { color: brown }

selects a code if it is somewhere inside a paragraph

A child selector > chooses a tag with a specific parent:

h3 > em { font-weight: bold }

selects an em only if its immediate parent is h3

An adjacent selector chooses an element that immediately follows another:

b + i { font-size: 8pt }

Example: I'm bold and I'm italic

Result will look something like: **I'm bold and** *I'm italic*

Examples

Selectors can apply to a type of element (example for any <p>), to a specific element (identified with an id), to a group of elements (identified by a class), or to more specific element defined by a combination of criterias. These are live examples of such selectors.

- The element selector
- The id selector
- The class selector (for all elements)
- The class selector (for only <p> elements)
- Grouping selectors

Attributes

It is possible to style HTML elements that have specific attributes or attribute values. A simple attribute selector allows you to choose elements that have a given attribute, regardless of its value:

Syntax: element[attribute] { ... }
Example: table[border] { ... }

An attribute value selector allows you to choose elements that have a given attribute with a given value:

Syntax: element[attribute="value"] { ... }
Example: table[border="0"] { ... }

The class attribute

Every HTML element can have a class attribute that defines one or many CSS classes. The class attribute allows you to have different styles for the same element:
In the style sheet:

```
p.important {font-size: 24pt; color: red}
p.fineprint {font-size: 8pt}
```

In the HTML:

```
<p class="important">The end is nigh!</p>
<p class="fineprint">Offer ends 1/1/97.</p>
```

To define a selector that applies to any element with that class, just omit the tag name (but keep the dot):

```
.fineprint {font-size: 8pt}
.label {background-color: blue}
```

Note that you can apply several classes to a single element:

```
<p class="important label">The end is nigh!</p>
```

The id attribute

The id attribute is used to identify a single element in the document: this means that the id value must be unique for each element in the document. So make sure to pick up different ids across your document. The id attribute is defined like the class attribute, but uses # instead of .
In the style sheet:

```
p#important {font-style: italic}     or
#important {font-style: italic}
```

In the HTML:

```
<p id="important">
```

class and id can both be used, and do not need to have different names:

<p class="important" id="important">

div and span

div and span are HTML elements whose only purpose is to hold CSS information. It's a bit like creating a block or node in the HTML tree structure. div ensures there is a line break before and after (so it's like a paragraph); span does not. Example:
CSS:

```
div {background-color: #66FFFF}
span.color {color: red}
```

HTML:

```
<div>This div is treated like a paragraph, but <span class="color">this span</span> is not.</div>
```

The box model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.
The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

Content - The content of the box, where text and images appear
Padding - Clears an area around the content. The padding is transparent
Border - A border that goes around the padding and content
Margin - Clears an area outside the border. The margin is transparent
Here is a CSS example to control these values:

```
div {
    width: 300px;
    border: 25px solid green;
    padding: 25px;
    margin: 25px;
}
```

What is the total width of the element?
Here is a live demo of the box model:

How to add CSS

There are three ways of inserting a style sheet in a HTML document:

1. External style sheet (example)
2. Internal style sheet (example)
3. Inline style (example)

External: This is the recommended technique since it is easy to share and modify independently from HTML documents. The HTML page includes a reference to the external style sheet file inside the <link> element. The <link> element goes inside the <head> section. For example:

```
<head>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
```

The document style.css is edited as a regular text file and contains pure CSS, for example:

```
body {
    background-color: lightblue;
}
h1 {
    color: red;
    margin-top: 20px;
}
```

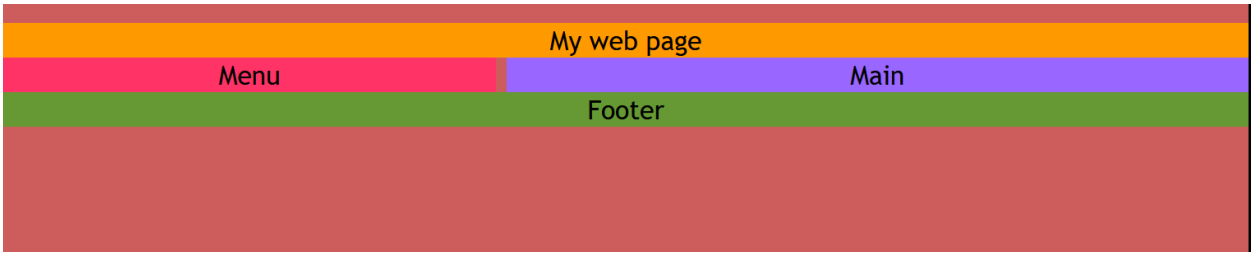
Internal: An internal style sheet is recommended if one single page has a unique style. Internal styles are defined within the <style> element, inside the <head> section of an HTML page, for example:

```
<head>
<style>
body {
    background-color: linen;
}
h1 {
    color: maroon;
    margin-left: 40px;
}
</style>
</head>
```

Inline: inline style is used to apply a unique style for a targeted element. For example:

```
<h1 style="color:red;margin-left:20px;">Example of inline style</h1>
```

Example



References

1. HTML cheatsheet: http://bloggerspath.com/ultimate-html5-cheat-sheet-for-web-developers/
2. Chrome inspector: https://youtu.be/wcFnnxfA70g
3. CSS tutorial: https://www.w3schools.com/css/default.asp

Useful heads-up

- ***Responsive web design*** (RWD) refers to web design that works well on a variety of devices and screen sizes.

- The impetus for doing this is the huge increase in the variety of devices (from smart phone to desktop) and the huge popularity of some of the smaller devices.

    - **flexible grids** – how to base your design on a grid-like arrangement, where the pieces scale for your screen size. We'll use standard templates.
    - **flexible or adaptive images** – how to make your images blow up/shrink appropriately for your screen size
    - **media queries** – how to use different style sheets for different types of media
    - **dynamic content** – content which is responsive to what your user is doing/using; this includes the user rotating a phone/tablet. Some folks don't think of this as "a thing" in RWD – everyone agrees on the first three items.

- the ***viewport*** refers to the screen window.
  RWD inevitably begins with

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

  This sets the size of the viewport (the screen real estate you will be working with) to be that of the device, and sets an initial level of how far you have zoomed in.
  This is not a w3c standard, but it is a de-facto one.
  Eventually you will want to read the details at https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag

- The default font-size for all browsers is 16 points. All of them.
  This is another de-facto standard that is not a w3c standard.

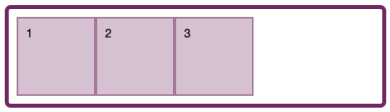
- **CSS pixel** refers to the pixel size you may specify in a style sheet --- and not the actual, physical pixels on a device (which depend on such things as pixel density.)
  If you wish to, you can learn more about this at Peter-Paul Koch's article "A pixel is not a pixel is not a pixel".
  http://www.quirksmode.org/blog/archives/2010/04/a_pixel_is_not.html

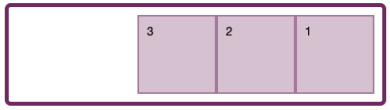
Read:

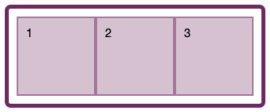
- To see what RWD can do watch the introduction to http://www.lynda.com/CSS-tutorials/About-exercise-files/110716/114016-4.html

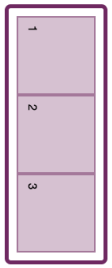
FLEXBOX CONTAINER CHEATSHEET

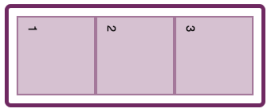
HTML	Other CSS ITEMS	Flexbox container CSS	How It Looks

 1
 2
 3
 	
	* {box-sizing: border-box;}
	
	.item {
	width: 100px;
	height: 100px;
	padding: 10px;
	background-color: rgba(111,41,97,.3);
	border: 2px solid rgba(111,41,97,.5);}
	
	.item ul {
	margin: 0;
	padding: 0;
	list-style: none;}
		.container {
		border: 5px solid rgb(111,41,97);
		border-radius: .5em;
		padding: 10px;
		display: flex;
			display:flex;
			 

		.container {
		border: 5px solid rgb(111,41,97);
		border-radius: .5em;
		padding: 10px;
		display: flex;
		flex-direction: row-reverse;}
			display: flex;
			flex-direction: row-reverse;
			 

		.container {
		border: 5px solid rgb(111,41,97);
		border-radius: .5em;
		padding: 10px;
		display: inline-flex;}
			display: inline-flex;
			 

		.container {
		border: 5px solid rgb(111,41,97);
		border-radius: .5em;
		padding: 10px;
		display: flex;
		flex-direction: row;
		writing-mode: vertical-lr;}
			display: flex;
			flex-direction: row;
			writing-mode: vertical-lr;
			 

		.container {
		border: 5px solid rgb(111,41,97);
		border-radius: .5em;
		padding: 10px;
		display: flex;
		flex-direction: column;
		writing-mode: vertical-lr;}
			display: flex;
			flex-direction: column;
			writing-mode: vertical-lr;
			 

Media Queries – some details

1. **@import** is used to get one style sheet to load all or part of another style sheet.

   Because it prevents parallel loading of style sheets, it slows down the page load, and **should be avoided if possible.** In other words, it is better to have (two) links to (two) stylesheets.

   @import goes right at the top of the head, just after the charset tag.

2. Firefox supports on two media types – screen and print (and FullerScreen for projection). It will work better if you use **media queries.**

   On the other hand, your aural and braille users won't be using Firefox anyway (since it doesn't support them adequately), so **@media** should be used for them.

3. Of course, your can have the media queries **inside a link to an external style sheet**.

   ```
   <link rel="stylesheet" type="text/css" media="only screen and (max-
   device-width: 780px)" href="max-device-width-780px.css" />
   ```

   or **within a style sheet**---

   ```
   <style>

   @media only screen and (max-device-width: 780px) {
          special styling goes here
          }
   general styling

   </style>
   ```

4. There is standard code for breakpoints at http://css-tricks.com/snippets/css/media-queries-for-standard-devices/

5. Next, there is excellent advice about the use (and not over-use) of breakpoints at http://bradfrost.com/blog/post/7-habits-of-highly-effective-media-queries/
   That said, you will need to use them.

6. There is a breakpoint plug-in for jQuery, with a very clear description at https://github.com/hejmartin/jquery-breakpoint (also accessible from http://plugins.jquery.com/breakpoint/)
   Basically, the plug-in monitors some condition and changes things when that condition

becomes true or false. This could be very useful if, for example, you are monitoring the orientation of a tablet or smartphone. Start at http://plugins.jquery.com/breakpoint/ and follow the links at the right side to the documentation and download.

There is also a breakpoints-js plug-in which adds classes to your elements based on breakpoints. See http://plugins.jquery.com/breakpoints-js/ and description at https://github.com/reusables/breakpoints.js

7. Many of the grid systems have code for breakpoints – e.g. responsivedesign has it at http://responsivedesign.is/develop/browser-feature-support/media-queries-for-common-device-breakpoints

8. Sass and mixins provide more structure for using breakpoints – see and http://www.sitepoint.com/managing-responsive-breakpoints-sass/ http://responsivedesign.is/develop/getting-started-with-sass or google: sass breakpoint tutorial
if you are interested.