Unit 2

USING STYLES

LEARNING GOALS

a) Identify possibilities to modify the HTML tags.

b) Be able to define styles directly.

c) Be able to define and associate global styles in outer sheets.

d) Be able to define alternative style sheets.

e) Identify the different properties of each element.

f) Be able to create style classes.

g) Be able to use validation tools of style sheets.

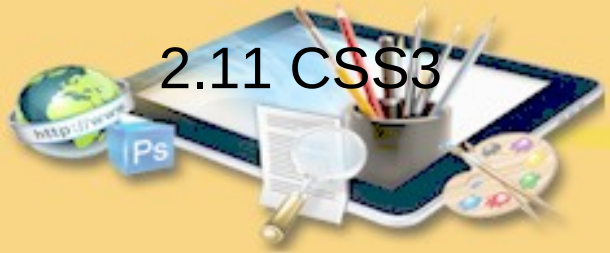
LEARNING GOALS

- CSS (Cascading Style Sheets) are very important in Web Application Design.

- CSS is a W3C standard widely recognized and used because of two aspects:

  1. saves time in web site design
  2. gets powerful effects supported by most browsers

CONTENTS



2.1 INTRODUCTION TO CSS

- CSS is the language for describing the presentation of Web pages, including colors, layout, and fonts
- It allows to adapt the presentation to different types of devices, such as large screens, small screens, or printers.
- CSS is independent of HTML and can be used with any XML-based markup language.
- The separation of HTML from CSS makes it easier to maintain sites, share style sheets across pages, and tailor pages to different environments. This is what we call as the separation of structure (or content) from presentation.

2.1 INTRODUCTION TO CSS

- **PHILOSOPHY**: use the HTML <body> tag to define the structures of the content displayed on the site (headers, paragraphs, images, etc..) And then in another file or in the HTML <head> is defined the appearance of each page using the CSS language.

CSS1 *is a simple style sheet mechanism that allows authors and readers to attach style (e.g. fonts, colours and spacing) to HTML documents. The CSS1 language is human readable and writable, and expresses style in common desktop publishing terminology.*

2.1 INTRODUCTION TO CSS

- **ADVANTAGE**: change the contents that are placed on the <body> without affecting the appearance defined in the CSS file (or <head> HTML), or to change the appearance in the CSS without affecting any of the <body> put in HTML.
- Another **ADVANTAGE**: adaptation of websites to different devices (iPad, mobile phone, personal computer, ...) where are going to be displayed.

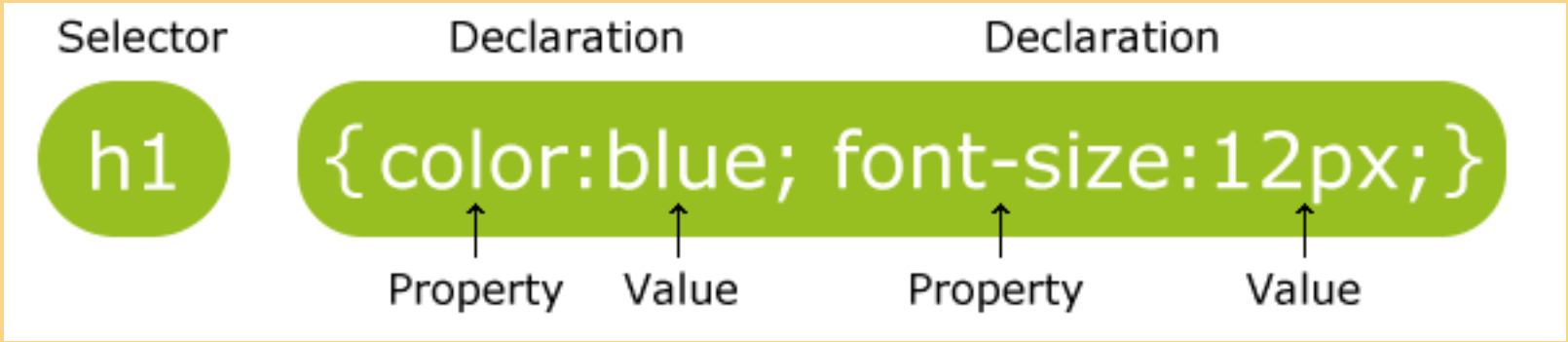
One of the fundamental features of CSS is that style sheets cascade; authors can attach a preferred style sheet, while the reader may have a personal style sheet to adjust for human or technological handicaps. The rules for resolving conflicts between different style sheets are defined in this specification.

2.1 INTRODUCTION TO CSS

- The CSS1 standard of W3C wants to establish the criteria for giving preference to one style above another (that is, cascading).
- The specification W3C defines the rules of preference that a compatible browser with CSS must respect:
  - so website developers don't have to design different sites for each browser
  - and users don't see a website with different appearance depending on the browser

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

- A CSS style sheet is made up of rules that indicate how the page will be displayed (style rules)
- **CSS Syntax**:
  - A CSS rule has two main parts: a selector, and one or more declarations.



2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

- The selector is normally the HTML **element** you want to style.
- Each declaration consists of a property and a value.
- The property is the style attribute you want to change. Each property has a value.
- A CSS declaration always ends with a semicolon, and declaration groups are surrounded by curly brackets:


```
p {color:red; text-align:center;}
```

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

- To make the CSS more readable, you can put one declaration on each line, like this:

```
p {
   color: red;
   text-align: center;
}
```

- Comments are used to explain your code, and may help you when you edit the source code at a later date. Comments are ignored by browsers.
- A CSS comment begins with "/*", and ends with "*/".

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.1 Selectors based on tags

- The easiest way to define a style rule is to use the HTML tags as selectors.

```
tagname {
   attribute: value;
}
```

- A style can be applied to any HTML tag following the syntax specified.
- The attribute refers to the feature that you want to change of the label, such as color. The value refers to the instance of the attribute, for example, blue. Ex:

```
h1 {
   color: blue;
}
```

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.1 Selectors based on tags

- If a browser finds a selector that doesn't understand, will ignore the whole declaration and will continue with the following one (it doesn't mind if the error is on the property, the value or the whole declaration).
- CSS provides syntax for defining attributes for multiple selectors with multiple attributes.

```
tagname1, tagname2 {
   attribute1: value1;
   attribute2: value2;
}
```

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.1 Selectors based on tags

- EXAMPLES:
  - If you want to apply the same style to two different selectors, the syntax would be:

```
h1, h2 {
   color: blue;
}
```

  - And if you want to apply different attributes to the same selector:

```
h1 {
   color: blue;
   background-color: red;
}
```

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.1 Selectors based on tags

- Where such statements should be placed so that the browser reads and interprets them?

```
<head>
  <title></title>
  <style>
    h1 {
      color: blue;
      background-color:red}
  </style>
</head>
```

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.2 Selectors based on classes

- The classes associated to HTML tags are defined as:

```
tagname.classname {
  attribute1:value1;
  attribute2:value2
}
```

- This alternative is more potent than what we have seen with tag-based selectors, allowing to apply different styles to the same tags.
- More generic classes do not apply to any HTML tags:

```
.classname {
  attribute1:value1;
  attribute2:value2;
}
```

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.2 Selectors based on classes

- EXAMPLE:

```
<head>
  <style>
    h1.roja {color: red}
    h1.verde {color: green}
    h1.azul {color: blue}
  </style>
</head>
…
<body>
  <h1 class="roja">A red heading</h1>
  <h1 class="azul">now blue</h1>
  <h1 class="verde">and now
green</h1>
</body>
```

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.2 Selectors based on classes

- EXAMPLE:

```
<head>
  <style>
    .textorojo { color: red }
    .fondoazul { background-color: blue }
  </style>
</head>
…
<body>
  <h3 class="textorojo fondoazul">red heading, blue background</h3>
  <p class="textorojo">in red color; background inherit</p>
</body>
```

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.3 Selectors based on identifiers

- Unlike classes, the identifiers can only be used on a single element.

```
tagname#identifier {attribute1: value1; attribute2: value2}
```

```
#identifier {attribute1: value1; attribute2: value2}
```

- As can be seen, this statement is identical to that used to define class selectors. The only difference is that instead of using a ".", we use a hash "#". However, the meaning, the semantics of both expressions, is very similar.

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.3 Selectors based on identifiers

- EXAMPLE:

```
<head>
   <style>
   #cabecera
   {  background: #CCC;
      Border: 1px solid #093;
      Margin: 10px 12px 20px 15px;
   }
</style>
</head>
…
<body>
   <div id="cabecera">Here is the heading</div>
   ...
   <a href="#cabecera">Go to the heading</a>
</body>
```

ACTIVITY 1

Create an HTML file with two texts, one is a heading and the other is a paragraph. Then add styles in the <head> with classes for showing the heading in green letters and blue background and, the paragraph in blue letters and green background. Solve the activity in two ways:

1) Using styles with 4 classes (`.greentext`, `.bluetext`, `.greenbackground` and `.bluebackground`)

2) Using styles with 2 classes (`.greentextbluebackground` and `.bluetextgreenbackground`).

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.4 CSS Pseudo-classes

A pseudo-class is used to define a **special state of an element**.

For example, it can be used to:

- Style an element when a user mouses over it (:hover)
- Style visited and unvisited links differently (:link, :visited)
- Style an element when it gets focus (:focus).

```
selector:pseudo-class {
   property: value;
}
```

2.2 SELECTORS: INLINE STYLES BASED ON TAGS, IN CLASSES AND IN IDENTIFIERS

2.2.5 CSS Pseudo-elements

A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter (::first-letter), or line (::first-line), of an element
- Insert content before (::before), or after (::after), the content of an element.


```
selector::pseudo-element {
   property: value;
}
```

2.3 GROUPING AND NESTING SELECTORS

- The selectors can be grouped and nested to achieve CSS styles more defined and specific, and to have a more optimized CSS file and easier to understand by the development team.

- Any selector can be grouped following this syntax:

```
selector1, selector2 {
   attribute1: value1;
   attribute2: value2;
...}
```

2.3 GROUPING AND NESTING SELECTORS

- This way you can apply the same style to a group of selectors at the same time. For example, if you want to apply a font type Arial to <h1>, <p> and <h3>, the rule would be:

```
h1,p,h3 {
 font-family: arial;
}
```

- The less optimized version of the same rule would be:

```
h1 {font-family: arial;}
 p {font-family: arial;}
h3 {font-family: arial;}
```

2.3 GROUPING AND NESTING SELECTORS

- For identifiers and classes:

```
#myheader, #myfooter {color: #00FF00}
.myheader, h2, #myfooter {color: #FF0000}
```

- The use of grouped selectors answers the best practices in CSS.
- To **minimize the code**, you should group selectors. In this way, it's easier to detect and correct errors.

2.3 GROUPING AND NESTING SELECTORS

- The selectors can be nested in order to achieve more specific and defined styles. This nesting is called in CSS, *contextual selectors*, that let you apply a style to an element depending on the elements you have around.
- We have 3 type of nested selectors:
  - Descendant selectors (or, *common nested selector*)
  - Child selectors
  - Adjacent sibling selectors

2.3 GROUPING AND NESTING SELECTORS

- At times, authors may want selectors to match an element that is the descendant of another element (e.g., "Match those *em* elements that are contained by an *h1* element"). A **descendant selector** is made up of two or more selectors separated by *white space*. A descendant selector of the form "A B" matches when an element B is an arbitrary descendant of some ancestor element A.

```
h1 em { color: blue }
```

- This rule will match the *em* in the following fragment:

```
<h1>This <span class="myclass">headline
is <em>very</em> important</span>
</h1>
```

2.3 GROUPING AND NESTING SELECTORS

- The common nesting explained above behaves in the same way if the labels are consecutive or if there are intermediate labels.
- A **child selector** matches when an element is the child of some element (without intermediate elements). A child selector is made up of two or more selectors separated by ">".
- The following rule sets the style of all *p* elements that are children of *body*:

```
body > P { line-height: 1.3 }
```

2.3 GROUPING AND NESTING SELECTORS

- **Adjacent sibling selectors** have the following syntax: E1 + E2, where E2 is the subject of the selector. The selector matches if E1 and E2 share the same parent in the document tree and E1 immediately precedes E2, ignoring non-element nodes (such as text nodes and comments).
- The next example reduces the vertical space separating an h1 and an h2 that immediately follows it:


```
h1 + h2 { margin-top: -5mm }
```

ACTIVITY 2

Interpret the following code. What color are the list elements?

Then check your answer on the browser.

```
<head>
  <meta charset="UTF-8">
  <title>SELECTORS</title>
  <style>
    h1+div {background-color: yellow}
    .level1 {color: green}
    .level2 {color: blue}
    ul ul .level2 {font-size: x-small}
    ul ul li {color: red}
  </style>
</head>
<body>
  <h1>Grouping and nesting selectors</h1>
  <div>Examples ...</div>
  <div>A list</div>
  <ul>
    <li class="level1">Grouping</li>
    <li class="level1">Grouping</li>
    <ul>
      <li class="level2">Nesting children</li>
      <li class="level2">Nesting siblings</li>
      <li>Common nesting</li>
    </ul>
  </ul>
</body>
```

2.4 GOOD PRACTICE WHEN WRITING CSS

- The selectors are named in lowercase, never begin with special characters or numeric.
- The name of the selectors must be specific and clear, so you have a greater expressive power.
- The class and identifiers names must not describe a visual feature such as color, size or position.
- The names must follow a vision more semantic than structural.
- Separate words with hyphens or uppercase.
- Don't make an excessive use of classes.
- Grouping the rules according to their selector whenever possible.
- At the start of a CSS, it is advisable to define the tag selectors.
- Structure visually the properties (no more than 3 in every line).

2.5 ATTRIBUTES. BOX MODEL.

Content - The content of the box, where text and images appear.

2.5 ATTRIBUTES. BOX MODEL.

2.5.1 Units of Measure

- Absolute:
  - Inches (in). One inch = 2.54 cm
  - Centimeter (cm).
  - Millimeters (mm).
  - Points (pt). One point = 1/72 of inches
  - Picas (pc). One pica = 12 points
- Relatives:
  - Pixels (px)
  - Em (em) (*font-size* of the element where it is use)

Example:

```
div { font-size: 15px;}
  p { font-size: 1.2em;}
```

2.5 ATTRIBUTES. BOX MODEL.

2.5.2 Position attributes

- They are: top, left, right and bottom.
- Top and bottom indicate the vertical distance where the layer will be placed and, left and right the horizontal.
- However, this distance depends on the **position** attribute that defines the type of position of the layer. If the position attribute is *absolute* (or fixed), top indicates the distance from the top of the layer with respect to the top of the page. If the attribute position was *relative*, top indicates the distance from where it was writing at this moment in the page to the top of the layer.

2.5 ATTRIBUTES. BOX MODEL.
2.5.3 Margin attributes

The attributes margin-left, margin-right, margin-top, margin-bottom indicate the separation between another box and the edge of the one shown. Example:

Margin - Clears an area around the border. The margin does not have a background color, it is completely transparent.

```
<head>
  <style>
    body { margin-top: 100px;
           margin-right: 100px;
           margin-bottom: 100px;
           margin-left: 100p;
           border: 3px dotted blue;}
    div { margin-top: 15px;
          margin-right:15px;
          margin-bottom: 15px;
          margin-left: 15px;
          border: 3px dotted red;}
  </style>
</head>
<body>
  <div style="font-size:0.5in;">Text of 0.5 inches</div>
</body>
```

Text of 0.5 inches

2.5 ATTRIBUTES. BOX MODEL.
2.5.4 Padding attributes

The attributes are: padding-to , padding-right, padding-bottom and padding-left .

Padding - Clears an area around the content. The padding is affected by the background color of the box.

```
<head>
  <style>
    body { margin: 100px 100px 100px 100px;
           border : 3px dotted blue;}
    div { margin-top: 15px;
          margin-right:15px;
          margin-bottom: 15px;
          margin-left: 15px;
          padding-left: 10px;
          border: 3px dotted red;}
  </style>
</head>
<body>
  <div style="font-size: 0.5in;">Text of 0.5
inches</div>
</body>
```

Text of 0.5 inches

2.5 ATTRIBUTES. BOX MODEL.
2.5.5 Border attributes

- The attributes are: border-top, border-bottom, border-right and border-left.
- Define the style and color of the edge of the box. Values: none, dotted, dashed, solid, double, groove, ridge, inset and outset.
- The border shorthand property sets all the border properties in one declaration.
- The properties that can be set, are (in order): border-width, border-style, and border-color.
- It does not matter if one of the values above are missing, e.g. border:solid #ff0000; is allowed.

2.5 ATTRIBUTES. BOX MODEL.
2.5.5 Border attributes

Border - A border that goes around the padding and content. The border is affected by the background color of the box.

```
<head>
  <style>
    body { margin: 100px 100px 100px 100px;
                border-style: inset;
                border-color: blue;
                border-radius: 15px;
                porder-width: thick;
                padding: 15px 15px 15px 15px;}
    div { margin: 15px;
            padding-left: 10px;
            border-top: 3px dotted red;
            border-right: 2px solid blue;
            border-bottom: 3px double green;
            border-left: 3px groove red;}
  </style>
</head>
<body>
  <div style="font-size: 0.5in;">Text of 0.5
inches</div>
</body>
```

Text of 0.5 inches

2.5 ATTRIBUTES. BOX MODEL.
2.5.6 Content attributes

- *Width* sets the distance between the boundary of the padding-left and padding-right.
- *Height*, same but between the padding-bottom and padding-top.
- **Important**: When you set the width and height properties of an element with CSS, you just set the width and height of the content area. To calculate the full size of an element, you must also add padding, borders and margins.
- Other attributes:
  - All attributes available in CSS 2.1 of the W3C:

    http://www.w3c.es/Divulgacion/GuiasReferencia/CSS21/#modeloCajas
  - This other URL shows the new additions to CSS3, where you can see *border-radius*
  - http://www.css3.info/preview/

ACTIVITY 3

What is the total width of this element?

```
div {
    width: 450px;
    padding: 20px;
    border: 10px solid gray;
    margin: 5px;
}
```

2.6 ELEMENTS: BACKGROUND COLORS, TEXT, LINKS, LISTS, TABLES, VISIBILITY, IMAGES

- **Fonts attributes**: *color, font-size, font-family, font-weight, font-style*.
- **Paragraphs attributes**: line-height, text-decoration, text-align,text-indent,text-transform.
- **Background attributes**: background-color, background-image.
- **Table attributes**: caption-side, table-layout, border-collapse, border-spacing, empty-cells.

2.6 ELEMENTS: BACKGROUND COLORS, TEXT, LINKS, LISTS, TABLES, VISIBILITY, IMAGES

- **Visibility attributes:** overflow, clip, visibility, display.
- **Lists attributes**: list-style-type, list-style-image,list-style-position.
- **Links attributes**:
  - **Normal links** (unvisited link): *a:link {attributes}*
  - **Visited links** (a link the user has visited): *a:visited {attributes}*
  - **Active links** (a link the moment it is clicked): *a:active {attributes}*
  - **Hover links** (a link when the user mouses over it): *a:hover {attributes}*

2.6 ELEMENTS: BACKGROUND COLORS, TEXT, LINKS, LISTS, TABLES, VISIBILITY, IMAGES

EXAMPLE:

```
<html>
  <head>
    <style>
      a.menus:link { text-decoration:none;
                     color: #000000;
                     border:#FFFFFF 1px solid; }

      a.menus:visited { text-decoration:none;
                        color:#cccccc; }


      a.menus:hover { text-decoration:underline;
                      color: #003399;
                      background: #red;
                      border:#FFFFFF 1px solid;}

      a.menus:active { text-decoration:none;
                       color: #003399;
                       background: #green;
                       border:#FFFFFF 1px solid; }
    </style>
  </head>

  <body>
    <a href="#" class="menus">Link one</a>
    <a href="#" class="menus">Link two </a>
    <a href="#" class="menus">Link three</a>
  </body>
</html>
```

2.7 OVERLAP AND PRECEDENCE OF STYLES

2.7.1 Overlapping boxes

- Managing these aspects involves understanding well the operation of the CSS and get very accurate results and personal.
- The *z-index* property specifies the stack order of an element. An element with greater stack order is always in front of an element with a lower stack order.
- **Note**: z-index only works on positioned elements (position: absolute, position: relative, or position: fixed).

2.7 OVERLAP AND PRECEDENCE OF STYLES
2.7.1 Overlapping boxes

- The **position** property specifies the type of positioning method used for an element. Values:
  - ***Static***. Default. The box is a normal box, laid out according to the normal flow of the page. The 'top', 'right', 'bottom', and 'left' properties do not apply.
  - ***Absolute***. The box's position is specified with the 'top', 'right', 'bottom', and 'left' properties. It's positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed). However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

    Absolutely positioned boxes are taken out of the normal flow.

Note: A "positioned" element is one whose position is anything except static.

2.7 OVERLAP AND PRECEDENCE OF STYLES

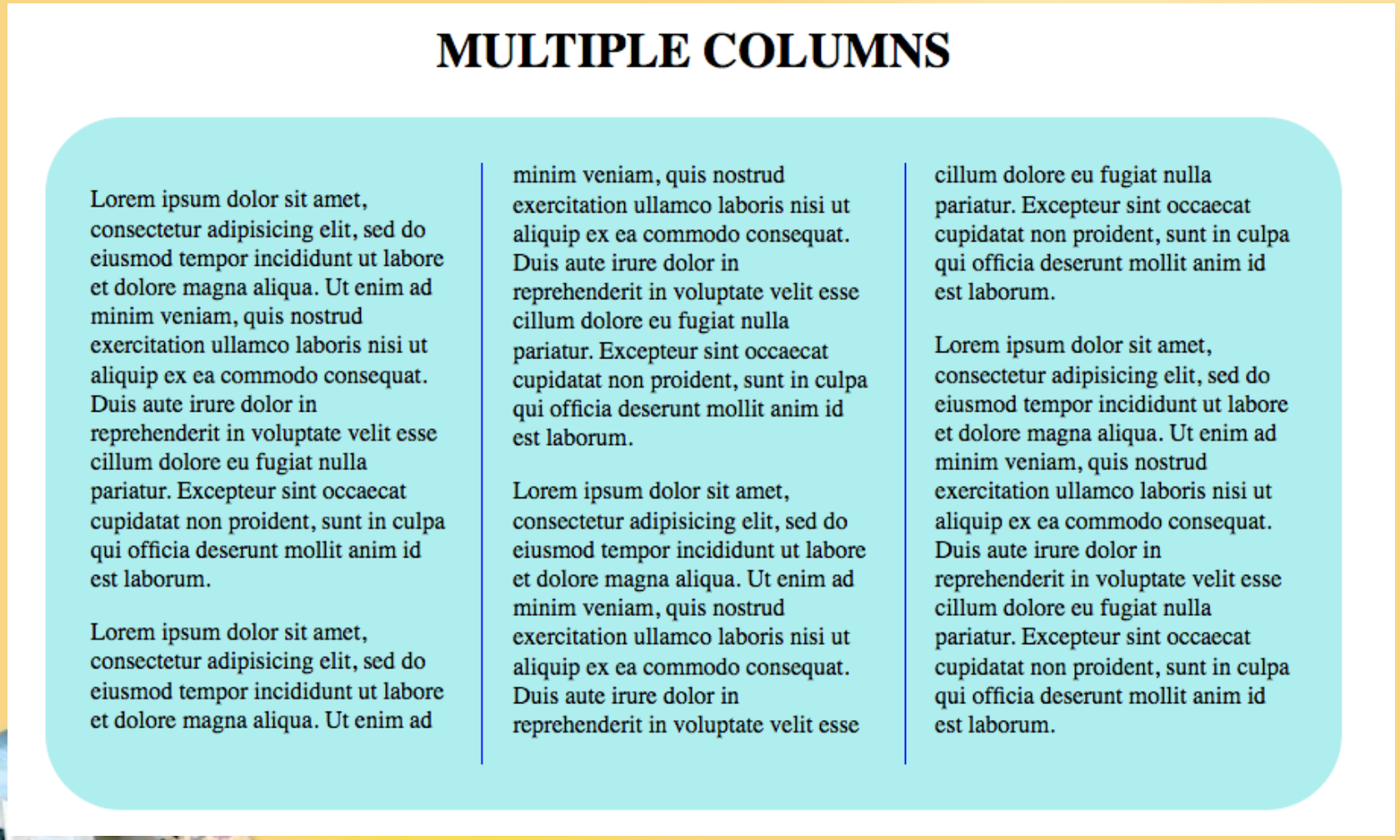
2.7.1 Overlapping boxes

- ***Relative***. The box's position is calculated according to the normal flow. Then the box is positioned relative to its normal position. Other content will not be adjusted to fit into any gap left by the element.
- ***Fixed***. The element is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

  A fixed element does not leave a gap in the page where it would normally have been located.
- ***Inherit***. The value of the position property is inherited from the parent element

ACTIVITY 4

Multiple column. Try to design a web page like in the image using the properties `column-count`, `column-rule` and `column-gap`.



2.7 OVERLAP AND PRECEDENCE OF STYLES

2.7.2 Precedence of styles

- The precedence of styles is associated with the concept of specificity of a rule.
- Specificity refers to the weight taken by each one of the elements of a style sheet. The more weight more specificity. The more specific rule has a less difficult to ensure that this and no other rules are going to be applied to certain content.

2.7 OVERLAP AND PRECEDENCE OF STYLES

2.7.2 Precedence of styles

- A simple way to calculate the specificity of a rule is adding points depending on selectors type that contains:
  - Is given a value of 1 point to a tag selector (for example, h1, p, div).
  - Is given a value of 10 points to a class selector.
  - Is given a value of 100 points to a identifier selector.
  - Is given a value of 1000 points to a style attribute.

2.7 OVERLAP AND PRECEDENCE OF STYLES

2.7.2 Precedence of styles

EXAMPLE:

```
<style>
   p {background: crimson;} /* 1 points specificity */

   .parrafo {background: pink;} /* 10 points specificity */

   p.parrafo {background: maroon;} /*11 points specificity*/

   #id-parrafo {background: orange;} /*100 points specificity*/

   p#id-parrafo {background: red;} /*101 points specificity*/

   p.parrafo#id-parrafo {background:green;} /*111 points
specificity*/
</style>
```

2.8 CREATE AND LINK STYLE SHEET

1. The first alternative is using the style attribute in the HTML tags (**integrated style rules**). An **inline style** loses many of the advantages of style sheets by mixing content with presentation. Use this method sparingly!

   To use inline styles you use the style attribute in the relevant tag. The style attribute can contain any CSS property. Eg

```
<p style="background: black;">
The background of this paragraph is black
</p>
```

2.8 CREATE AND LINK STYLE SHEET

2. Another alternative is to use the tag <style> within the same file (**incrusted style rules**), in the head section of an HTML page. An **internal style sheet** should be used when a single document has a unique style.

   Attributes more important of the tag <style>:

   - ***Type*** (value *text/css*). Specifies the MIME type of the style sheet.
   - ***Media***. Specifies what media/device the media resource is optimized for.
   - ***Title***. Specifies extra information about an element.

2.8 CREATE AND LINK STYLE SHEET

3. An **external style sheet** is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing one file. Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the head section:

```
<head>
    <link rel="stylesheet" href="mystyle.css">
</head>
```

An external style sheet can be written in any text editor. The file should not contain any html tags. Your style sheet should be saved with a .css extension.

2.8 CREATE AND LINK STYLE SHEET

The rel attribute is used to define the relationship between the linked file and the HTML document:

- **rel = *stylesheet*** specifies a persistent style.
- **rel = *stylesheet*** and the **title** attribute specifies a preferred style.
- **rel = "*alternate stylesheet*"** defines an alternate style

4. Another alternative to link external CSS is to use @import rule. This rule is included within <style>. An example of use of this rule is as follows:


```
<style>
  @import url("estilos.css");
</style>
```

2.10 TOOLS AND VERIFICATION TEST

- W3C: http://jigsaw.w3.org/css-validator/
- XHTML-CSS http://xhtml-css.com/
- Firebug https://getfirebug.com/
- Pendule
  https://chrome.google.com/webstore/detail/pendule/gbkffbkamcejhkcaocmkdeiiccpmjfdi
- List-o-matic
  http://www.accessify.com/tools-and-wizards/developer-tools/list-o-matic/
- CSS Layout Generator http://www.pagecolumn.com/
- CSS Text Wrapper http://csstextwrap.com/

2.11 CSS3

- Rounded Corners
- Colors
- Shadows
- Gradients
- Multiple background
- Transitions
- Transforms
- Animations
- Media Queries
- Box Sizing
- Flexbox

2.11 CSS3

border-radius property

- The border-radius property is used to add rounded corners to an element.
  - The border-radius property is a shorthand property for setting the four border-*-radius properties.
  - If you specify only one value for the border-radius property, this radius will be applied to all 4 corners.
  - However, you can specify each corner separately if you wish.

```
div { padding: 10px 40px;
      background: cyan;
      width: 300px;
     border-radius: 25px;
}
```

The border-radius property allows you to add rounded corners to elements.

2.11 CSS3
COLORS

RGBA colors.
`rgba(red,green,blue,alpha)`. The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

rgba(0,0,255,0)
rgba(0,255,0,0.2)
rgba(0,255,0,0.4)
rgba(0,255,0,0.6)
rgba(0,255,0,0.8)
rgba(0,255,0,1)

2.11 CSS3
COLORS

HSL colors.

`hsl(hue,saturation,lightness)`:

1. Hue is a degree on the color wheel (from 0 to 360):
   - 0 (or 360) is red
   - 120 is green
   - 240 is blue
2. Saturation is a percentage value: 100% is the full color.
3. Lightness is also a percentage; 0% is dark (black) and 100% is white.

2.11 CSS3
COLORS

hsl(120,100%,0%)	hsl(120,50%,0%)
hsl(120,100%,20%)	hsl(120,50%,20%)
hsl(120,100%,40%)	hsl(120,50%,40%)
hsl(120,100%,60%)	hsl(120,50%,60%)
hsl(120,100%,80%)	hsl(120,50%,80%)
hsl(120,100%,100%)	hsl(120,50%,100%)

2.11 CSS3
COLORS

HSLA colors.

`hsla(hue,saturation,lightness,alpha)`, where the alpha parameter defines the opacity. The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

hsla(240,100%,50%,0)
hsla(240,100%,50%,0.2)
hsla(240,100%,50%,0.4)
hsla(240,100%,50%,0.6)
hsla(240,100%,50%,0.8)
hsla(240,100%,50%,1)

2.11 CSS3
COLORS

Opacity.

The opacity property value must be a number between 0.0 (fully transparent) and 1.0 (fully opaque).

background-color:red;opacity:0.5;

background-color:green;opacity:0.5;

background-color:blue;opacity:0.5;

ACTIVITY 5

What is the difference between:

```
p {background-color:rgb(0,255,0);
   opacity:0.5;}
p {background-color:rgba(0,255,0,0.5);}
```

?

2.11 CSS3
GRADIENTS

CSS3 gradients let you display smooth transitions between two or more specified colors.

Advantages (vs using images for these effects):

1) reduce download time and bandwidth usage

2) look better when zoomed (because the gradient is generated by the browser)

CSS3 defines two types of gradients:

- Linear Gradients (goes down/up/left/right/diagonally)
- Radial Gradients (defined by their center)

2.11 CSS3
GRADIENTS

Linear gradients:

- you must define at least two color stops (colors you want to render smooth transitions among).
- you can also set a starting point and a direction (or an angle) along with the gradient ef

```
<linear-gradient> = linear-gradient(
[
        [ [top | bottom] || [left | right] ]
        |
        <angle>
,]?
<color-stop>[, <color-stop>]+
);
```

2.11 CSS3
GRADIENTS

EXAMPLES:

Linear Gradient - Top to Bottom (this is default)

```
linear-gradient(blue,yellow)
```

Linear Gradient - Left to Right

```
linear-gradient(to right,blue,yellow)
```

Linear Gradient - Diagonal

```
linear-gradient(to bottom right,blue,yellow)
```

2.11 CSS3
GRADIENTS

Using Angles

```
linear-gradient(45deg, blue, yellow);
```

Using Multiple Color Stops

```
background: linear-gradient(to right,
red,orange,yellow,green,blue,indigo,violet);
```

Using Multiple Color Stops

```
background: linear-
gradient(red,yellow,green);
```

2.11 CSS3
GRADIENTS

Using Transparency

```
linear-gradient(to
right,rgba(255,0,0,0),rgba(255,0,0,1));
```

...

```
linear-gradient(to right,white,red);
```

Repeating a linear-gradient

```
repeating-linear-gradient(red, yellow 10%,
green 20%);
```

2.11 CSS3
GRADIENTS

Repeating a linear-gradient

```
repeating-linear-gradient(to right,blue,red
10%,blue 20%);
```

Repeating a linear-gradient

```
repeating-linear-gradient(45deg,red,yellow
7%,green 10%);
```

Repeating a linear-gradient

```
repeating-linear-gradient(190deg,red,yellow
7%,green 10%);
```

2.11 CSS3
GRADIENTS

Radial gradients:

- it is defined by its center.
- you must also define at least two color stops.
- By default, shape is ellipse, size is
  position is center.

```
<radial-gradient> = radial-gradient(
        [<bg-position>,]?
        [
                [<shape> || <size>]
                |
                [<length> | <percentage>]{2}
        ,]?
        <color-stop>[, <color-stop>]+
)
```

```
<shape> = [ circle | ellipse ]
```


```
<size> = [ closest-side | closest-corner | farthest-side | farthest-corner |
```

2.11 CSS3
GRADIENTS

EXAMPLES:

Radial Gradient - Evenly Spaced Color Stops (this is default)

```
radial-gradient(red,yellow,green);
```

Radial Gradient - Evenly Spaced Color Stops (this is default)

```
radial-gradient(red,yellow);
```

Radial Gradient - Differently Spaced Color Stops

```
radial-gradient(red 5%,yellow 15%,green 60%);
```

2.11 CSS3
GRADIENTS

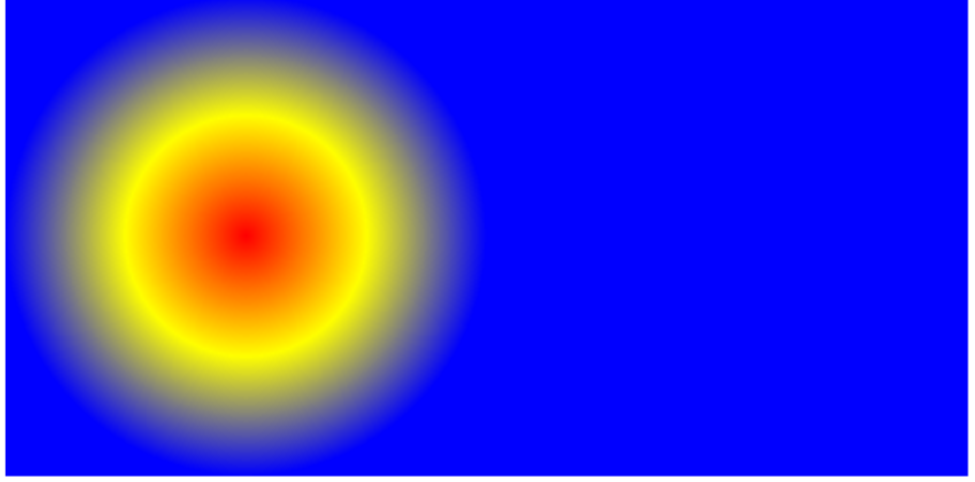
Set Shape

```
radial-gradient(circle,red,yellow,green);
```

2.11 CSS3
GRADIENTS

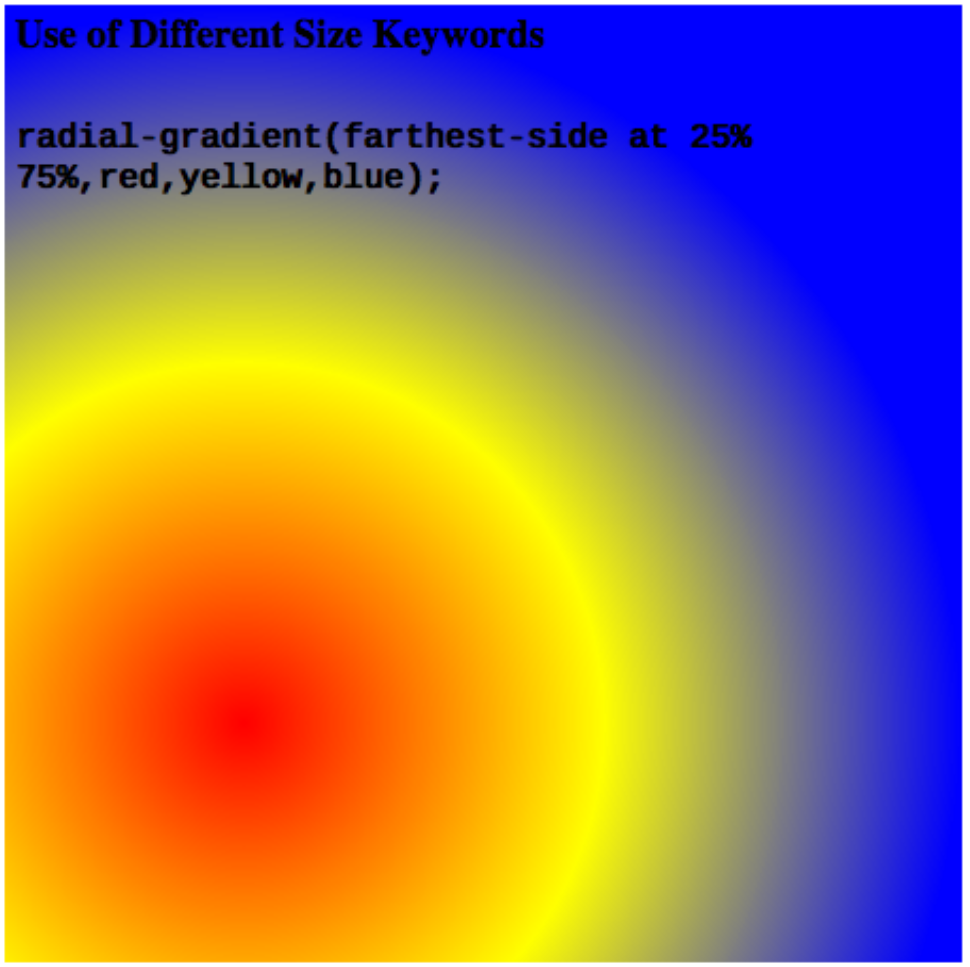
Use of Different Size Keywords

```
radial-gradient(closest-side at 25%
75%,red,yellow,blue);
```



Use of Different Size Keywords

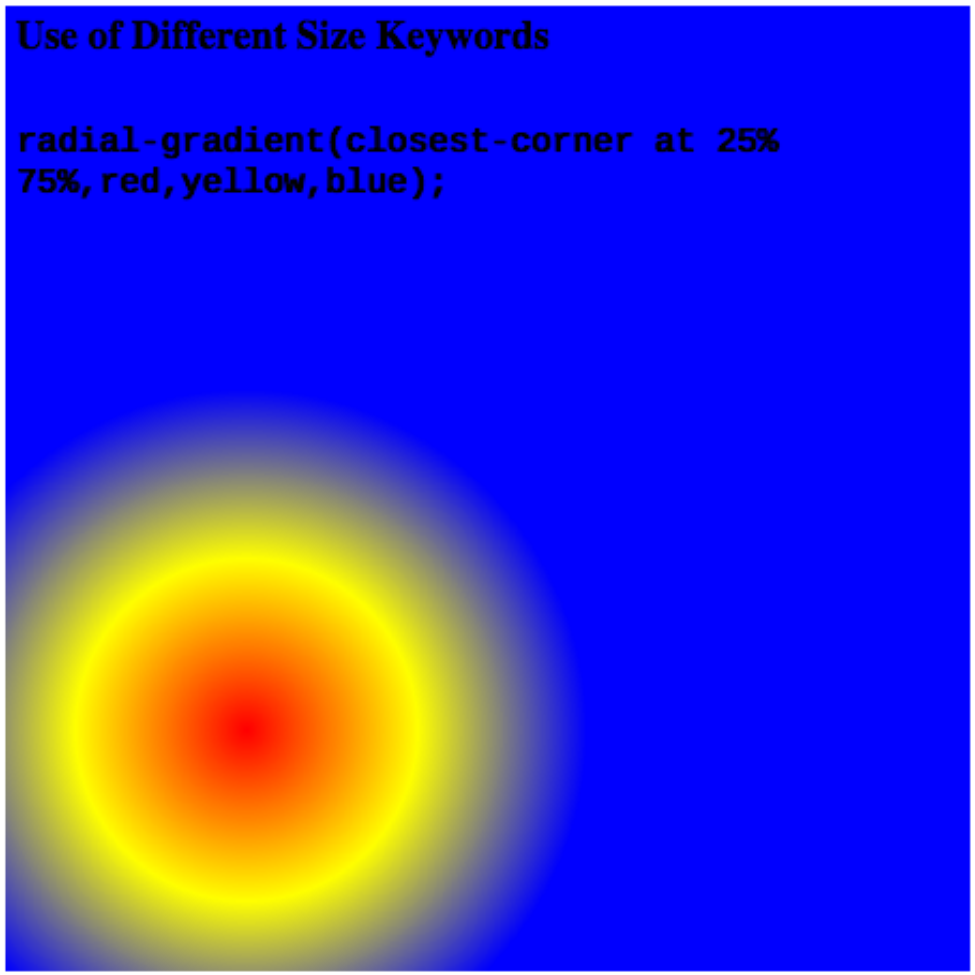
```
radial-gradient(farthest-side at 25%
75%,red,yellow,blue);
```



2.11 CSS3
GRADIENTS

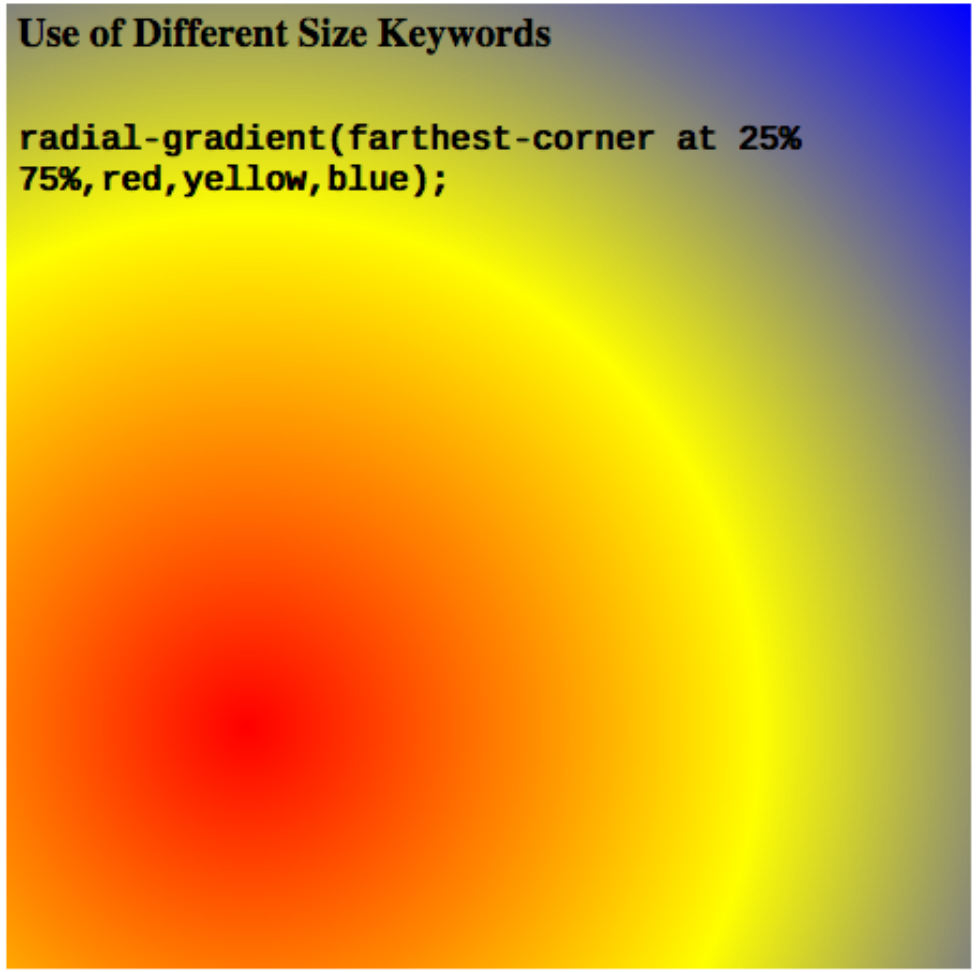
Use of Different Size Keywords

```
radial-gradient(closest-corner at 25%
75%,red,yellow,blue);
```

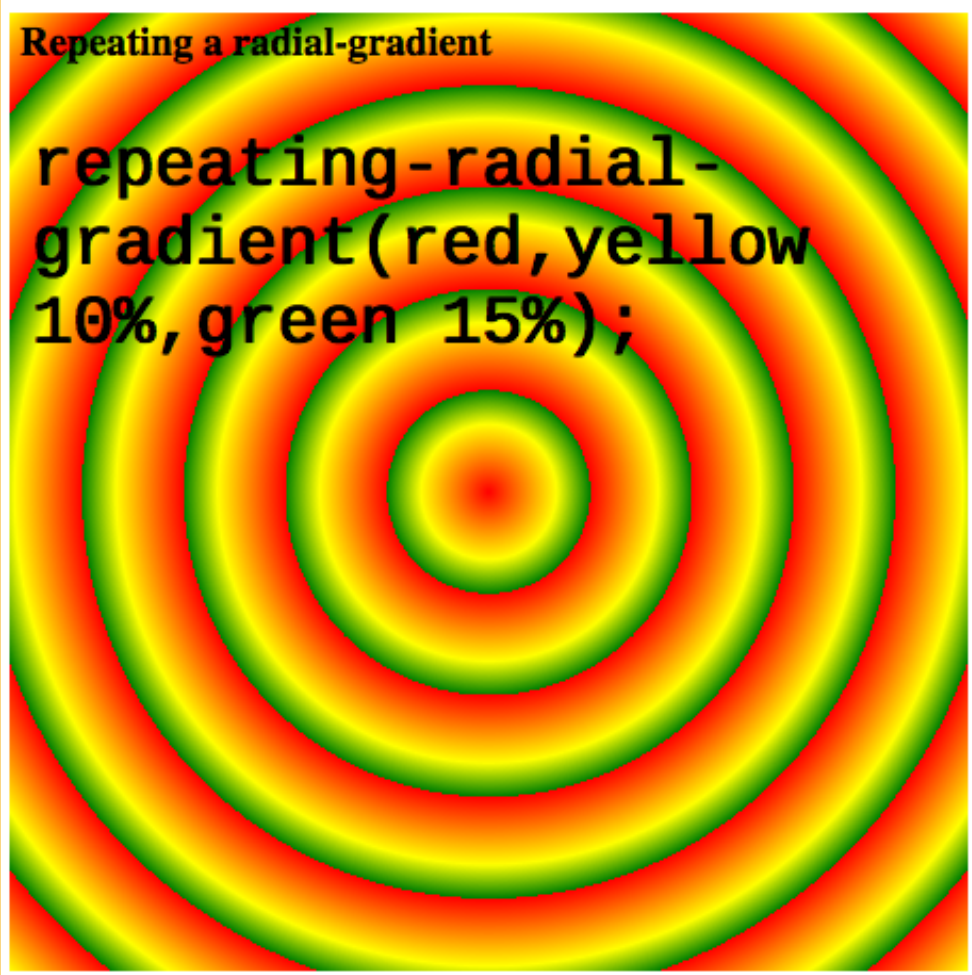


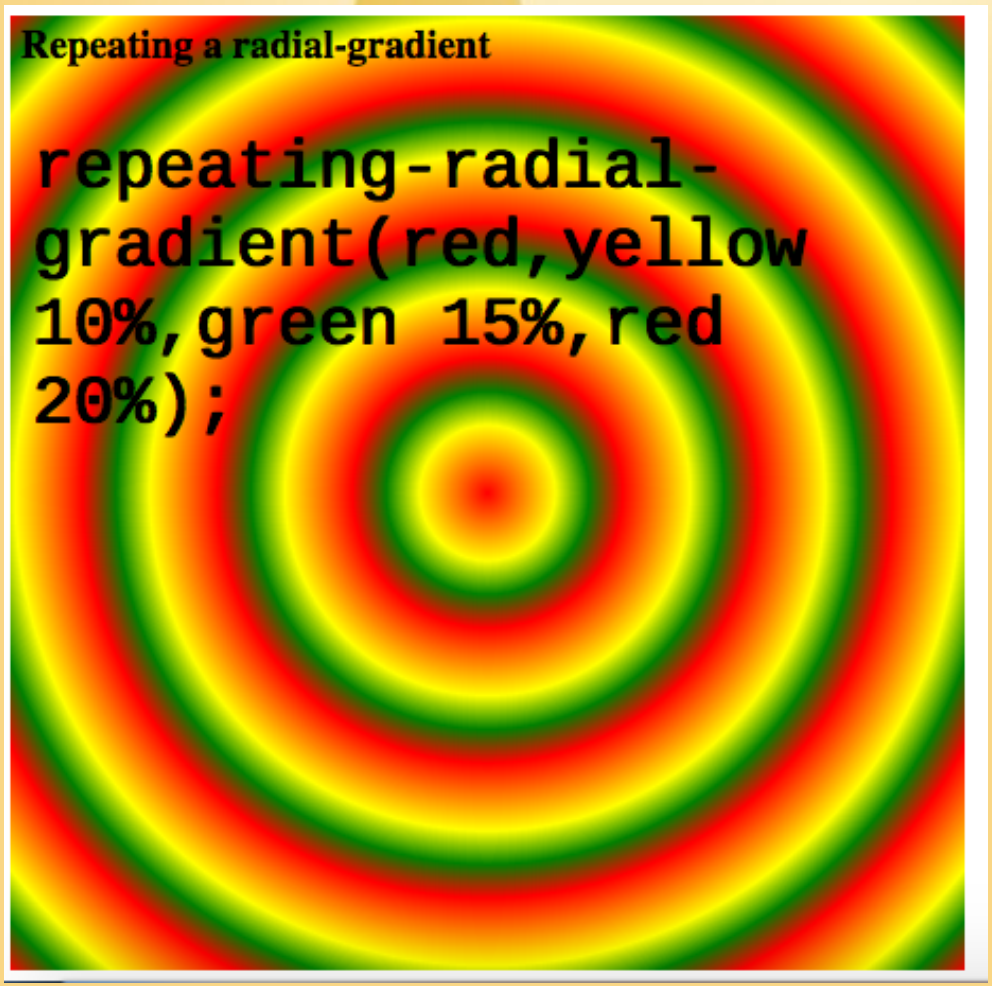
Use of Different Size Keywords

```
radial-gradient(farthest-corner at 25%
75%,red,yellow,blue);
```



2.11 CSS3
GRADIENTS





2.11 CSS3
MULTIPLE BACKGROUNDS

- CSS3 allows you to add multiple background images for an element, through the `background-image` property.
- The different background images are separated by commas, and the images are stacked on top of each other, where the first image is closest to the viewer.
- Multiple background images can be specified using either the individual background properties or the background shorthand property.

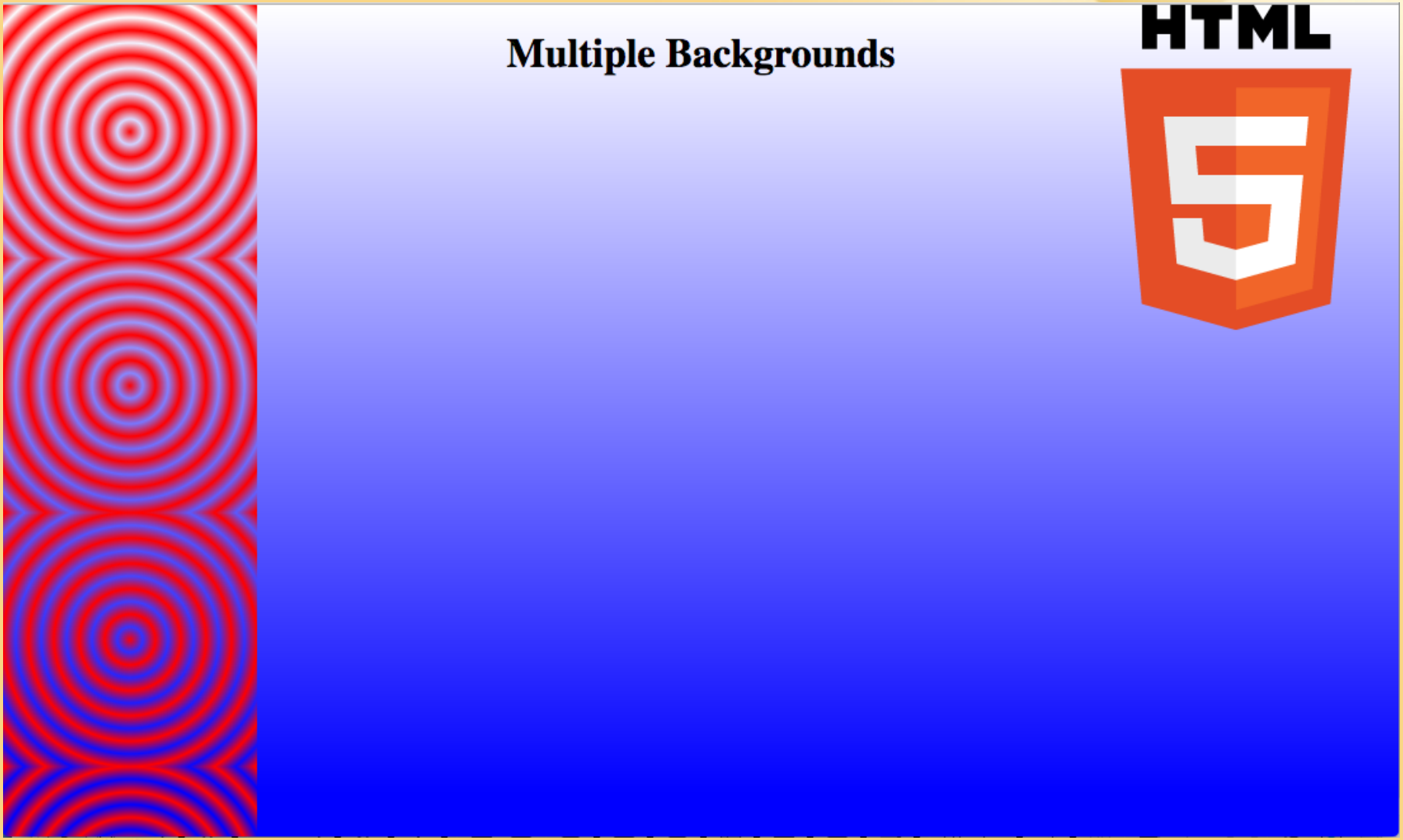
2.11 CSS3
MULTIPLE BACKGROUNDS
EXAMPLE

```
html {
  height: 100%;
  background-image: url(HTML5.png),
                    repeating-radial-gradient(circle farthest-side,transparent 10%,red 20%,transparent 30%),
                    linear-gradient(to top,blue 5%,transparent);
  background-size: auto,200px 200px,auto;
  background-position: right top,left top;
  background-repeat: no-repeat,repeat-y;
}
```

The same using the shorthand property:

```
html {
  height: 100%;
  background: url(HTML5.png) right top no-repeat,
                repeating-radial-gradient(circle farthest-side,transparent 10%,red 20%,
                  transparent 30%) left top /200px 200px repeat-y,
                linear-gradient(to top,blue 5%,transparent);
}
```

2.11 CSS3
MULTIPLE BACKGROUNDS



ACTIVITY 5

Create an empty html giving styles with CSS to get the following background. You can only use one image for the grass on the footer.

2.11 CSS3
TRANSITIONS

- To change property values smoothly (from one value to another), over a given duration.
- To create a transition effect, you must specify two things:
  - the CSS property you want to add an effect to
  - the duration of the effect
- **Note**: If the duration part is not specified, the transition will have no effect, because the default value is 0.

2.11 CSS3
TRANSITIONS

The following example shows a 100px * 100px red element. The element has also specified a transition effect for the width property, with a duration of 2 seconds.

- The transition effect will start when the specified CSS property (width) changes value.
- We specify a new value for the width property when a user mouses over the element.

```
<!DOCTYPE html>
<html>
<head>
  <style>
    div {
      width: 100px;
      height: 100px;
      background: red;
      transition: width 2s;
    }

    div:hover {
      width: 300px;
    }
  </style>
</head>
<body>
  <div></div>

  <p>Hover over the div element above,
    to see the transition effect.</p>
</body>
</html>
```

2.11 CSS3
TRANSITIONS

Change Several Property Values:

- The following example adds a transition effect for both the width and height property, with a duration of 2 seconds for the width and 4 seconds for the height:

```
transition: width 2s, height 4s;
```

2.11 CSS3
TRANSITIONS

Specify the Speed Curve of the Transition

- The transition-timing-function property specifies the speed curve of the transition effect. Possible values:
  - **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

2.11 CSS3
TRANSITIONS

Delay the Transition Effect

- The transition-delay property specifies a delay (in seconds) for the transition effect.

```
transition-delay: 1s;
```

Using the shorthand property transition:

```
transition: width 2s linear 1s;
```

CSS Syntax

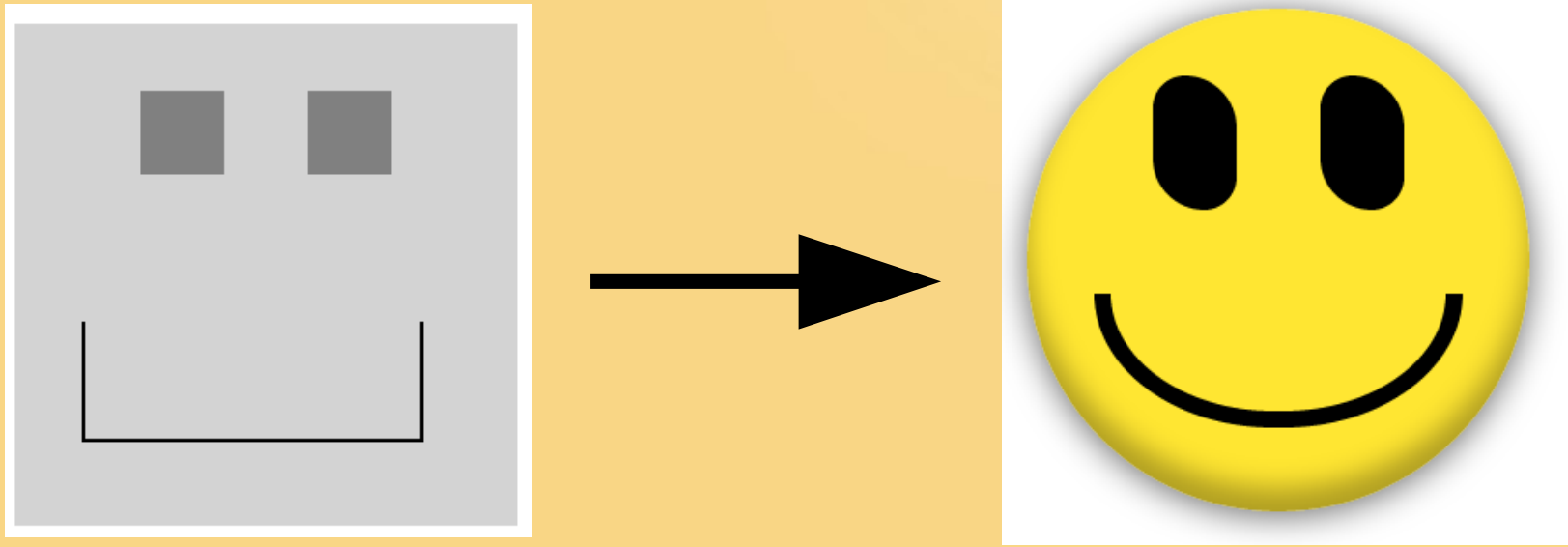
```
transition: property duration timing-function
delay|initial|inherit;
```

ACTIVITY 6

Create an interface with icons (at the bottom of the window) that grow with the style of Mac OS X and using the transition effects previously seen.

ACTIVITY 7

Design a smile like the picture on the left and when you mouse over it, make a transition to the smile on the right.



2.11 CSS3
TRANSFORMS

- Transforms allow you to translate, rotate, scale, and skew elements.
- A transformation is an effect that lets an element change shape, size and position.
- CSS3 supports 2D and 3D transformations.
- 2D transformation methods:
  - translate()
  - rotate()
  - scale()
  - skew()
  - matrix()

2.11 CSS3
TRANSFORMS

- **SYNTAX**:

```
transform: none|transform-functions|
initial|inherit;
```

- The translate() method moves an element from its current position (according to the parameters given for the X-axis and the Y-axis). Ex:

```
transform: translate(100px,-50px);
```

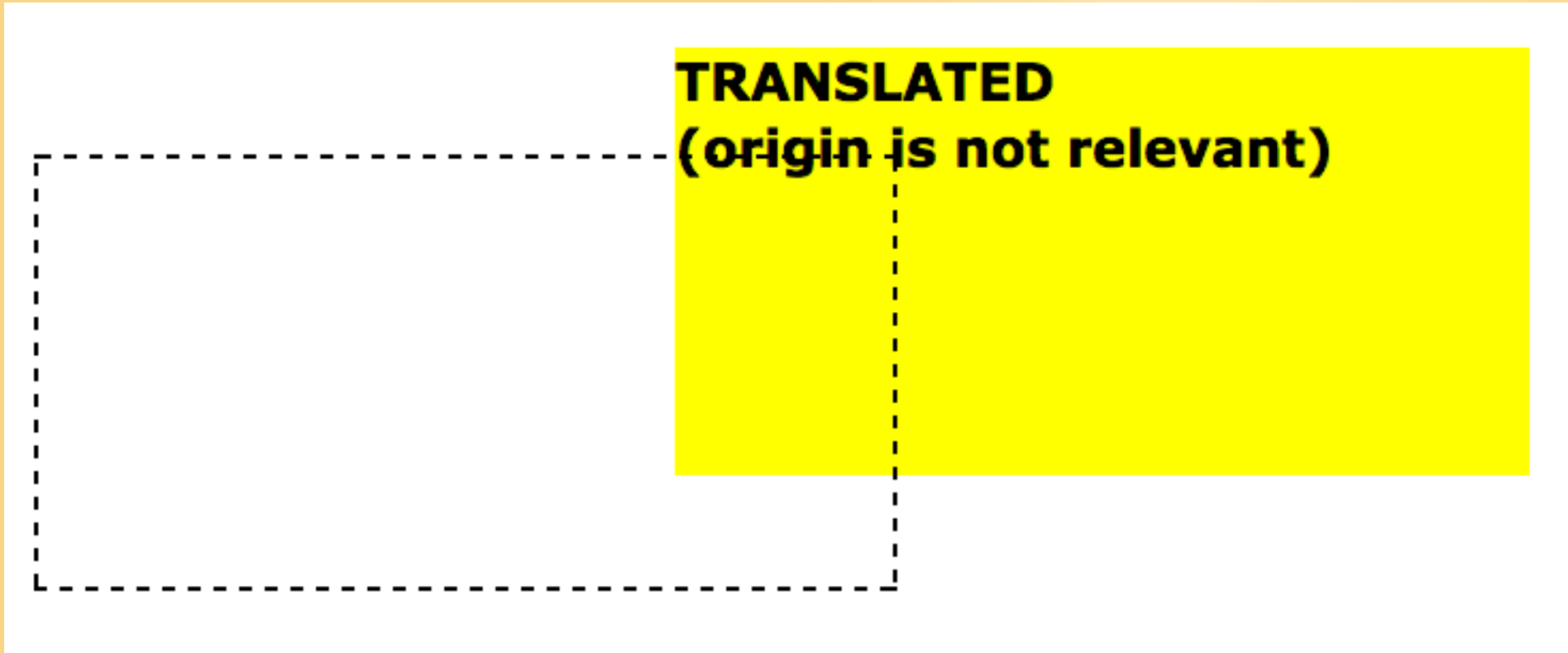
- The rotate() method rotates an element clockwise or counter-clockwise according to a given degree. Ex:

```
transform: rotate(30deg);
```

2.11 CSS3
TRANSFORMS

EXAMPLE:

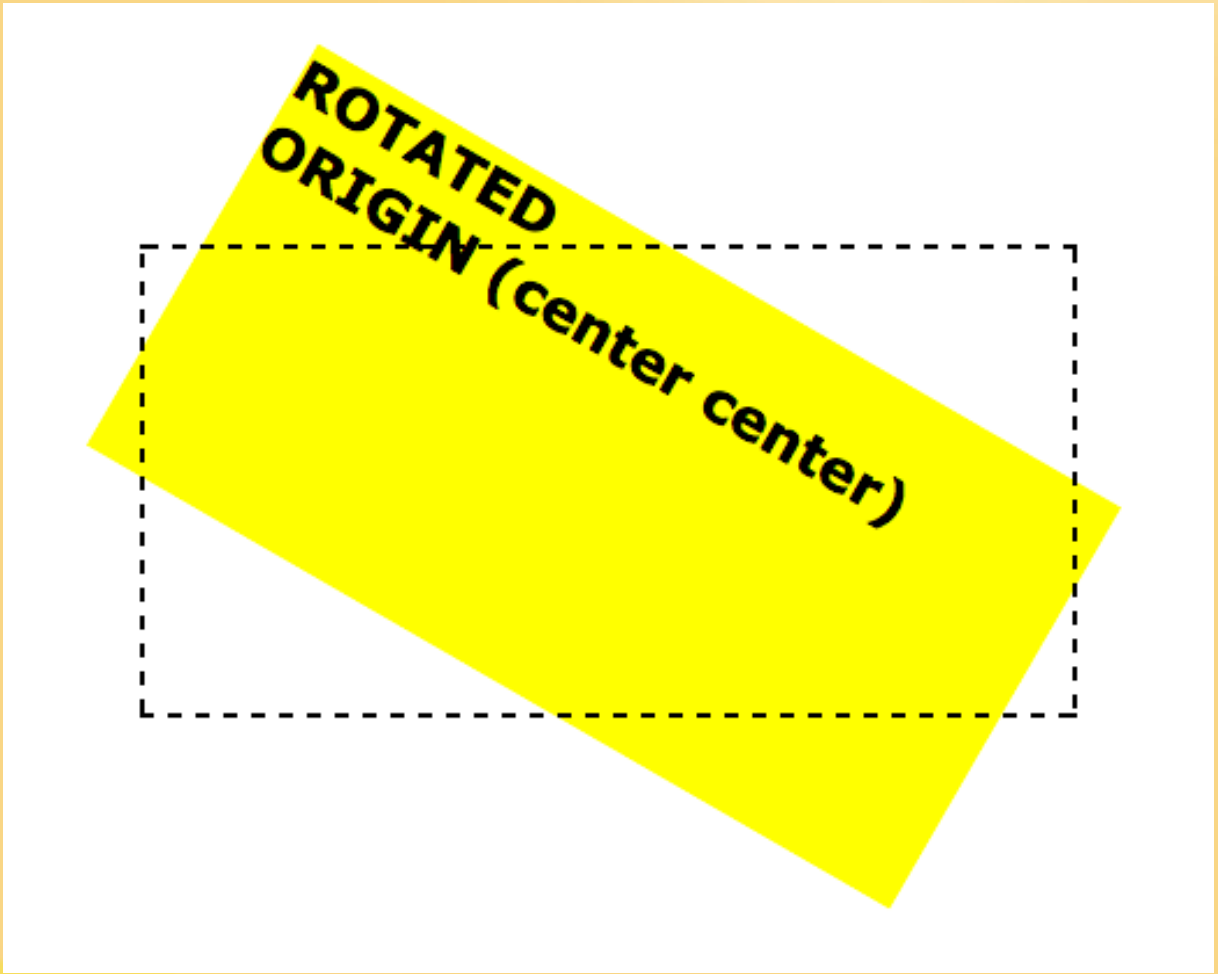
```
transform: translate(300px, -50px);
```



2.11 CSS3
TRANSFORMS

EXAMPLE:

```
transform: rotate(30deg);
```



2.11 CSS3
TRANSFORMS

- The scale() method increases or decreases the size of an element (according to the parameters given for the width and height).

```
transform: scale(2,0.5);
```

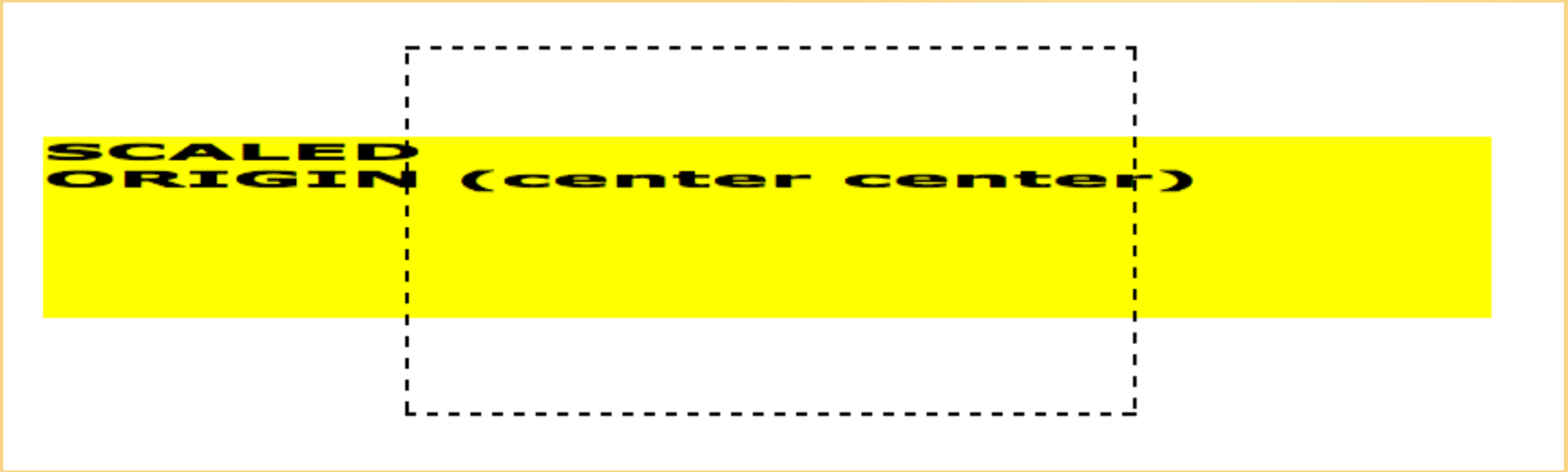
- The skewX() method skews an element along the X-axis by the given angle and/or along de Y-axis by the given angle. Ex:

```
transform: skew(20deg,10deg);
```

2.11 CSS3
TRANSFORMS

EXAMPLE:

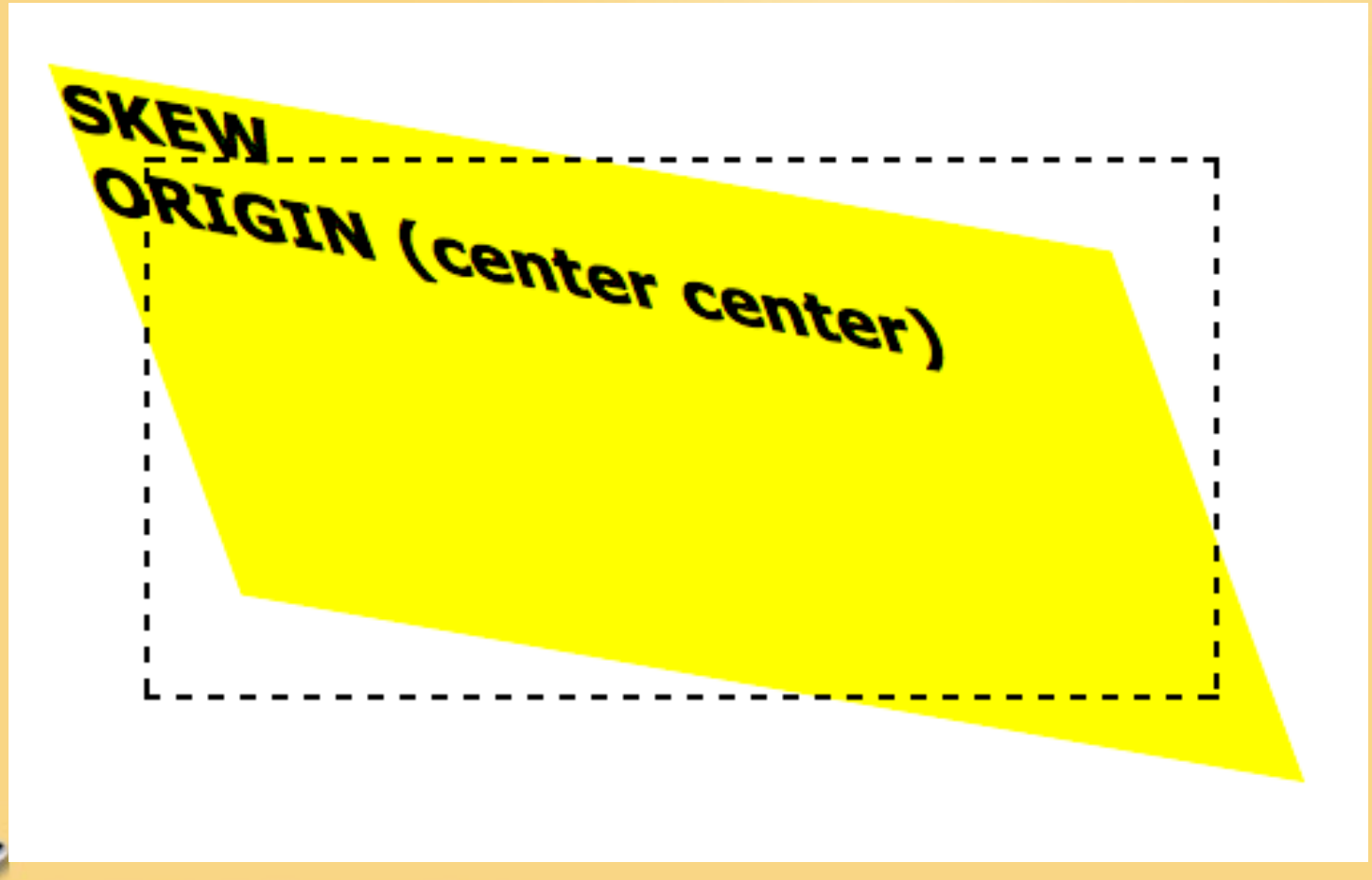
```
transform: scale(2,0.5);
```



2.11 CSS3
TRANSFORMS

EXAMPLE:

```
transform: skew(20deg,10deg);
```



2.11 CSS3
TRANSFORMS

- The matrix() method combines all the 2D transform methods into one.
- The matrix() method take six parameters, containing mathematic functions, which allows you to rotate, scale, move (translate), and skew elements.
- The parameters are as follow: matrix(scaleX(),skewY(),skewX(),scaleY(), translateX(),translateY()):

2.11 CSS3
TRANSFORMS

- CSS3 3D Transforms allows you to format your elements using 3D transformations.
  - The rotateX() method rotates an element around its X-axis at a given degree:
  - The rotateY() method rotates an element around its Y-axis at a given degree:
  - The rotateZ() method rotates an element around its Z-axis at a given degree:
- There are also translateX, translateY, translateZ, scaleX, scaleY, scaleZ, ...

2.11 CSS3
TRANSFORMS

```
transform: rotateX(180deg);
```


```
transform: rotateY(180deg);
```


```
transform: rotateZ(180deg);
```

2.11 CSS3
TRANSFORMS

- The transform-origin property allows you to change the position of transformed elements.
- 2D transformations can change the x- and y-axis of an element. 3D transformations can also change the z-axis of an element.
- **Note**: This property must be used together with the transform property.
- **SYNTAX**:

```
transform-origin: x-axis y-axis z-axis|
initial|inherit;
```

2.11 CSS3
ANIMATIONS

CSS3 animations allows animation of most HTML elements without using JavaScript or Flash!

- An animation lets an element gradually change from one style to another.
- You can change as many CSS properties as many times you want.
- To use CSS3 animation, you must first specify some keyframes for the animation.
- Keyframes hold what styles the element will have at certain times.

2.11 CSS3
ANIMATIONS

- When you specify CSS styles inside the @keyframes rule, the animation will gradually change **from** the current style **to** the new style at certain times.
- To get an animation to work, you must bind the animation to an element.
- **Note**: If the **animation-duration** property is not specified, the animation will have no effect, because the default value is 0.

```
<style>
  div {
    width: 100px;
    height: 100px;
    background-color: red;
    animation-name: example;
    animation-duration: 4s;
  }

  @keyframes example {
    from {background-color: red;}
    to {background-color: yellow;}
  }
</style>
```

2.11 CSS3
ANIMATIONS

- It is also possible to use percent. By using percent, you can add as many style changes as you like.

```
#ex2 {
  width: 300px;
  height: 100px;
  margin: 50px;
  background-color: red;
  animation-name: example2;
  animation-duration: 4s;
}

@keyframes example2 {
    0%   {background-color: red;}
    25%  {background-color: yellow;}
    50%  {background-color: blue;}
    100% {background-color: green;}
}
```

2.11 CSS3
ANIMATIONS

- You can combine more than one changes in the same animation:

```
#ex3 {
  position: relative;
  width: 120px;
  height: 100px;
  margin: 50px;
  background-color: red;
  animation-name: example3;
  animation-duration: 4s;
}

@keyframes example3 {
  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;}
}
```

2.11 CSS3
ANIMATIONS

- The **animation-delay** property specifies a delay for the start of an animation.

```
#ex4 {
  position: relative;
  width: 120px;
  height: 100px;
  margin: 50px;
  background-color: red;
  animation-name: example3;
  animation-duration: 4s;
  animation-delay: 2s;
}

@keyframes example3 {
  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;}
}
```

2.11 CSS3
ANIMATIONS

- The **animation-iteration-count** property specifies the number of times (can also be `infinite`) an animation should run.

```
#ex5 {
  position: relative;
  width: 120px;
  height: 100px;
  margin: 50px;
  background-color: red;
  animation-name: example3;
  animation-duration: 4s;
  animation-delay: 2s;
  animation-iteration-count: 3;
}

@keyframes example3 {
  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;}
}
```

2.11 CSS3
ANIMATIONS

- The **animation-direction** property is used to let an animation run in `reverse` direction or `alternate` cycles.

```
#ex7 {
  position: relative;
  width: 120px;
  height: 100px;
  margin: 50px;
  background-color: red;
  animation-name: example3;
  animation-duration: 4s;
  animation-delay: 2s;
  animation-iteration-count: 3;
  animation-direction: reverse;
}

@keyframes example3 {
  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;}
}
```

2.11 CSS3
ANIMATIONS

- The **animation-timing-function** property specifies the speed curve of the animation.
- Remember you can also use the shorthand animation property.

```
#div1,#div2,#div3,#div4,#div5 {
  width: 100px;
  height: 50px;
  background-color: cyan;
  font-weight: bold;
  position: relative;
  animation: mymove 5s infinite;
}

#div1 {animation-timing-function: linear;}
#div2 {animation-timing-function: ease;}
#div3 {animation-timing-function: ease-in;}
#div4 {animation-timing-function: ease-out;}
#div5 {animation-timing-function: ease-in-out;}

@keyframes mymove {
  from {left: 0px;}
  to {left: 300px;}
}
```

2.11 CSS3
MEDIA QUERIES

- Media queries in CSS3 extend the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.
- Media queries can be used to check many things, such as:
  - width and height of the viewport
  - width and height of the device
  - orientation (is the tablet/phone in landscape or portrait mode?)
  - resolution
- Using media queries are a popular technique for delivering a tailored style sheet to tablets, iPhone, and Androids.

2.11 CSS3
MEDIA QUERIES

Media Query Syntax:

```
@media not|only mediatype and (expressions) {
    CSS-Code;
}
```

- When a media query is true, the corresponding style sheet or style rules are applied, following the normal cascading rules.
- Unless you use the `not` or `only` operators, the media type is optional and the all type will be implied.
- Different stylesheets for different media:

```
<link rel="stylesheet" media="mediatype and|not|only (expressions)" href="print.css">
```

2.11 CSS3
MEDIA QUERIES

CSS3 Media Types:

Value	Description
all	Used for all media type devices
print	Used for printers
screen	Used for computer screens, tablets, smart-phones etc.
speech	Used for screenreaders that "reads" the page out loud

- For a full overview of all the media types and features/expressions, please look at the @media

2.11 CSS3
MEDIA QUERIES

EXAMPLE:

Resize the browser window to see the effect!

The media query will only apply if the media type is screen and the viewport is 480px wide or wider.

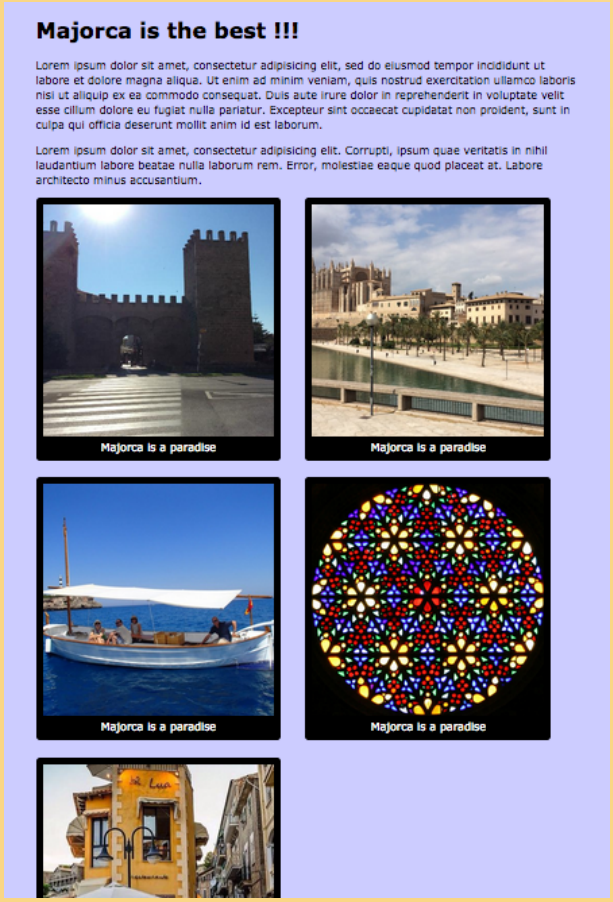
Resize the browser window to see the effect!

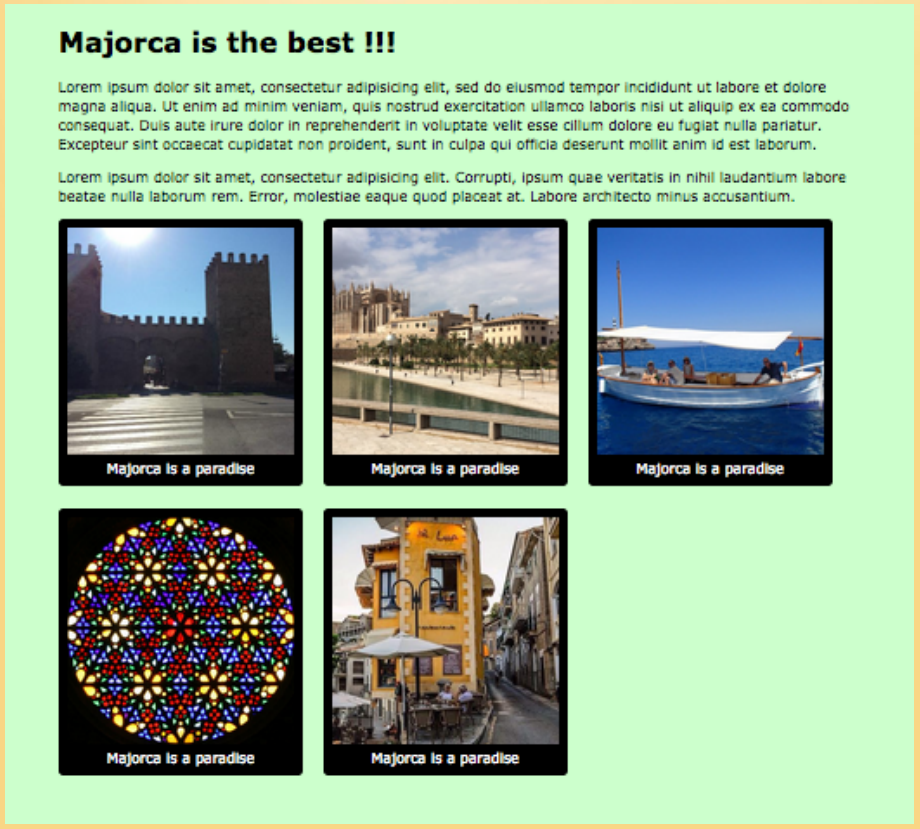
The media query will only apply if the media type is screen and the viewport is 480px wide or wider.

```
<!DOCTYPE html>
<html>
<head>
  <style>
  body {
    background-color: pink;
  }

  @media screen and (min-width: 480px) {
    body {
      background-color: lightgreen;
    }
  }
  </style>
</head>
<body>
  <h1>Resize the browser window to see the
  effect!</h1>
  <p>The media query will only apply if the
  media type is screen and the viewport is
  480px wide or wider.</p>
</body>
</html>
```

ACTIVITY 7

Using media queries, try to design depending on different screen sizes ...





2.11 CSS3
BOX SIZING

- Box-sizing property allows us to include the padding and border in an element's total width and height.
- If you set `box-sizing: border-box;` on an element, padding and border are included in the width and height:
- The code below ensures that all elements are sized in this more intuitive way. Applying this to all elements is safe and wise:


```
* {
    box-sizing: border-box;
}
```

2.11 CSS3
BOX SIZING

```
  <style type="text/css">
    div {
      width: 300px;
      height: 100px;
      background-color: cyan;
      margin: 10px;
      font-weight: bold;
      font-size: 1.5em;
    }

    .pad {
      padding: 30px;
    }

    .bz {
      box-sizing: border-box;
    }
  </style>
</head>
<body>
  <h1>3 boxes are 300px * 100px</h1>
  <div></div>
  <div class="pad">with padding</div>
  <div class="pad bz">with padding and BOX-SIZING</div>
</body>
```

3 boxes are 300px * 100px

with padding

with padding and BOX-SIZING

2.11 CSS3
FLEXBOX

- Flexible boxes, or flexbox, is a **new layout** mode in CSS3.
- With flexbox elements behave predictably when the page layout must accommodate **different screen sizes** and **different display devices**.
- Flexible box model can provides an improvement over the block model because:
  - does not use floats,
  - flex container's margins don't collapse with the margins of its contents.

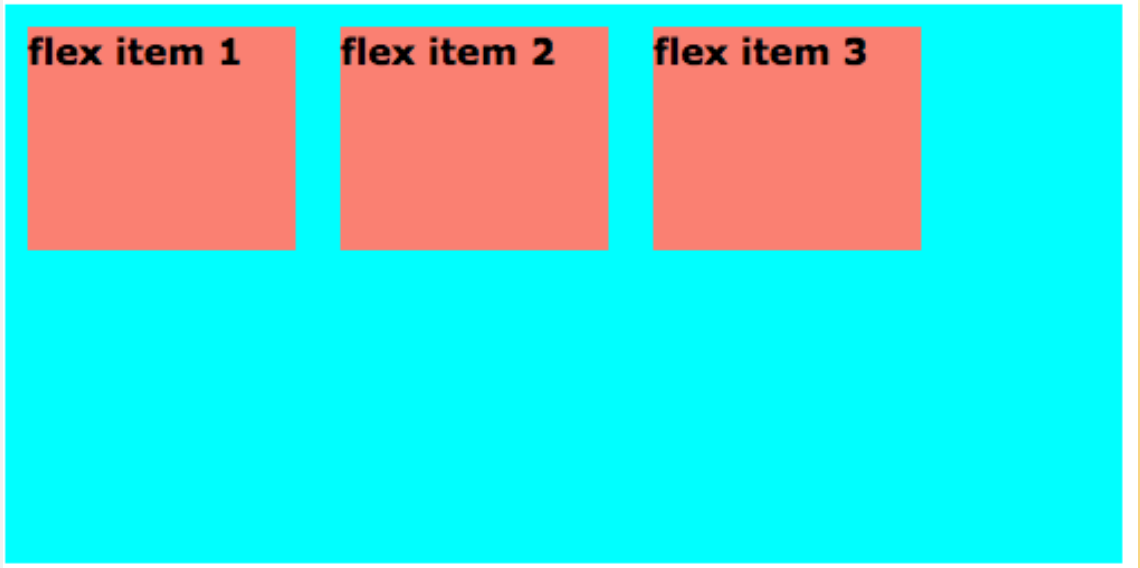
2.11 CSS3
FLEXBOX

Flexbox Concepts:

- Flexbox consists of:
  - flex containers and
  - flex items.
- A flex container is declared by setting the **`display`** property of an element to:
  - **`flex`** (rendered as a block) or
  - **`inline-flex`** (rendered as inline)
- Inside a flex container there is one or more flex items.
- Flex items are positioned inside a flex container along a flex line.

2.11 CSS3
FLEXBOX

```
  .flex-container {
    display: flex;
    width: 500px;
    height: 250px;
    background-color: cyan;
  }

  .flex-item {
    background-color: salmon;
    width: 120px;
    height: 100px;
    margin: 10px;
  }
  </style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">flex item 1</div>
    <div class="flex-item">flex item 2</div>
    <div class="flex-item">flex item 3</div>
  </div>
</body>
```



2.11 CSS3
FLEXBOX

```
  .flex-container {
    display: flex;
    direction: rtl; /* right-to-left */
    width: 500px;
    height: 250px;
    background-color: cyan;
  }

  .flex-item {
    background-color: salmon;
    width: 120px;
    height: 100px;
    margin: 10px;
  }
  </style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">flex item 1</div>
    <div class="flex-item">flex item 2</div>
    <div class="flex-item">flex item 3</div>
  </div>
</body>
```

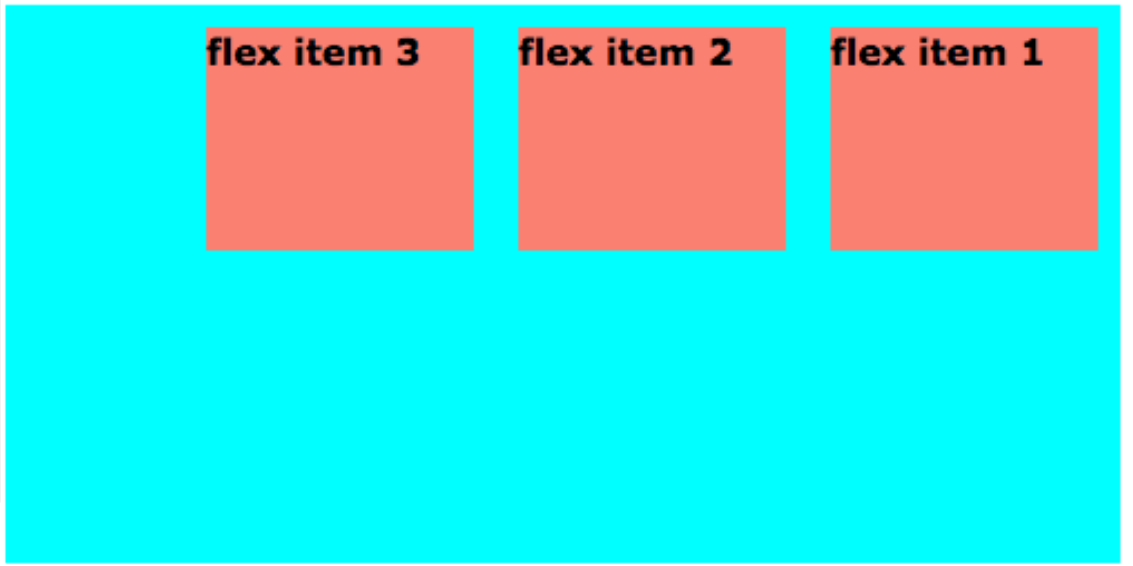
2.11 CSS3
FLEXBOX

The default value of **`flex-direction`** is row.
Other values:

- **`row-reverse`** - If the writing-mode (direction) is left to right, the flex items will be laid out right to left
- **`column`** - If the writing system is horizontal, the flex items will be laid out vertically
- **`column-reverse`** - Same as column, but reversed

2.11 CSS3
FLEXBOX

```
  .flex-container {
    display: flex;
    flex-direction: row-reverse;
    width: 500px;
    height: 250px;
    background-color: cyan;
  }

  .flex-item {
    background-color: salmon;
    width: 120px;
    height: 100px;
    margin: 10px;
  }
  </style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">flex item 1</div>
    <div class="flex-item">flex item 2</div>
    <div class="flex-item">flex item 3</div>
  </div>
</body>
```



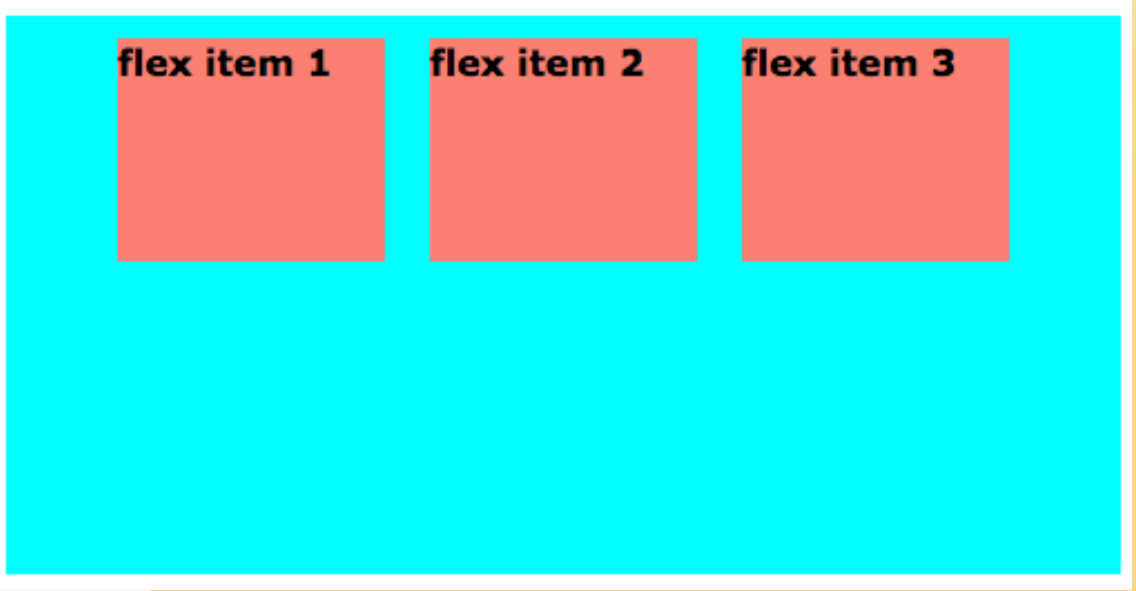
2.11 CSS3
FLEXBOX

The **`justify-content`** property horizontally aligns the flexible container's items when the items do not use all available space on the main-axis. Possible values:

- **`flex-start`** - Default value. Items are positioned at the beginning of the container
- **`flex-end`** - Items are positioned at the end of the container
- **`center`** - Items are positioned at the center of the container
- **`space-between`** - Items are positioned with space between the lines
- **`space-around`** - Items are positioned with space before, between, and after the lines

2.11 CSS3
FLEXBOX

```
  <style>
  * {
    font-family: verdana;
    font-weight: bold;
  }
  .flex-container {
    display: flex;
    justify-content: center;
    width: 500px;
    height: 250px;
    background-color: cyan;
  }

  .flex-item {
    background-color: salmon;
    width: 120px;
    height: 100px;
    margin: 10px;
  }
  </style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">flex item 1</div>
    <div class="flex-item">flex item 2</div>
    <div class="flex-item">flex item 3</div>
  </div>
</body>
```



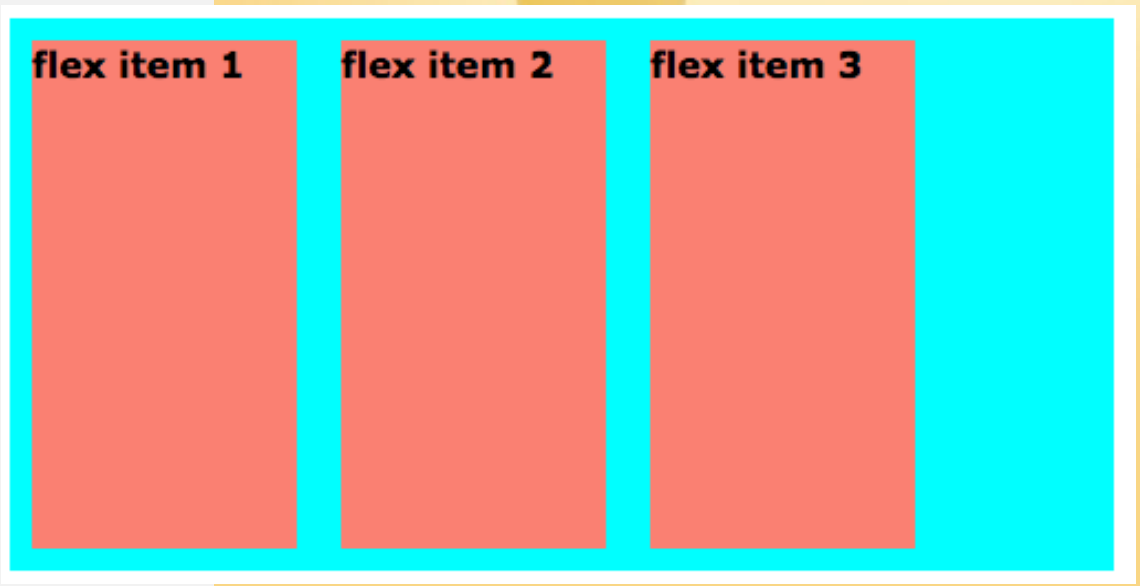
2.11 CSS3
FLEXBOX

The **`align-items`** property vertically aligns the flexible container's items when the items do not use all available space on the cross-axis. Possible values:

- **`stretch`** - Default value. Items are stretched to fit the container
- **`flex-start`** - Items are positioned at the top of the container
- **`flex-end`** - Items are positioned at the bottom of the container
- **`center`** - Items are positioned at the center of the container (vertically)
- **`baseline`** - Items are positioned at the baseline of the container

2.11 CSS3
FLEXBOX

```
  <style>
  * {
    font-family: verdana;
    font-weight: bold;
  }
  .flex-container {
    display: flex;
    align-items: stretch;
    width: 500px;
    height: 250px;
    background-color: cyan;
  }

  .flex-item {
    background-color: salmon;
    width: 120px;
  /* height: 100px; */
    margin: 10px;
  }
  </style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">flex item 1</div>
    <div class="flex-item">flex item 2</div>
    <div class="flex-item">flex item 3</div>
  </div>
</body>
```



2.11 CSS3
FLEXBOX

The **`flex-wrap`** property specifies whether the flex items should wrap or not, if there is not enough room for them on one flex line. Possible values:

- **`nowrap`** - Default value. The flexible items will not wrap
- **`wrap`** - The flexible items will wrap if necessary
- **`wrap-reverse`** - The flexible items will wrap, if necessary, in reverse order

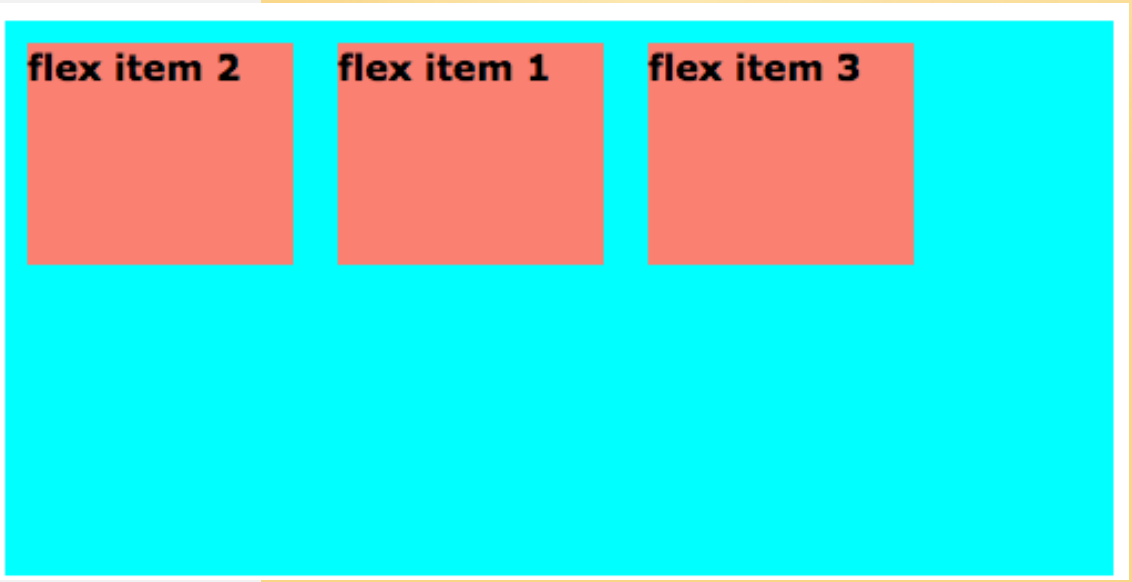
2.11 CSS3
FLEXBOX

```
  .flex-container {
    display: flex;
    flex-wrap: wrap;
    width: 300px; /* container is smaller */
    height: 250px;
    background-color: cyan;
  }

  .flex-item {
    background-color: salmon;
    width: 120px;
    height: 100px;
    margin: 10px;
  }
  </style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">flex item 1</div>
    <div class="flex-item">flex item 2</div>
    <div class="flex-item">flex item 3</div>
  </div>
</body>
```

2.11 CSS3
FLEXBOX

The **`order`** property specifies the order of a flexible item relative to the rest of the flexible items inside the same container:

```
  <style>
  * {
    font-family: verdana;
    font-weight: bold;
  }
  .flex-container {
    display: flex;
    width: 500px;
    height: 250px;
    background-color: cyan;
  }
  .first {
    order: -1;
  }

  .flex-item {
    background-color: salmon;
    width: 120px;
    height: 100px;
    margin: 10px;
  }
  </style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">flex item 1</div>
    <div class="flex-item first">flex item 2</div>
    <div class="flex-item">flex item 3</div>
  </div>
</body>
```

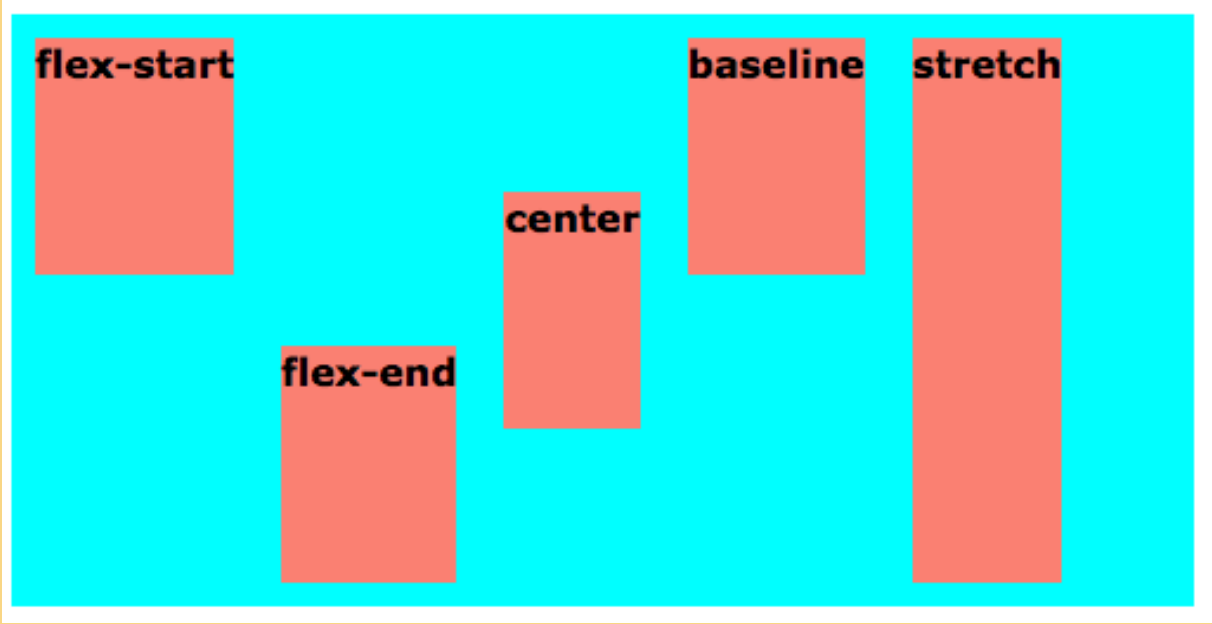


2.11 CSS3

FLEXBOX

```
.flex-container {
  display: flex;
  width: 500px;
  height: 250px;
  background-color: cyan;
}

.flex-item {
  background-color: salmon;
  min-height: 100px;
  margin: 10px;
}
.item1 {
  align-self: flex-start;
}
.item2 {
  align-self: flex-end;
}

.item3 {
  align-self: center;
}

.item4 {
  align-self: baseline;
}

.item5 {
  align-self: stretch;
}
</style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item item1">flex-start</div>
    <div class="flex-item item2">flex-end</div>
    <div class="flex-item item3">center</div>
    <div class="flex-item item4">baseline</div>
    <div class="flex-item item5">stretch</div>
  </div>
</body>
```

The **`align-self`** property of flex items overrides the flex container's `align-items` property for that item. It has the same possible values as the align-items property.



END
Unit 2