
# Internet Programming

## Web Page Layout

Dan Brandon, Ph.D., PMP

# Display Style

- Two broad classifications of HTML elements
  - Block elements: such as paragraphs or headings (box model applies)
  - Inline elements: such as emphasized text or inline images
- Define the display style for any page using the `display` property:

```
display: type;
```

where `type` defines the display type

# Display Style (con't)

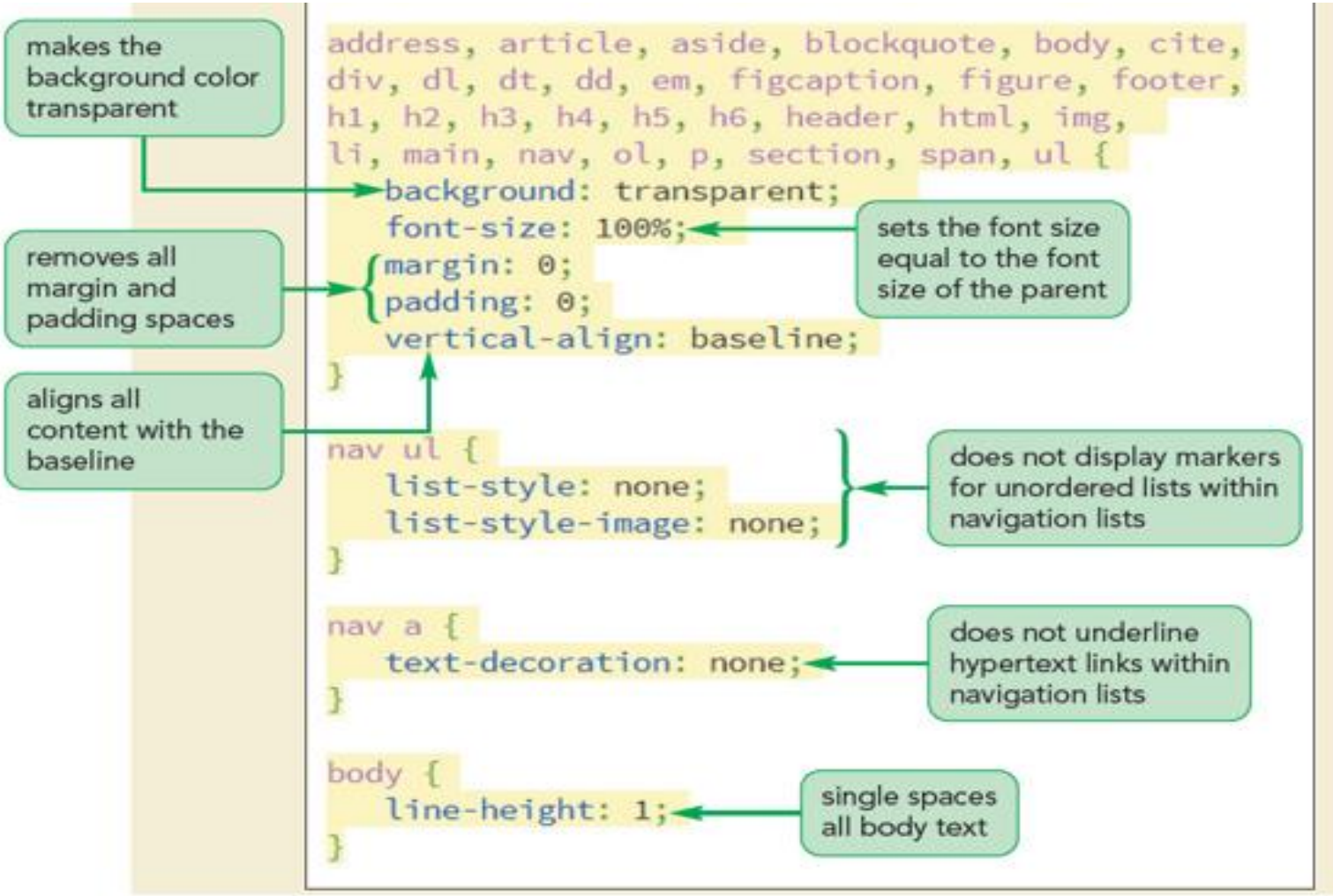
| Display Value | Appearance |
|---|---|
| `block` | Displayed as a block |
| `table` | Displayed as a web table |
| `inline` | Displayed inline within a block |
| `inline-block` | Treated as a block placed inline within another block |
| `run-in` | Displayed as a block unless its next sibling is also a block, in which case, it is displayed inline, essentially combining the two blocks into one |
| `inherit` | Inherits the display property of the parent element |
| `list-item` | Displayed as a list item along with a bullet marker |
| `none` | Prevented from displaying, removing it from the rendered page |

# Reset Style Sheets

- **Reset style sheet** supersedes a browser's default styles and provides a consistent starting point for page design
- The first style rule in a sheet is the `display` property used to display HTML5 structural elements as blocks
- Premade reset style sheets are freely available on the web that contain many style rules used to reconcile the differences between browsers and devices

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

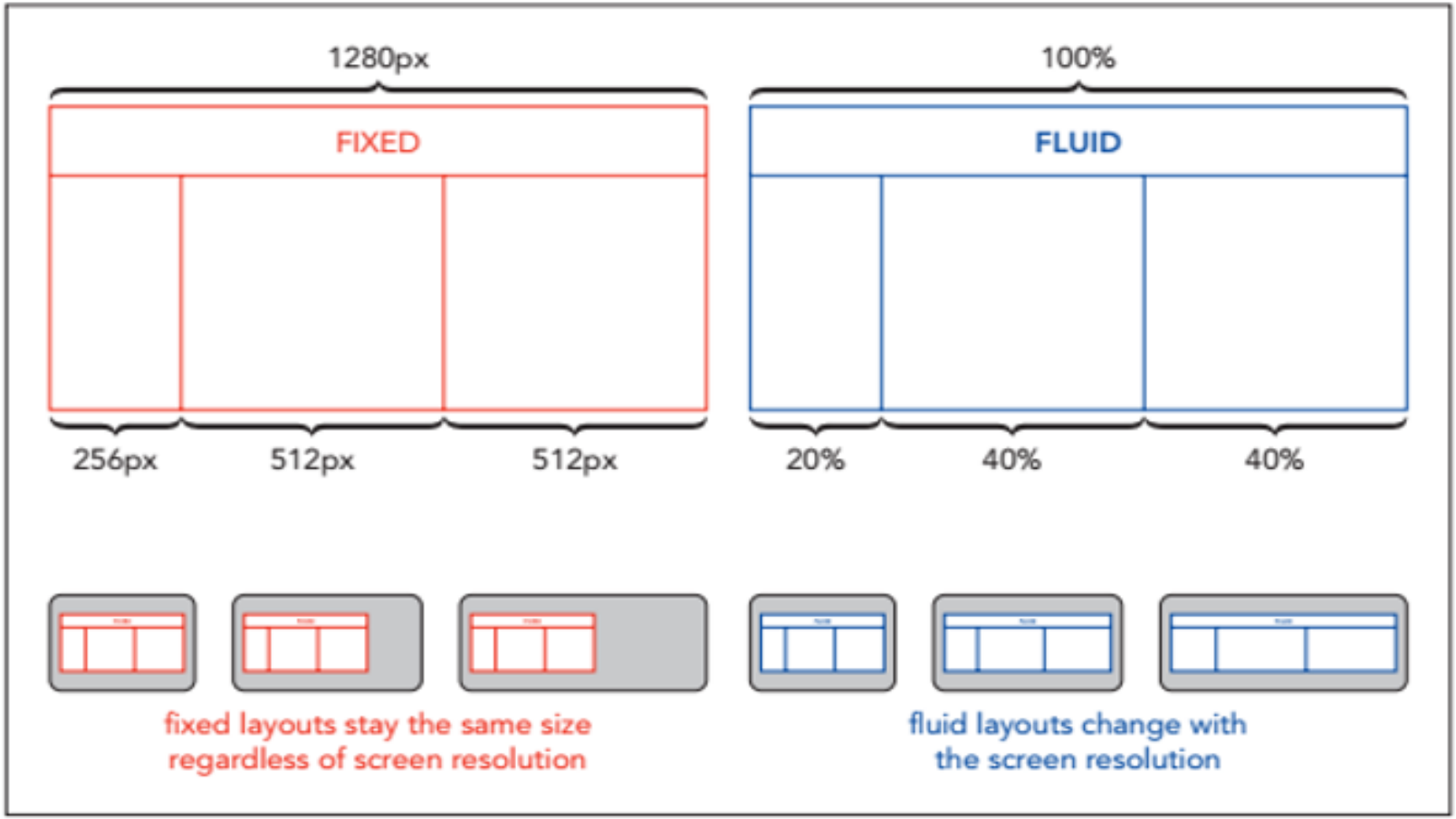
# Reset Style Sheets (con't)



# Layout Categories

- Web page layouts fall into three categories
  - **Fixed layout** – Size of the page and page elements are fixed, usually using pixels as the unit of measure
  - **Fluid layout** – The width of the page elements are set as a percent of the available screen width
  - **Elastic layout** – Images and text are always sized in proportion to each other in em units
- **Responsive design** – The layout and design of a page changes in response to the device that is rendering it

# Layout Categories (con't)



# Element Width & Height

- The width and height of an element are set using the following properties:

```
width: value;
height: value;
```

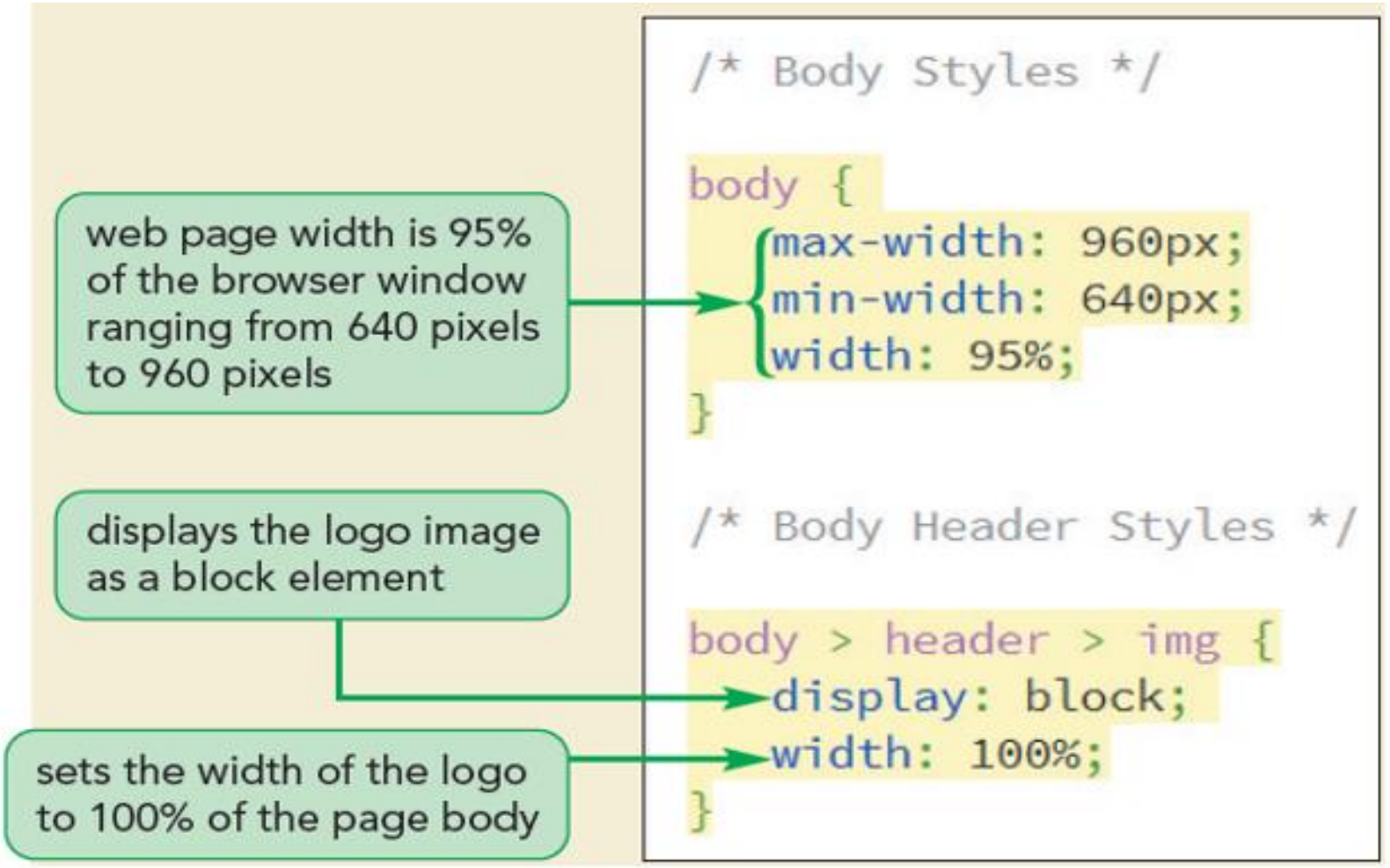
  where `value` is the width or height using one of the CSS units of measurement or as a percentage of the width or height of the parent element

- Set limits on the width or height of a block:

```
min-width: value; min-height: value;
max-width: value; max-height: value;
```

  where `value` is a length expressed in one of the CSS units of measure (usually pixels to match the display device measurement unit)

# Element Width & Height (con't)



# Centering a Block Element

- Block elements can be centered horizontally within their parent element by setting both the left and right margins to `auto`
- Example: center the page body within the browser window using the style rule

```
body {
   margin-left: auto;
   margin-right: auto;
}
```

# Centering (con't)

- Centering an element vertically can be accomplished by displaying the parent element as a table cell and setting the `vertical-align` property to `middle`
- Example: to vertically center the following `h1` heading within the `div` element:

```
<div>
   <h1>Pandaisia Chocolates</h1>
</div>
```

# Centering (con't)

- Apply the style rule

```
div {
    height: 40px;
    display: table-cell;
    vertical-align: middle;
}
```

- Using this style rule, the `h1` heading will be vertically centered

# Centering (con't)

- One can vertically center a single line of text within its parent element
- Set text line height to be larger than font size

```
h1 {
      font-size: 1.4em;
      line-height: 2em;
    }
```

- This only works for a single line of text

# Floating an Element

- **Floating** an element takes it out of position and places it along the left or right side of its parent element
- To float an element, apply

```
float: position;
```

where *position* is `none` (the default), `left` to float the object on the left margin or `right` to float the object on the right margin

# Floating (con't)

- If sibling elements are floated along the same margin, they are placed alongside each other within a row
- For elements to be placed within a single row, the combined width of the elements cannot exceed the total width of their parent element
  - Otherwise excess content will automatically wrap to a new row

# Floating(con't)

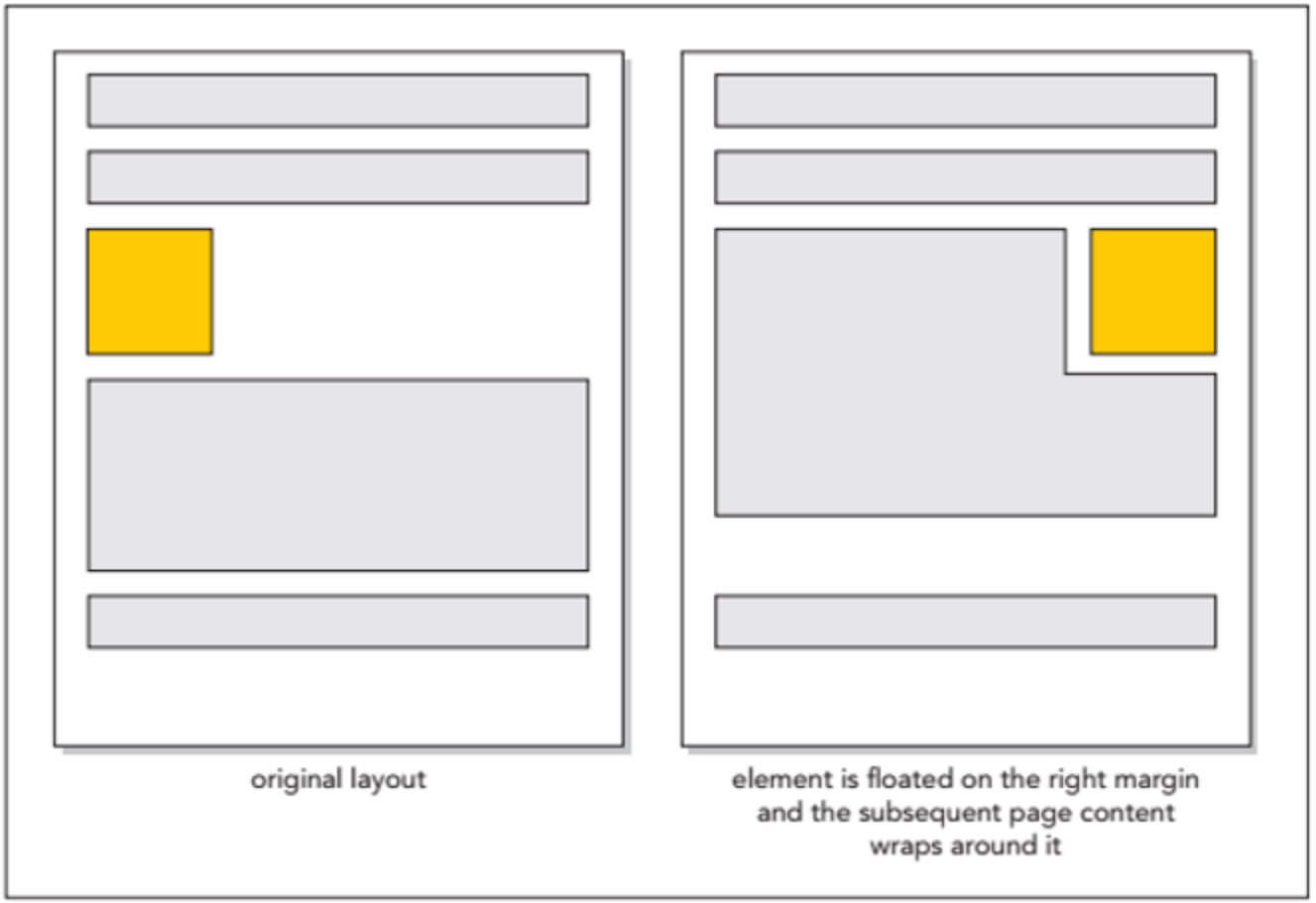
- To ensure that an element is always displayed below floated elements, use

  `clear: position;`

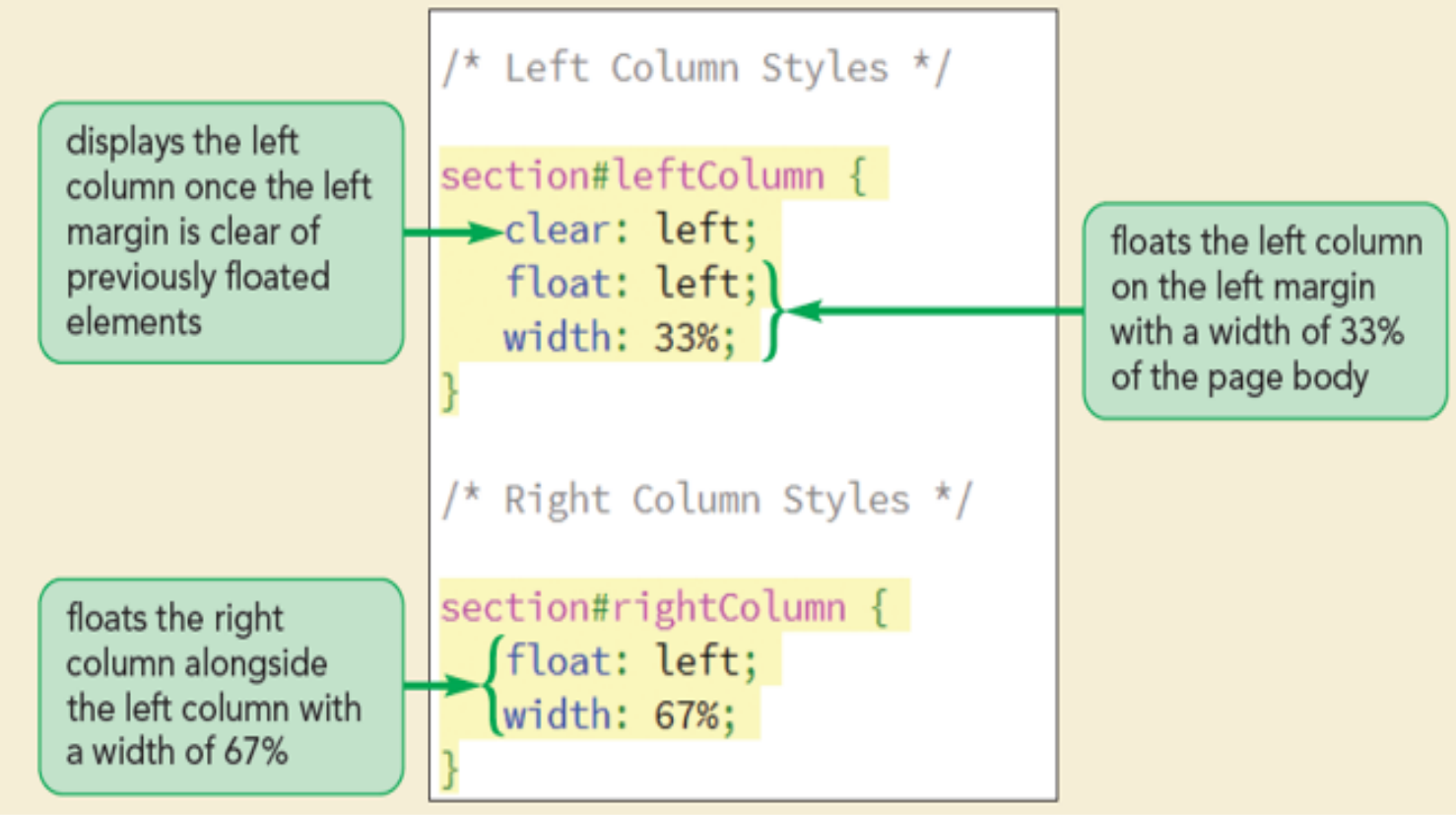
  where `position` is `left`, `right`, `both`, or `none`

  - `left` – Displays the element only when the left margin is clear of floating objects
  - `right` – Displays the element only when the right margin is clear of floating objects
  - `both` – Displays the element only when both margins are clear of floats
  - `none` – Displays the element alongside any floated objects

# Floating (con't)



# Floating (con't)



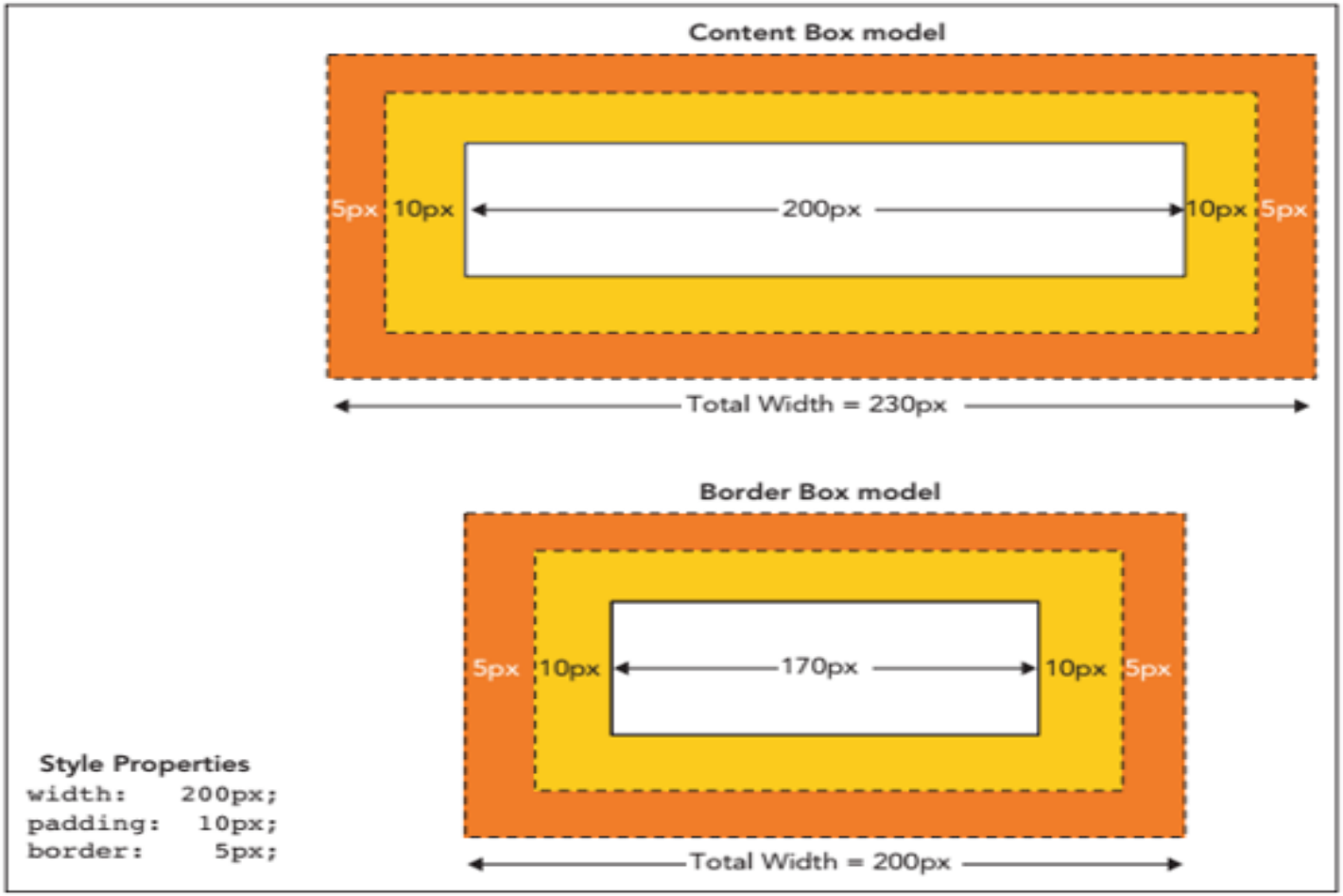
# Box Model Sizing

- **Content box model** – The `width` property refers to the width of an element content only
  - Additional space include padding or borders
- **Border box model** – The `width` property is based on the sum of the content, padding, and border spaces
  - Additional space taken up by the padding and border is subtracted from space given to the content
- The layout model can be chosen using

  `box-sizing: type;`

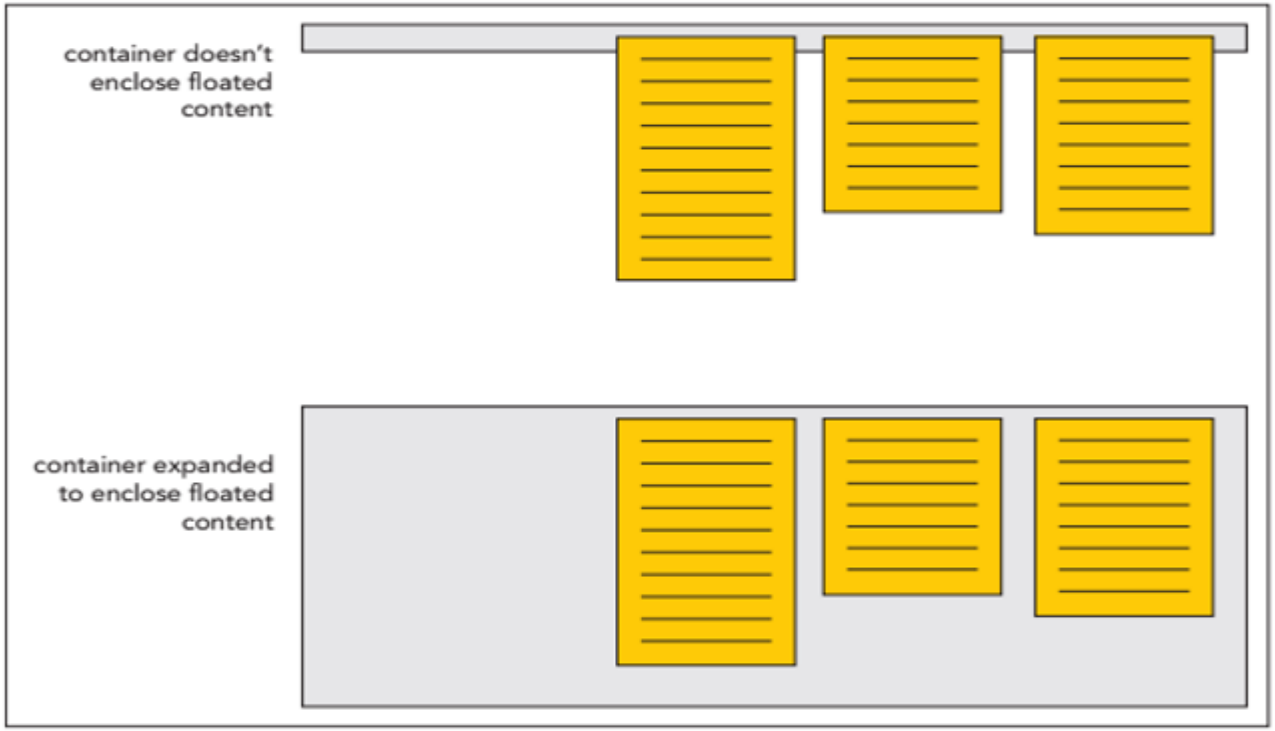
  where *`type`* is `content-box` (the default), `border-box`, or `inherit` (to inherit the property defined for the element's container)

# Box Model Sizing (con’t)



# Container Collapse

- **Container collapse** – An empty container with no content
- Elements in the container are floated



# Collapse (con't)

- One can use the `after` pseudo-element to add a placeholder element after a layout element such as a footer
- The general style rule is

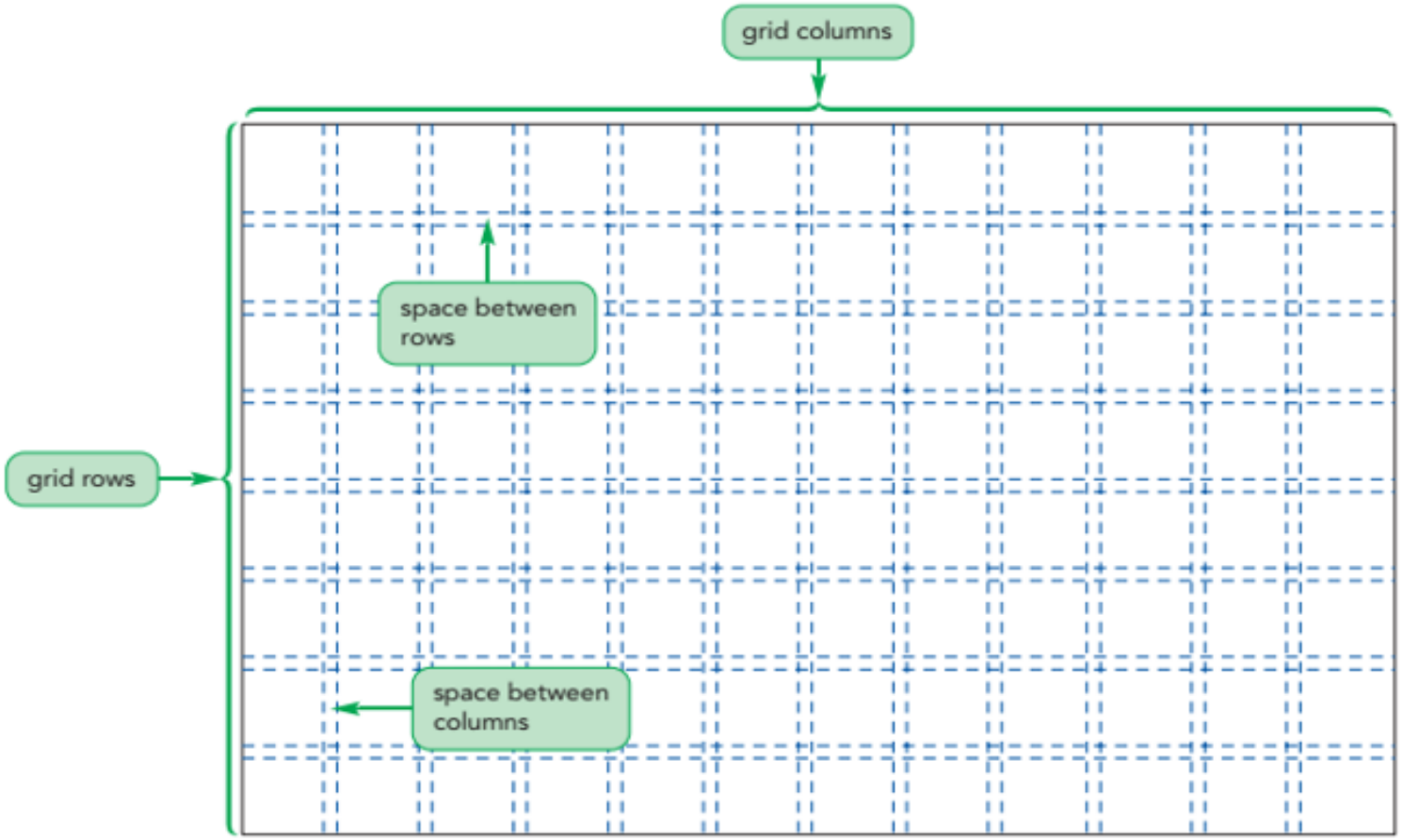
```
container::after {
        clear: both;
        content: "";
        display: table;
}
```

  where *`container`* is the selector for the element containing floating objects
- The `clear` property keeps the placeholder element from being inserted until both margins are clear of floats

# Grid-Based Layouts

- In a **grid layout**, the page is comprised of a system of intersecting rows and columns that form a grid
- The rows are based on the page content
- The number of columns is based on the number that provides the most flexibility in laying out the page content
- Many grid systems are based on 12 columns

# CSS Grids (con't)



# CSS Grids (con't)

- Advantages of using a grid:
  - Grids add order to the presentation of page content
  - A consistent logical design gives readers the confidence to find the information they seek
  - New content can be easily placed within a grid in a manner consistent with previously entered data
  - It is easily accessible for users with disabilities and special needs
  - It increases development speed with a systematic framework for the page layout

# CSS Grids (con't)

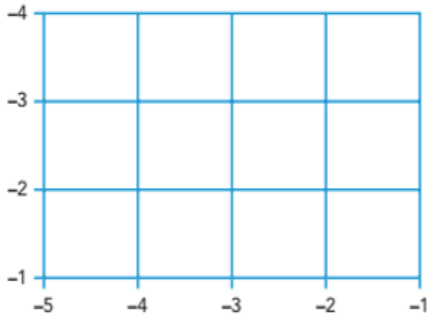
- **Fixed grids** – Every column has a fixed position
  - Widths of the columns and margins are specified in pixels
- **Fluid grids** – Provides more support across different devices with different screen sizes
  - Column width is expressed in percentages

# CSS Grids (con't)

- The **CSS grid model** is a set of CSS design styles used to create grid-based layouts
- Each CSS grid is laid out in a set of row and column gridlines
- To reference positions within a grid, the CSS grid model numbers the gridlines in the horizontal and vertical directions
  - Start from the top-left corner of the grid with the row gridlines and then moving left to right with the column gridlines along the bottom
- Both gridlines start with a value of "1" and increase in value down and across the grid

# CSS Grids (con't)

- Gridlines can be referenced in the reverse order starting
  - Start from the bottom-right corner with the first row and column gridlines are given a value of "-1"
- The advantage of using both positive and negative gridline numbers
  - Can always reference both the first gridline (1) and the last gridline (-1) no matter the size of the grid



# CSS Grids (con't)

- The cells that are created from the intersection of the horizontal and vertical gridlines will contain the elements from the web page
- An element can be contained within a single cell or it can span several cells within a grid area
- Note that grid areas must be rectangular; you cannot have an L-shaped grid area

# CSS Grids (con't)

- To create a CSS grid, first identify a page element as the grid container using the following display property:

```
display: grid;
```

- The entire grid itself is considered a block-level element and thus could be floated or resized within the web page just like any other block-level element

# CSS Grids (con't)

HTML Content

```
<body>
   <div id="outer">
      <div id="item1">1</div>
      <div id="item2">2</div>
      <div id="item3">3</div>
      <div id="item4">4</div>
      <div id="item5">5</div>
      <div id="item6">6</div>
   </div>
</body>
```

CSS Styles

```
div#outer {
   display: grid;
}
```

outer element is marked as a grid

Web Page

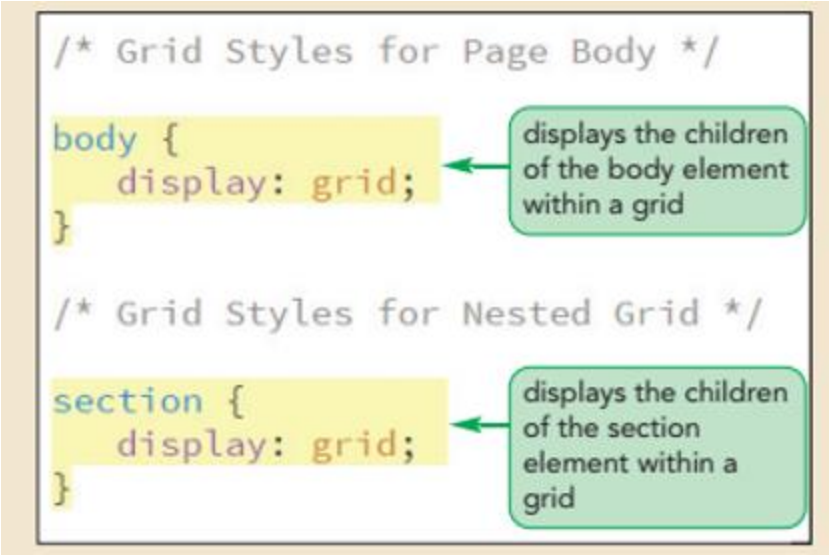

each child element is part of the grid

# CSS Grids (con't)

- Grids can also be created as inline elements using the style:

```
display: inline-grid;
```

which creates the grid inline with other elements in the web page



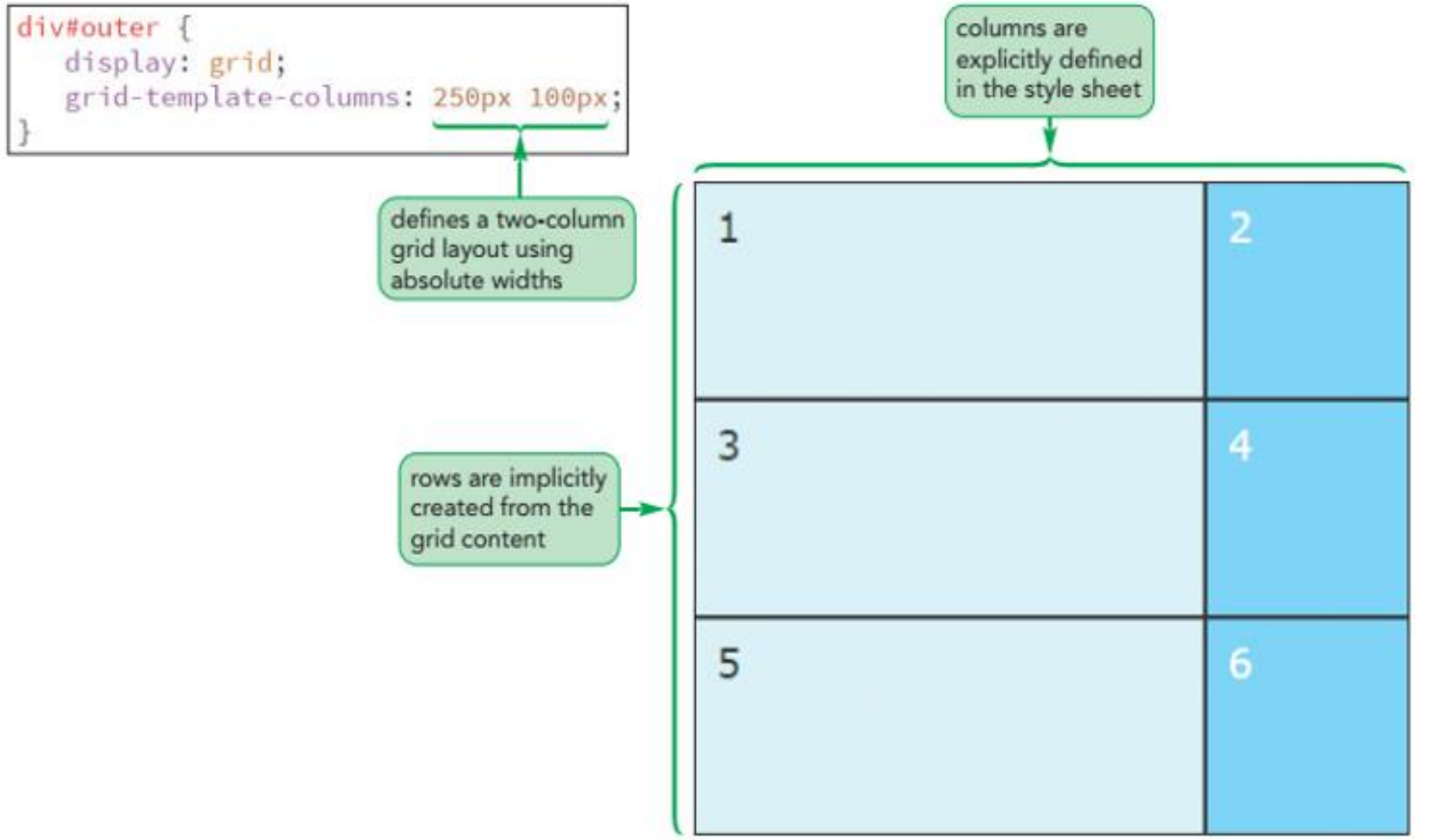
# CSS Grids (con't)

- To define the number and size of grid columns, use the following `grid-template-columns` style:

  ```
  grid-template-columns: width1 width2 …;
  ```

  where `width1`, `width2`, etc. is a space-separated list that defines the width of the columns or tracks within the grid
- Column widths can be expressed using any CSS unit measures such as pixels, em units, and percentages
- The keyword `auto` can be used to allow the column width to be automatically set by the browser

# CSS Grids (con't)



# CSS Grids (con't)

- An **explicit grid** completely defines the number and size of the grid rows and columns
- An **implicit grid** contains rows and/or columns that are generated by the browser as it populates the grid with items from the grid container
- In most grid layouts, columns are explicitly defined and the browser fills out the grid rows drawn from the web page content

# CSS Grids (con't)

- To explicitly define the number of rows and their height, use the following

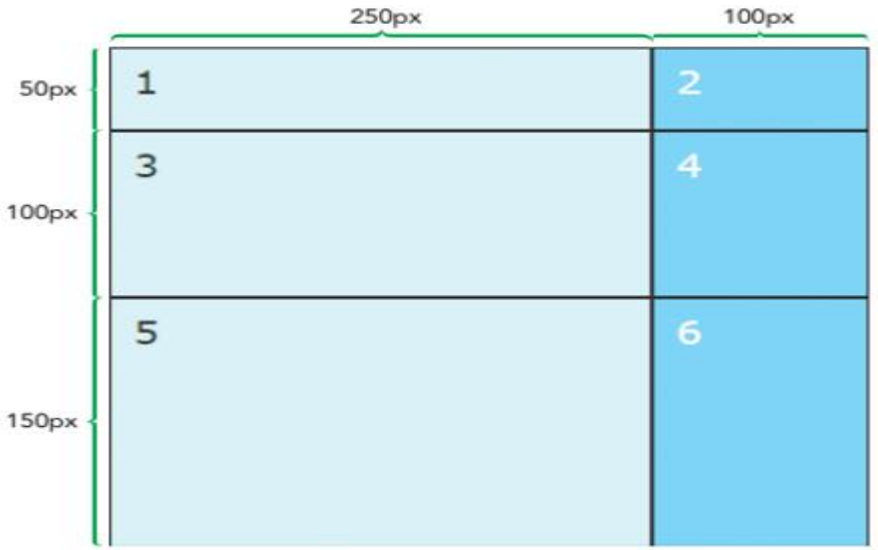
  `grid-template-rows property:`

  `grid-template-rows: height1 height2 …;`

  where `height1`, `height2`, etc. define the heights of the grid rows

```
div#outer {
   display: grid;
   grid-template-columns: 250px 100px;
   grid-template-rows: 50px 100px 150px;
}
```

explicitly defines both the rows and columns of the grid



# CSS Grids (con't)

- A grid layout can adapt to devices of various screen widths and sizes by using flexible units
- A **fr (fractional) unit**, indicated by the unit abbreviation fr, creates grid tracks that expand or contract in size to fill available space while retaining their relative proportions to one another
- Fractional units are often combined with absolute units to create grid layouts that are both fixed and flexible
- The following style rule generates a grid in which the width of the first column is set to 250 pixels with the remaining space allotted to the other two columns in a proportion of 2 to 1

```
grid-template-columns: 250px 2fr 1fr
```

# CSS Grids (con't)

- Some grid layouts involve many columns so it is difficult to specify column sizes
- The layout style can be simplified by using the following `repeat()` function:

```
repeat(repeat, tracks)
```

  where `repeat` is the number of repetitions of the tracks specified in `tracks`
- In place of a `repeat` value, the keyword `auto-fill` can be used to fill up the grid with as many columns (or rows) that will fit within the grid container
- The following style uses the `auto-fill` keyword to fill the grid with as many 100 pixel-wide columns that will fit within the container:

```
grid-template-columns: 250px repeat(auto-fill, 100px)
```

# CSS Grids (con't)

- It is possible to switch between fixed and flexible track sizes using the following function

  ```
  minmax(min, max)
  ```

  where `min` is the minimum track size for a row and column and `max` is the maximum

- Example:

  ```
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
  ```

# CSS Grids (con't)

- Outlines - Lines drawn around an element, enclosing the element content, padding, and border spaces
  - `Outline-width: value;` – Specifies the width of a line in CSS units or `thin`, `medium`, or `thick`
  - `Outline-color: color;` – Specifies the color of a line
    - Properties of *`color`* are: CSS color name or value

# CSS Grids (con't)

- `Outline-style: style;` - Specifies the design of a line
  - Properties of *`style`* are: `solid`, `double`, `dotted`, `dashed`, `groove`, `inset`, `ridge`, or `outset`
- Outline properties can be combined:

  ```
  width style color;
  ```

  where `width`, `style`, and `color` are the values for the line's width, design, and color

# CSS Grids (con't)

- By default, grid items are laid out in document order going from left to right and up to down, with each item placed within a single cell
- In many layouts however, it might be desirable to move items around or a have a single item occupy multiple rows and column
- To place the `article` element in a **grid cell** located in the first row and second column of the grid, apply the following style rule:

```
article {
grid-row: 1;
grid-column: 2;
}
```

# CSS Grids (con't)

- To move a grid item to a specific location within the grid, use the following `grid-row` and `grid-column` properties:

  ```
  grid-row: row;
  grid-column: column;
  ```

  where `row` is the row number and `column` is the column number
- To extend a grid item so that it covers multiple rows or multiple columns, include the starting and ending gridline in the style property as follows:

  ```
  grid-row: start/end;
  grid-column: start/end;
  ```

  where `start` is the starting gridline and `end` is the ending gridline

# CSS Grids (con't)

- Starting and ending gridlines can be expressed in the following four properties:

```
grid-column-start: integer;
grid-column-end: integer;
grid-row-start: integer;
grid-row-end: integer;
```

- Another way of setting the size of a grid cell is with the `span` keyword
- The general syntax is:

```
grid-row: span value;
grid-column: span value;
```

where `value` is the number of rows or columns covered

# CSS Grids (con't)

- To specify both the location and the size of the item, include the starting gridline in the style rule
- Example:

```
article {
grid-row: 1/span 2;
grid-column: 4/span 3;
}
```

  In the grid areas approach to layout you identify sections of the grid with item names, creating a textual representation of the layou

# CSS Grids (con't)

- To create a textual representation in a style sheet, use the following `grid-template-areas` property:

```
grid-template-areas: "row1"
"row2"
…;
```

  where `row1`, `row2`, etc. are text strings containing the names of the areas for each row

- To assign elements to grid areas, use the following `grid-area` property:

```
grid-area: area;
```

  where `area` is the name of an area defined in the `grid-template-areas` property

# CSS Grids (con't)

- The grid-area property can be used as a shorthand to place and size grid items using gridline numbers
- The general syntax is:

```
grid-area: row-start/col-start/row-end/col-end;
```

where `row-start`, `col-start`, `row-end`, and `col-end` are the starting and ending gridline numbers from the grid's rows and columns

# CSS Grids (con't)

- Another part of grid layout is defining the space between items in a grid
- The gap size is defined using the following grid-gap property:

```
grid-gap: row column;
```

where `row` is the internal space between grid rows and `column` is the internal space between grid columns

# CSS Grids (con't)

- Grid gaps for rows and columns can also be set using the following properties:

```
grid-column-gap: value;
grid-row-gap: value;
```

  where `value` is the size of the gap in one of the CSS units of measure
- Gap size setting is applied only to the interior space between the grid items

# CSS Grids (con't)

- The content within the grid cell can be positioned using the `align-items` and `justify-items` properties
  - The `align-items` property sets the vertical placement of the content
  - The `justify-items` property sets the horizontal placement
- The syntax of both properties is:

```
align-items: placement;
justify-items: placement;
```

# CSS Grids (con't)

where `placement` is:

- `stretch` to expand the content between the top/bottom or left/right edges, removing any spacing between the content and the cell border (the default)
- `start` to position the content with the top or left edge of the cell
- `end` to position the content with the bottom or right edge of the cell
- `center` to center the content vertically or horizontally within the cell

# CSS Grids (con't)

- To align and justify only one cell, apply the `align-self` and `justify-self` properties to the content within the grid cell
- Example

```
article {
align-self: center;
justify-self: center;
}
```

# CSS Grids (con't)

- To modify grid position use the `align-content` and ju`stify-content` properties:

```
align-content: placement;
justify-content: placement;
```

Where `placement` is:

- `start` to position the grid with the top or left edge of the container (the default)
- `end` to position the grid with the bottom or right edge of the container

# CSS Grids (con't)

- `center` to center the grid vertically or horizontally within the container
- `space-around` to insert an even amount of space between each grid item with no space at the far ends
- `space-between` to insert an even amount of space between each grid item, with no space at the far ends
- `space-evenly` to insert an even amount of space between each grid item, including the far ends

# CSS Positioning

- To place an element at a specific position within its container, use

```
position: type;
top: value;
right: value;
bottom: value;
left: value;
```

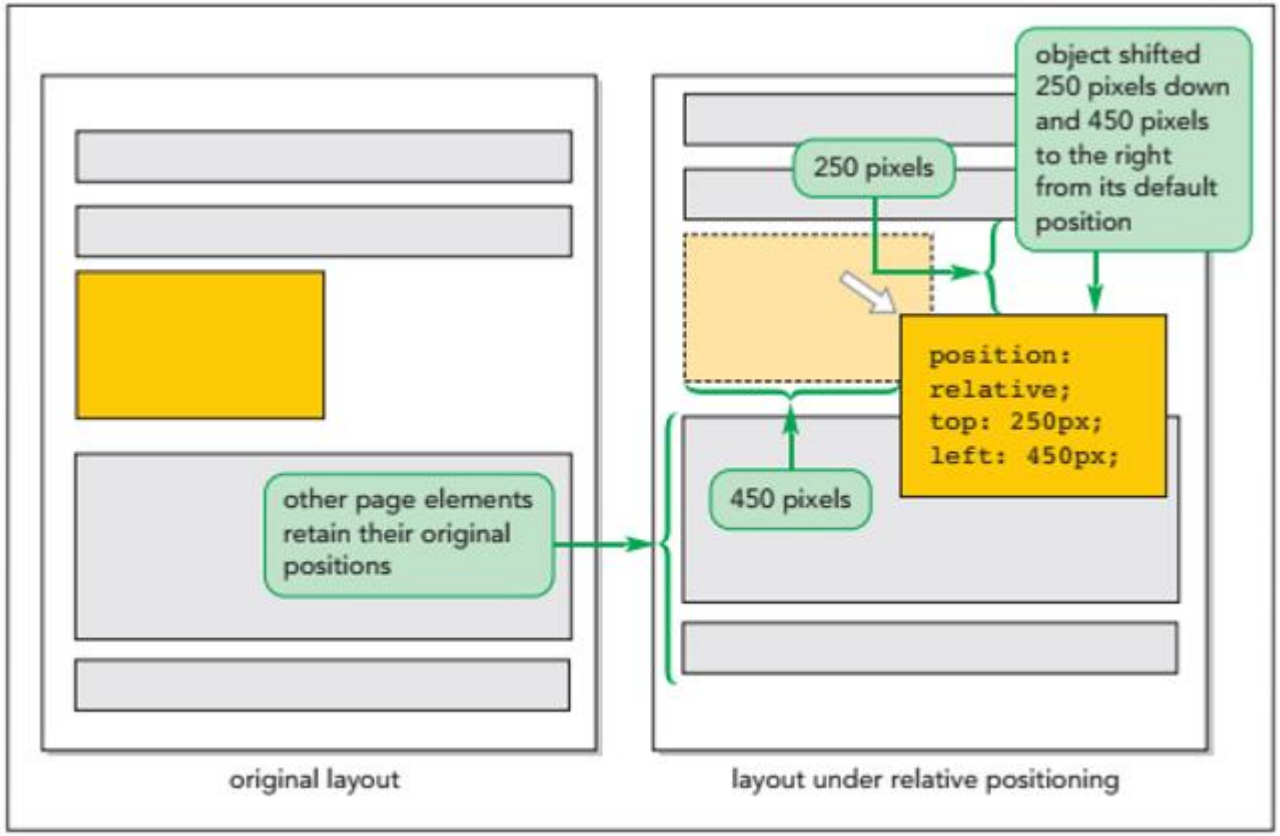
where *type* indicates the kind of positioning applied to the element and `top, right, bottom, and left` properties indicate the coordinates of the element

# CSS Positioning (con't)

- **Static positioning** – The element is placed where it would have fallen naturally within the flow of the document
- Relative positioning – The element is moved out of its normal position in the document flow
- Absolute positioning – The element is placed at specific coordinates within containers

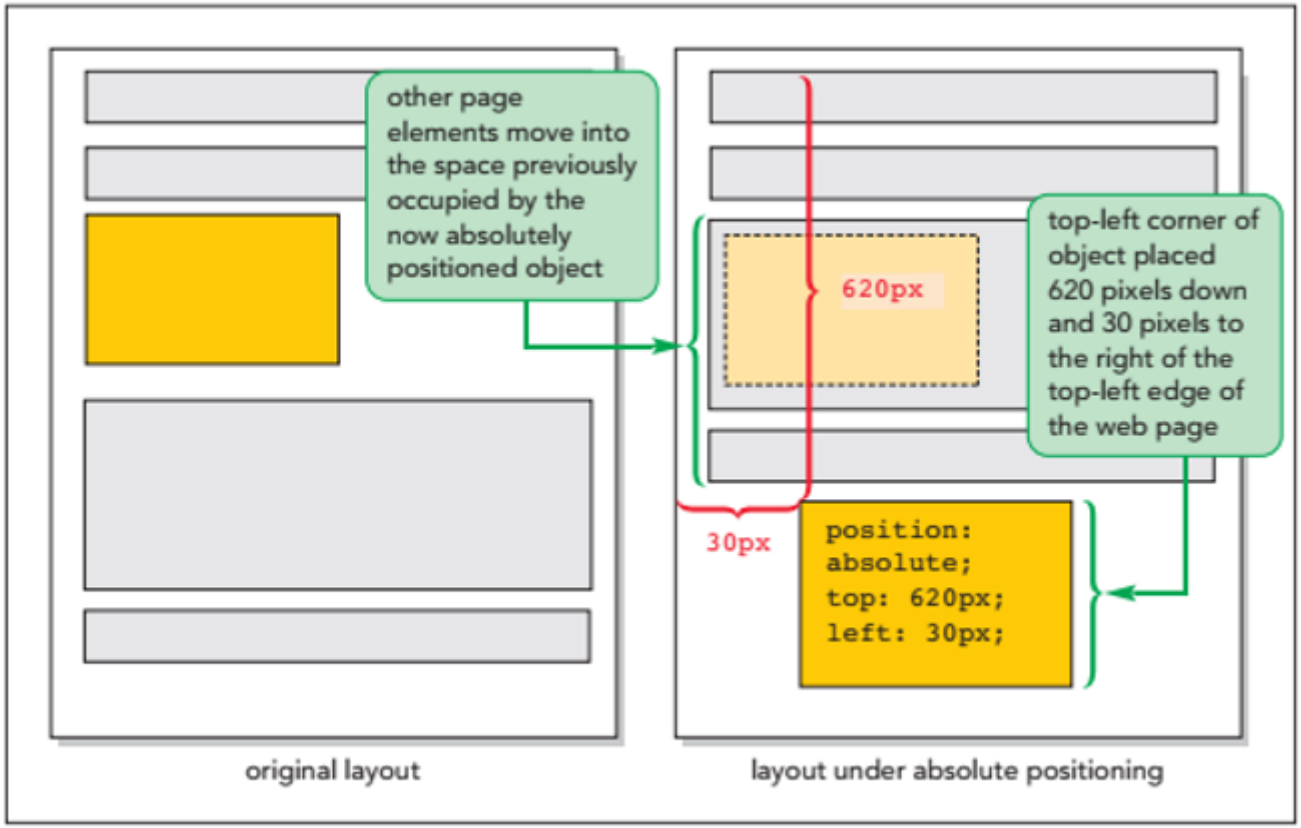
# CSS Positioning (con't)

- Moving an object via relative positioning:



# CSS Positioning (con't)

- Using absolute positioning:



# CSS Positioning (con't)

- Fixed positioning – Fixes an object within a browser window to avoids its movement
- Inherited positioning – Allows an element to inherit the position value of its parent element

# CSS Positioning (con't)

- Setting positioning type in parent object:

```
/* Infographic Styles */

div.infobox {
   position: absolute;
}

/* First Infographic */

div#info1 {
   display: block;
   top: 20px;
   left: 5%;
}
```

places every information box using absolute positioning

places the first box 20 pixels from the top edge of the main element and 5% from the left

```
/* Second Infographic */

div#info2 {
   display: block;
   top: 185px;
   left: 42%;
}

/* Third Infographic */

div#info3 {
   display: block;
   top: 135px;
   left: 75%;
}
```

places the second box 185 pixels from the top and 42% from the left

places the third box 135 pixels from the top and 75% from the left

# CSS Positioning (con't)

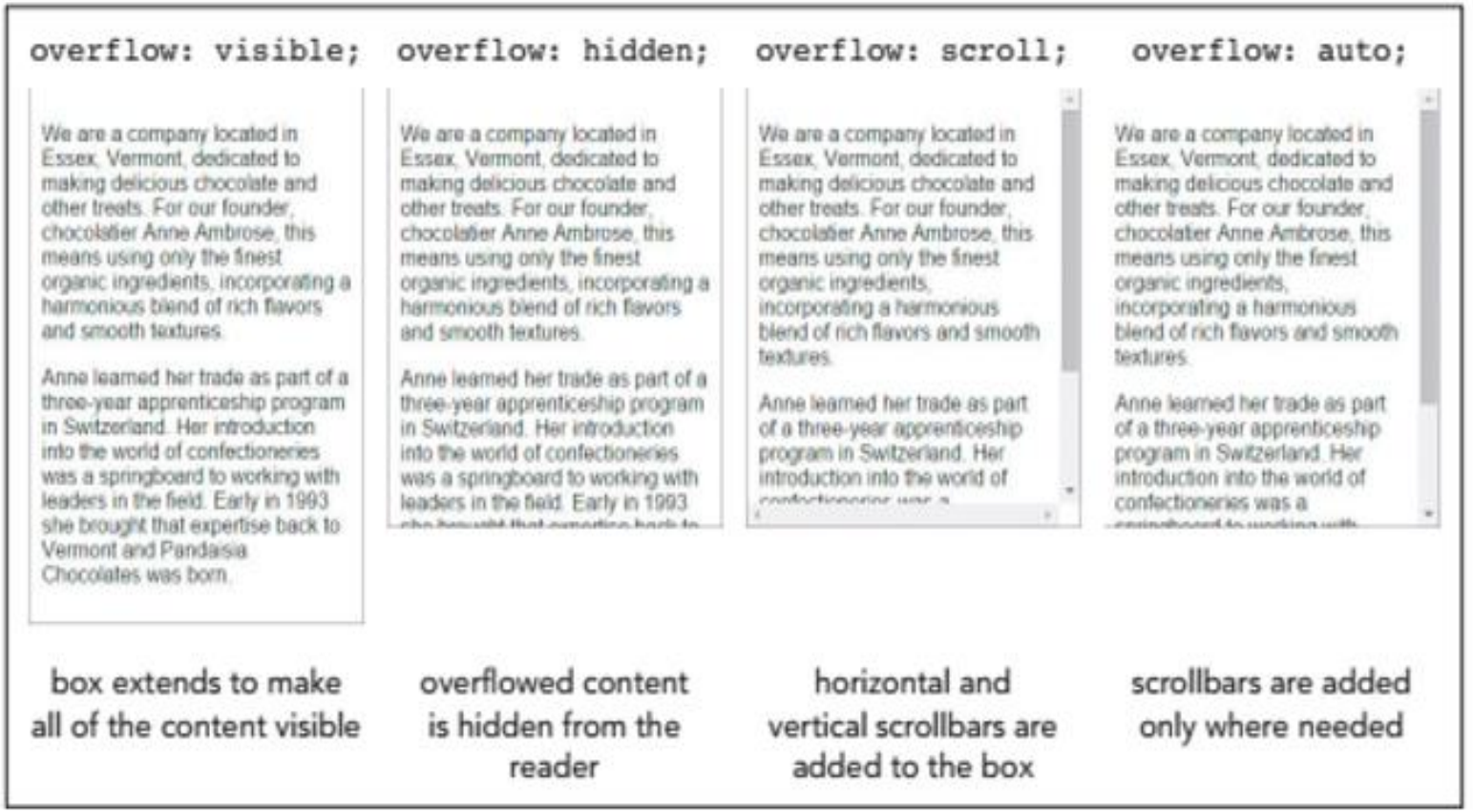
- `Overflow` property – Controls a browser that handles excess content

  `overflow: type;`

  where *`type`* is `visible` (the default), `hidden`, `scroll`, or `auto`

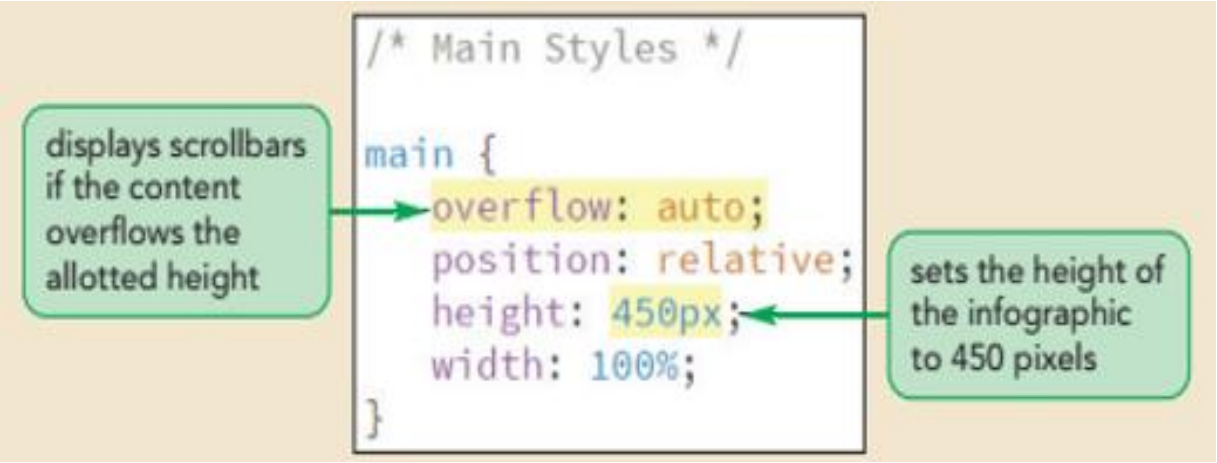
  - `visible` – Instructs browsers to increase the height of an element to fit overflow contents
    - `hidden` – Keeps an element at the specified height and width, but cuts off excess content
    - `scroll` – Keeps an element at the specified dimensions, but adds horizontal and vertical scroll bars
    - `auto` – Keeps an element at the specified size, adding scroll bars when they are needed

# CSS Positioning (con't)



# CSS Positioning (con't)

- CSS3 provides the `overflow-x` and `overflow-y` properties to handle overflow specially in the horizontal and vertical directions



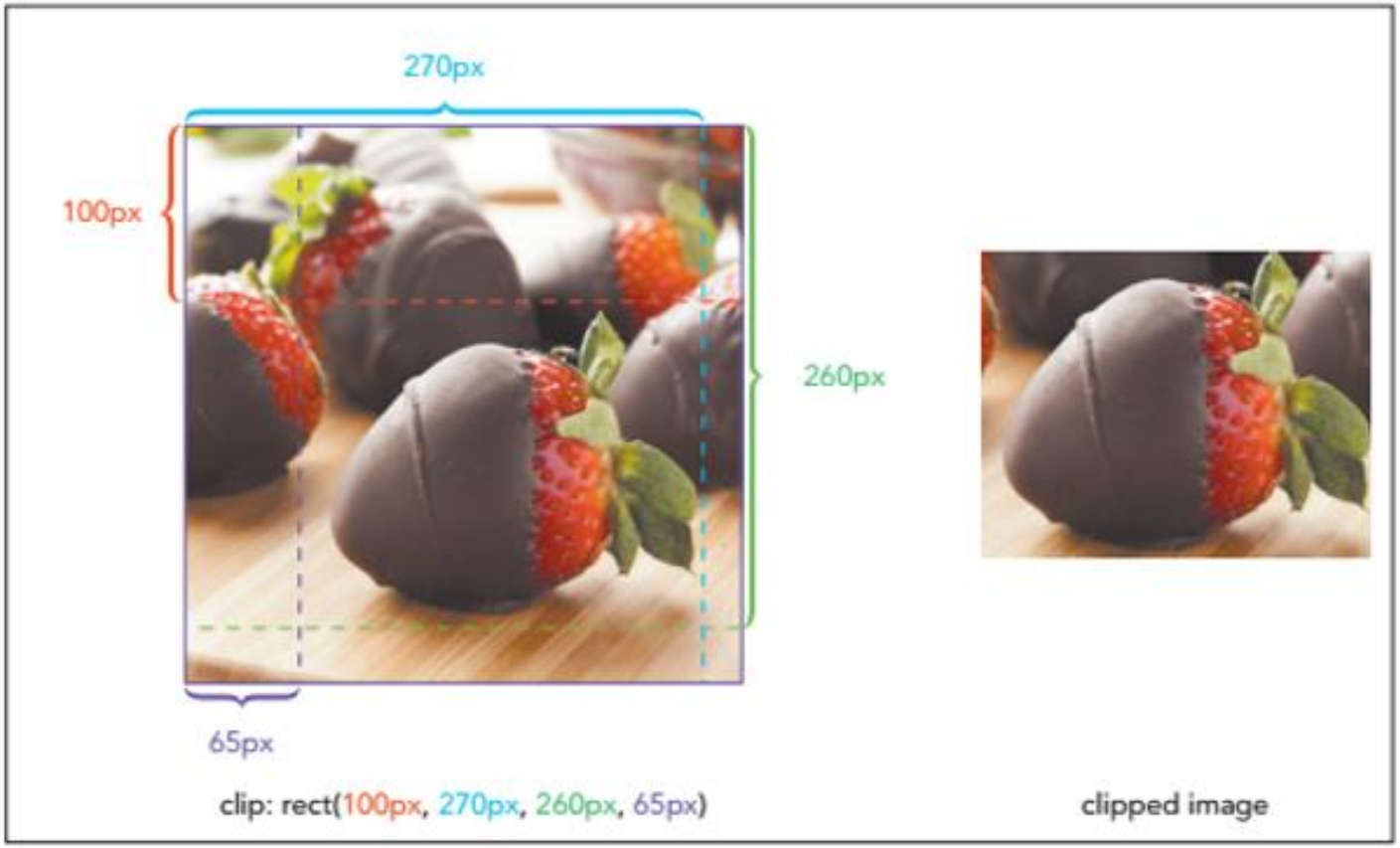
# Clipping an Object

- The `clip` property defines a rectangular region through which an element's content can be viewed
- Anything that lies outside the boundary of the rectangle is hidden
- The `clip` property syntax is

```
clip: rect(top, right, bottom,
left);
```

where *top*, *right*, *bottom*, and *left* define the coordinates of the clipping rectangle

# Clipping (con’t)



# Stacking Elements

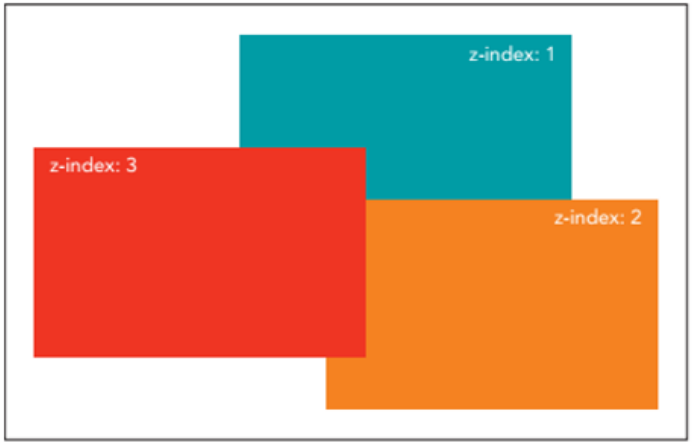
- By default, elements that are loaded later by a browser are displayed on top of elements that are loaded earlier
- To specify different stacking order, use the following `z-index` property:

```
z-index: value;
```

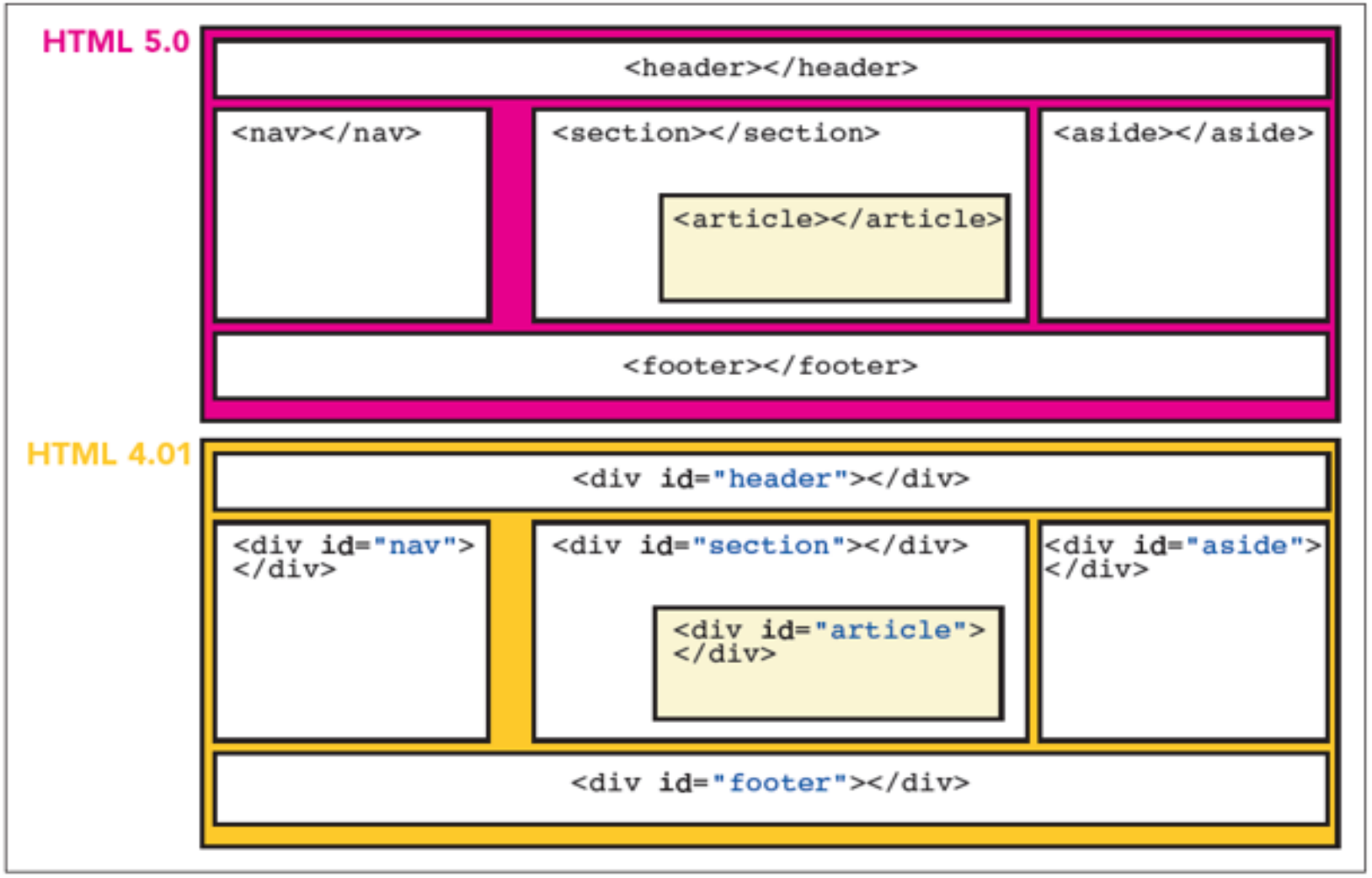
where *`value`* is a positive or negative integer, or the keyword `auto`

# Stacking (con't)

- The `z-index` property works only for elements that are placed with absolute positioning
- An element's z-index value determines its position relative only to other elements that share a common parent



# HTML 5 vs Earlier Sectioning



# Revised John Doe Homepage with CSS Layouts

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>John Doe's Home Page</title>
		<style>
			/* structural styles */
			article, aside, footer, header, main, nav, section {
				display: block;
			}
			/* layout styles */
			header {margin-left:auto; margin-right:auto}
			nav {float:left; width:15%; border:3px solid gray; background-color:tan; padding:3px; margin:3px;}
			aside {float:right; width:15%; border:3px solid gray; background-color:tan; padding:3px; margin:3px;}
			section {border:3px; padding:6px; margin:3px;}
			footer {clear:both;}
			/* grid style */
			div#mygrid {display:grid;}
			div#address-cell {outline: 3px solid gray;}
			div#hobby-cell {outline: 3px solid gray;}
			div#girl-cell {outline: 3px solid gray;}
			div#pet-cell {outline: 3px solid gray;}
		</style>
	</head>
```

# Revised Home page (con't)

```
<body>
        <header>
                <div align="center">
                        <h1>John Doe</h1>
                        <img align="center" src="smiling.gif" width=200 alt="[Picture of me]" hspace=50>
                        <h2>&#x0C38;&#x0C41;&#x0C38;&#x0C3E;&#x0C35;&#x0C17;&#x0C24; (Hello in Japanese)</h2>
                        <p>
                        This is a picture of me.
                        <p>
                                I am so happy to be in this class.
                        <p>
                                I always do my reading and homework on time, and I never miss a class.
                        <p>
                        <br clear=left>
                </div>
        </header>
        <nav>
                <h2>Links:</h2>
                <ul>
                        <li><a href="#add">address</a>
                        <li><a href="#hob">hobbies</a>
                        <li><a href="#girl">girlfriend</a>
                        <li><a href="#pets">pets</a>
                </ul>
        </nav>
        <aside>
        <h2>My Favorite Classes</h2>
        <ul>
                <li>MIS 470
                <li>MIS 471
                <li>MIS 351
                <li>MIS231
        </ul>
        </aside>
```

# Revised Home page (con't)

```
<section>
        <div id="mygrid">
          <div id="address-cell" align="center">
                <a name="add"><h2>My address</h2>
                <div style="font-style:italic">
                        11 Peach Street<BR>
                        Memphis, TN 38108
                </div>
                <p>
          </div>
          <div id="hobby-cell" align="center">
                <a name="hob"><h2>My Hobbies</h2>
                <img align=right src="hercules.gif" alt="[Picture of me at gym]" width=200 hspace=50>
                <p>
                        My hobby is going to the gym.
                <p>
                        I take my text books with me to
                        <br>read while I use the treadmill.
                <p>
                        I tried to study while in the spa,
                        <br>but I dropped my homework in the water.
                <p>
                        I wish they would put internet terminals
                        <br>on the treadmills and stationery bikes.
                <p>
                        Here is a picture of me.
                        <br clear=right>
                <p>
          </div>
          <div id="girl-cell">
                <a name="girl"><h2 align="center">My Girlfriend</h2>
                <img align=right src="hercgirl.gif" alt="[Picture of my girlfriend]" width=200 hspace=50>
                <div style="font-size:large">This is my girlfriend !</div>
                <p>
                These are the things we like to do:
                <ul type=circle>
                        <li>Ride in my 1983 Ford Pinto
                        <li>Help pick up litter on our campus
                        <li>Eat in the school snack bar
                        <li>Sign up for 8am classes together
                        <li>Discuss student affairs with our school administration
                        <li>Study our math class notes together &#171 What Fun &#187
                        <li>Surf the Net !
                </ul>
                <br clear=right>
                <p>
          </div>
```

# Revised Home page (con't)

```
                    <div id="pet-cell" align="center">
                        <a name="pets"><h2>My Pets</h2>
                        <p>
                        <table border width=400>
                            <caption align=top>Pet Types and Names</caption>
                            <tr>
                                <th width=40%>Name</th><th width=60%>Type</th>
                            </tr>
                            <tr>
                                <td>Fred</td><td>Pigeon</td>
                            </tr>
                            <tr>
                                <td>Alice</td><td>Skunk</td>
                            </tr>
                            <tr>
                                <td>Mike</td><td>Aligator</td>
                            </tr>
                        </table>
                        <p>
                    </div>
                </div>
            </section>
            <footer>
                <p><h3 align="center">Copyright John Doe</h3>
            </footer>
        </body>
</html>
```

# htmllab12.htm

## John Doe

### సుసావగత (Hello in Japanese)

This is a picture of me.

I am so happy to be in this class.

I always do my reading and homework on time, and I never miss a class.

**Links:**

- address
- hobbies
- girlfriend
- pets

### My address

*11 Peach Street*
*Memphis, TN 38108*

### My Hobbies

My hobby is going to the gym.

I take my text books with me to
read while I use the treadmill.

I tried to study while in the spa,
but I dropped my homework in the water.

I wish they would put internet terminals
on the treadmills and stationery bikes.

Here is a picture of me.

### My Girlfriend

This is my girlfriend !

These are the things we like to do:

- Ride in my 1983 Ford Pinto
- Help pick up litter on our campus
- Eat in the school snack bar
- Sign up for 8am classes together
- Discuss student affairs with our school administration
- Study our math class notes together « What Fun »
- Surf the Net !

### My Pets

Pet Types and Names

| Name | Type |
|---|---|
| Fred | Pigeon |
| Alice | Skunk |
| Mike | Aligator |

### My Favorite Classes

- MIS 470
- MIS 471
- MIS 351
- MIS231

# htmllab12.htm

Section is set up as a grid.

{header}

## John Doe

### సుసావగత (Hello in Japanese)

This is a picture of me.

I am so happy to be in this class.

I always do my reading and homework on time, and I never miss a class.

{nav}

**Links:**

- address
- hobbies
- girlfriend
- pets

{section}

### My address

*11 Peach Street*
*Memphis, TN 38108*

### My Hobbies

My hobby is going to the gym.

I take my text books with me to
read while I use the treadmill.

I tried to study while in the spa,
but I dropped my homework in the water.

I wish they would put internet terminals
on the treadmills and stationery bikes.

Here is a picture of me.

### My Girlfriend

This is my girlfriend !

These are the things we like to do:

- Ride in my 1983 Ford Pinto
- Help pick up litter on our campus
- Eat in the school snack bar
- Sign up for 8am classes together
- Discuss student affairs with our school administration
- Study our math class notes together « What Fun »
- Surf the Net !

### My Pets

Pet Types and Names

| Name | Type |
|---|---|
| Fred | Pigeon |
| Alice | Skunk |
| Mike | Aligator |

{aside}

**My Favorite Classes**

- MIS 470
- MIS 471
- MIS 351
- MIS231

{footer} 

# CSS Frameworks

- A **framework** is a software package that provides a library of tools to design a website
  - Includes style sheets for grid layouts and built-in scripts to provide support for a variety of browsers and devices
- Some popular CSS frameworks include
  - **Bootstrap**, **Neat**, **Unsemantic**, **Profound Grid**, **HTML5 Boilerplate**, **Skeleton**

# Bootstrap

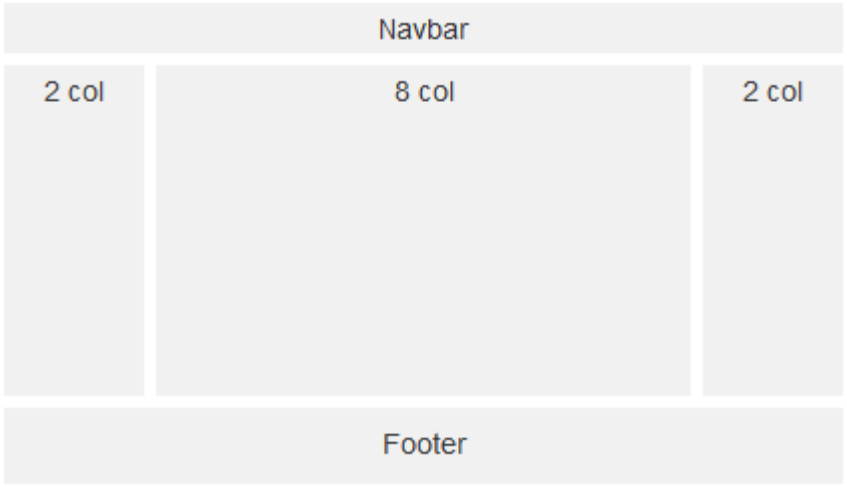
Bootstrap is likely the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites. Bootstrap is completely free to download and use.

# Bootstrap (con't)

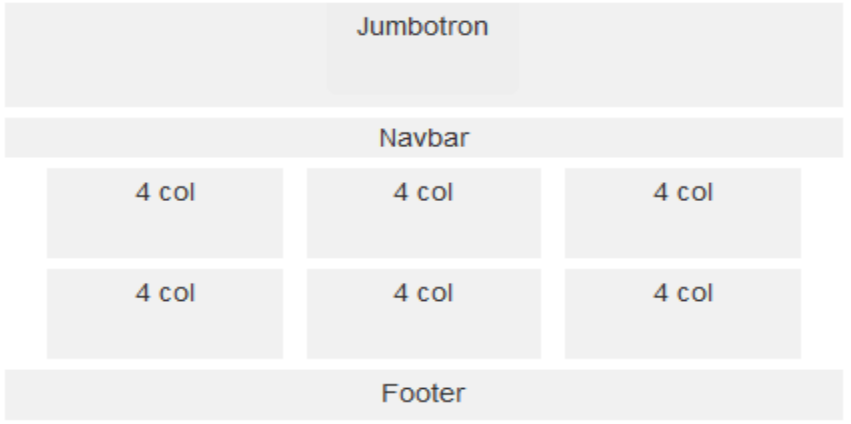
```
<div class="jumbotron text-center">
  <h1>My First Bootstrap Page</h1>
  <p>Resize this responsive page to see the effect!</p>
</div>

<div class="container">
  <div class="row">
    <div class="col-sm-4">
      <h3>Column 1</h3>
      <p>Lorem ipsum dolor..</p>
    </div>
    <div class="col-sm-4">
      <h3>Column 2</h3>
      <p>Lorem ipsum dolor..</p>
    </div>
    <div class="col-sm-4">
      <h3>Column 3</h3>
      <p>Lorem ipsum dolor..</p>
    </div>
  </div>
</div>
```

# Bootstrap Templates




Typical Website




Online Store

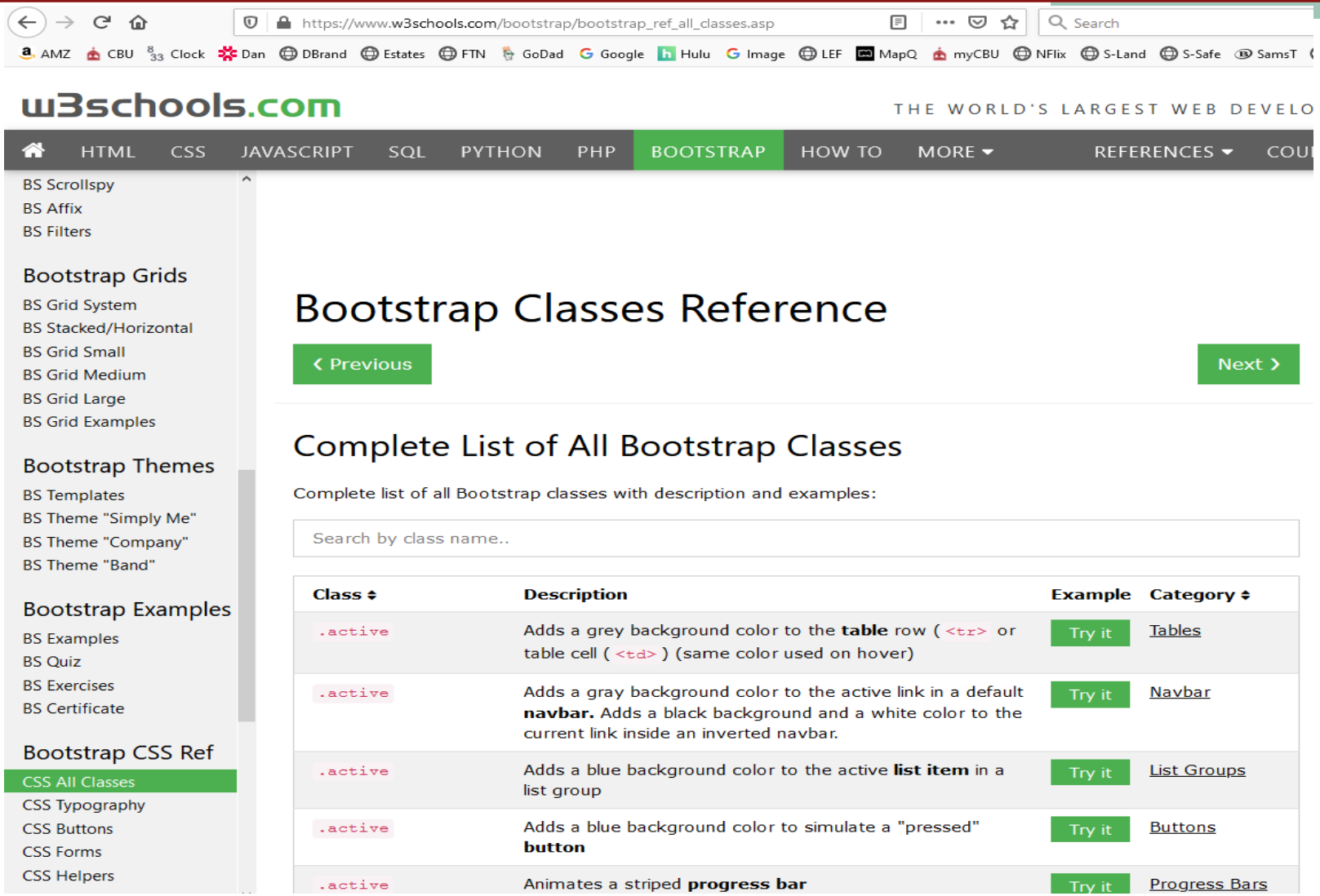
# Bootstrap Content Delivery Network

- Bootstrap provides its files thru its Content Delivery Network (CDN)
- This requires one to be connected to the internet when developing/using bootstrap boilerplate CSS files
  - One can also download the Bootstrap source code, but that is not necessary; there are over 10,000 lines of code in the Bootstrap CSS file
- The next slide shows the code that needs to be included into your HTML file to include the Bootstrap code (CSS and JavaScript files)
  - Bootstrap uses jQuery and Popper

# Links to Load Bootstrap

```
<html>
    <head>
        <title>Welcome</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
    </head>
    <body>
        <h1>Welcome to My Website</h1>
        <p>
            Some text…
</p>
        <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>
        <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>
    </body>
</html>
```

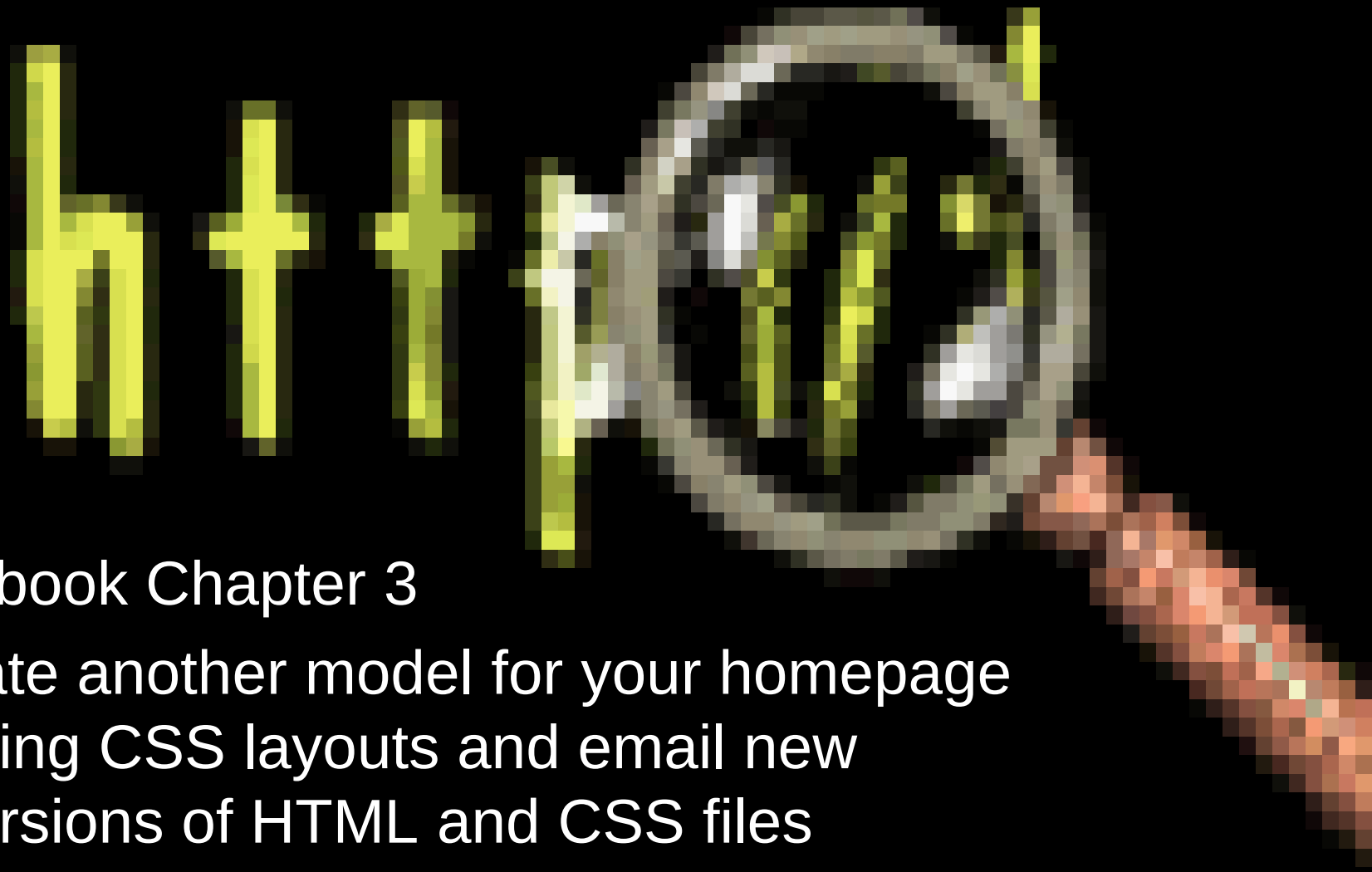
# Bootstrap Classes Reference



# References

- Learn Enough CSS & Layout to Be Dangerous: A tutorial introduction to CSS and page layout (Learn Enough Web Basics) by Lee Donahoe and Michael Hartl
- Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics by Jennifer Robbins
- Web Design - Start Here: A No-Nonsense, Jargon Free Guide to the Fundamentals of Web Design by Stefan Mischook

# Homework



Textbook Chapter 3

Create another model for your homepage using CSS layouts and email new versions of HTML and CSS files