



*NEW PERSPECTIVES*

# HTML5, CSS3, and JavaScript

## 6<sup>th</sup> Edition

### Tutorial 3

## Designing a Page Layout

# Objectives – 1/2

---

- Create a reset style sheet
- Explore page layout designs
- Center a block element
- Create a floating element
- Clear a floating layout
- Prevent container collapse
- Explore grid-based layouts

# Objectives – 2/2

---

- Create a layout grid
- Format a grid
- Explore the CSS grid styles
- Explore positioning styles
- Work with relative positioning
- Work with absolute positioning
- Work with overflow content

# Page Layout with Floating Elements



# Introducing the *display* Style - 1/2

- HTML elements are classified into
  - Block elements, such as paragraphs or headings
  - Inline elements, such as emphasized text or inline images
- The display style can be defined for any page element using

`display: type;`

where *type* defines the display type

# Introducing the *display* Style - 2/2

Figure 3-1

Some values of the display property

Display Value	Appearance
block	Displayed as a block
table	Displayed as a web table
inline	Displayed in-line within a block
inline-block	Treated as a block placed in-line within another block
run-in	Displayed as a block unless its next sibling is also a block, in which case, it is displayed in-line, 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

© 2016 Cengage Learning

# Creating a Reset Style Sheet – 1/2

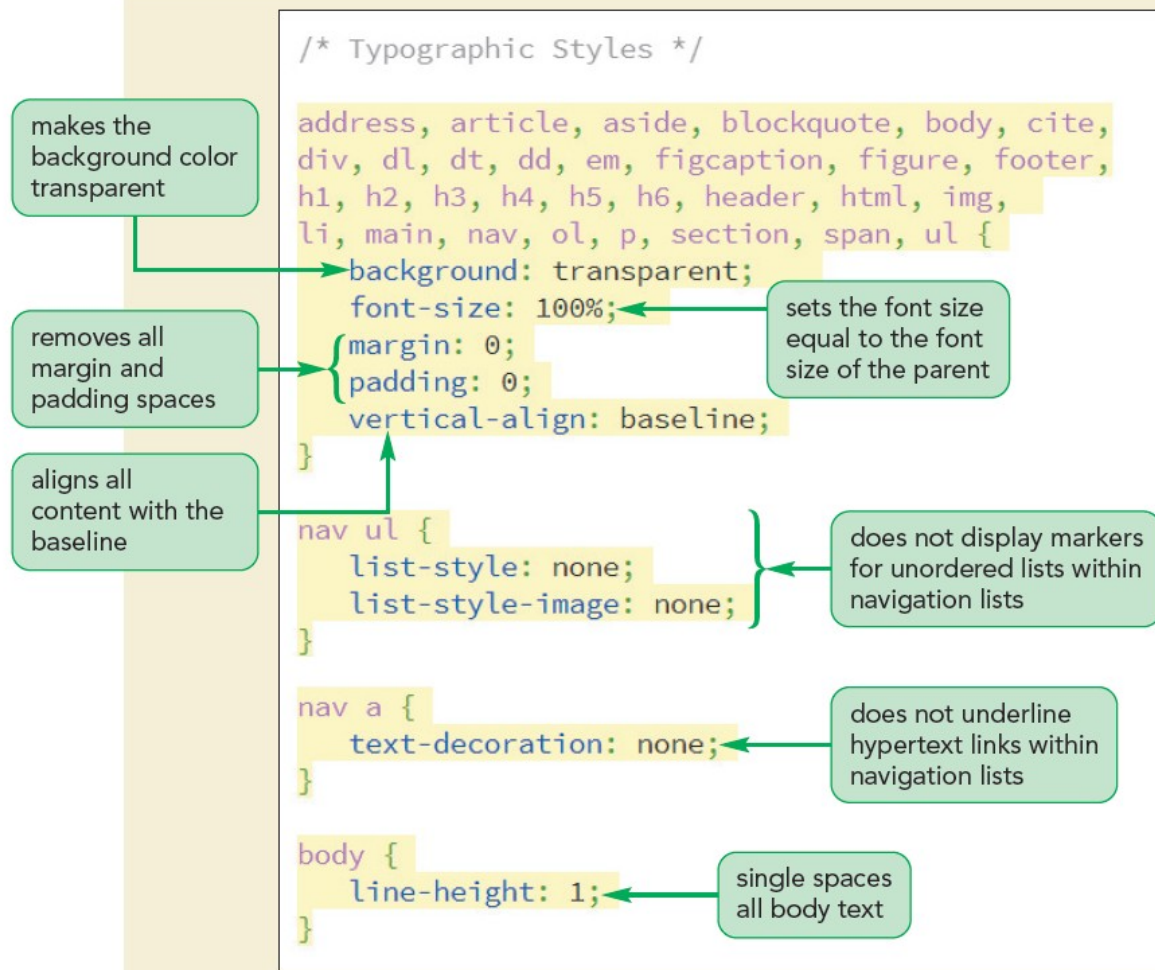
- **Reset style sheet** supersedes a browser's default styles and provides a consistent starting point for page design
- The first style rule in a sheet is the `display` property used to display HTML5 structural elements

**Figure 3-2** Displaying HTML5 structural elements as blocks

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

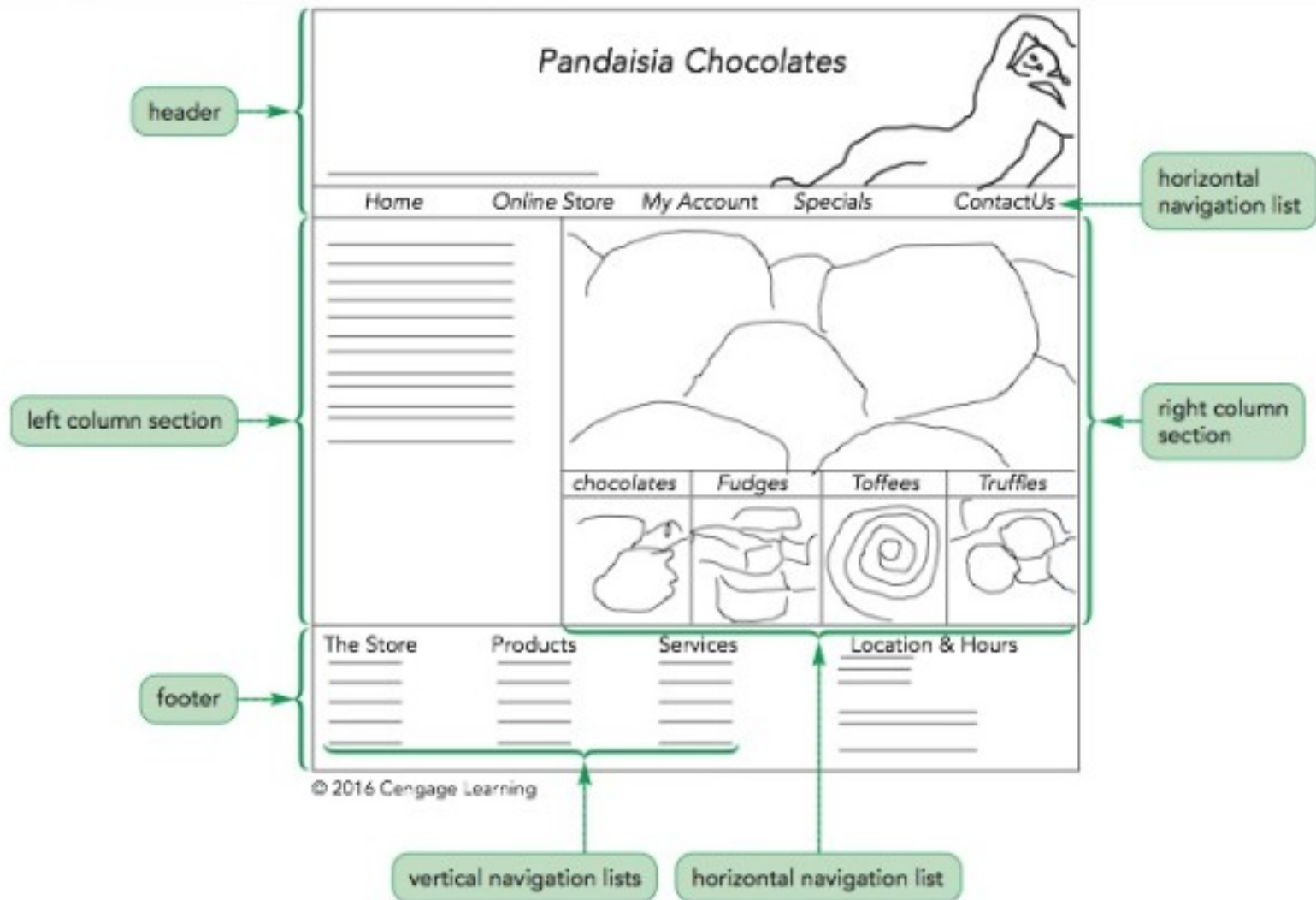
# Creating a Reset Style Sheet – 2/2

Figure 3-3 Completing the reset style sheet





# Page Layout Design – Pandaisia Chocolates



# Exploring Page Layout Designs – 1/2

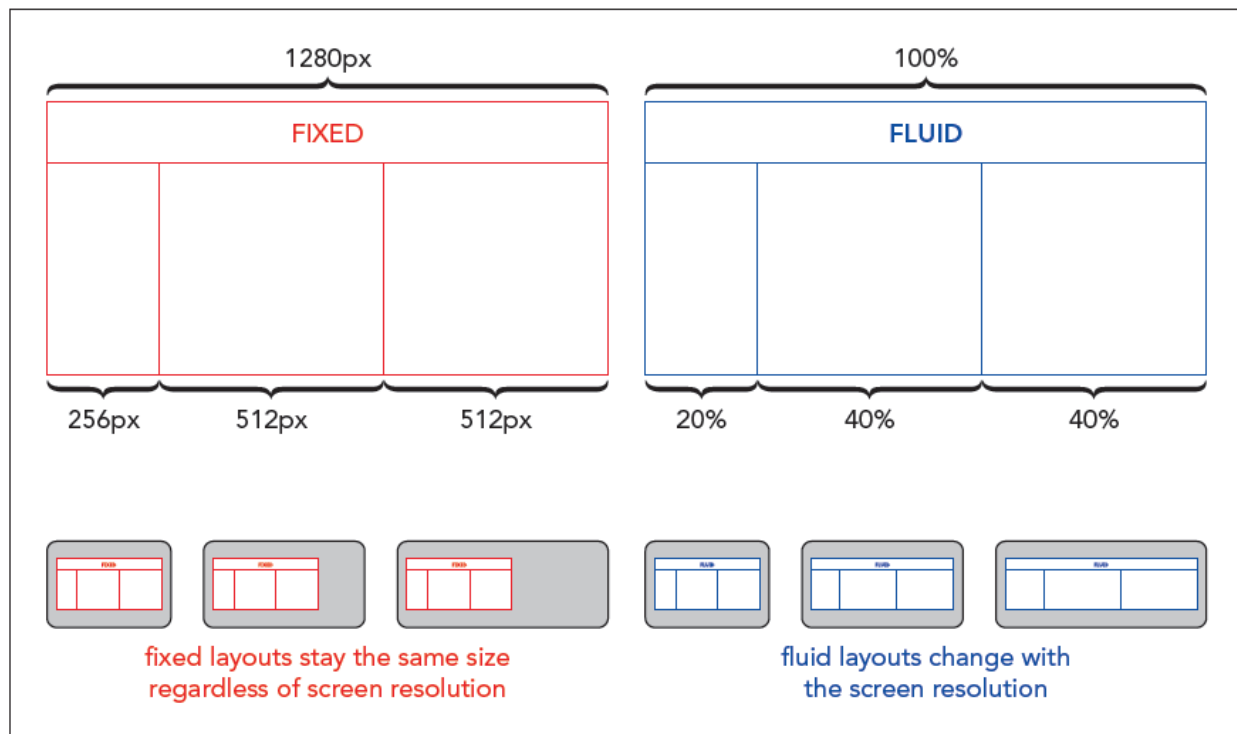
---

- 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

# Exploring Page Layout Designs - 2/2

- **Responsive design** – The layout and design of a page changes in response to the device that is rendering it

**Figure 3-5** Fixed layouts vs. fluid layouts



# Working with Width and Height – 1/2

---

- 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

# Working with Width and Height -2/2

Figure 3-6 Setting the width of the page body and logo

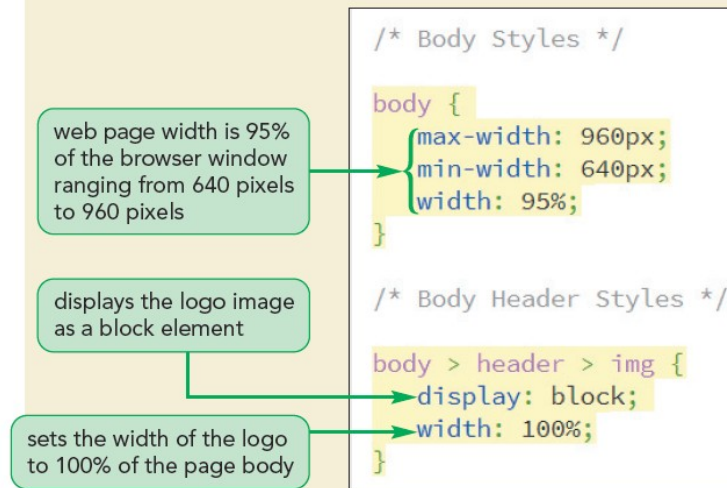
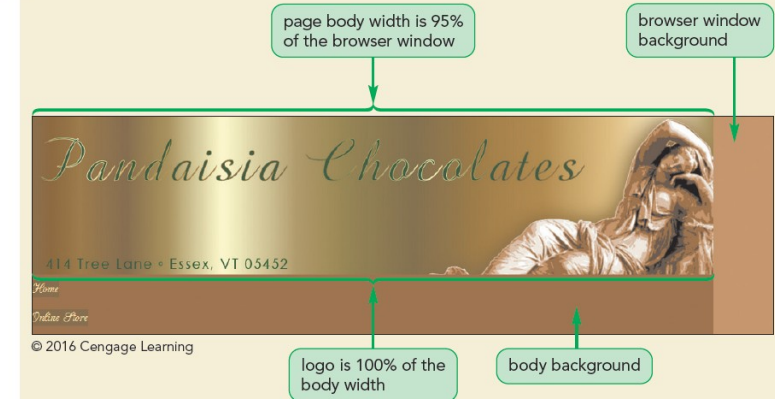


Figure 3-7 Initial view of the body header



- The height:

```
html { height: 100%; }
```

```
Body { min-height: 100%; }
```

# Centering a Block Element – 1/3

---

- Block elements can be centered horizontally within their parent element by setting both the left and right margins to `auto`

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

# Vertical Centering – 2/3

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

```
<div>  
    <h1>Pandaisia Chocololates</h1>  
</div>
```

# Vertical Centering – 3/3

---

- Apply the style rule

```
div {  
    height: 40px;  
    display: table-cell;  
    vertical-align: middle;  
}  
  
h1 { font-size: 1.4em;  
     line-height: 2em; }
```

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



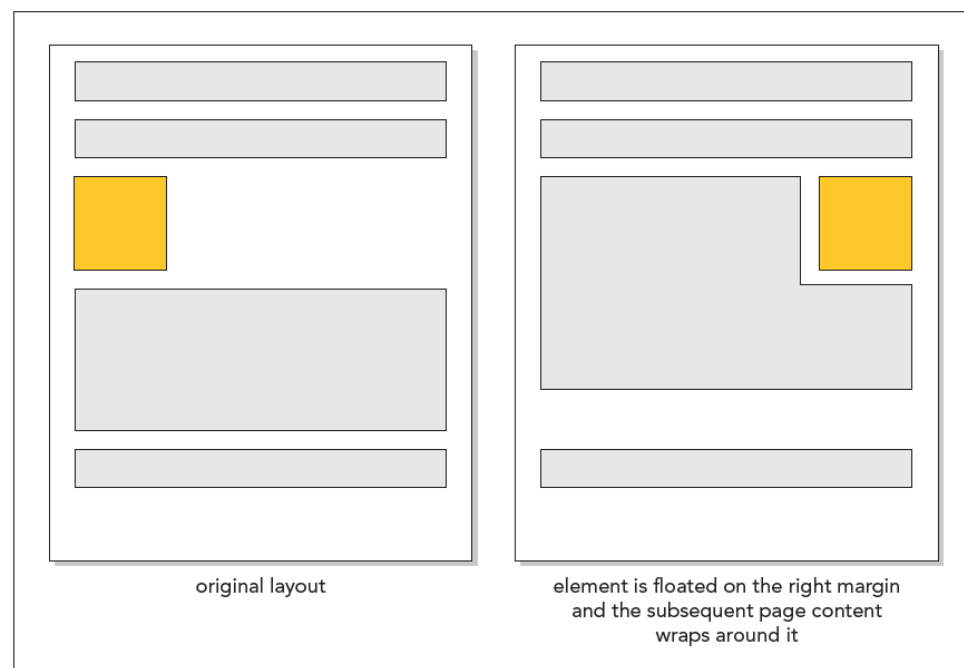
# Floating Page Content – 1/3

- **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 Page Content – 2/3

- For elements to be placed within a single row, the combined width of the elements cannot exceed the total width of their parent element

Figure 3-9 Floating an element



© 2016 Cengage Learning

# Floating Page Content – 3/3

Figure 3-13

Formatting hyperlinks in horizontal navigation lists

```
/* Horizontal Navigation Styles */  
  
nav.horizontalNavigation li {  
    display: block;  
    float: left;  
}  
  
nav.horizontalNavigation a {  
    display: block;  
    text-align: center;  
}
```

displays the  
link as a block

centers the  
link text within  
the block

Figure 3-14

Links in the body header

each hypertext  
link displayed as  
a block with the  
link text centered  
within the block



© 2016 Cengage Learning

# Clearing a Float – 1/5

---

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

`clear: position;`

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

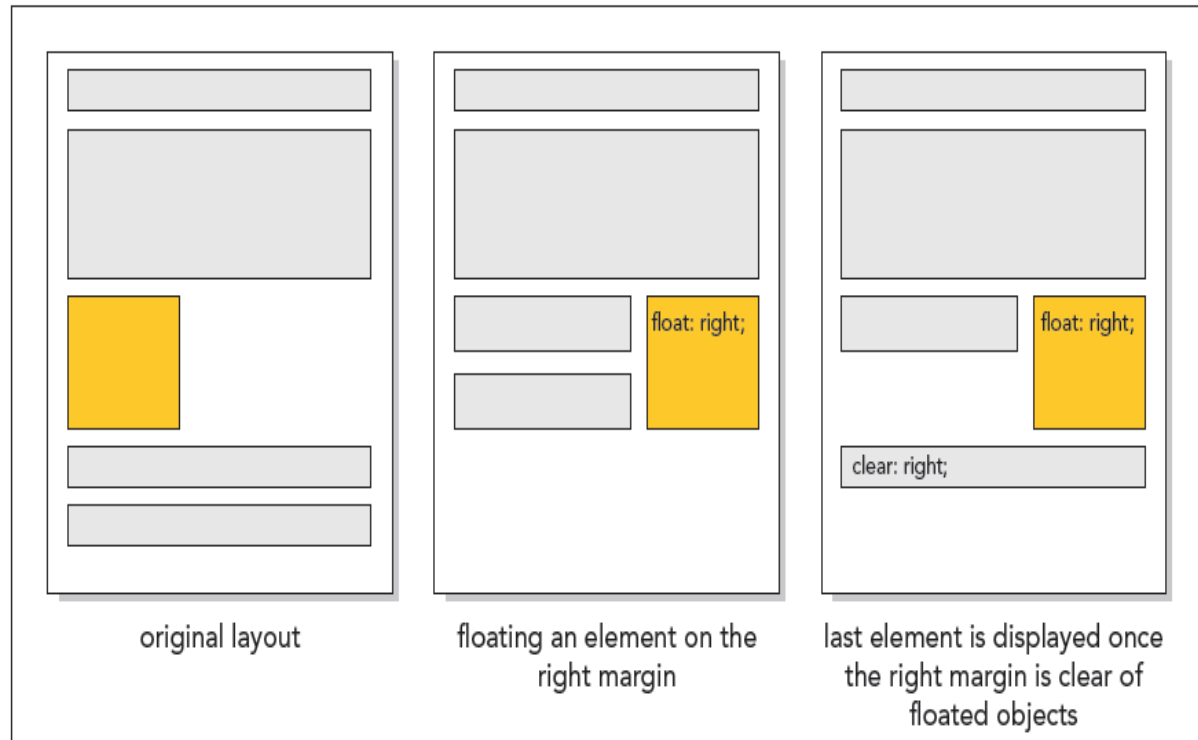
# Clearing a Float – 2/5

---

- `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

# Clearing a Float – 3/5

**Figure 3-15** Clearing a float



© 2016 Cengage Learning

# Clearing a Float – 4/5

Figure 3-16 Float the left and right column sections

```
/* Left Column Styles */
section#leftColumn {
  clear: left;
  float: left;
  width: 33%;
}

/* Right Column Styles */
section#rightColumn {
  float: left;
  width: 67%;
}
```

displays the left column once the left margin is clear of previously floated elements

floats the left column on the left margin with a width of 33% of the page body

floats the right column alongside the left column with a width of 67%

Figure 3-17 Formatting the right column section

```
/* Right Column Styles */
section#rightColumn {
  float: left;
  width: 67%;
}

section#rightColumn img {
  display: block;
  width: 100%;
}

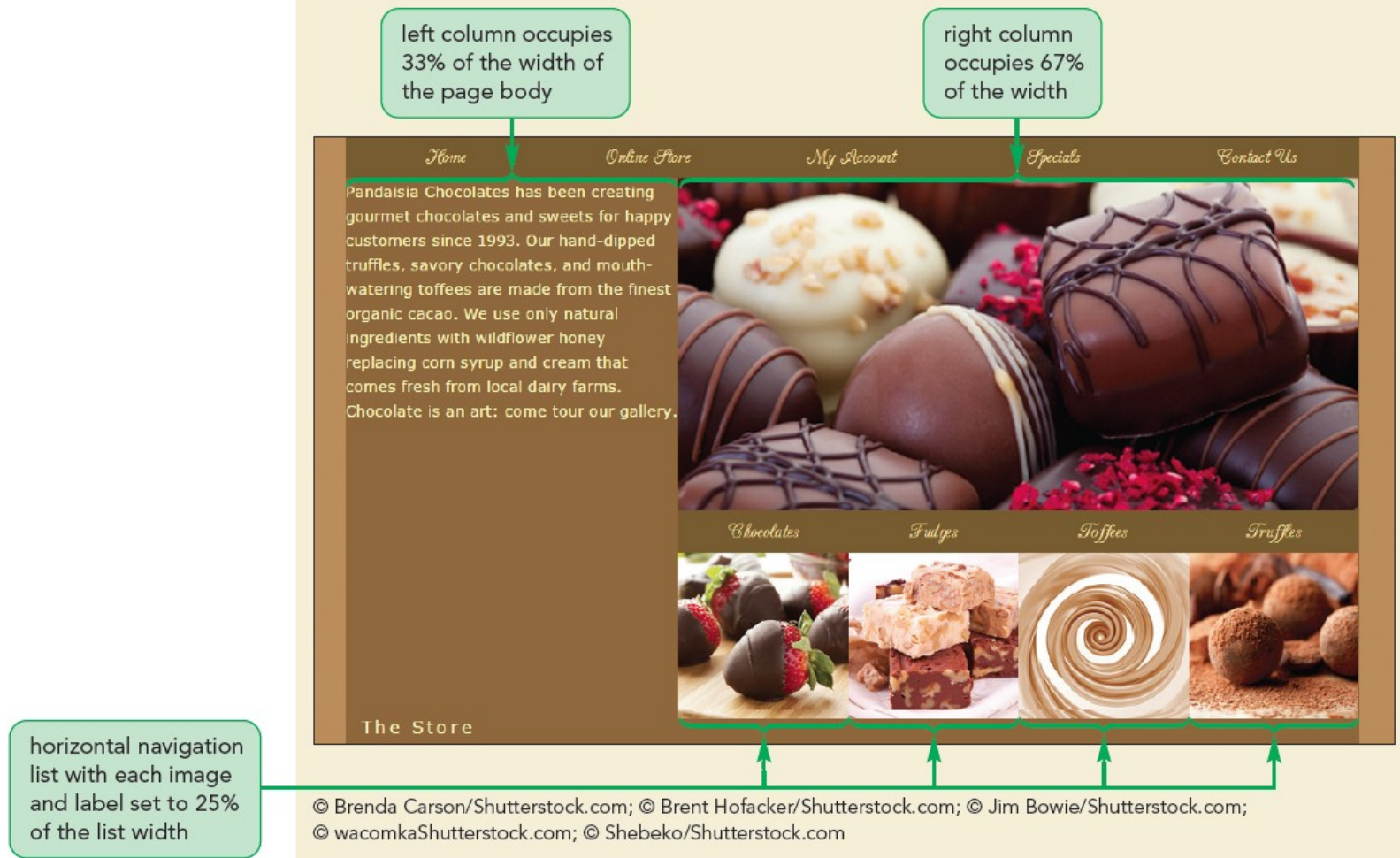
section#rightColumn > nav.horizontalNavigation li {
  width: 25%;
}
```

displays every image in the right column as a block with a width equal to the width of its parent element

sets the width of each list item to 25% of the width of the navigation list

# Clearing a Float – 5/5

Figure 3-18 Layout of the left and right columns





# Refining a Floated Layout – 1/3

---

- **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

# Refining a Floated Layout – 2/3

---

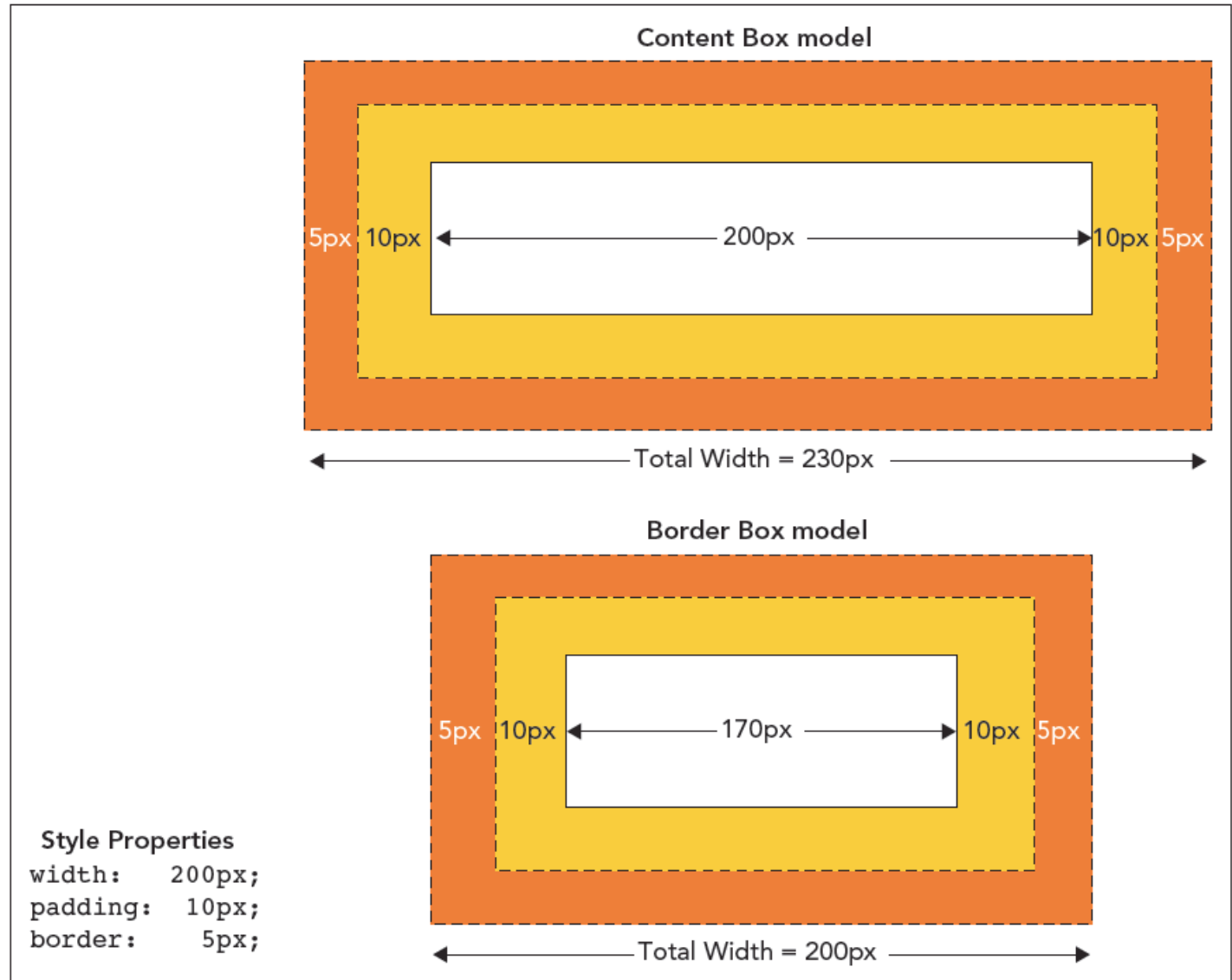
- The layout model can be chosen using

`box-sizing: type;`

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

# Refining a Floated Layout – 3/3

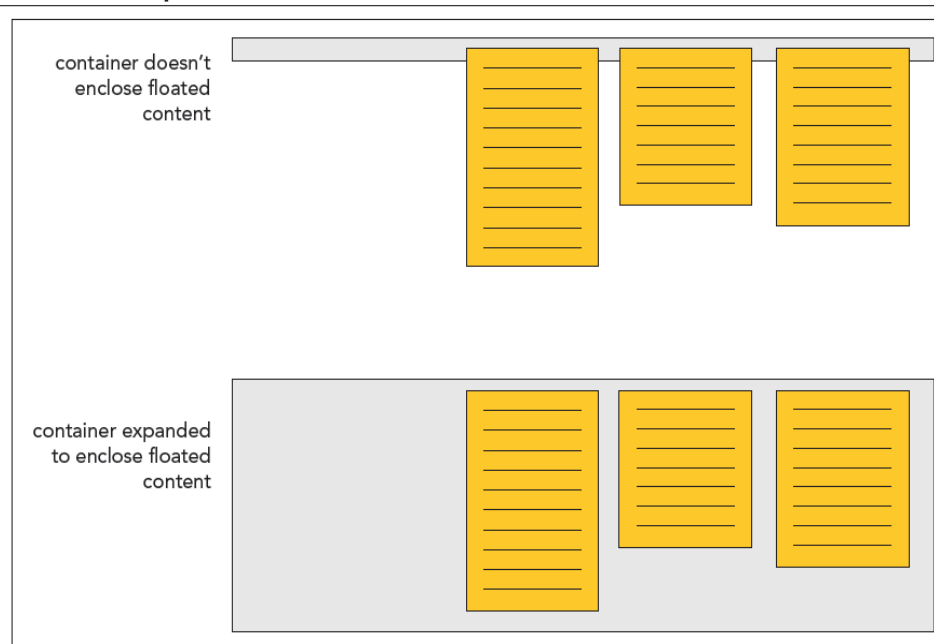
Figure 3-21 Comparing the Content Box and Border Box models



# Working with Container Collapse – 1/3

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

Figure 3-26 Container collapse



© 2016 Cengage Learning

## Working with Container Collapse – 2/3

- Use the `after` pseudo-element to add a placeholder element after the footer
- The general style rule is

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

where *container* is the selector for the element containing floating objects

# Working with Container Collapse – 3/3

---

- The `clear` property keeps the placeholder element from being inserted until both margins are clear of floats
- The element itself is a web table and contains an empty text string

# Section 3.2: Page Layout Grids

## Page Layout Grids

A grid layout arranges the page content within grid rows with grid columns floated inside those rows.

Red outline indicates the location of grid rows and columns.


Grid rows are displayed starting on a new line.

The grid columns are floated with their rows.

**Pandaisia Chocolates**  
414 Tree Lane • Essex, VT 05452  
Home Online Store My Account Specials Contact Us

### About Pandaisia Chocolates

#### Our Company



We are a company located in Essex, Vermont, dedicated to making delicious chocolate and other treats. For our founder, chocolatier Anne Ambrose, this means using only the finest organic ingredients, incorporating a harmonious blend of rich flavors and smooth textures.

#### FAQ

Do you do weddings?

Yes! That's our favorite thing to do. We sell bulk chocolates in a wide variety of box designs perfect for weddings or other special occasions.

How long do your chocolates last?

We recommend that you store our chocolates in a dark, cool place and consume them within two weeks of purchase.

What varieties are you selling?

We're constantly updating our product list to match seasonal expectations. Typically we have between 12 and 18 varieties of any one time.

Can I customize my own box?

Of course! We have special gift boxes but if you want to create a box of your favorites, we're glad to oblige.

Can I request a special variety?

We're happy to consider requests, but remember that some varieties are seasonal and cannot always be made to order.

#### About Chocolate

##### Enjoying Chocolate

The best chocolate is fresh chocolate. Preservatives change the flavor and texture of chocolate. For the best results, our chocolates should be consumed within a few days of purchase. Store them in a cool, dark place at a temperature of 60° to 70° and then enjoy!

##### Healthy Chocolate

Chocolate has a bad reputation because of the poor quality of mass-produced bars loaded with lots of milk, sugar, and butter – which are tasty but not healthy. We keep the processed sugars to a minimum and produce dark chocolate that is 73% cacao.

Pandaisia Chocolates © 2017 All Rights Reserved

# Overview of Grid-Based Layouts – 1/3

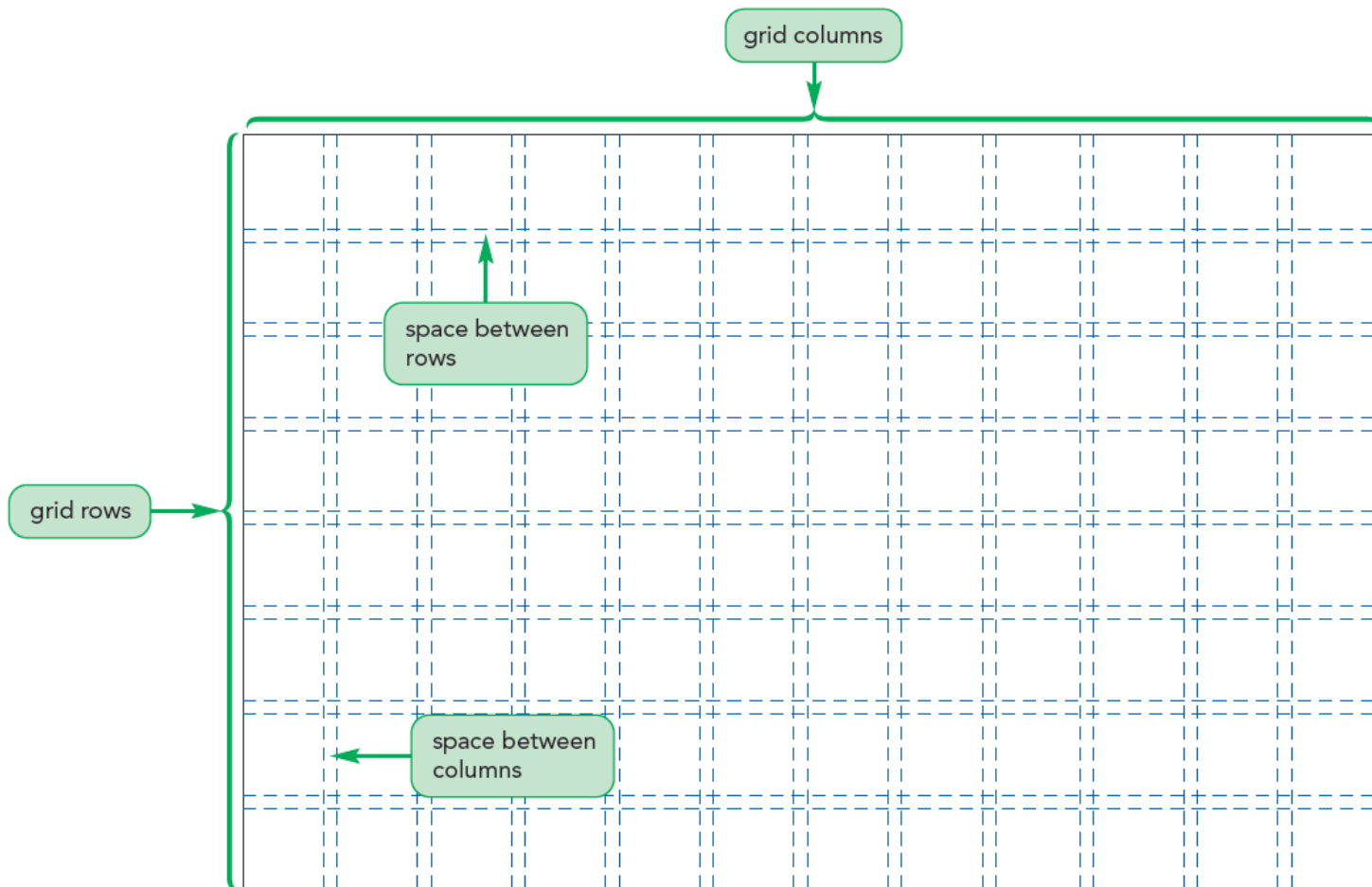
---

- Rows and columns form a grid
  - The number of rows is 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



# Overview of Grid-Based Layouts - 2/3

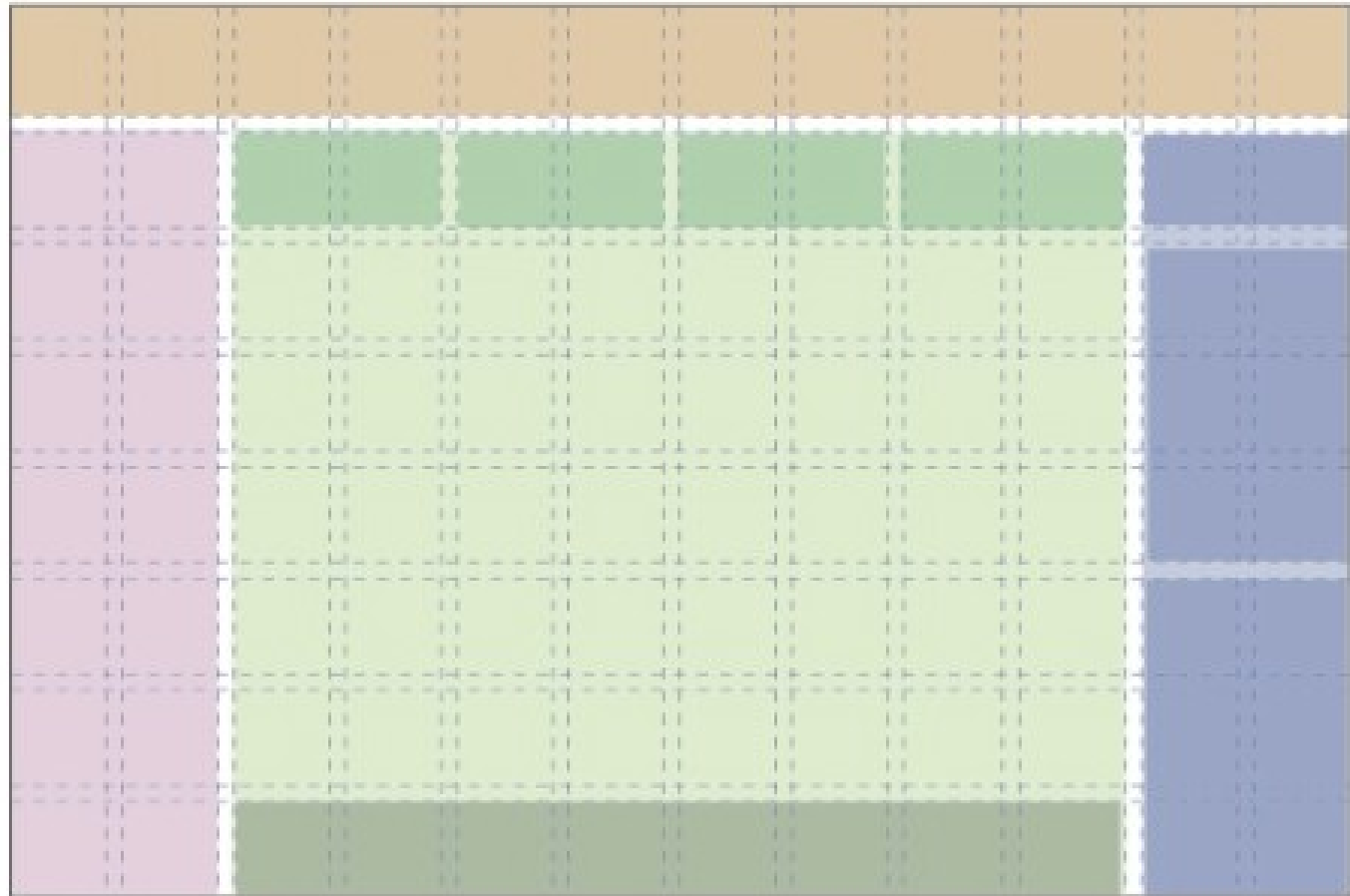
Figure 3-29 Page grid



© 2016 Cengage Learning

# Overview of Grid-Based Layouts – 2/3

**Figure 3-30** Layout based on a grid



© 2016 Cengage Learning

# Overview of Grid-Based Layouts – 3/3

---

- 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
  - It is easily accessible for users with disabilities and special needs
  - It increases the development speed with a systematic framework for the page layout

# Fixed and Fluid Grids

---

- **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 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**
  - **YAML4**
  - **960 Grid System**
  - **Foundation 3**

# Setting up a Grid – 1/4

---

- A grid layout is based on rows of floating elements
- Each floating element constitutes a column
- The set of elements floating side-by-side establishes a row
- Many grid layouts use the `div` (or division) element to mark distinct rows and columns of the grid

# Setting up a Grid – 2/4

- This is an example of a simple grid consisting of a single row with two columns:

```
<div class="row">  
  <div class="column1"></div>  
  <div class="column2"></div>  
</div>
```

The page content is placed within the `div` elements

# Setting up a Grid – 2/4

- This is an example of a simple grid consisting of a single row with two columns:

```
<div class="row">
  <div class="column1">
    <div class="row">
      <div class="column1a"></div>
      <div class="column1b"></div>
    </div>
    <div class="row">
      <div class="column1c"></div>
      <div class="column1d"></div>
    </div>
  </div>
  <div class="column2"></div>
</div>
```

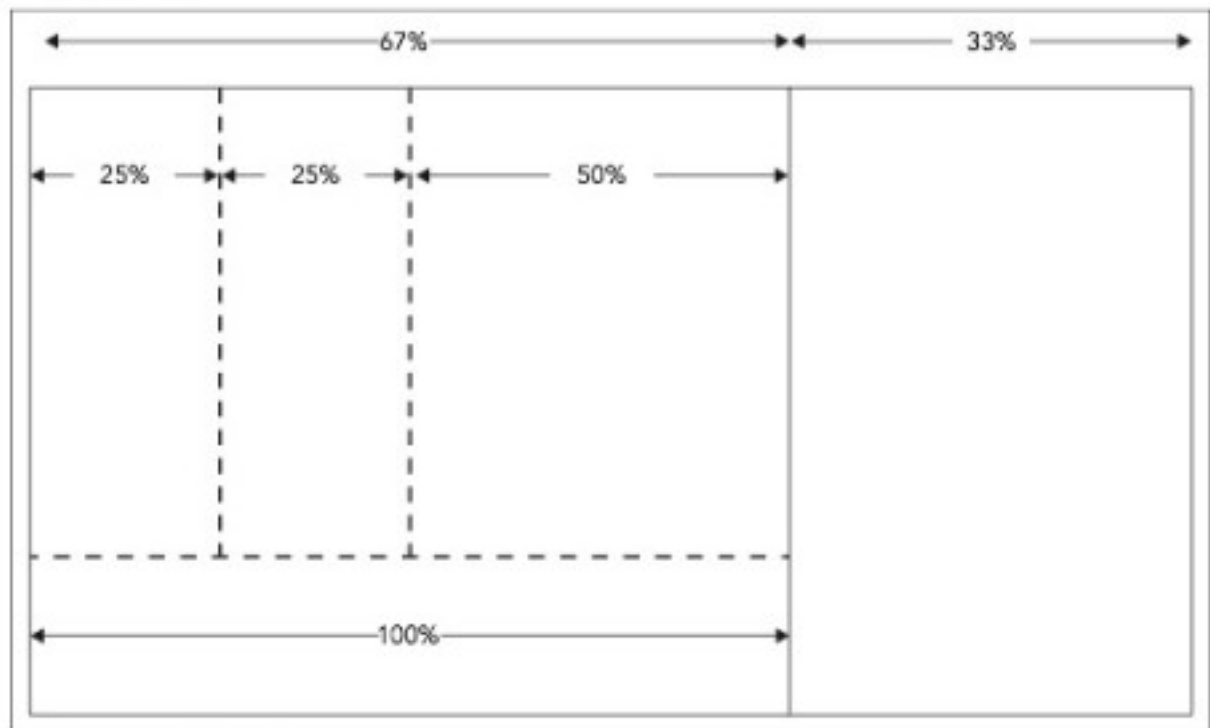


# Setting up a Grid – 2/4

- This is an example of a simple grid consisting of a single row with one columns:

```
<div class="row">
  <div class="col-2-3">
    <div class="row">
      <div class="col-1-4"></div>
      <div class="col-1-4"></div>
      <div class="col-1-2"></div>
    </div>
    <div class="row">
      <div class="col-1-1"></div>
    </div>
  </div>
  <div class="col-1-3"></div>
</div>
```

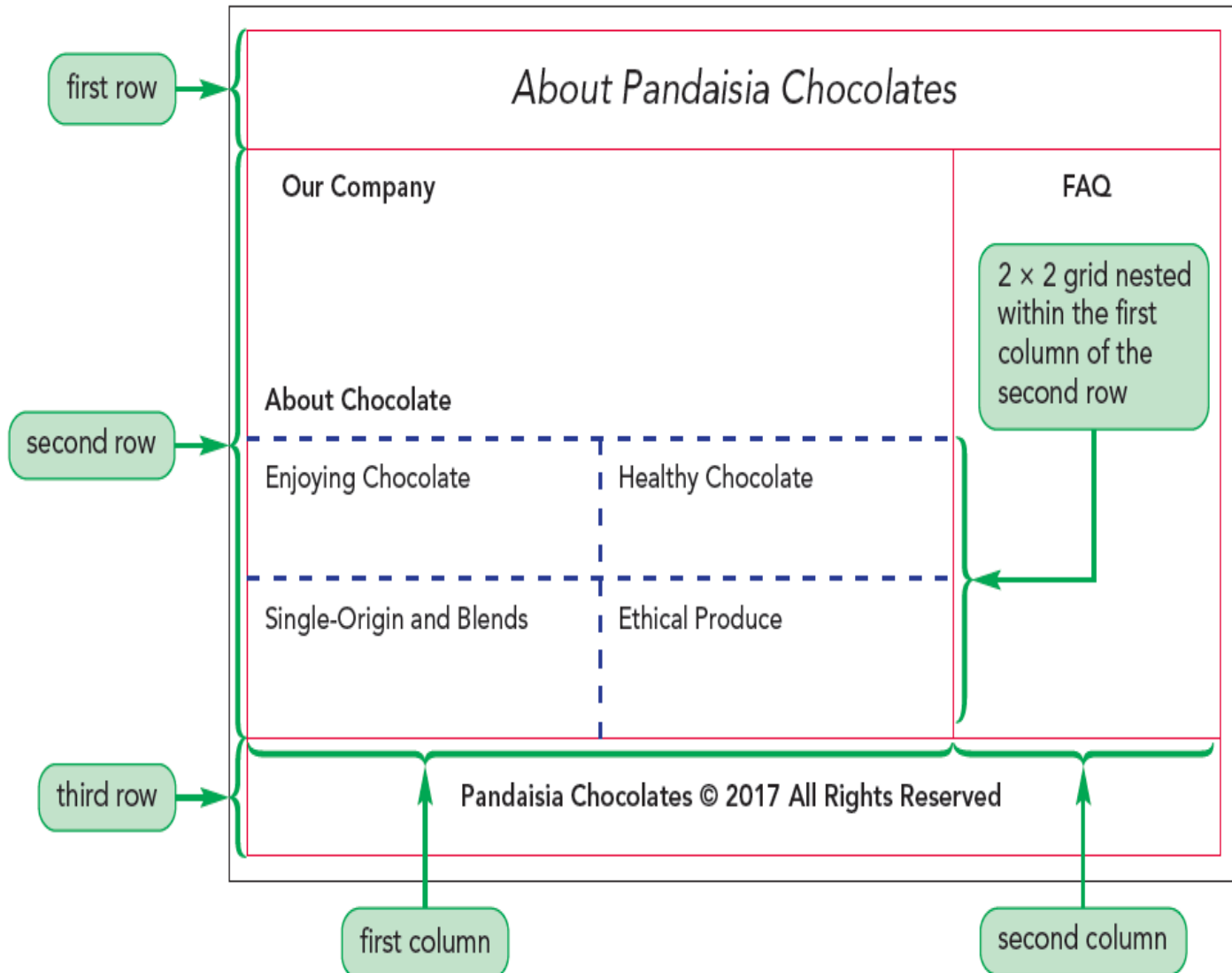
Sample grid layout



© 2016 Cengage Learning

# Setting up a Grid – 3/4

Figure 3-32 Proposed grid layout for the About Pandaisia Chocolates page

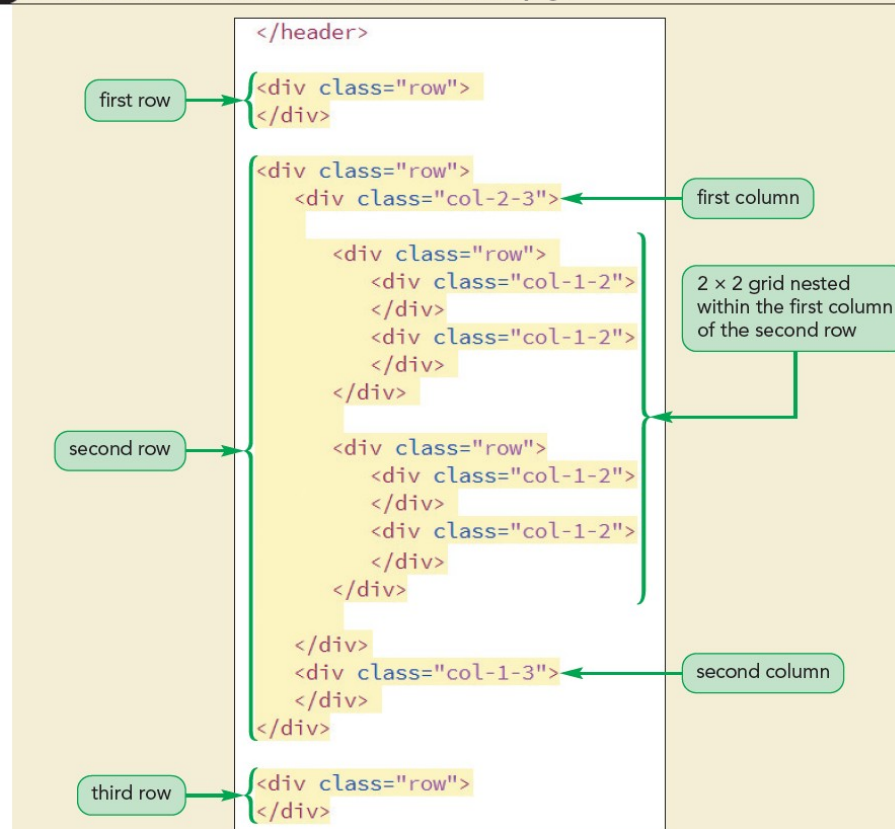


© 2016 Cengage Learning

# Setting up a Grid – 4/4

- The code for the grid layout for the Pandaisia Chocolates website is as follows:

Figure 3-33 div elements in the About Pandaisia Chocolates page



# Designing the Grid Rows

- Grid rows contain floating columns
- Since a grid row starts a new line within a page, it should only be displayed when both margins are clear of previously floated columns

Figure 3-34

Styles for row div elements

```
/* Grid Rows Styles */
```

```
div.row {  
  clear: both;  
}
```

displays the row only when both margins are clear of previously-floated columns

```
div.row::after {  
  clear: both;  
  content: "";  
  display: table;  
}
```

automatically expands the row to cover floating columns

# Designing the Grid Columns – 1/2

- Every grid column needs to be floated within its row
- Grid columns are placed within a `div` element having the general class name

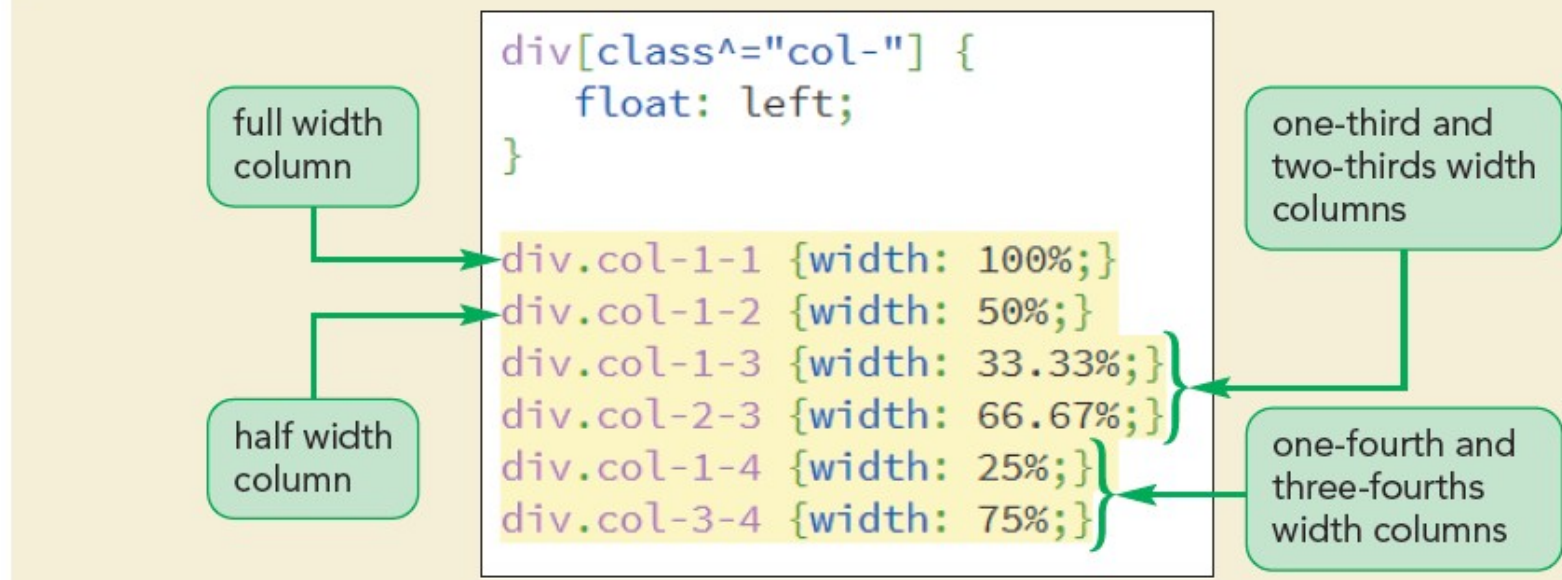
`class="col-numerator-denominator"`

where *numerator-denominator* provides the fractional width of the column

# Designing the Grid Columns - 2/2

Figure 3-36

Setting the column widths



# Adding the Page Content

Figure 3-39 Adding content about chocolate

row heading

content about  
chocolate pasted  
into the first nested  
column

```
<div class="row">
  <h2>About Chocolate</h2>
  <div class="col-1-2">
    <h3>Enjoying Chocolates</h3>
    <p>We believe that the best chocolate is fresh chocolate.
    Preservatives change the flavor and texture of chocolate.
    For the best results, our chocolates should be consumed
    within a few days of purchase. Store them in a
    cool, dark place at a temperature of 60&deg; to
    70&deg; such as a refrigerator or wine cellar.</p>
  </div>
  <div class="col-1-2">
  </div>
</div>

<div class="row">
  <div class="col-1-2">
  </div>
  <div class="col-1-2">
  </div>
  <div class="col-1-2">
  </div>
</div>
</div>
```

# Outlining a Grid – 1/2

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



# Outlining a Grid – 2/2

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

```
outline: width style color;  
{  
    outline: 1px dotted green;  
}
```

# Defining a CSS Grid – 1/2

- To create a grid display without the use of `div` elements, use the following grid-based properties:

```
selector {  
    display: grid;  
    grid-template-rows: track-list;  
    grid-template-columns: track-list;  
}
```

- *grid* – Selected elements in a grid
- *track-list* – Space-separated list of row heights or column widths

# Defining a CSS Grid – 2/2

- **fr unit** – Represents the fraction of available space left on the grid after all other rows or columns have attained their maximum allowable size
- For example, the following style creates four columns with the dimension specified in the style rule:

```
grid-template-columns: 200px 250px  
1fr 2fr;
```

# Assigning Content to Grid Cells – 1/2

---

- Elements in a CSS grid are placed within a **grid cell** at the intersection of a specified row and column
- By default, all of the specified elements are placed in the grid cell located at the intersection of the first row and first column

# Assigning Content to Grid Cells – 2/2

- To place an element in a different cell, use

`grid-row-start: integer;`

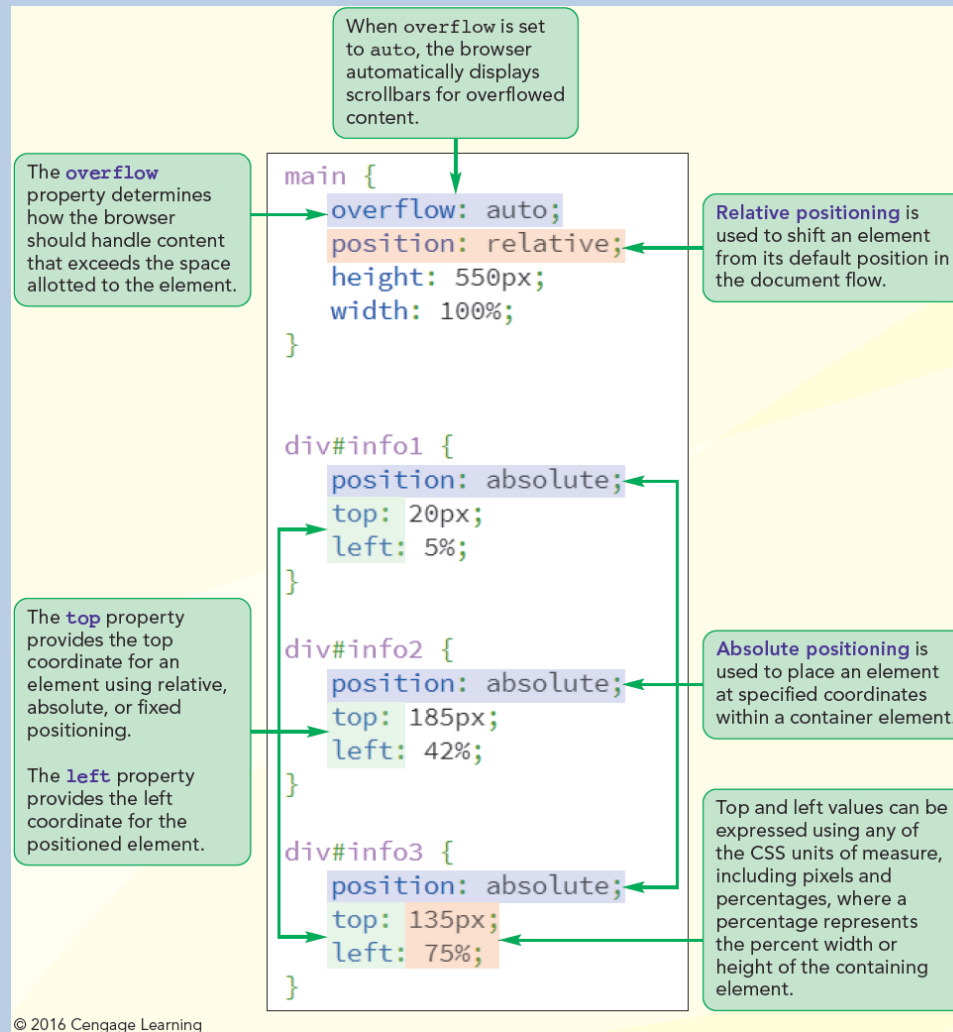
`grid-row-end: integer;`

`grid-column-start: integer;`

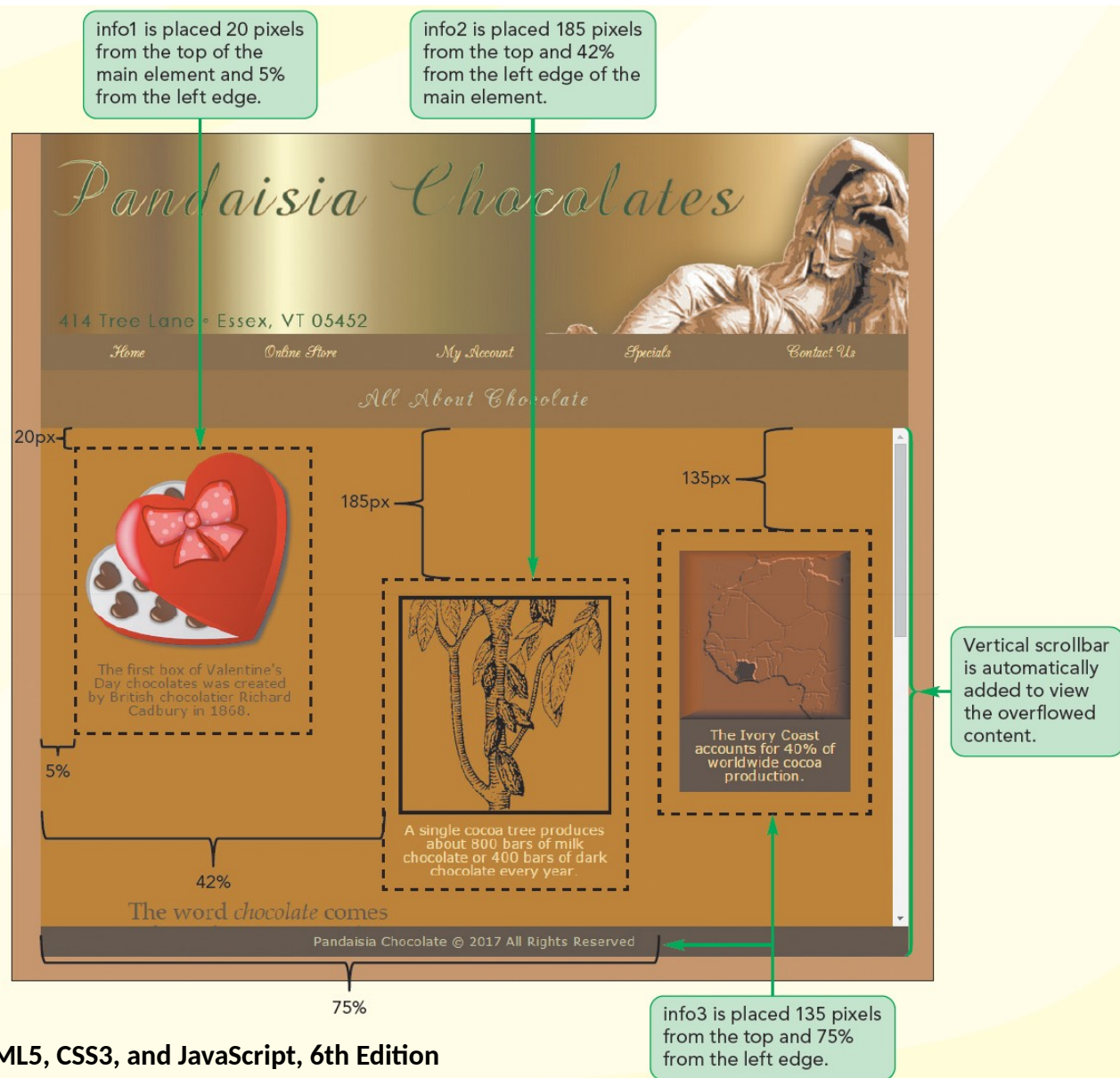
`grid-column-end: integer;`

where *integer* defines the starting and ending row or column that contains the content

# Section 3.3: Layout with Positioning Styles – 1/2



# Layout with Positioning Styles – 2/2



# The CSS positioning Styles – 1/4

- 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



# The CSS Positioning Styles – 2/4

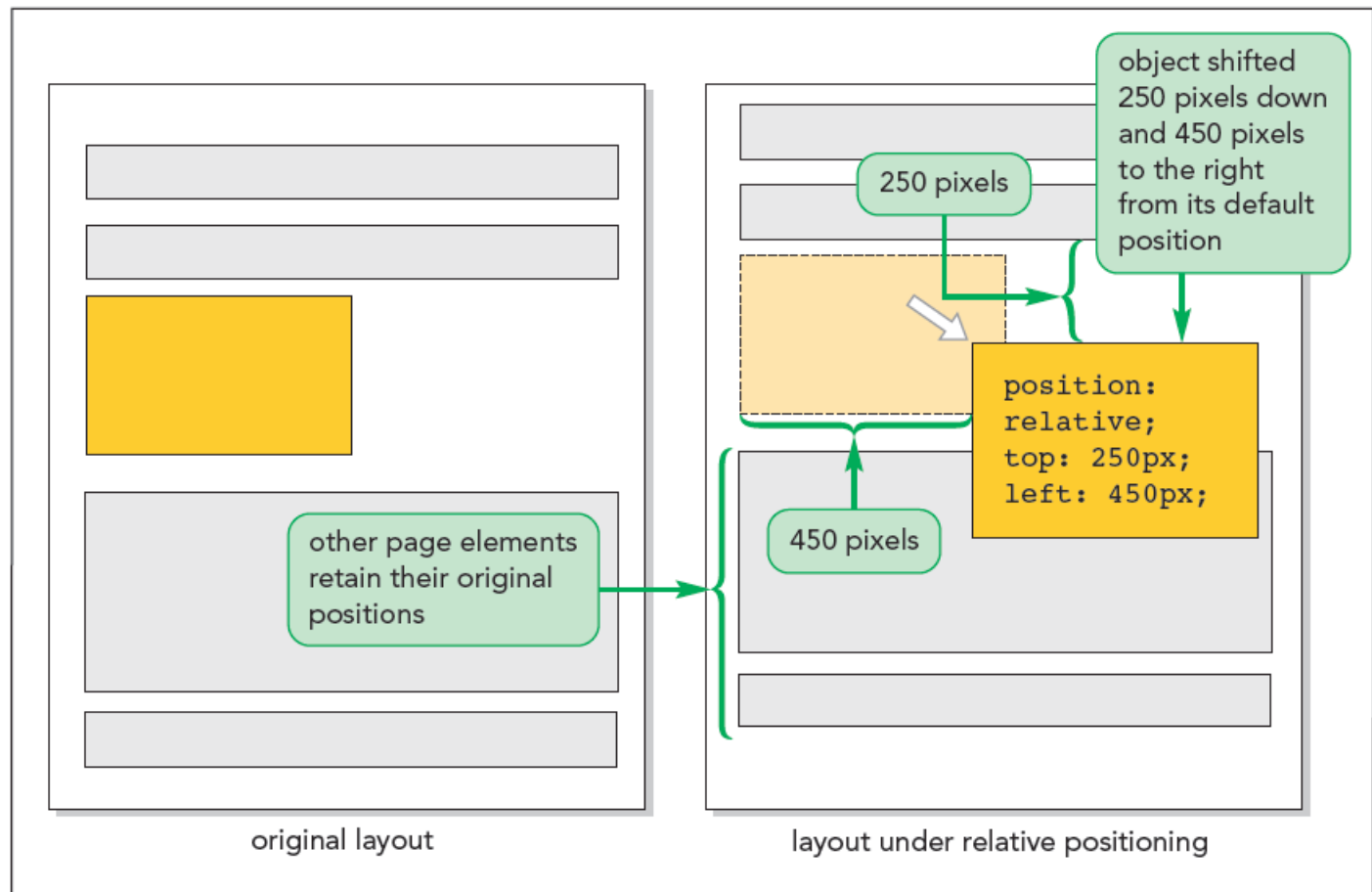
---

- **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

# The CSS Positioning Styles - 3/4

Figure 3-46

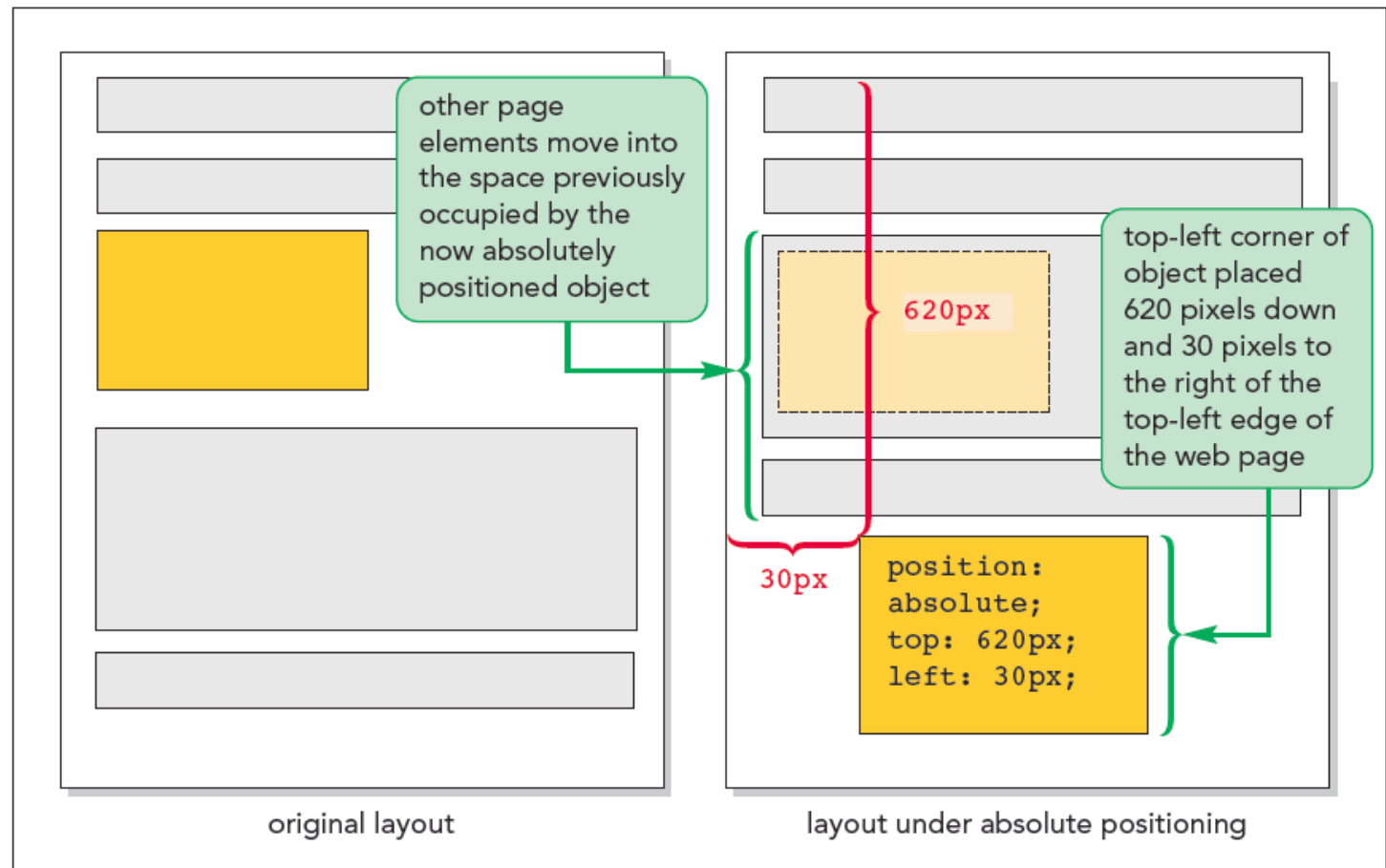
Moving an object using relative positioning



© 2016 Cengage Learning

# The CSS Positioning Styles - 4/4

**Figure 3-47** Moving an object using absolute positioning



© 2016 Cengage Learning

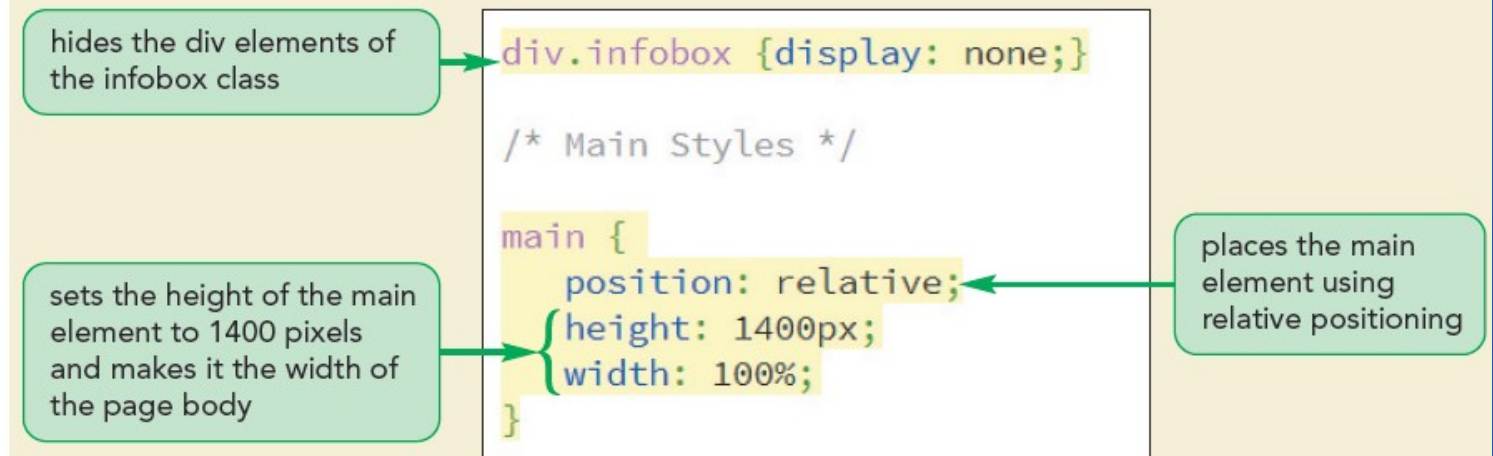
# Fixed and Inherited Positioning

---

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

# Using the Positioning Styles – 1/5

**Figure 3-50** Setting the display styles of the main element



# Using the Positioning Styles - 2/5

**Figure 3-51** Placing the first information box

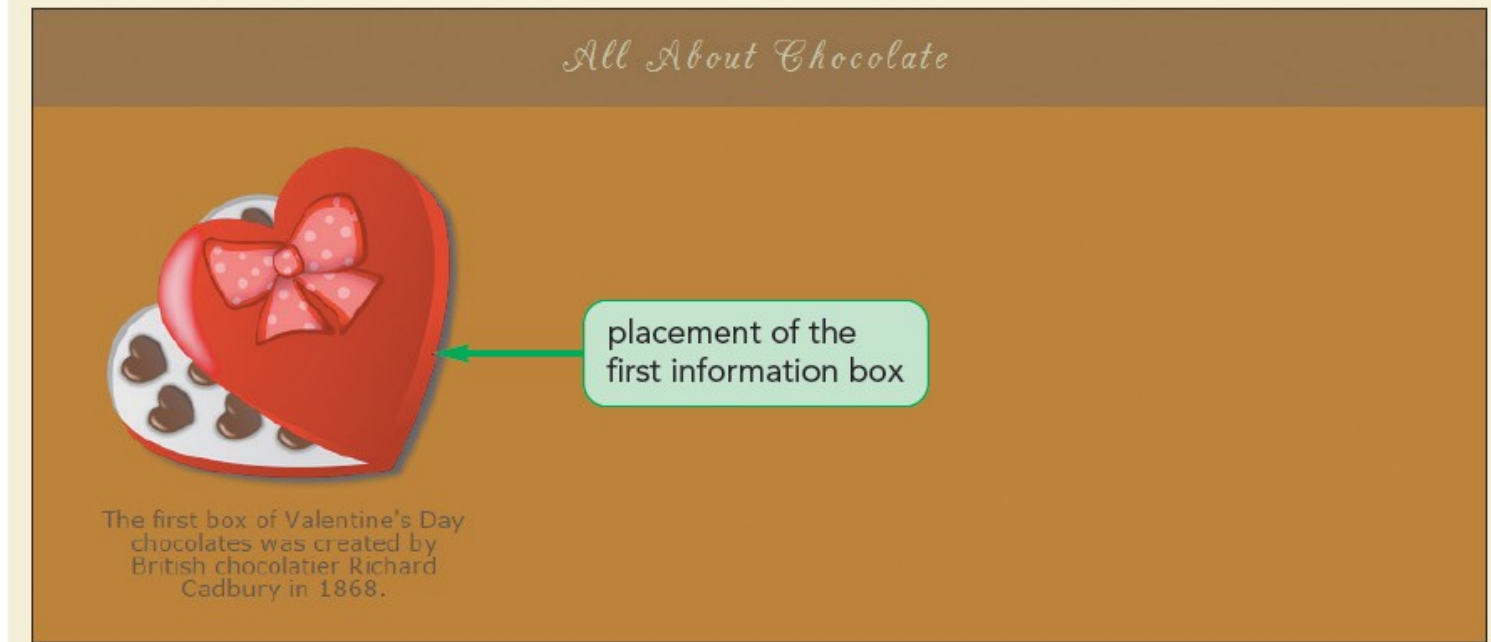
```
/* Infographic Styles */  
  
div.infobox {  
    position: absolute;  
}  
  
/* First Infographic */  
  
div#info1 {  
    display: block;  
    top: 20px;  
    left: 5%;  
}
```

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

# Using the Positioning Styles - 3/5

**Figure 3-52** Appearance of the first information box



# Using the Positioning Styles - 4/5

**Figure 3-53**

Positions of the second and third boxes

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

```
/* Second Infographic */
```

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

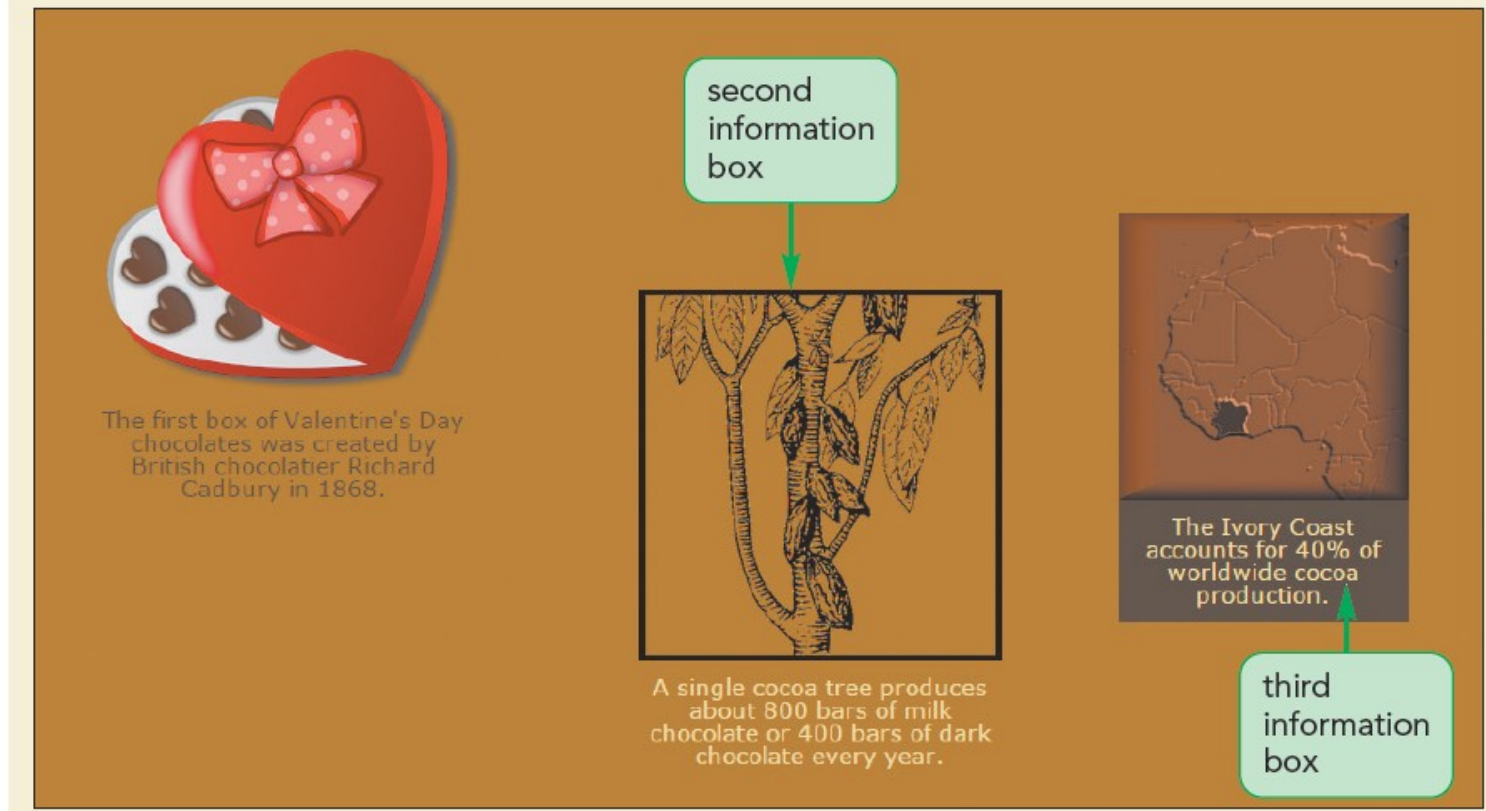
```
/* Third Infographic */
```

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



# Using the Positioning Styles - 5/5

**Figure 3-54** Placement of the first three boxes



# Handling Overflow – 1/4

- `overflow` – 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

# Handling Overflow – 2/4

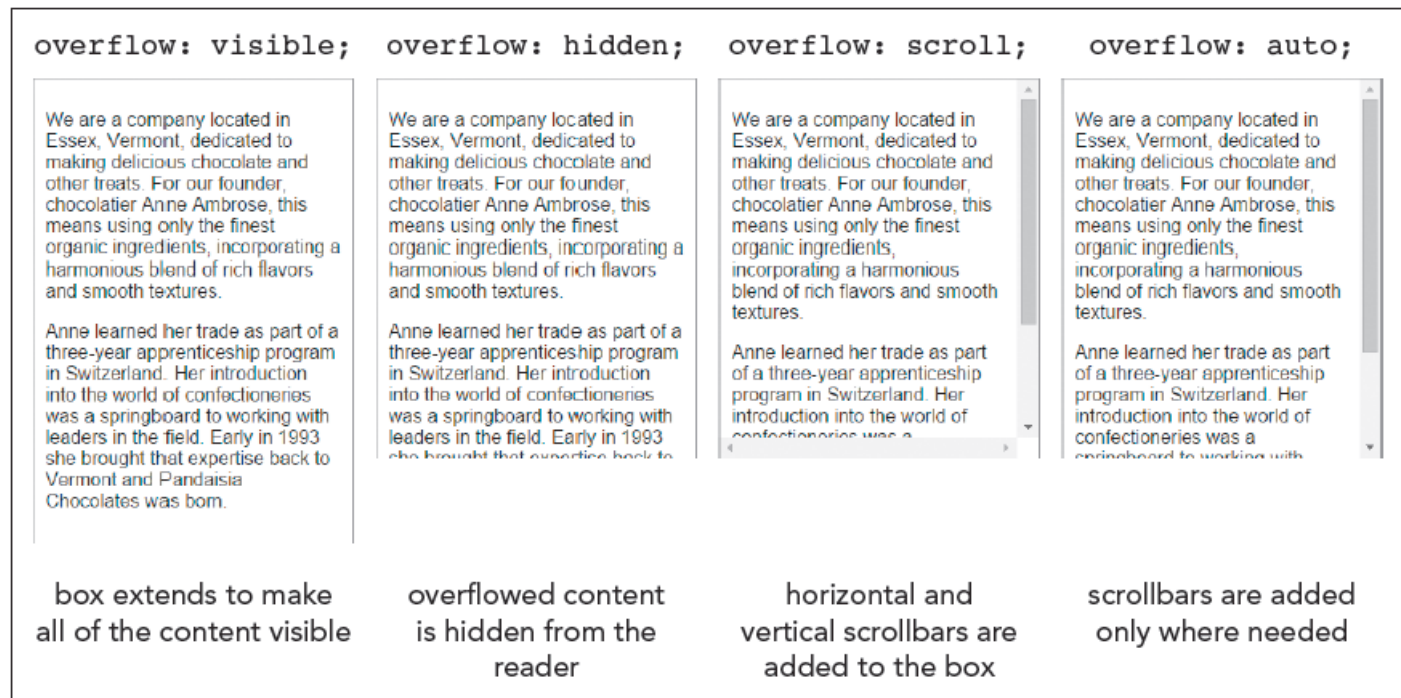
---

- `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

# Handling Overflow – 3/4

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

Figure 3-59 Values of the overflow property



# Handling Overflow – 4/4

**Figure 3-60** Setting the overflow property

displays scrollbars  
if the content  
overflows the  
allotted height

```
/* Main Styles */  
  
main {  
  overflow: auto;  
  position: relative;  
  height: 450px;  
  width: 100%;  
}
```

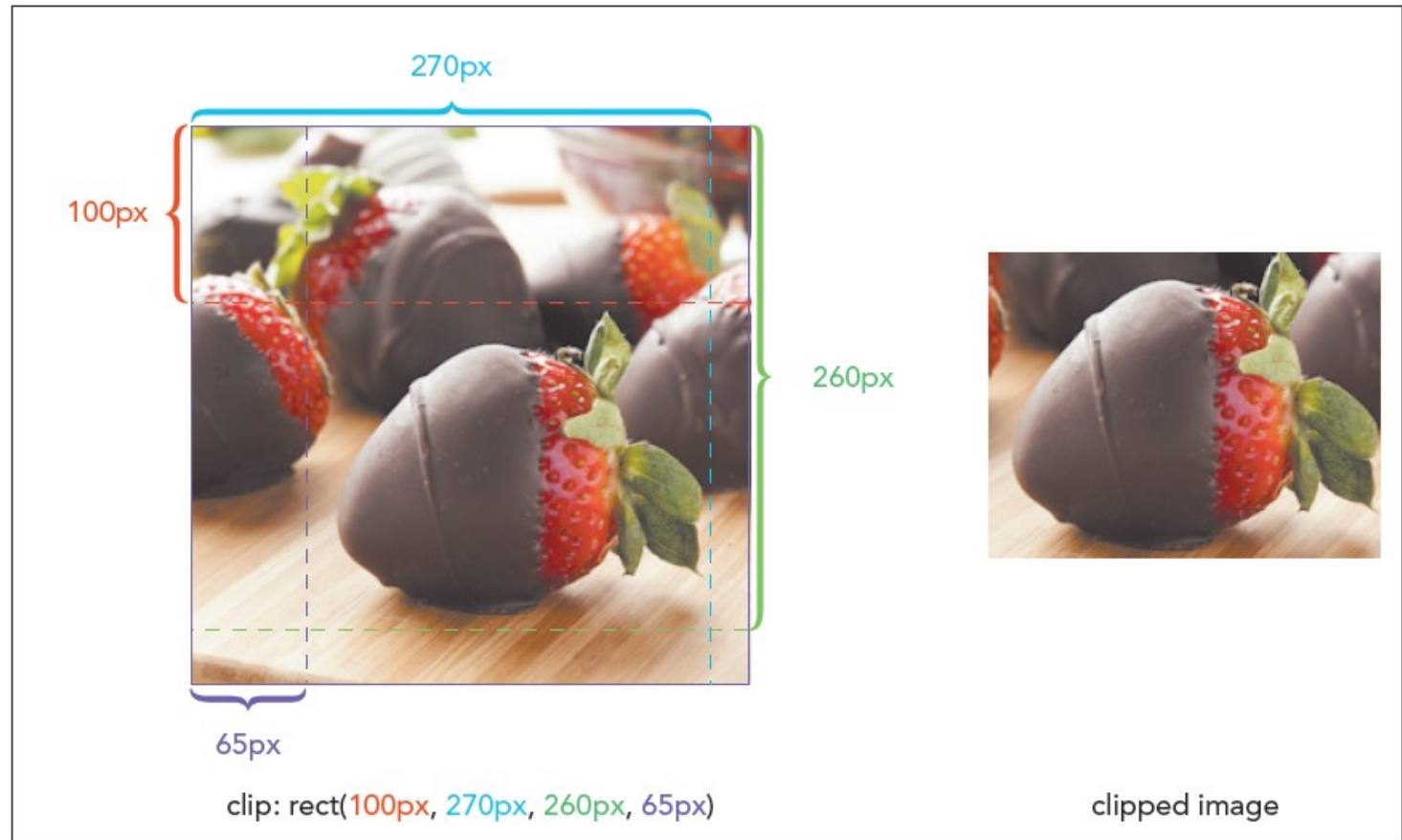
sets the height of  
the infographic  
to 450 pixels

# Clipping an Element – 1/2

- **Clip** – 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 syntax of the `clip` property is  
`clip: rect(top, right, bottom, left);`  
where *top*, *right*, *bottom*, and *left* define the coordinates of the clipping rectangle

# Clipping an Element - 2/2

Figure 3-62 Clipping an image



© Brent Hofacker/Shutterstock.com

# Stacking elements – 1/3

- 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;`

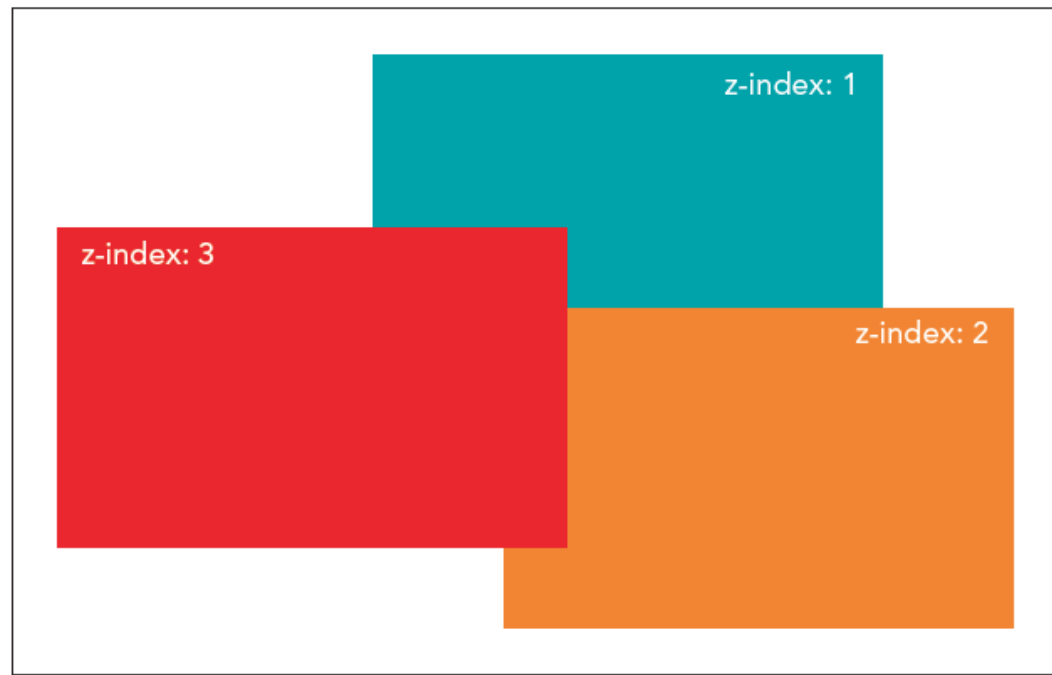
where *value* is a positive or negative integer, or the keyword `auto`



# Stacking elements – 2/3

- The `z-index` property works only for elements that are placed with absolute positioning

**Figure 3-63** Using the `z-index` property to stack elements

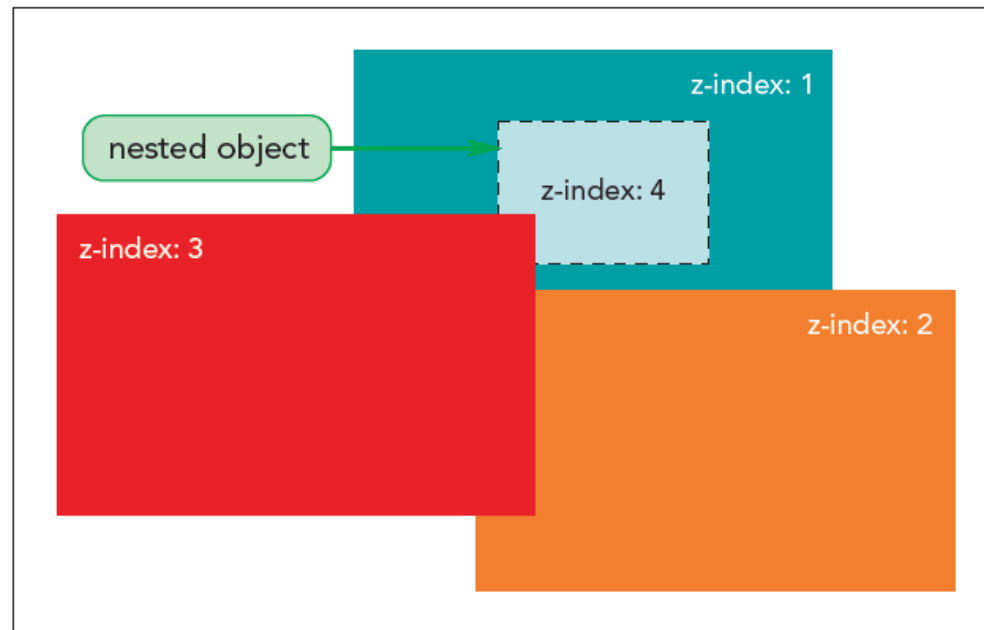


© 2016 Cengage Learning

# Stacking elements – 3/3

- An element's z-index value determines its position relative only to other elements that share a common parent

**Figure 3-64** Stacking nested objects



© 2016 Cengage Learning