Tutorial 05

Designing for the Mobile Web

Objectives

- Create a media query
- Work with the browser viewport
- Apply a responsive design
- Create a pulldown menu with CSS
- Create a flexbox

Objectives (continued)

- Work with flex sizes
- Explore flexbox layouts
- Create a print style sheet
- Work with page sizes
- Add and remove page breaks

Introducing Responsive Design

User Experience	Mobile	Desktop
Page Content	Content should be short and to the point.	Content can be extensive, giving readers the opportunity to explore all facets of the topic.
Page Layout	Content should be laid out within a single column with no horizontal scrolling.	With a wider screen size, content can be more easily laid out in multiple columns.
Hypertext Links	Links need to be easily accessed via a touch interface.	Links can be activated more precisely using a cursor or mouse pointer.
Network Bandwidth	Sites tend to take longer to load over cellular networks and thus overall file size should be kept small.	Sites are quickly accessed over high-speed networks, which can more easily handle large file sizes.
Lighting	Pages need to be easily visible in outdoor lighting through the use of contrasting colors.	Pages are typically viewed in an office setting, allowing a broader color palette.
Device Tools	Mobile sites often need access to devices such as phone dialing, messaging, mapping, and built-in cameras and video.	Sites rarely have need to access desktop devices.

Introducing Responsive Design (continued)

- The three primary components of responsive design theory identified by Ethan Marcotte are:
  - **flexible layout** so that the page layout automatically adjusts to screens of different widths
  - **responsive images** that rescale based on the size of the viewing device
  - **media queries** that determine the properties of the device rendering the page so that appropriate designs can be delivered to specific devices

Introducing Media Queries

- Media queries associate a style sheet or style rule with a specific device or list of device features
- Create a media query within an HTML file, by adding a media attribute to either the `link` or `style` element in the document head:

  `media="devices"`

  where `devices` is a comma-separated list of supported media types associated with a specified style sheet

Introducing Media Queries (continued)

Media Type	Used For
`all`	All output devices (the default)
`braille`	Braille tactile feedback devices
`embossed`	Paged Braille printers
`handheld`	Mobile devices with small screens and limited bandwidth
`print`	Printers
`projection`	Projectors
`screen`	Computer screens
`speech`	Speech and sound synthesizers, and aural browsers
`tty`	Fixed-width devices such as teletype machines and terminals
`tv`	Television-type devices with low resolution, color, and limited scrollability

The @media Rule

- Media queries can be used to associate specific style rules with specific devices using the following:

```
@media devices {
   style rules
}
```

where `devices` are supported media types and `style rules` are the style rules associated with those devices

Media Queries and Device Features

- To target a device based on its features, add the feature and its value to the media attribute using the syntax:

  `media="devices and|or (feature:value)"`

  where `feature` is the name of a media feature and `value` is the feature's value
- The `and` and `or` keywords are used to create media queries that involve different devices or features, or combinations of both

Media Queries and Device Features (continued)

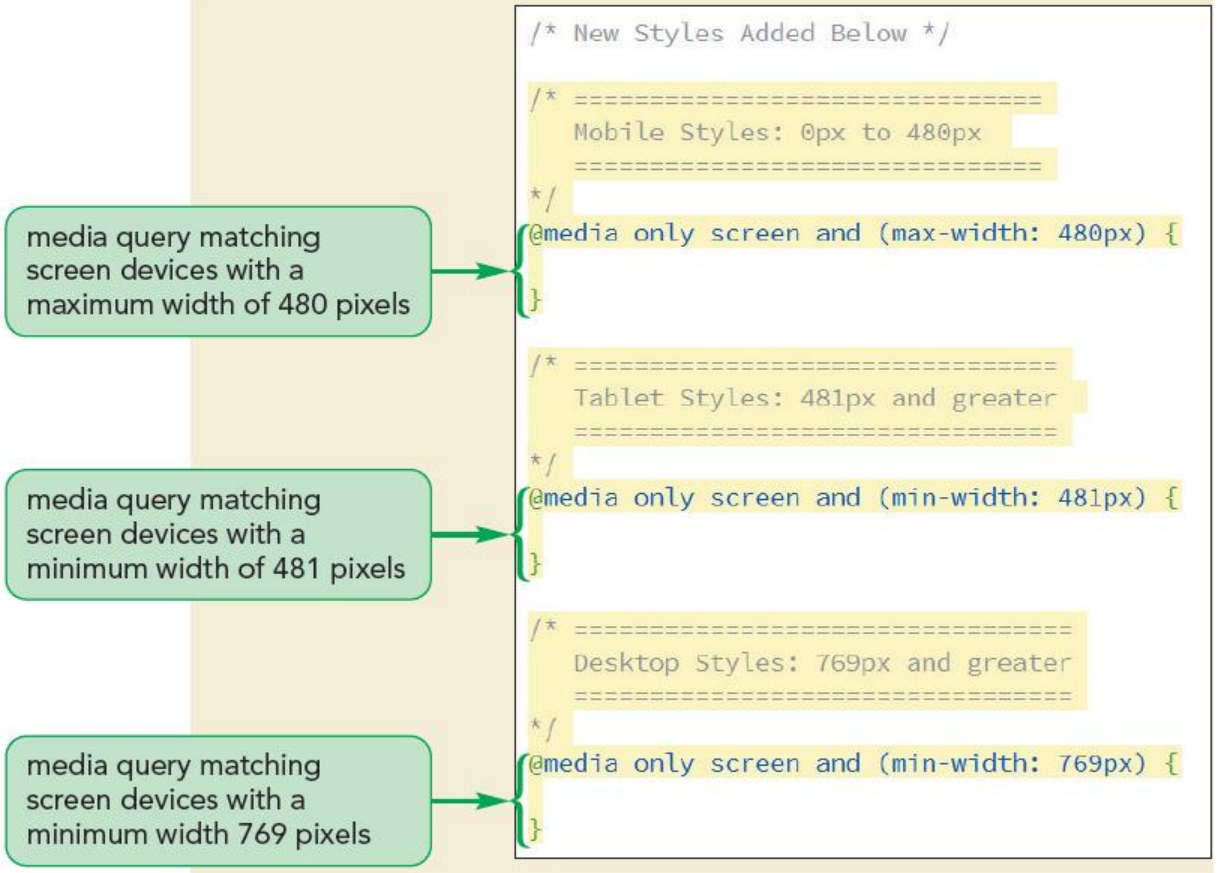
Feature	Description
`aspect-ratio`	The ratio of the width of the display area to its height
`color`	The number of bits per color component of the output device; if the device does not support color, the value is 0
`color-index`	The number of colors supported by the output device
`device-aspect-ratio`	The ratio of the device-width value to the device-height value
`device-height`	The height of the rendering surface of the output device
`device-width`	The width of the rendering surface of the output device
`height`	The height of the display area of the output device
`monochrome`	The number of bits per pixel in the device's monochrome frame buffer
`orientation`	The general description of the aspect ratio: equal to `portrait` when the height of the display area is greater than the width; equal to `landscape` otherwise
`resolution`	The resolution of the output device in pixels, expressed in either dpi (dots per inch) or dpcm (dots per centimeter)
`width`	The width of the display area of the output device

Applying Media Queries to a Style Sheet

- The **mobile first** principle is one in which the overall page design starts with base styles that apply to all devices followed by style rules specific to mobile devices
- Tablet styles are applied when the screen width is 481 pixels or greater
- Desktop styles build upon the tablet styles when the screen width exceeds 768 pixels
- As the screen width increases, more features found in smaller devices are added or replaced

Applying Media Queries to a Style Sheet (continued)

Figure 5-6 Creating media queries for different screen widths

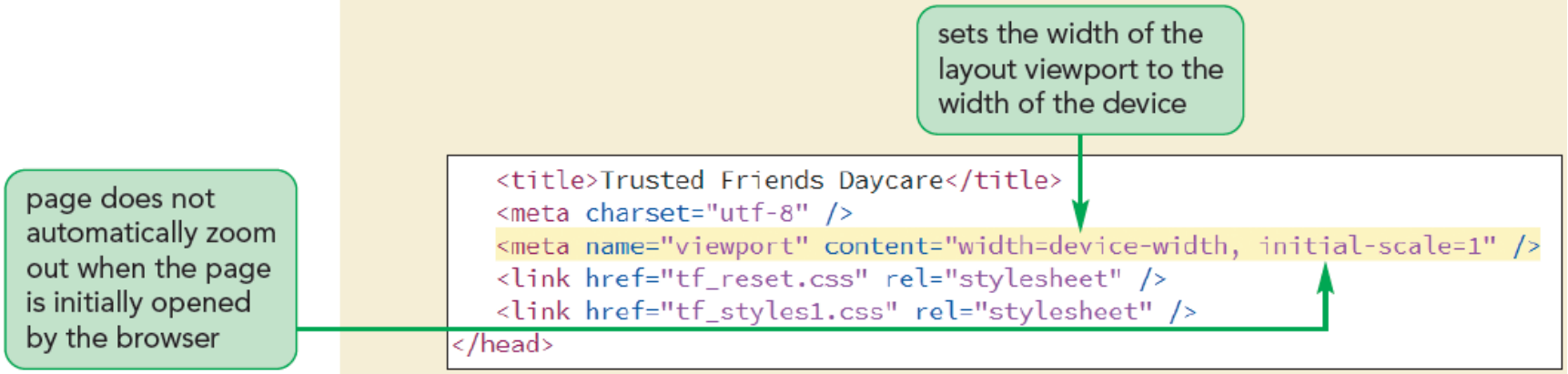


Exploring Viewports and Device Width

- Web pages are viewed within a window called the viewport
- Mobile devices have two types of viewports:
  - **Visual viewport** – displays the web page content that fits within a mobile screen
  - **Layout viewport** – contains the entire content of the page, some of which may be hidden from the user

Exploring Viewports and Device Width (continued)

Figure 5-8 Setting the properties of the viewport

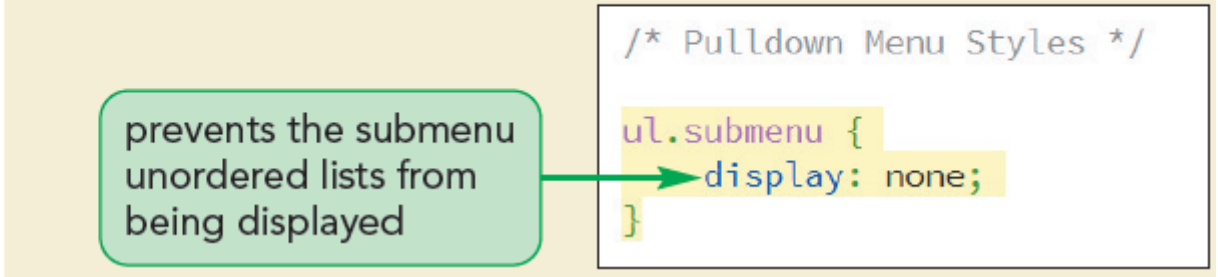


Creating a Mobile Design

- Design principles
  - Have the most important information up-front and easily accessible
    - Means the home page on a mobile device needs to be free of unnecessary clutter
  - Limit the choices offered to users
    - Means that ideally, there should only be a few navigation links on the screen at any one time

Creating a Pulldown Menu with CSS

Figure 5-11 Hiding the navigation list submenus



Creating a Pulldown Menu with CSS (continued 1)

Figure 5-12 Navigation list with hidden submenus

Creating a Pulldown Menu with CSS (continued 2)

- The following selector can be used to select the submenu that is immediately preceded by a hovered submenu title:

```
a.submenuTitle:hover+ul.submenu
```

- In order to keep the submenu visible as the pointer moves away from the title and hovers over the now-visible submenu, use the following:

```
a.submenuTitle:hover+ul.submenu,
ul.submenu:hover
```

Creating a Pulldown Menu with CSS (continued 3)

- To make a submenu visible, change its `display` property back to block, using the following style rule:

```
a.submenuTitle:hover+ul.submenu,
ul.submenu:hover {
   display: block;
}
```

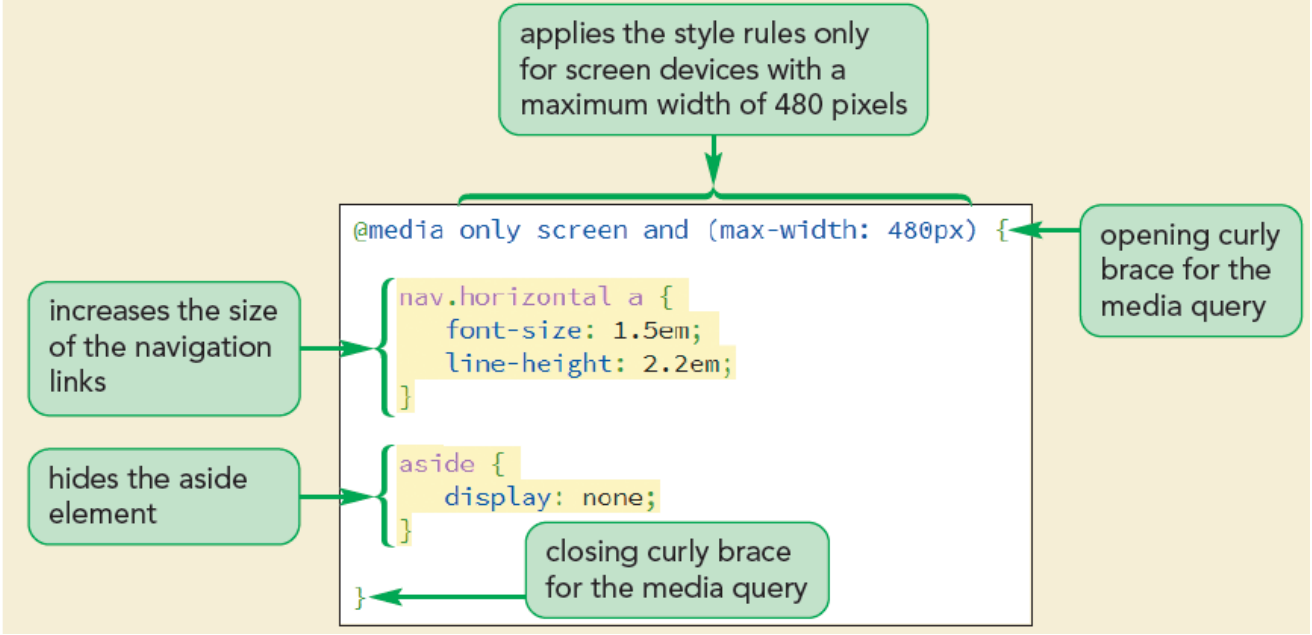
- Do not use only the `ul.submenu:hover` selector
  - Because you cannot hover over the submenu until it is visible and it won't be visible until you first hover over the submenu title

Testing Your Mobile Website

Mobile Emulator	Description
Android SDK	Software development kit for Android developers(*developer.android.com/sdk*)
iOS SDK	Software development kit for iPhone, iPad, and other iOS devices(*developer.apple.com*)
Mobile Phone Emulator	Online emulation for a variety of mobile devices(*www.mobilephoneemulator.com*)
Mobile Test Me	Online emulation for a variety of mobile devices (*mobiletest.me*)
Opera Mobile SDK	Developer tools for the Opera Mobile browser(*www.opera.com/developer*)

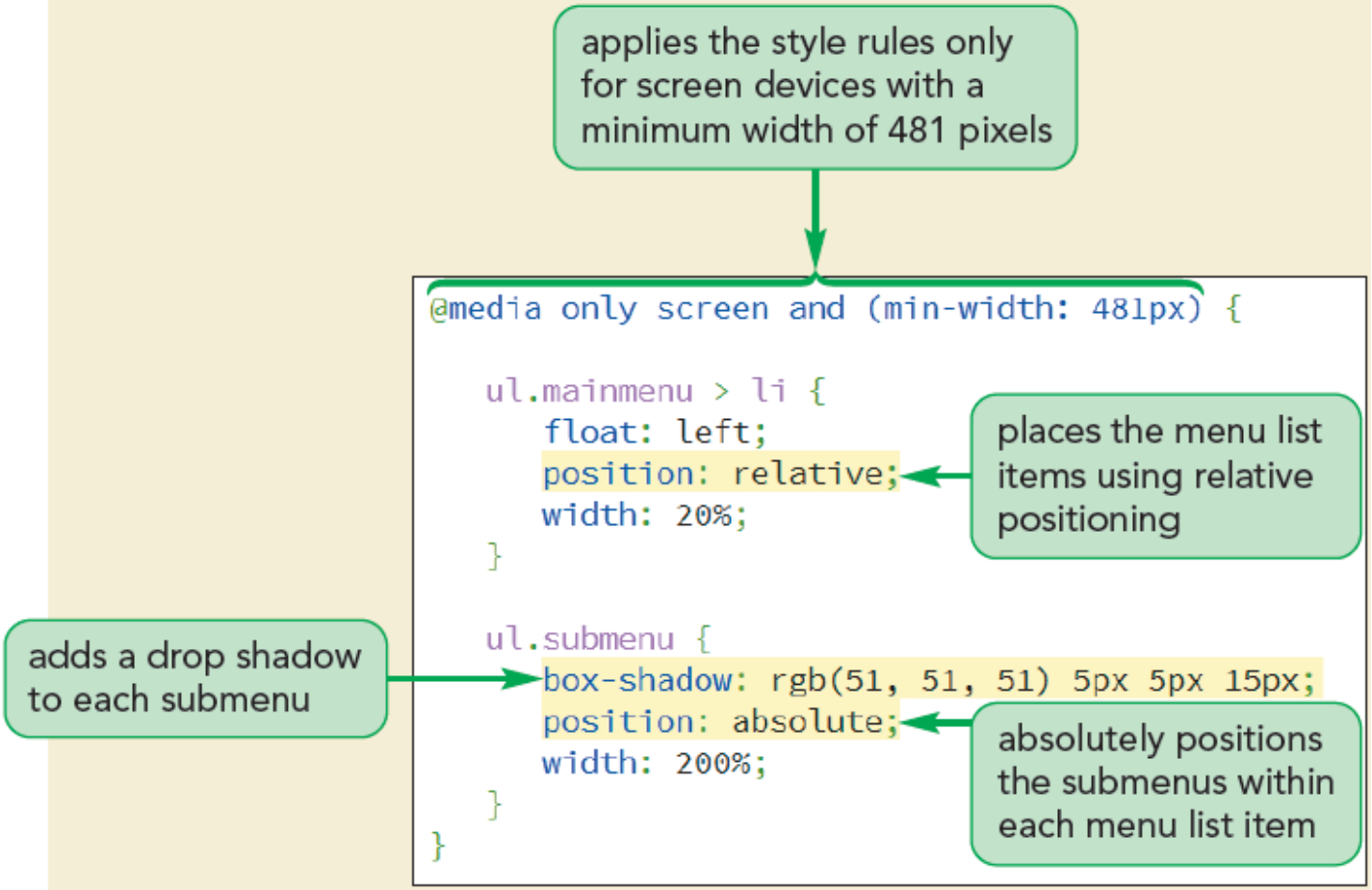
Testing Your Mobile Website (continued)

Figure 5-17 Hiding the aside element for mobile devices



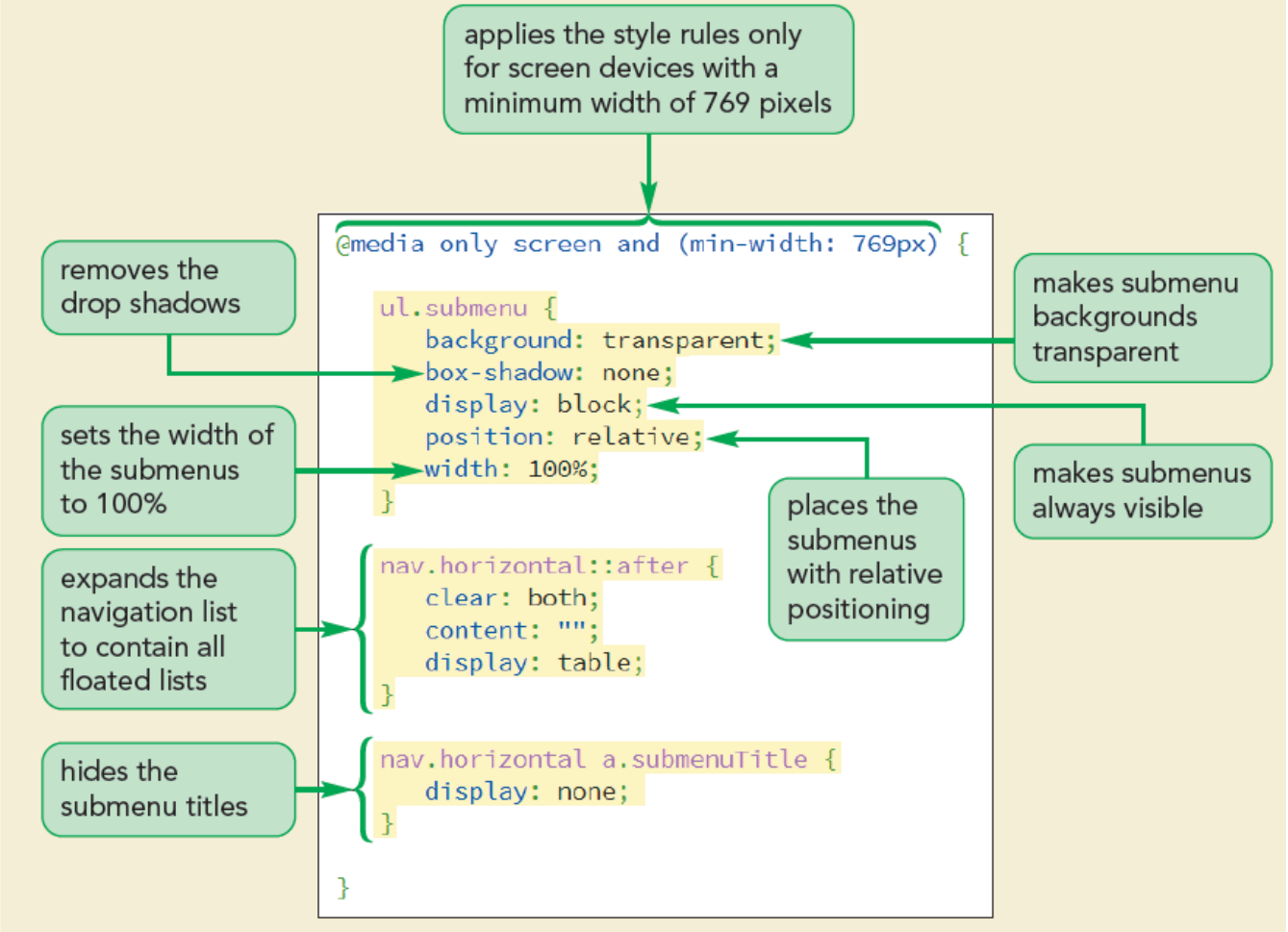
Creating a Tablet Design

Figure 5-21 Placing the pulldown menus with absolute positioning



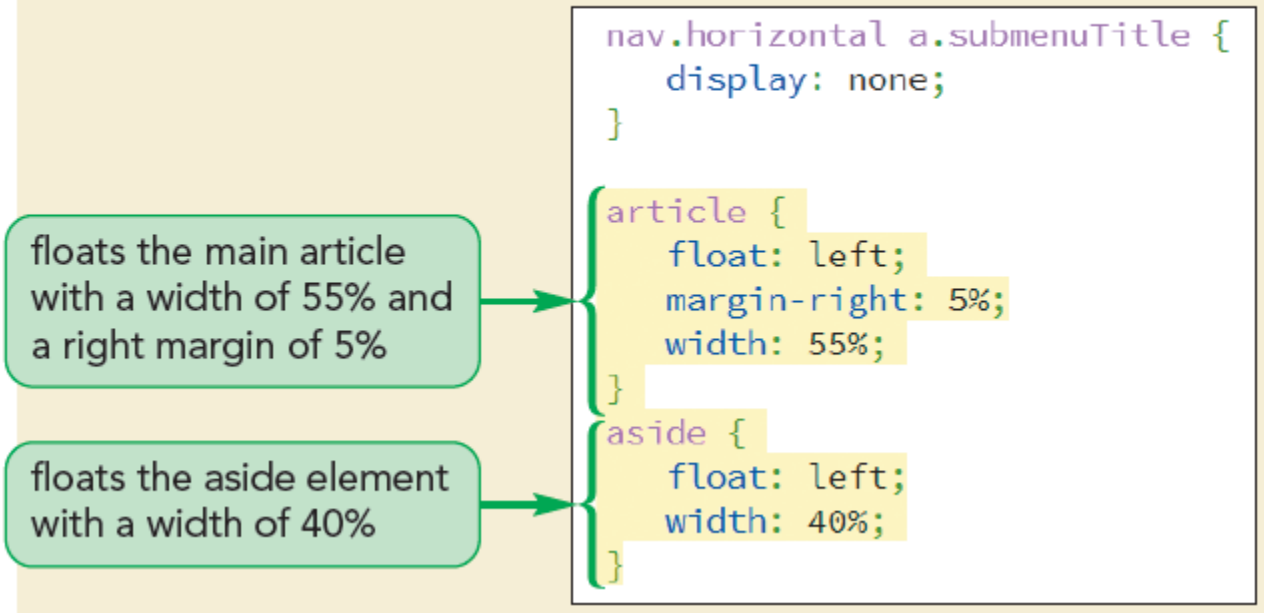
Creating a Desktop Design

Figure 5-23 Adding design styles for the browser background and page body



Creating a Desktop Design (continued)

Figure 5-24 Styles for the article and aside elements

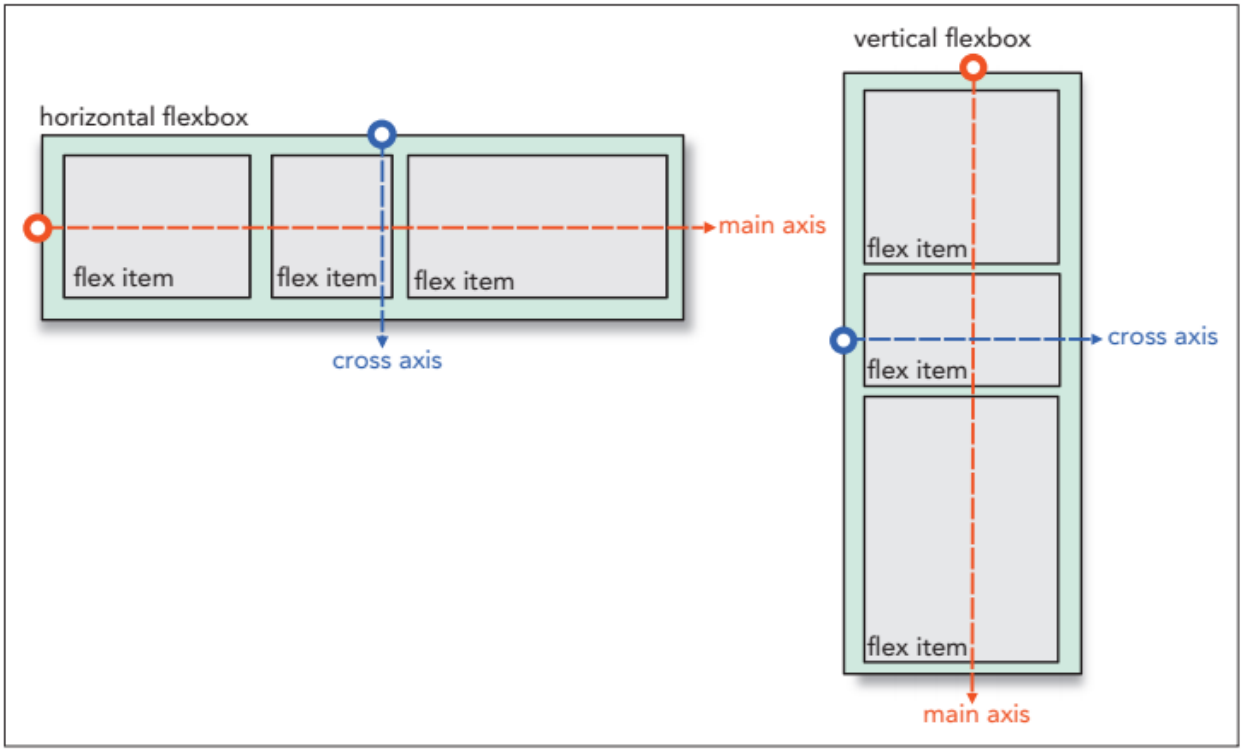


Defining a Flexible Box

- A flexible box or flexbox is a box containing items whose sizes can shrink or grow to match the boundaries of the box
- Items within a flexbox are laid out along a main axis
- The **main axis** can point in either the horizontal or vertical direction
- **Cross axis** is perpendicular to the main axis and is used to define the height or width of each item

Defining a Flexible Box (continued 1)

Figure 5–26 Horizontal and vertical flexboxes



Defining a Flexible Box (continued 2)

- To define an element as a flexbox, apply either of the following display styles:

```
display: flex;
```

or

```
display: inline-flex;
```

where a value of `flex` starts the flexbox on a new line and a value of `inline-flex` keeps the flexbox in-line with its surrounding content

Cross-Browser Flexboxes

- The complete list of browser extensions that define a flexbox is entered as:

```
display: -webkit-box;
display: -moz-box;
display: -ms-flexbox;
display: -webkit-flex:
display: flex;
```

Setting the Flexbox Flow

- By default, flexbox items are arranged horizontally starting from the left and moving to the right
- The orientation of a flexbox can be changed using:

```
flex-direction: direction;
```

where `direction` is `row` (the default), `column`, `row-reverse`, or `column-reverse`

Setting the Flexbox Flow (continued 1)

- The `row` option in a `flex-direction` lays out the flex items from left to right
- The `column` option in a `flex-direction` creates a vertical layout starting from the top and moving downward
- The `row-reverse` and `column-reverse` options in a `flex-direction` lay out the items bottom-to-top and right-to-left respectively

Setting the Flexbox Flow (continued 2)

- Flex items try to fit within a single line, either horizontally or vertically
- Flex items can wrap to a new line using the following property:

```
flex-wrap: type;
```

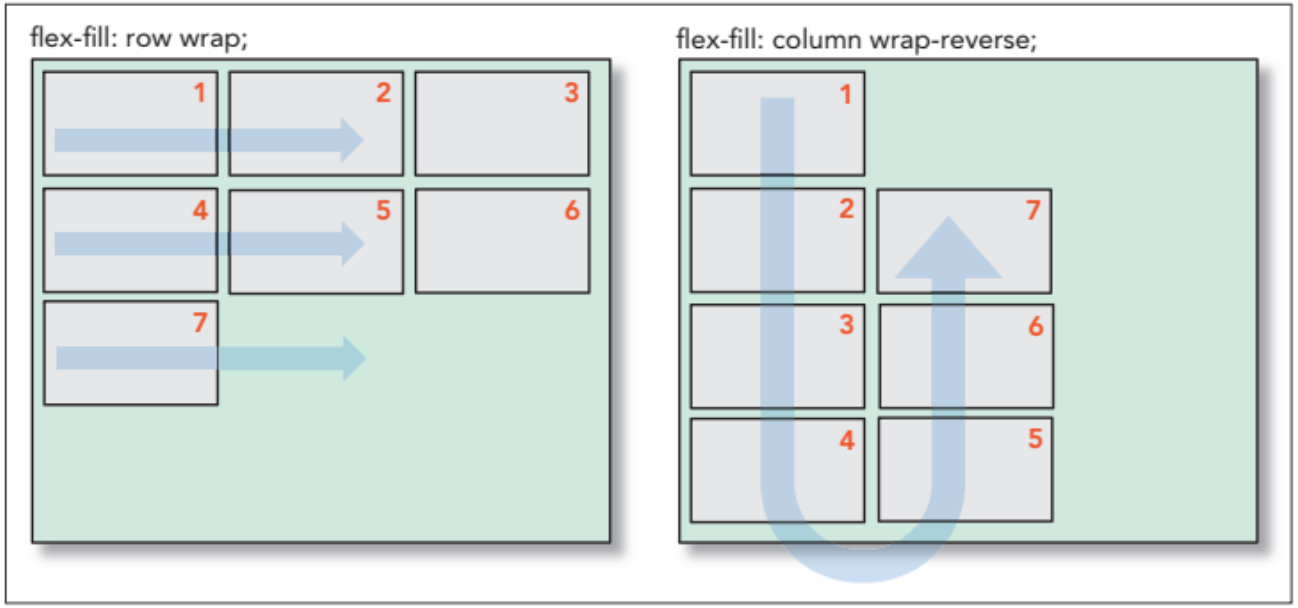
Setting the Flexbox Flow (continued 3)

where `type` is either:

- `nowrap` (default)
- `wrap` to wrap the flex items to a new line
- `wrap-reverse` to wrap flex items to a new line starting in the opposite direction from the current line

Setting the Flexbox Flow (continued 4)

Figure 5–27 Flexbox layouts



Setting the Flex Basis

- The flex items are determined by three properties:
  - base size
  - growth value
  - shrink value
- The basis size defines the initial size of the item before the browser attempts to fit it to the flexbox

Setting the Flex Basis (continued)

- The basis size is set using the following:

  ```
  flex-basis: size;
  ```

  where `size` is one of the CSS units of measurement, which sets the initial size of the flex item based on its content or the value of its width or height property
- For example:

  ```
  aside {flex-basis: 200px;}
  ```

Defining the Flex Growth

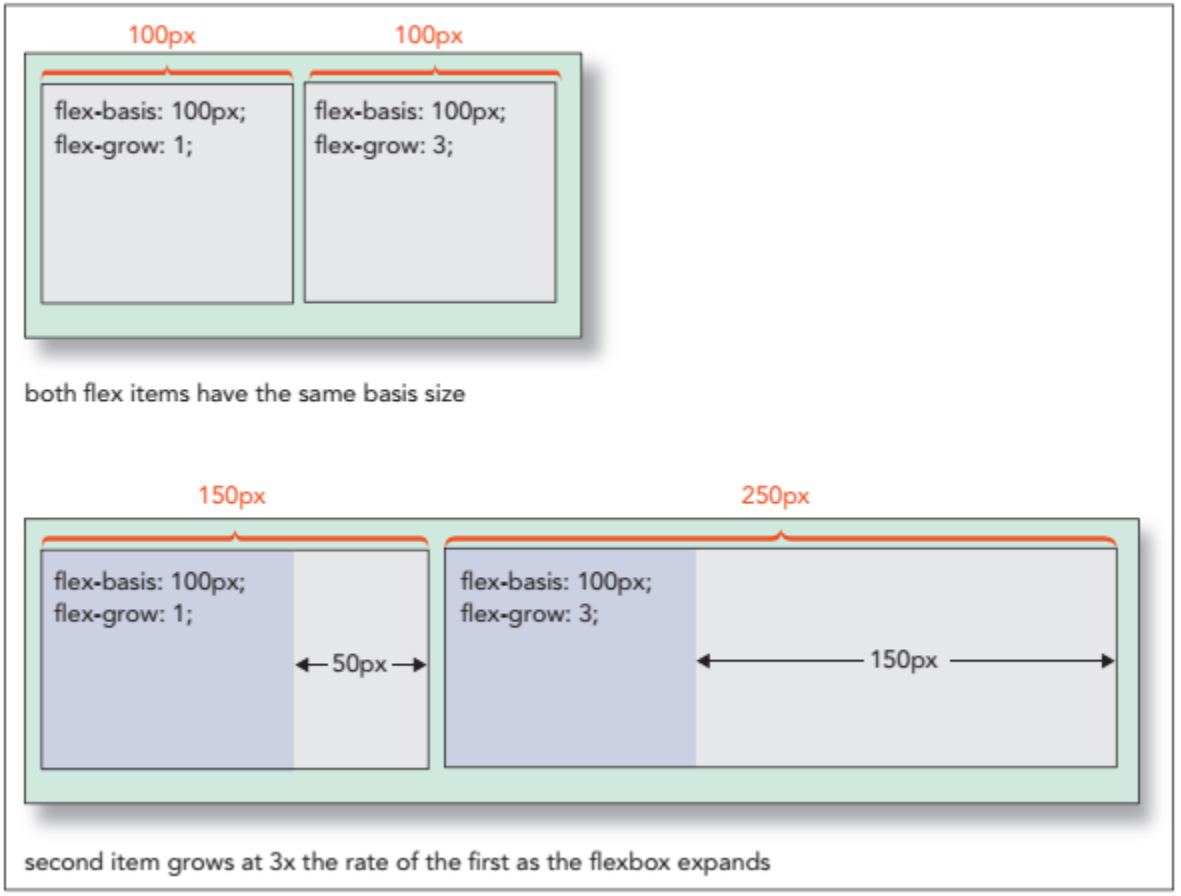
- The rate at which a flex item grows from its basis size is determined by the `flex-grow` property

  ```
  flex-grow: value;
  ```

  where `value` is a non-negative value that expresses the growth of the flex item relative to the growth of other items in the flexbox
- The default `flex-grow` value is 0, which is equivalent to the flex item remaining at its basis size

Defining the Flex Growth (continued 1)

Figure 5–29 Growing flex items beyond their basis size



Defining the Flex Growth (continued 2)

- The following style rule creates a layout for navigation list in which each list item is assigned an equal size and grows at the same rate

```
nav ul {
    display: flex;
}
nav ul li {
    flex-basis: 0px;
    flex-grow: 1;
}
```

Defining the Shrink Rate

- When the flexbox size falls below the total space allotted to its flex items:
  - Two possibilities occur depending on whether the flexbox is defined to wrap its contents to a new line
- First, if the `flexbox-wrap` property is set to `wrap`, one or more of the flex items will be shifted to a new line and expanded to fill in the available space on that line

Defining the Shrink Rate (continued 1)

Figure 5–30 Shrinking flex items smaller than their basis size

Defining the Shrink Rate (continued 2)

- Second, if the flexbox does not wrap to a new line as it is resized, then the flex items will continue to shrink, still sharing the same row or column
- The rate at which flexboxes shrink below their basis size is given by the following property:

```
flex-shrink: value;
```

Defining the Shrink Rate (continued 3)

where `value` is a non-negative value that expresses the shrink rate of the flex item relative to the shrinkage of the other items in the flexbox

- The default `flex-shrink` value is 1
- If the `flex-shrink` value is set to 0, then the flex item will not shrink below its basis

The `flex` Property

- The syntax for the `flex` property is:

  ```
  flex: grow shrink basis;
  ```

  where `grow` defines the growth of the flex item, `shrink` provides its shrink rate, and `basis` sets the item's initial size
- The default flex value is:

  ```
  flex: 0 1 auto;
  ```

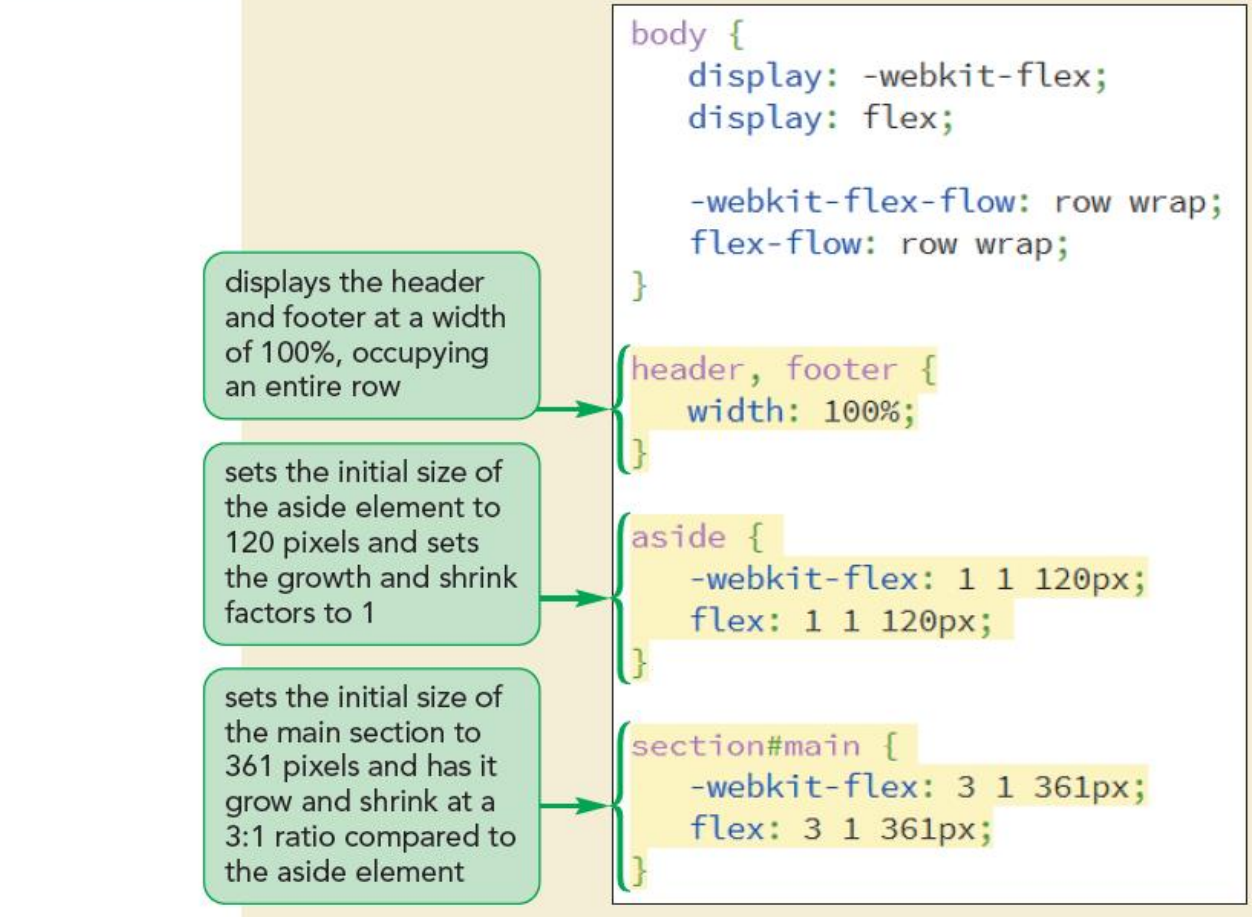
  which automatically sets the size of the flex item to match its content or the value of its `width` and `height` property

The `flex` Property (continued)

- The `flex` property supports the following keywords:
  - `auto` – Use to automatically resize the item from its default size (equivalent to `flex: 1 1 auto;`)
  - `initial` – The default value (equivalent to `flex: 0 1 auto;`)
  - `none` – Use to create an inflexible item that will not grow or shrink (equivalent to `flex: 0 0 auto;`)
  - `inherit` – Use to inherit the flex values of its parent element

Applying a Flexbox Layout

Figure 5-32 Set the flex properties of the flex items in the page body



Applying a Flexbox Layout (continued)

Figure 5–33 Flex layout under different screen widths

Reordering Page Content with Flexboxes

- The flexbox model allows to place the flex items in any order using the following order property:

  ```
  order: value;
  ```

  where `value` is an integer where items with smaller `order` values are placed before items with larger `order` values

Reordering Page Content with Flexboxes (continued)

Figure 5-37 Setting the order of a flex item

places the aside element before the body footer

places the body footer at the end of the flexbox

```
/* ==========================
   Mobile Styles: 0 to 480px
   ========================== */

@media only screen and (max-width: 480px) {

   aside {
      -webkit-order: 99;
      order: 99;
   }

   footer {
      -webkit-order: 100;
      order: 100;
   }

}
```

Aligning Items along the Main Axis

- By default, flex items are laid down at the start of the main axis
- To specify a different placement, apply the following `justify-content` property

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

  where `placement` is one of the following keywords:

  - `flex-start` – Items are positioned at the start of the main axis (the default)

Aligning Items along the Main Axis (continued)

`flex-end` – Items are positioned at the end of the main axis

`center` – Items are centered along the main axis

`space-between` – Items are distributed evenly with the first and last items aligned with the start and end of the main axis

`space-around` – Items are distributed evenly along the main axis with equal space between them and the ends of the flexbox

Aligning Flex Lines

- The `align-content` property is similar to the `justify-content` property except that it arranges multiple lines of content along the flexbox's cross axis
- The syntax of the align-content property is:

  ```
  align-content: value;
  ```

  where `value` is one of the following keywords:

  `flex-start` - Lines are positioned at the start of the cross axis

Aligning Flex Lines (continued 1)

`flex-end` - Lines are positioned at the end of the cross axis

`stretch` - Lines are stretched to fill up the cross axis (the default)

`center` - Lines are centered along the cross axis

`space-between` - Lines are distributed evenly with the first and last lines aligned with the start and end of the cross axis

Aligning Flex Lines (continued 2)

`space-around` - Lines are distributed evenly along the cross axis with equal space between them and the ends of the cross axis

Aligning Items along the Cross Axis

- The align-items property aligns each flex item about the cross axis
- The syntax is:

  `align-items: value;`

  where `value` is one of the following keywords:

  `flex-start` – Items are positioned at the start of the cross axis

  `flex-end` – Items are positioned at the end of the cross axis

Aligning Items along the Cross Axis (continued 1)

- `center` – Items are centered along the cross axis
- `stretch` – Items are stretched to fill up the cross axis (the default)
- `baseline` – Items are positioned so that the baselines of their content align

Aligning Items along the Cross Axis (continued 2)

- The `align-items` property is only impactful when there is a single line of flex items
- The `align-content` property is used to layout the flexbox content for multiple lines of flex items

Aligning Items along the Cross Axis (continued 3)

- To align a single item out of a line of flex items, use the following `align-self` property:

  ```
  align-self: value;
  ```

  where `value` is one of the alignment choices supported by the `align-self` property

Creating a Navicon Menu

- **Navicon** – It is used to indicate the presence of hidden navigation menus in mobile websites
- The navicon is a symbol represented as three horizontal lines
- When a user hovers or touches the navicon, the navigation menu is revealed

Creating a Navicon Menu (continued)

Figure 5-41 Inserting the navicon

```
<nav class="horizontal">
   <a id="navicon" href="#"><img src="tf_navicon.png" alt="" /></a>
   <ul>
      <li><a href="tf_home.html">Home</a></li>
      <li><a href="#">Infants</a></li>
      <li><a href="#">Toddlers</a></li>
      <li><a href="#" id="currentPage">Pre-K</a></li>
      <li><a href="#">After School</a></li>
   </ul>
</nav>
```

navicon image

Designing for Printed Media

- A **print style sheet** formats the printed version of a web document
- Browsers support their own internal style sheet to format the print versions of the web pages they encounter
  - Their default styles might not always result in the best printouts

Applying a Media Query for Printed Output

- To apply a print style sheet, the `media` attribute is used in the `link` elements to target style sheets to either screen devices or print devices

Applying a Media Query for Printed Output (continued)

Figure 5-45 Style sheets for different devices

Working with the @page Rule

- Every printed page in CSS is defined as a **page box**
- A page box is composed of two areas:
  - **Page area** – contains the content of the document
  - **Margin area** – contains the space between the printed content and the edges of the page

Working with the @page Rule (continued)

- Styles are applied to the page box using:

```
@page {
style rules
}
```

  where *style rules* are the styles applied to the page
- The styles are limited to defining the page size and the page margin

Setting the Page Size

- The following `size` property allows web authors to define the dimensions of a printed page:

  `size: width height;`

  where *`width`* and *`height`* are the width and height of the page
- The keyword `auto` lets browsers determine the page dimensions
- The keyword `inherit` inherits the page size from the parent element

Using the Page Pseudo-Classes

- Different styles can be defined for different pages by adding the following:

```
@page:pseudo-class {
    style rules
}
```

where *pseudo-class* is `first` for the first page of the printout, `left` for the pages that appear on the left in the double-sided printouts, or `right` for pages that appear on the right in double-sided printouts

Page Names and the Page Property

- To define styles for pages other than the first, left, or right, create a page name as follows:

```
@page name {
    style rules
}
```

where *name* is the label given to the page

Page Names and the Page Property (continued)

- To assign a page name to an element, use

```
selector {
    page: name;
}
```

where *selector* identifies the element that will be displayed on its own page, and *name* is the name of a previously defined page style

Formatting Hypertext Links for Printing

- To append the text of a link's URL to the linked text, apply the following style rule:

```
a::after {
    content: " (" attr(href) ") ";
}
```

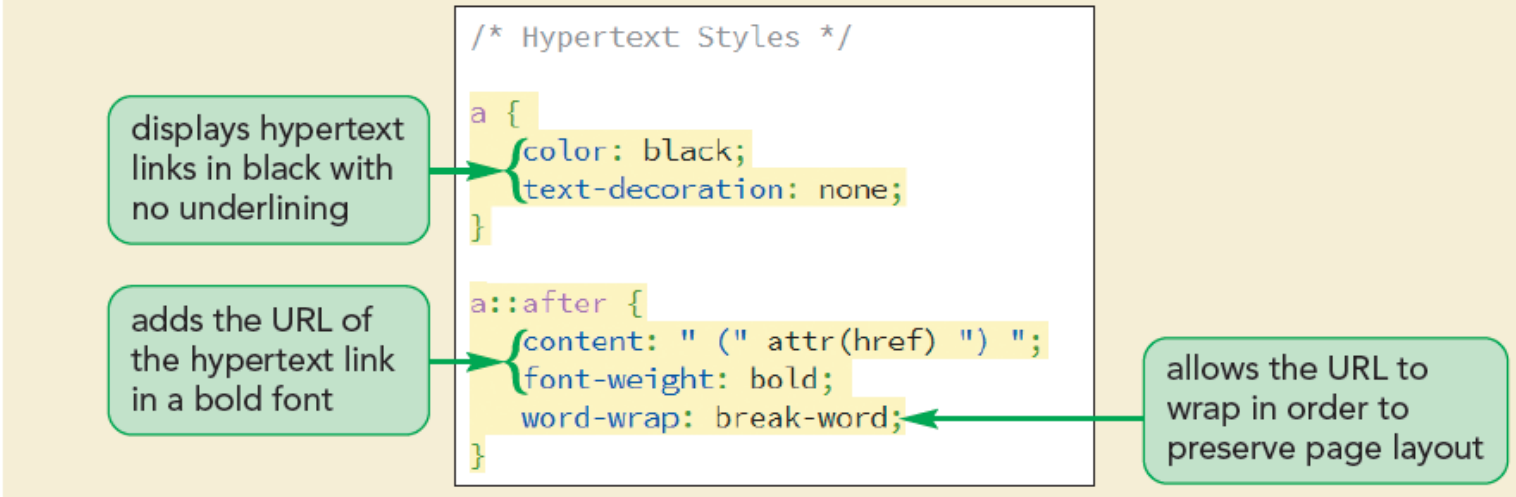
This style rule uses the `after` pseudo-element along with the `content` property and the `attr()` function to retrieve the text of the `href` attribute and add it to the contents of the `a` element

Formatting Hypertext Links for Printing (continued 1)

- The `word-wrap` property is used to break long text strings at arbitrary points if it extends beyond the boundaries of its container

Formatting Hypertext Links for Printing (continued 2)

Figure 5-51 Formatting printed hypertext links



Working with Page Breaks

- Page breaks can be inserted either directly before or after an element, using the following properties:

```
page-break-before: type;
page-break-after: type;
```

where *type* has the following possible values:

- `always` – Use to always place a page break before or after the element
- `avoid` – Use to never place a page break

Working with Page Breaks (continued)

- `left` – Use to place a page break where the next page will be a left page
- `right` – Use to place a page break where the next page will be a right page
- `auto` – Use to allow the printer to determine whether or not to insert a page break
- `inherit` – Use to insert the page break style from the parent element

Preventing Page Breaks

- Page breaks can be prevented by using the keyword `avoid` in the `page-break-after` or `page-break-before` properties
- For example, the following style rule prevents page breaks from being added after any heading

```
h1, h2, h3, h4, h5, h6 {
    page-break-after: avoid;
}
```

Working with Widows and Orphans

- Page breaks within block elements, such as paragraphs, often leave behind widows and orphans
- A widow is a fragment of text left dangling at the top of a page
- An orphan is a text fragment left at the bottom of a page

Working with Widows and Orphans (continued)

- To control the size of widows and orphans, CSS supports the following properties:

```
widows: value;
orphans: value;
```

where *value* is the number of lines that must appear within the element before a page break can be inserted by printer