Media Queries

Fluid Layouts

- "You must be shapeless, formless, like water. When you pour water in a cup, it becomes the cup. When you pour water in a bottle, it becomes the bottle. When you pour water in a teapot, it becomes the teapot."

  Bruce Lee

Fluid Layouts

- Fluid layouts are a great start. They eliminate the constraints of a fixed-width layout and enable your site to display nicely on a larger number of screens.
- However, they can only take you so far.

Media Queries

- Media queries let you define which styles should apply under specific circumstances by allowing you to query the values of features such as
  - Resolution,
  - Colour depth
  - Height, and
  - Width
- By carefully applying media queries, you can iron out the remaining wrinkles in your layout.

- By using Media Queries you can:
  - Set the viewport of your site.
  - Adjust your site design.
  - Identify the needed breakpoints.
  - Improve the navigation experience across multiple screen sizes.

The Problem

- Consider the site for Figure 3.1 in the text
- If we resize the window to be very wide, the line length increases.
  - The wider we go, the further the line length of the article gets from the ideal.
  - Other than that, the situation is not all that bad; the layout holds up pretty well.

The Problem

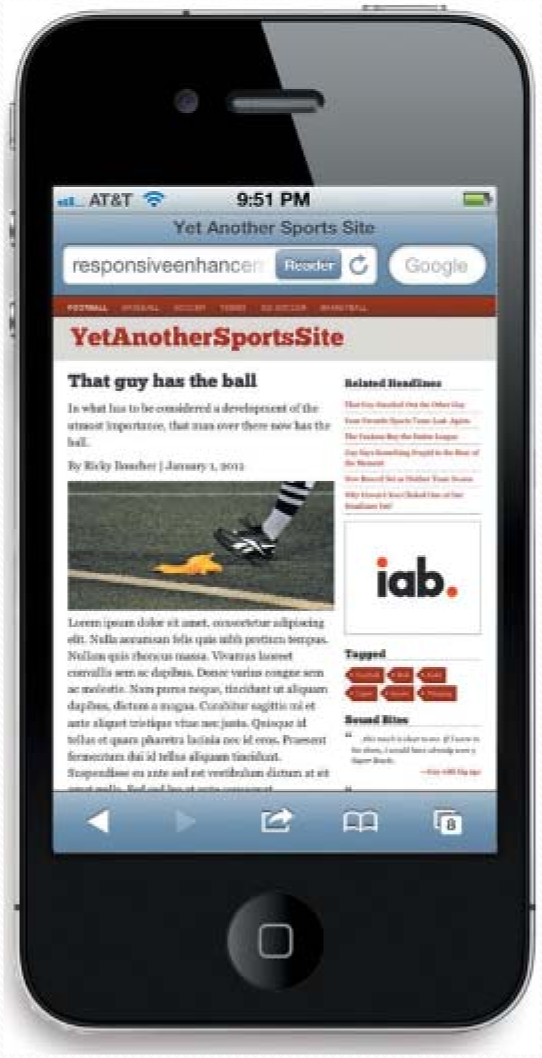
- As we resize the window to be narrower, the layout begins to look like it was hit repeatedly with a big stick.
  - The window does not have to get very narrow before the first navigation item falls under the rest of the links
    - This is not particularly elegant, but it is not necessarily a deal-breaker.
  - The line length of the primary column also gets a bit too short.
    - Remember, ideally we want that line length to fall somewhere between 45 and 70 characters. Anything under or above those numbers can have a negative impact on the reading experience.

The Problem

- As the window gets narrower, the issues get worse.
- By `360px` or so, the navigation is a complete mess.
- The primary column can barely fit three words per line, and even the sidebar is cramped for space.
- Clearly we have a Fluid layout but it does not seem to be scaling/adjusting properly
- And there is more …

The Problem

- Shrinking a browser is not the same as viewing the site on a Mobile device
- On a mobile device the, the page maintains its original layout, but it is zoomed out so that the text and site appear quite small.



Viewports

- The concept of a viewport is a simple one in terms of desktop browsers:
  - the viewport is the visible area of the browser, the browser width.
- It is so simple, in fact, that no one really bothers to even think about it.
- But that all changes with phones. Despite having much smaller screens, they attempt to display the "full" site in order to provide a full web experience. Suddenly things get a little more complicated.

When is a Pixel not a Pixel?

- When it comes to browsers, there are two kinds of pixels:
  - Device pixels and
  - CSS pixels
- Device pixels behave the way you would expect a pixel to behave.
  - If you have a screen that is `1024px` wide, then you can fit two `512px` elements side-by-side in it.

When is a Pixel not a Pixel?

- CSS pixels are a bit less stable.
- CSS pixels deal not with the screen, but with the visible area within the browser window.
- This means that CSS pixels may not line up exactly with device pixels.
- While on many devices, one CSS pixel is the same as one device pixel, on a high-resolution display such as the Retina display of the iPhone, one CSS pixel is actually equal to two device pixels.
- Just wait… it is about to get even more fun!

When is a Pixel not a Pixel?

- Any time a user zooms in or out of a page, the CSS pixels change.
  - If a user zooms to 300%, then the pixels stretch to three times the height and three times the width they were set at originally.
  - If the user zooms to 50%, then the pixels are reduced to half the height and half the width.

When is a Pixel not a Pixel?

- The entire time, the device pixel count does not change— the screen is, after all, the same width.
- The CSS pixel count, however, does. The number of CSS pixels that can be viewed within the browser window changes.

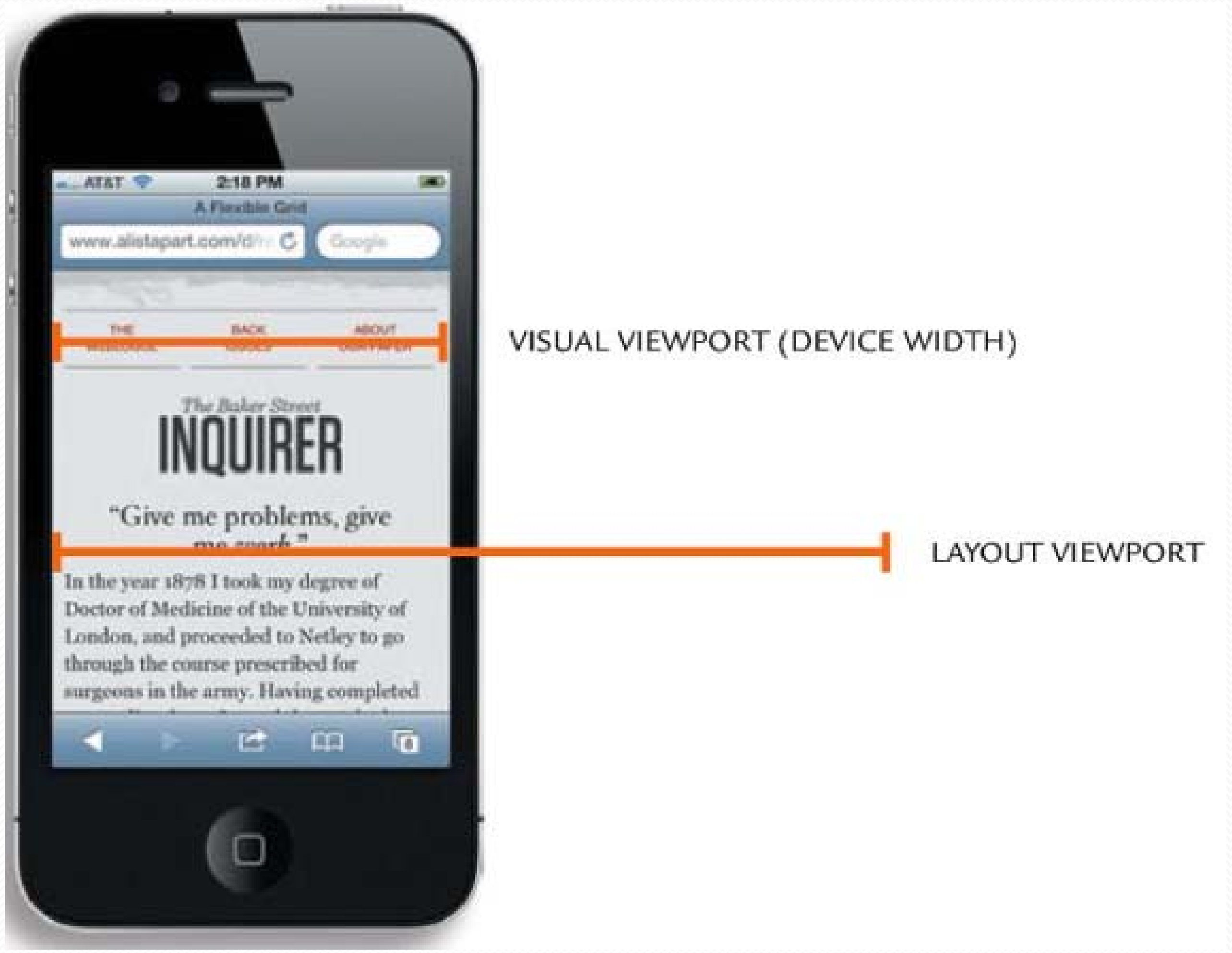
Viewports

- The changing number of CSS pixels comes into play in considering the viewport.
- Remember there are two different viewports to consider:
  - Physical or Visual Viewport– the screen size
  - The Layout Viewport – the size of the layout

Viewports

- The layout viewport is similar to device pixels in that its measurements are always the same, regardless of orientation or zoom level.
- The visual viewport, however, varies. This is the part of the page that is actually shown on the screen at any given point.

Viewports



In the year 1878 I took my degree of Doctor of Medicine of the University of London, and proceeded to Netley to go through the course prescribed for surgeons in the army. Having completed

Viewports

- On a mobile device, this can complicate things.
- To allow for a "full web" experience, many mobile devices return high layout viewport dimensions.
  - iPhone has a layout viewport width of `980px`,
  - Opera Mobile returns `850px`, and
  - Android WebKit returns `800px`.
- This means that if you create a `320px` element on the iPhone, it will fill up only about a third of the screen real estate.

Viewports

- The `viewport` meta tag lets us control the scaling and Layout Viewport of many devices.

- Basic Syntax

```
<meta name =“viewport”
      content =“directive, directive"
/>
```

Viewports

- As it turns out, the meta `viewport` element is actually non-normative.
- In plain English, it is not a definitive standard.
- In fact, a close look at the W3C documents reveals that the only reason it is still included in the specification is to provide a road map for browsers to migrate to the new CSS `@viewport` syntax.
- But it is still widely used and support for `@viewport` is Work in Progress

Viewport Properties - `width`

- The `width` directive lets you set the viewport to a specific width, or to the width of the device

- Example

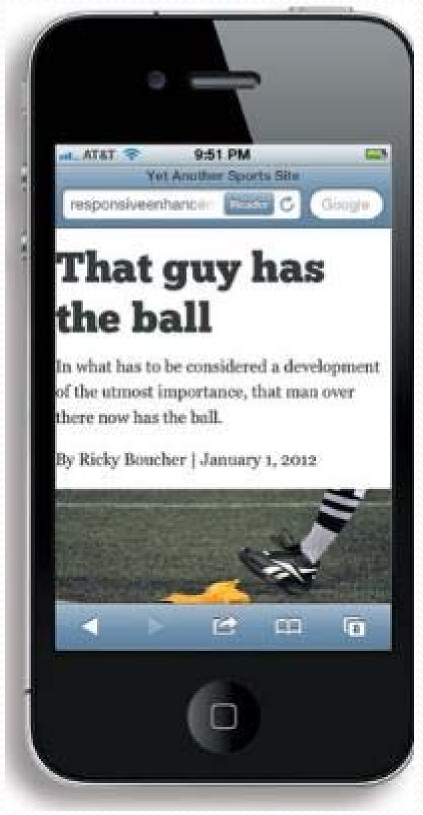
```
<meta name =“viewport”
      content =“width = device-width”
/>
```

Viewport Properties - `width`

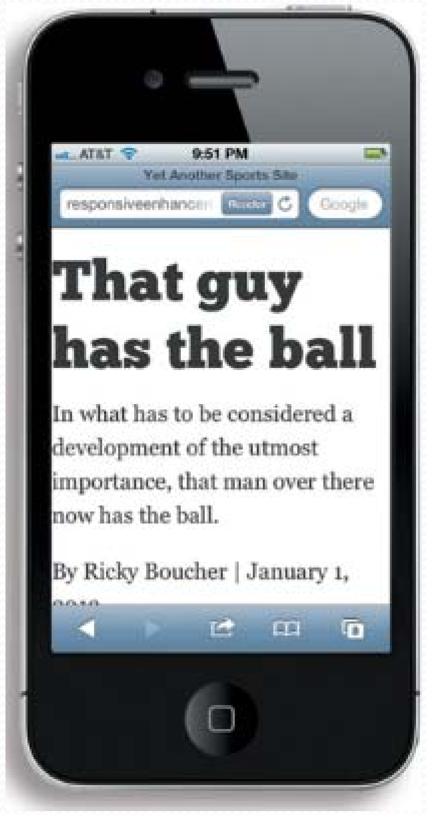
- Using `device-width` is the best solution. This way, your screen's layout viewport will equal the screen of the device— in device pixels.
- If you use a specific width instead, such as `240px`, most devices that do not have a width of `240px` will scale to match.
  - So, if your device has a screen width of `320px`, everything will be scaled up by a factor of `1.33 (320/240)` in an attempt to display the page neatly

Viewport Properties - `width`

Set to **`device-width`**


AT&T
9:51 PM
Yet Another Sports Site
responsiveenhancem
Reader
Google
That guy has the ball
In what has to be considered a development of the utmost importance, that man over there now has the ball.
By Ricky Boucher | January 1, 2012

Set to **`320px`**


AT&T
9:51 PM
Yet Another Sports Site
responsiveenhancem
Reader
Google
That guy has the ball
In what has to be considered a development of the utmost importance, that man over there now has the ball.
By Ricky Boucher | January 1,

Viewport Properties - `height`

- The counterpart to `width`, `height` lets you specify a particular height

- Example

```
<meta name =“viewport”
      content =“height = device-height”
/>
```

- This sets the layout viewport equal to the height of the screen.

Viewport Properties - `height`

- In practice, you probably will not use height very much.
- The only time it is handy is if you do not want to let the page scroll vertically, which does not happen that often.

Viewport Properties – `user-scaleable`

- The user-scalable directive tells the browser whether or not the user can zoom in and out on the page

- Example

```
<meta name =“viewport”
      content =“user-scaleable = no”
/>
```

Viewport Properties – `user-scaleable`

- You will often find pages that set user-scalable to `no`, typically to ensure the “pixel-perfect” display of a design.
- This is counter to the nature of the Web, and detrimental to users with accessibility needs.
- If you do not set the user-scalable directive, it will default to `yes`.
- As a result, it is best to stay clear of this one.

Viewport Properties – `initial-scale`

- Given a number between `0.1` (`10%`) and `10.0` (`1000%`), the initial-scale declarative sets the initial zoom level of the page.

- Example

```
<meta name =“viewport"
      content =“initial-scale = 1,
                 width = device-width”
/>
```

Viewport Properties – `initial-scale`

- Using the previous declaration
  - If the width of the device is `320px`, the page will display at `320px`
  - If the width is `200px`, the page will display at `200px`

Viewport Properties – initial-scale

- Another Example

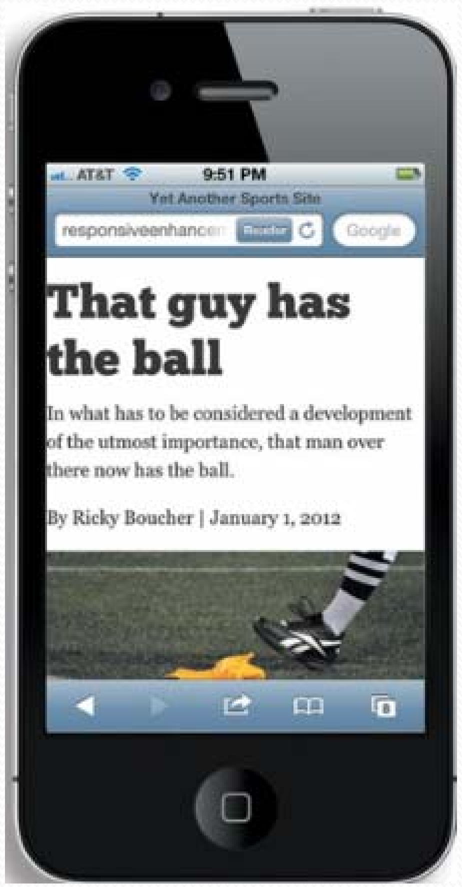
```
<meta name =“viewport"
      content =“initial-scale = 0.5,
                width = device-width”
/>
```

Viewport Properties – `initial-scale`

- With the second example, the width attribute is set to the width of the device and the initial-scale is set to `0.5` (50%).
- This means that the browser will display everything zoomed out.
  - On a `320px`-wide device, the page will display at `640px`
  - On a `200px`-wide device, it will display at `400px`

Viewport Properties – `initial-scale`

`initial-scale` set to `1`

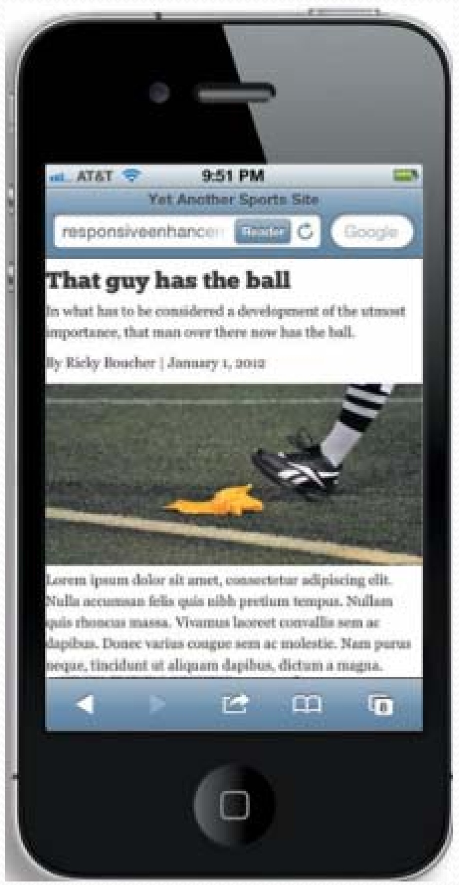


That guy has the ball

In what has to be considered a development of the utmost importance, that man over there now has the ball.

By Ricky Boucher | January 1, 2012

`initial-scale` set to `0.5`



That guy has the ball

In what has to be considered a development of the utmost importance, that man over there now has the ball.

By Ricky Boucher | January 1, 2012

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla accumsan felis quis nibh pretium tempus. Nullam quis rhoncus massa. Vivamus laoreet convallis sem ac dapibus. Donec varius congue sem ac molestie. Nam purus neque, tincidunt ut aliquam dapibus, dictum a magna.

Viewport Properties – `maximum-scale`

- The `maximum-scale` declarative tells the browser how far the user can zoom in on a page. In mobile Safari, the default is `1.6` (`160%`), but any number between `0.1` (`10%`) and `10.0` (`1000%`) will work.

- Example

```
<meta name =“viewport"
      content =“maximum-scale = 1.0,
                width = device-width”
/>
```

Viewport Properties – `maximum-scale`

- With the previous example, we set the `maximum-scale` declarative to `1.0` (`100%`).
- This disabled the user's ability to zoom in.
- As you have already seen, this limits accessibility and should be avoided.

Viewport Properties – `minimum-scale`

- The `minimum-scale` declarative tells the browser how far the user can zoom out on a page. In mobile Safari, the default is `0.25` (25%), but any number between `0.1` (10%) and `10.0` (1000%) will work.

- Example

```
<meta name =“viewport"
      content =“minimum-scale = 1.0,
                width = device-width”
/>
```

Viewport Properties – `maximum-scale`

- With the previous example, we set the `minimum-scale` declarative to `1.0` (`100%`).
- This disabled the user's ability to zoom out.
- As you have already seen, this limits accessibility and should be avoided.

Problems with meta `viewport`

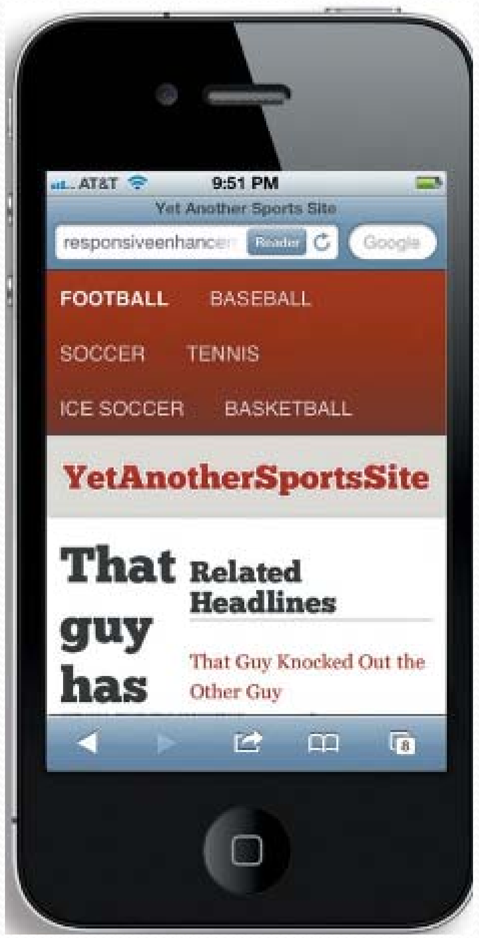
- It does not take a keen eye to tell that by setting the `viewport`, we have actually just made the situation worse
- Now, our site looks equally beaten up on the mobile and the desktop.
- One recommendation advocated by Visual Designers is
  - "Do not use viewport unless you know what you are doing"
- Put another way,
  - "Beware of unintended consequences"

Problems with meta `viewport`

- It does not take a keen eye to tell that by setting the viewport, we have actually just made the situation worse
- Now, our site looks equally beaten by the stick on the phone and the desktop.
- It is time to call on our friend, the media query, for help.

Problems with meta `viewport`

- With the viewport set, the site displays just as it did on the desktop, only zoomed out



Media Queries

- It is time to call on our friend, the media query, for help.

Media Queries

- Media queries let you question the browser to determine if certain expressions are true.
- If they are, you can load a specific block of styles intended for that situation and tailor the display.

Media Queries

- The general form of a media query is:

```
@media [not | only] type [and] (expr) {
     rules
}
```

Media Queries

- A media query has four basic components:
  1. **Media types**: specify the type of device to target
  2. **Media expressions**: test against a feature and evaluate to either true or false
  3. **Logical keywords**: keywords (such as `and`, `or`, `not`, or `only`) that let you create more complex expressions
  4. **Rules**: basic styles that adjust the display

Media Types

- The Web extends far beyond the screen devices you are used to (phones, tablets, desktops, and laptops.
- Information can be printed or accessed via
  - Braille tactile feedback devices,
  - Speech synthesizers,
  - Projectors,
  - Televisions
  - And many more.

Media Types

- Media types were developed to bring order to this chaos.
- Each media type tells the user agent (such as a browser) whether or not to load that stylesheet for a given type of media.

Media Types defined by CSS

Type	Target Device
`all`	All Devices (default)
`braille`	Braille tactile feedback devices
`embossed`	Paged braille printers
`handheld`	Handheld devices (typically small scree and possibly monochrome)
`print`	Printing or print preview
`projection`	Projected presentations
`screen`	Colour computer screen
`speech`	Speech synthesizers
`tty`	Media using a fixed-pitch character grid (terminals or teletypes)
`tv`	Television devices

Example Media Types

- CSS

```
@media print {
}
```

- HTML

```
<link rel    =“stylesheet”
      href   =“print.css”
      media  =“print”
/>
```

Example Media Types

- Every media query must include a media type. If you do not set one, it should default to `all`, but the actual behavior varies from browser to browser.
- In practice, you will find yourself using `all`, `screen` and `print` almost exclusively.

Media Expressions

- The power of media queries is their ability to test against different features of a device using expressions that evaluate to either true or false.
- A simple example would be to determine whether the width of the viewport is greater than `320px`

```
@media screen and (min-width: 320px) {
}
```

Media Features

Feature	Definition	Value	Min/ Max
`width`	Width of the display area of the device	Length in pixels	yes
`height`	Height of the display area of the device	Length in pixels	yes
`device-width`	Width of the rendering surface area of the device	Length in pixels	yes
`device-height`	Height of the rendering surface area of the device	Length in pixels	yes
`orientation`	Indicates if the device is in portrait or landscape orientation	`portrait | landscape`	no
`aspect-ratio`	Ratio of the value of the width feature to the value of the height feature	Ratio `height/width`	yes

Media Features

Feature	Definition	Value	Min/ Max
`resolution`	Pixel density of the device • Dots per inch (dpi) • Dots per cm (dpcm)	Integer value	no
`scan`	Scanning process of `tv` device	`progressive|` `interlace`	no
`grid`	Returns whether a device is a grid device (1) or a bitmap device(2)	Integer value of `0` or `1`	no

Media Features

Feature	Definition	Value	Min/ Max
`device-aspect-ratio`	Ratio of the value of the device-width feature to the value of the device-height feature	Ratio `device-height/device-width`	yes
`color`	Number of bits per color components of the device, Zero if not a color device	Integer value	yes
`color-index`	Number of entries in the color look-up table for the device	Integer value	yes
`monochrome`	Number of bits per pixel on a monochrome device, Zero if not a monochrome device	Integer value	yes

Media Features

- Primarily, you will stick to using
  - `width`,
  - `height`,
  - `orientation`,
  - `resolution` and
  - `aspect-ratio`

Logical Keywords - and

- You can use and to test against more than one expression:

```
@media screen and (color)
```

- The above example tests to make sure the device has a color screen.

Logical Keywords - `or`

- There is no `or` keyword for media queries, but the comma acts as one.

```
@media screen and (color),
       projection and (color)
```

- In the example above, the query evaluates as true if the device is either a colour screen device, or a colour projection device.

Logical Keywords - `not`

- The `not` keyword negates the result of the entire expression, not just a portion of it.

```
 @media not screen and (color) {
 } // equates to not (screen and (color))
```

- For the media query above, the query returns false for any device that has a color screen.
- You cannot use the `not` keyword to negate a single test. If you are using `not` it must be the first entry in the media query after the `@media` keyword.

Logical Keywords - `only`

- Many older browsers support media types, but not media queries. This sometimes results in the browser attempting to download styles that you do not want.
- The `only` keyword can be used to hide media queries from older browsers, as they will not recognize the media query.
- Browsers that do support the `only` keyword process the media query as if the `only` keyword was not present. This is generally a very good idea.
- If you are using `only` it must be the first entry in the media query after the `@media` keyword.

Logical Keywords - `only`

- Consider the following media query

```
@media only screen and (color)
```

- If a device does not support media queries, it ignores the query above entirely.
- If the device does support media queries, it evaluates the query the same way it would evaluate the following:

```
@media screen and (color)
```

Other Logical Keywords

- As Responsive Design takes hold other logical keywords are being considered for media queries

Keyword	Proposed Meaning
`script`	Test to see if ECMAscript is supported, and if that support is active (i.e., it has not been disabled).
`pointer`	Query about the accuracy of the pointing device (such as a mouse or finger).
`hover`	Query whether or not the primary pointing method can hover.

Rules

- The last piece in the media query puzzle is the actual style rules you want to apply.
- These are basic CSS rules. The only special thing about them is that they are included within a media query

```
@media only screen and
            (min-width: 320px) {
       h1{color: blue}
}
```

Embedded versus External

- Media queries can be embedded in the main stylesheet or placed in the media attribute of a link element to include an external stylesheet.

Embedded versus External

- Embedded

```
h1{
    text-decoration:none;
}
@media screen and (min-width: 300px) {
    h1{
        text-decoration: underline;
    }
}
```

Embedded versus External

- External

```
<link href =“style.css"
      media =“only screen and (min-width: 300px)"
/>
```

Embedded versus External

- For media queries that are embedded in a single stylesheet, all styles are downloaded regardless of whether or not they are needed
  - but the benefit is that you have to make only one HTTP request.
- This is an important consideration for performance, particularly if the device is being used on a mobile network.

Embedded versus External

- Mobile networks suffer from high latency, that is, the time it takes for the server to receive and process a request from the browser.
- Every time an HTTP request is made on a mobile network, it could be taking as much as four or five times as long as it would take on a typical wired Internet connection.

Embedded versus External

- The downside, of course, is that this one CSS file can get to be very large. So while you have saved a few requests, you have created a heavy file that can be difficult to maintain.

Embedded versus External

- You might be surprised to learn that external media queries still result in all the styles being downloaded, even if they are not applicable.
- The rationale for this is that if the browser window size or orientation is changed, those styles are ready and waiting.
- Unfortunately, this results in several HTTP requests instead of just one.

Embedded versus External

- The advantage of external media queries is that the files will be smaller, helping to make them easier to maintain.
- You can also serve up a low-weight, simplified stylesheet to devices that do not support media queries and again, thanks to the only keyword, you do not have to worry about them applying styles they will not need.

Media Query Order

- Two ways of building a Responsive site are:
  - From the desktop down or
  - From mobile up

- The approach selected impacts the structuring of your CSS

Desktop Down

- Responsive design, as it is still most commonly implemented, is built from the desktop down.
- The default layout is what you typically see on the screen of a browser on a laptop or desktop computer. Then, using a series of media queries (typically `max-width`), the layout is simplified and adjusted for smaller screens.

Desktop Down

- A stylesheet structured from a Desktop Down approach might look like the following:

```
/* base styles */
@media all and (max-width: 768px) {
     ...
}
@media all and (max-width: 320px) {
     ...
}
```

Desktop Down

- Unfortunately, building from the desktop down results in some serious issues.
- Media query support on mobile devices, while improving, is still somewhat sketchy. The following are example devices that lack media query support
  - BlackBerry (pre-version 6.0),
  - Windows Phone 7, and
  - NetFront (which powers pre-third generation Kindle devices)

Mobile Up

- If you flip things around and build the mobile experience first, and then use media queries to adjust the layout as the screen size gets larger, you can largely circumvent the support issue.
- Building the mobile experience first will ensure that mobile devices that do not support media queries will still be served an appropriate layout.
- The only desktop browser that you will need to contend with is Internet Explorer.
  - Prior to version 9, Internet Explorer does not support media queries.

Mobile Up

- A stylesheet structured from a Mobile Up approach might look like the following:

```
/* base styles, for the small-screen
experience, go here */
@media all and (min-width: 320px) {
     ...
}
@media all and (min-width: 768px) {
     ...
}
```

Media Query Order

- Support is not the only advantage of building mobile up. Creating the mobile experience first can help reduce the complexity of your CSS as well.

Media Query Order

Desktop Down

```
aside{
  display: table-cell;
  width: 300px;
}
@media all and
    (max-width:
320px){
  aside{
      display: block;
      width: 100%;
  }
```

Mobile Up

```
@media all and
    (min-width:
320px){
  aside{
   display: table-
cell;
   width: 300px;
  }
}
```

Desktop Down v Mobile Up

- Give an Role Play Exercise
  - UI Changes v Biz Changes v DB Changes
  - Frequency of Changes
  - Impact of Changes
  - Impact on Desktop Down v Mobile Up Approach

Create your Core Experience

- Read Book Chapter 3 Pages 76-94. It gives a practical example of application of the material covered

Wrapping it up

- Fluid layouts are a start, but they can only take us so far.
- At some point, we need to adjust the layout, sometimes dramatically, to better accommodate different devices.
- Smartphones try to let us experience the full Web. If the meta viewport element is not being used, most smartphones display a zoomed version of the site.

Wrapping it up

- Media queries let us test for features like width and height and adjust the CSS we apply to our design accordingly.
- They can be used both externally and internally. Each method has benefits and limitations, so it is important to choose the approach that best meets the project requirements.
- While it is common to pick specific device widths for breakpoints, a better approach is to let the content dictate where you need to include a media query.