Example

```html
/* Core */

import url(https://fonts.googleapis.com/css?family=Shadows+Into+Light);

html {
  background-image: url('mint_stripes.jpg');
  /* Stripes background mint green:
   Licensed under CC BY 1.0 (https://creativecommons.org/publicdomain/zero/1.0/) */

  background-color: rgb(204, 239, 219);
}

* {
  margin: 0;
  padding: 0;
}

body {
  width: 70%;
  margin: 1em auto;
  background-color: rgb(1, 92, 3);
  color: white;
  border: 1em solid rgb(1, 92, 3);
  border-radius: 1em;
  font-size: 16px;
  font-family: 'Shadows Into Light', cursive;
}

nav, main, aside {
  background-color: white;
  color: black;
  padding: 1em;
  border: 0.0625em solid rgb(204, 239, 219);
  border-radius: 1em;
}

.cocktail {
  color: rgb(1, 92, 3);
  font-weight: bold;
}

img {
  max-width: 100%;
}
```
/* Header */
header {
    padding: 2em;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: space-between;
    align-items: baseline;
}
header > h1 {
    font-size: 3em;
}
header > p {
    font-size: 2em;
}

/* Nav */
nav {
    padding: 1em 0;
    background-color: rgb(204, 239, 219);
    text-align: center;
}
nav ul {
    list-style: none;
    padding: 0 0.625em;
}
nav li li {
    background-color: white;
    border: 0.0625em solid rgb(204, 239, 219);
}

/* Main */
main {
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: space-around;
    align-items: flex-start;
}
main > h1, main > figure, main > p {
    flex-basis: 100%;
}
main > section {
    flex-basis: 48%;
    min-width: 15em;
    margin: 0.5em 0 1em 0;
    border: 0.0625em solid rgb(204, 239, 219);
    border-radius: 1em;
}
main > h1 {
    font-size: 2em;
}
main > section > h1 {
    font-size: 1.5em;
    padding: 0 1rem;
}
main > section > p {
    padding: 0 1rem;
}
main > p {
    margin: 0.5em 0 1em 0;
}
main ul, main ol {
    margin: 0.5em 0 1em 0;
    padding: 0 0 2.5em;
}
main > figure, main > warning {
    text-align: center;
}
.attribution {
    display: block;
    font-size: 0.6em;
}
Example, continued

```css
/* Aside */
aside > h1 {
  font-size: 1.5em;
}
aside > p {
  margin: 0.5em 0 3em 0;
  border-bottom: 0.0625em solid rgb(204, 239, 219);
}
/* Footer */
footer { padding: 2em;
}
```

Example, continued

```css
/* Two-column layout */
@media screen and (min-width: 50em) {
  nav ul {
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: space-around;
  }
  nav ul li {
    flex-basis: 10em;
  }
  body {
    display: grid;
    grid-template-columns: 75% 25%;
    grid-template-rows: auto;
    grid-template-areas: "upper-top upper-top"
                        "lower-top lower-top"
                        "middle-left middle-right"
                        "bottom bottom";
  }
  header {
    grid-area: upper-top;
  }
  nav {
    grid-area: lower-top;
  }
  main {
    grid-area: middle-top;
  }
  aside {
    grid-area: middle-left;
  }
  footer {
    grid-area: middle-right;
  }
  footer { grid-area: bottom; }
}
```
Example, continued

/* Three-column layout */
@media screen and (min-width: 70em) {
  nav {
    text-align: left;
    background-color: white;
    padding: 1em 0;
  }
  nav ul {
    margin: 0.5em 0 1em 0;
    padding: 0 0 0 2.5em;
  }
  nav li li {
    border: 0;
    border-bottom: 0.0625em solid rgb(204, 239, 219);
  }
  body {
    display: grid;
    grid-template-columns: 20% 55% 25%;
    grid-template-rows: auto;
    grid-template-areas: "top top top" "middle-left middle-middle middle-right" "bottom bottom bottom"
  }
  header {
    grid-area: top;
  }
  nav {
    grid-area: middle-left;
  }
  main {
    grid-area: middle-middle;
  }
  aside {
    grid-area: middle-right;
  }
  footer {
    grid-area: bottom;
  }
}

Example, continued

/* Fixed-width three-column layout */
@media screen and (min-width: 80em) {
  body {
    width: 60em;
  }
}

Common breakpoints

<table>
<thead>
<tr>
<th>Breakpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>320px</td>
<td>For small screen devices, like phones, held in portrait mode</td>
</tr>
<tr>
<td>480px</td>
<td>For small screen devices, like phones, held in landscape mode</td>
</tr>
<tr>
<td>600px</td>
<td>Smaller tablets held in portrait mode</td>
</tr>
<tr>
<td>768px</td>
<td>Ten-inch tablets held in portrait mode</td>
</tr>
<tr>
<td>1024px</td>
<td>Ten-inch tablets held in landscape mode, as well as some laptop, netbook and desktop displays</td>
</tr>
<tr>
<td>1200px</td>
<td>For widescreen displays, primarily laptop and desktop browsers</td>
</tr>
</tbody>
</table>

But,
- there is a much wider range of devices and viewports than this table assumes
- the table is not future-proof
Breakpoints

- Let content determine breakpoints:
  - “Start with the small screen first, then expand until it looks like shit. Time for a breakpoint!” — Stephen Hay
  - A breakpoint is any point at which you need a media query to get the page or one of its components to look right
- Some people now recommend using ems instead of pixels for the media queries
  - This defines breakpoints proportionally and hence also works when the user scales the content
  - It also encourages you to define breakpoints based on readability: people are most comfortable with lines of 45-75 characters

A final fix

- Many small-screen devices draw a distinction between the ‘layout viewport’ and the ‘visual viewport’
  - E.g. iPhone layout viewport is 980 px wide but visual viewport is 320 px wide
  - Web pages are drawn onto the layout viewport (e.g. CSS widths are based on this)
  - But what you see is just part of it — whatever fits into the visual viewport
  - Zoom out to see more of it
- Question: Why do you think this is a problem for us?
  - For RWD on these devices we need to override this default behaviour by making the two viewports of equal size
  - Two alternative ways of doing this
    - Add the following inside the head element of your HTML:
      ```html
      <meta name="viewport" content="initial-scale=1.0, width=device-width" />
      ```
      (but this is not part of standard HTML and it is undesirable to define visual styles in HTML)
    - Or add the following to your CSS:
      ```css
      @viewport
      {
      width: device-width
      }
      ```
      (but this is so new that it is not yet widely supported by browsers)
  - For now, maybe do both!
Revision: background images

- We use CSS to give an element a background image. E.g.:

```html
html {
  background-image: url(swirly-pattern.gif);
}
```
- Any element can have a background image, not just `body`
- Other relevant CSS includes:
  - `background-repeat` with values `repeat` (default), `repeat-x`, `repeat-y`, `no-repeat`
  - `background-position` with values `top/center/bottom` and `left/center/right`
  - `background-attachment` with values including `scroll` (default) and `fixed`
  - Also `background-size`, `background-origin`, `background-clip`

Revision: foreground images

- We use HTML to include a foreground image. E.g.:

```html
<img src="wombat.jpg" alt="Wombats are furry, have protruding front teeth and walk on all fours."
     title="A wombat walking along a road" />
```
- There are also width and height attributes — see below
- You can also nest the `img` element inside a `figure` element, optionally along with a `figcaption` element

Foreground images in fixed-width layouts

- In fixed-width layouts, you know how much space (in pixels) is available for your images
- As much as possible, try to create the original image so that it is the desired size
- If this is not possible, then scale it using image manipulation software (e.g. Photoshop, GIMP)
- It is possible but less satisfactory to let the browser scale the image by specifying the desired size
- either in the HTML, e.g.

```html
<img src="wombat.jpg" width="152" alt="..." />
```
- or in the CSS

```css
img {
  width: 152px;
}
```

Question: Why are these less satisfactory than using correctly-sized images?

Foreground images for liquid layouts

- In liquid layouts, you don't know how much space is available for your images
- If the image is wider than the available space, it will overflow its container — unlikely to look good!
- One option is to use CSS to handle the overflow:

```css
figure {
  overflow: hidden;
}
```

The `overflow` property has values that include `visible` (default), `hidden`, `scroll` and `auto`
Foreground images for liquid layouts

- A more flexible option (recommended in *Responsive Web Design*) is to set the max width to 100% in the CSS:

```css
img {
  max-width: 100%;
}
```

- If the image is narrower than its container, it will display at its normal size
- If it is wider than its container, it will be scaled down to match the width of the container
- Or (less common) you could set the width to 100%:

```css
img {
  width: 100%;
}
```

  Then the image scales up or down to match the container

**Question:** This looks good. But what are the disadvantages?

Background images for liquid layouts

- Background images are less of a problem
  - By default, they tile
    - So they will cover the element (irrespective of its size) without scaling
  - Or you can use the `background-size` property to scale them, e.g.

```css
html {
  background-image: url('swirly-pattern.gif');
  background-size: cover;
}
```

- Or you can use media queries to load different images for different viewport sizes

Videos for liquid layouts

- Revision: how to embed your own video into a web page:

  ```html
  <video width="640" height="360" controls>
    <source src="mojito.webm" type="video/webm"/>
  </video>
  ```

- Now, for RWD, we can do what we did with images:

```css
video {
  max-width: 100%;
}
```

- There seems to be some disagreement about whether it is better to also remove the height and width attributes, or just the height, or neither
- See also this more extensive treatment of this topic