CSS Frameworks

CSS frameworks contain a collection of useful classes and other CSS code. Frameworks aim to save designers and developers time and effort by providing pre-written CSS classes that can be used to style and arrange website content quickly. There are plenty of CSS frameworks out there, and *Bootstrap* is possibly the most popular.

CSS frameworks vary from 'lightweight' to 'heavy', and some include additional features like JavaScript, CSS precompiler support, icon fonts, and more.

Bootstrap is a heavier, more feature-rich type of framework. That said, it is modularised should you wish to utilise a specific set of features. We'll be using Bootstrap for this workshop. If you'd like to try out something more lightweight and straightforward, *Skeleton* is one such option:

http://getskeleton.com/

Bootstrap

"Bootstrap, originally named *Twitter Blueprint*, was developed by Mark Otto and Jacob Thornton at Twitter as a framework to encourage consistency across internal tools."
-- *from wikipedia.org*

To make use of Bootstrap, you'll need to download the framework files:

https://getbootstrap.com/docs/4.3/getting-started/download/

Don't worry about the source, sass, etc. links -- what you are after is the *compiled* version.

Once downloaded, extract the zip archive. You'll discover two folders within it:

- css (Bootstrap's CSS files)
- js (Bootstrap's JavaScript files)

You'll need to link these files to an HTML page. Begin by creating a new directory in your 289.101 repo named "bootstrap". Move the Bootstrap css and js directories into this. Using your preferred editor (Brackets?) create a new "index.html" file and save it alongside the css/js directories.

```
bootstrap
├─ css
├─ js
└─ index.html
```

Now add the necessary code to your `head` element:

index.html

```
...

<head>
  <meta charset="utf-8" />
  <title>Bootstrap demo</title>

  <!-- add the following 4 lines for Bootstrap -->
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <link rel="stylesheet" href="css/bootstrap.min.css" />
  <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
  <script src="js/bootstrap.min.js"></script>
</head>

...
```

To explain the above:

- the `<meta name="viewport" ...` tag ensures that your website doesn't scale-down/zoom-out on mobile devices (like, when you have to zoom and pan about the webpage), but rather re-flows as you'd expect a responsive page to do.

- The `link` and `script` tags link to the necessary Bootstrap JavaScript files (this is required for the animated- and popover-type features). Some Bootstrap features rely on *jQuery*, so it's often necessary to include jQuery as well. If you're not interested in using Bootstrap's more interactive features (modals, collapses, etc.), you can leave out the JavaScript (`<script>`) lines altogether. For this lesson, you'll need them.

You'll notice that the minified (.min) files have been referenced to reduce the overall download size. Minified files have no 'whitespace' (space characters, new-lines).

Outline All Elements

It will be far easier to see what Bootstrap is doing if you outline all of your elements:

index.html

```
  ...

  <!-- outline all elements -->
  <style>
    * { outline: #EEE solid 1px }
  </style>

</head>

...
```

Of course, it's preferable to use an external stylesheet, but an internal stylesheet will keep things simpler for now. You should also note that the Bootstrap files are better left alone (unedited). If you wish to override Bootstrap styles, do so using your own selectors in a stylesheet that is linked somewhere below the Bootstrap files, i.e.:

incorrect:

```
  ...
  <link rel="stylesheet" href="css/custom_theme.css" />
  ...
  <link rel="stylesheet" href="css/bootstrap.min.css" />
  ...
```

correct:

```
  ...
  <link rel="stylesheet" href="css/bootstrap.min.css" />
  ...
  <link rel="stylesheet" href="css/custom_theme.css" />
  ...
```

Grids

Grids are used to control layout. Many CSS frameworks include some type of grid feature, as grids are handy for building responsive web-pages.

Bootstrap bases its grid on a 12-column system. This means that each row of your layout must contain 'cells' that span twelve columns in total. For example, a row could contain:

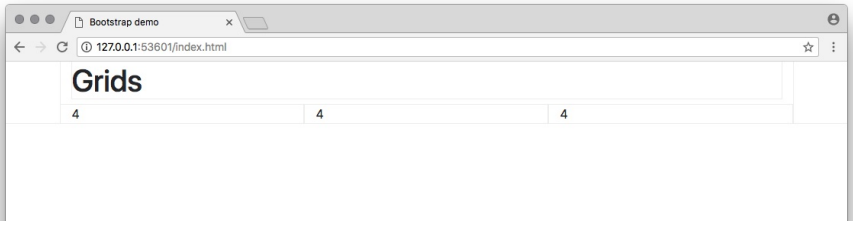
- 3 × 4-column-wide columns

- 2 × 6-column-wide columns
- 1 × 8-column-wide column, and 2 × 2-column-wide columns
- and so forth ...

Add a 3 × 4-column arrangement to your HTML file using the following code:

index.html

```
...

<body>

  <div class="container">

    <h1>Grids</h1>

    <div class="row">
      <div class="col-sm-4"> 4 </div>
      <div class="col-sm-4"> 4 </div>
      <div class="col-sm-4"> 4 </div>
    </div>

  </div>

...
```



Test this code in your browser, adjusting the window size to see how things adapt. The results should be fairly self-explanatory. The important thing to note is how the Bootstrap classes are used:

`class="container"`

- This creates a fixed-width(s) container that maxes-out at 1200px (as opposed to a a "container-fluid" that spans the entire viewport width).

`class="row"`

- Each row of columns is contained within a `row` element.

`class="col-sm-4"`

- The `-4` indicates a how many columns-wide the cell is -- but remember: these must add up to twelve in each row; if a row's sum total exceeds 12, columns will wrap into a new row. The `-sm` represents a *break-point* -- in other words: the point at which the cell becomes 100%-wide and occupies its own row -- like, how content jumps from being adjacently-arranged to vertically-stacked for mobile devices. The various break-points represent the following widths:
  `col-...` (portrait phones and up)
  `col-sm-...` (landscape phones and up)
  `col-md-...` (tablets and up)
  `col-lg-...` (desktops and up)
  `col-xl-...` (large desktops and up)

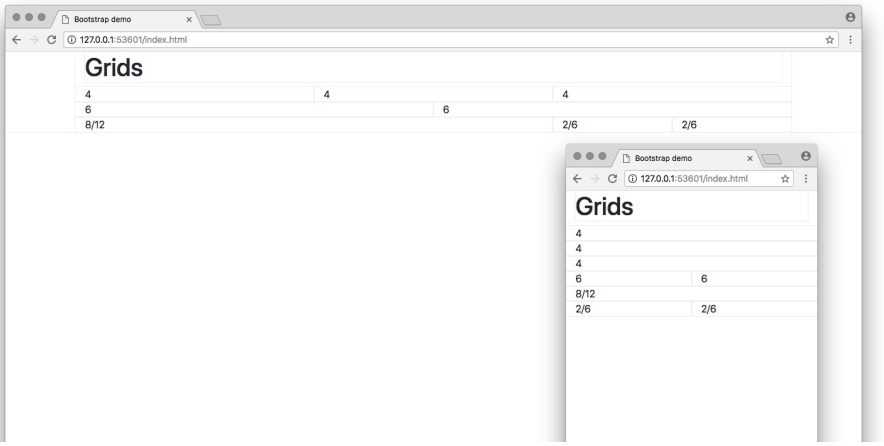
To get a better idea of how things work, add two more rows:

index.html

```
...

<body>

  <div class="container">

    <h1>Grids</h1>

    <div class="row">
      <div class="col-sm-4"> 4 </div>
      <div class="col-sm-4"> 4 </div>
      <div class="col-sm-4"> 4 </div>
    </div>

    <div class="row">
      <div class="col-6"> 6 </div>
      <div class="col-6"> 6 </div>
    </div>

    <div class="row">
      <div class="col-12 col-sm-8"> 8/12 </div>
      <div class="col-6 col-sm-2"> 2/6 </div>
      <div class="col-6 col-sm-2"> 2/6 </div>
    </div>
```

```
  </div>

...
```

Resize your web browser window to see how the classes take effect. Pay attention, in particular, to how the third-row targets multiple break-points, and how the "2/6" cells 'wrap' into a new row once the columns add up to more than 12 (because the "8/12" row becomes 100% wide).



By default, Bootstrap columns include some horizontal padding. To remove this, add a `no-gutters` class to the relevant row element. Try this on the first row:

index.html

```
...

  <h1>Grids</h1>

  <div class="row no-gutters">
    ...
```

There's no more padding on the 4 divs.

The grid system has many other features, including classes for things like alignment, ordering, offsetting, nesting, and more. You can refer to the Bootstrap documentation for further information:

https://getbootstrap.com/docs/4.3/layout/grid/

Icon Fonts

Earlier versions of Bootstrap included an icon font -- namely, *Glyphicons*. Glyphicons isn't a 'font' per se, but rather a collection of useful icons, much like dingbats such as *Webdings* or *Wingdings*. Like font characters, the icons are vector-based so that you don't experience pixelation issues. Moreover, changing

icon colours is a simple matter of utilising the CSS `color` property.

Bootstrap now recommends that one uses either *Font Awesome*, *Iconic*, or *Octicons*, rather than Glyphicons. We'll go for Font Awesome.

You *could* download the Font Awesome set, and extract it into your working folder. However, to avoid unnecessary complication, just use the CDN (Content Delivery Network) for now:

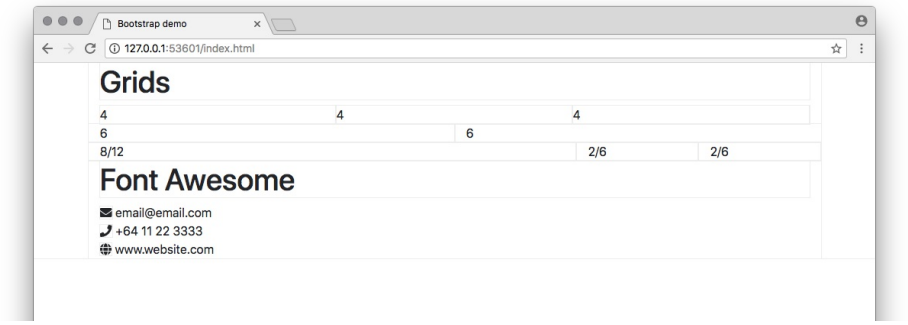
index.html

```
...
<!-- font awesome -->
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.1/css/all.css">

<!-- outline all elements -->
...
```

You can now insert icons using the `i` (italic) tag. Add the following code to see this in action:

index.html

```
    ...
    <div class="col-6 col-sm-2"> 2/6 </div>
  </div>

  <h1>Font Awesome</h1>

  <i class="fas fa-envelope"></i> email@email.com
  <br />
  <i class="fas fa-phone"></i>    +64 11 22 3333
  <br />
  <i class="fas fa-globe"></i>    www.website.com

</div>

...
```



The syntax should be obvious enough, but you'll need the reference for the rest of the icon names:

https://fontawesome.com/icons

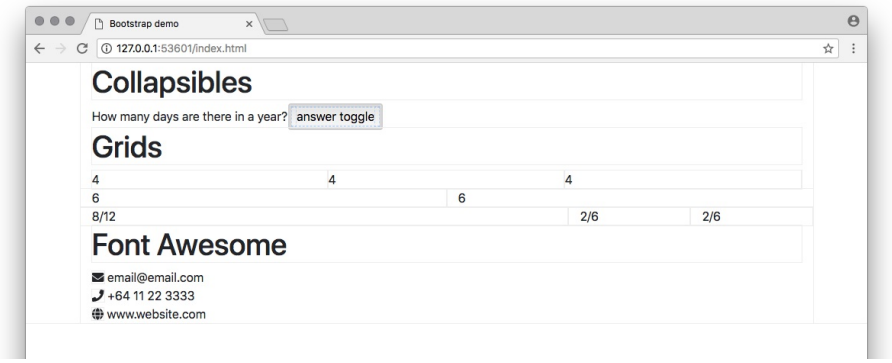
Collapsible Content

Collapsible content areas can be hidden or shown at the click of a button, with some animation effect to smooth the transition. Add the following code above your grid to see what happens:

index.html

```
...

<body>

  <div class="container">

    <h1>Collapsibles</h1>

    How many days are there in a year?

    <button data-toggle="collapse" data-target="#answer"> answer toggle </button>

    <div id="answer" class="collapse">
      365.2422
    </div>

    <h1>Grids</h1>

    ...
```

Note how the `data-target="#answer"` is used to target the relevant CSS selector (in this case an `id`). Always be sure to include the necessary `data-toggle="collapse"` and `class="collapse"` attributes. Preview the result and click the "answer toggle" button to see the effect in action:



Using panel-groups you can create more complex collapsible elements, such as an accordion. Refer to the Bootstrap documentation for more info:

https://getbootstrap.com/docs/4.3/components/collapse/#accordion-example

Further Reading

There is *a lot more* to Bootstrap than what's covered here. You can find out more using the Bootstrap documentation:

https://getbootstrap.com/docs/4.3/

If you're looking for a simpler (but less feature-packed) framework, perhaps give Skeleton a try:

http://getskeleton.com/

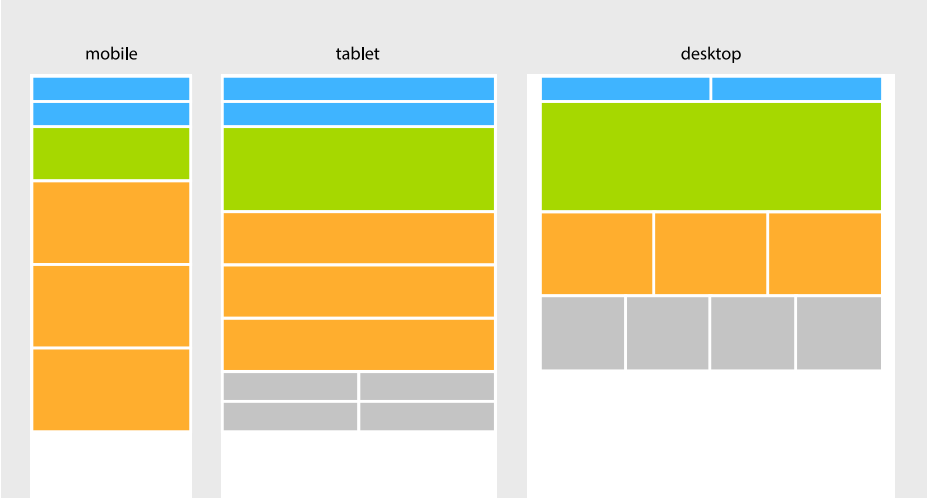
Foundation is another, more 'heavy-weight' option worth exploring:

http://foundation.zurb.com/

Layout Tasks

The lesson files include two layout challenges. Getting the exact heights of elements correct isn't important -- getting the layouts to adapt responsively is what's important here.

Let's get started with `layout_01.svg`:



Create a new file in your 289.101 "bootstrap" directory named "layout_1.html".

```
bootstrap
├─ css
├─ js
├─ index.html
└─ layout_1.html
```

Add the following code:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>Layout 1</title>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <link rel="stylesheet" href="css/bootstrap.min.css" />

  <style>
    * { outline: white 8px solid }

    .blue {
      background-color: #3FB4FF;
      height: 50px;
    }
  </style>
</head>

<body>
```

```
  <div class="container">

    <div class="row">
      <div class="col-lg-6 blue"></div>
      <div class="col-lg-6 blue"></div>
    </div>

  </div>
</body>

</html>
```

To complete the layout, you will need to make use of media queries that match Bootstrap's viewport widths, as well as classes to hide/show elements at different break-points.

Matching Media Queries with Bootstrap Break-Points

In addition to using Bootstrap's grid classes, you may wish to add your own media queries that match the Bootstrap break-points. Add the following code to your internal stylesheet:

```
    ...

    /* xs devices */
    body { background-color: red }

    /* sm devices */
    @media (min-width: 576px) {
      body { background-color: orange }
    }

    /* md devices */
    @media (min-width: 768px) {
      body { background-color: yellow }
    }

    /* lg devices */
    @media (min-width: 992px) {
      body { background-color: green }
    }

    /* xl devices */
    @media (min-width: 1200px) {
      body { background-color: blue }
```

```
    }
  }
  </style>
</head>

...
```

Test and confirm that the colour of your webpage background changes as you resize the window. You can now add CSS code specific to each of the `@media` conditions (with the page background colour to guide you).

Think mobile-first when using media queries. In other words: all of the CSS that sits above your media queries should target "xs" mobile devices -- meaning that larger devices are the 'exception to the rule'. Also, make sure that the `@media` conditions appear in the correct order (narrowest first).

Hiding & Showing Elements

If you need to hide elements beyond certain break-points, add the appropriate `d-...` class(es):

- `d-none`
- `d-sm-none`
- `d-md-none`
- `d-lg-none`
- `d-xl-none`

The table below provides a list of the combinations you are likely to use:

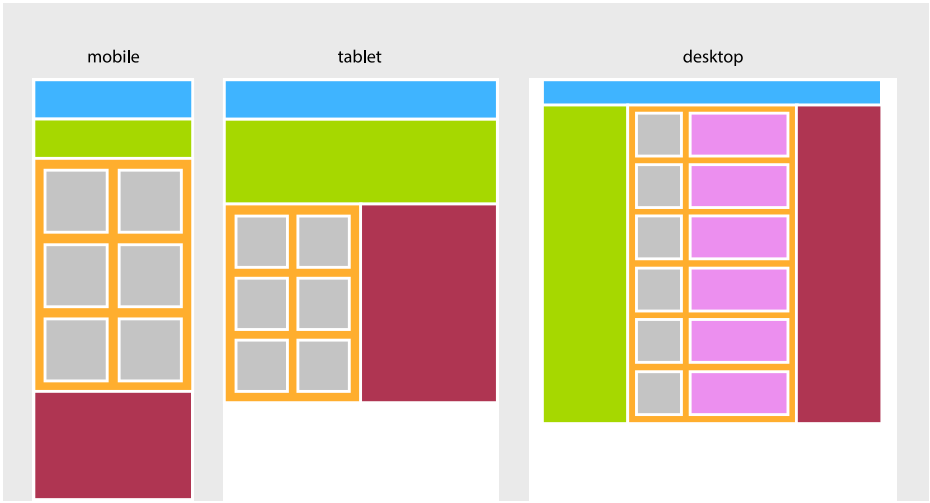
screen size	**class**
hidden on all	`d-none`
hidden only on xs	`d-none d-sm-block`
hidden only on sm	`d-sm-none d-md-block`
hidden only on md	`d-md-none d-lg-block`
hidden only on lg	`d-lg-none d-xl-block`
hidden only on xl	`d-xl-none`

visible on all	`d-block`
visible only on xs	`d-block d-sm-none`
visible only on sm	`d-none d-sm-block d-md-none`
visible only on md	`d-none d-md-block d-lg-none`
visible only on lg	`d-none d-lg-block d-xl-none`
visible only on xl	`d-none d-xl-block`

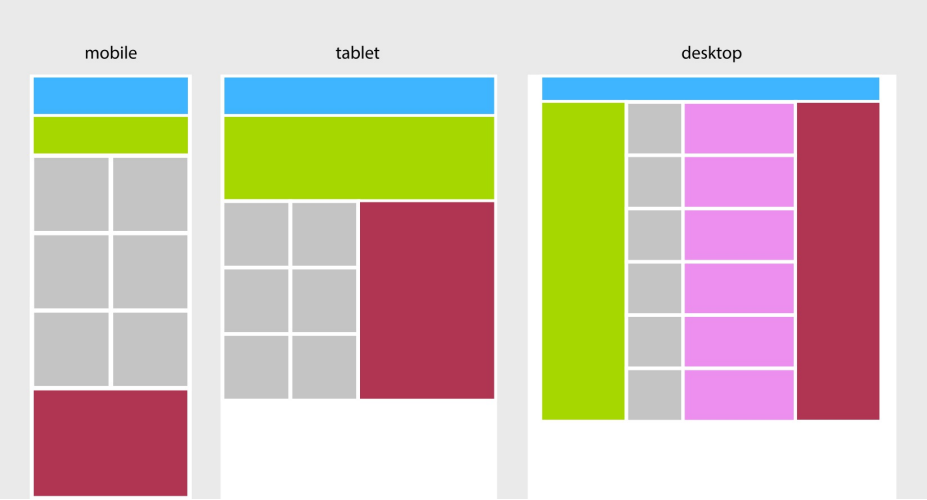
The `d-none d-sm-block` is required for hiding the grey blocks in *layout_01* (mobile view):

Layout 02

The purple elements in *layout_02* (mobile and tablet views) also require `d-...` classes:



Note that the orange div need not be visible. It serves as a container for the grey and pink elements. Without margins, these will block-out any orange background colour showing through. Therefore, your final *layout_02* looks more like this:



You might find the CSS `height: auto` useful for instructing bootstrap columns to match the height of the row (i.e. where the green element jumps from something with a finite height on tablet, to 100% high on desktop).

end