Comp 324/424 - Client-side Web Design

Fall Semester 2018 - Week 6

Dr Nick Hayward

CSS Basics - fonts - part 1

- fonts can be set for the body or within an element's specific ruleset
- we need to specify our font-family,

```
body {
font-family: "Times New Roman", Georgia, Serif;
}
```

- value for the font-family property specifies preferred and fall-back fonts
  - *Times New Roman, then the browser will try Georgia and Serif*
  - *" " - quotation marks for names with spaces...*

n.b. " " added due to CSS validator requesting this standard - it's believed to be a legacy error with the validator...

CSS Basics - fonts - part 2

- useful to be able to modify the size of our fonts as well

```
body {
  font-size: 100%;
}
h3 {
  font-size: x-large;
}
p {
  font-size: larger;
}
p.p1 {
  font-size: 1.1em;
}
```

- set base font size to 100% of font size for a user's web browser
- scale our other fonts relative to this base size
  - *CSS absolute size values, such as `x-large`*
  - *font sizes relative to the current context, such as `larger`*
  - *em are meta-units, which represent a multiplier on the current font-size*
  - *relative to current element for required font size*
  - *1.5em of 12px is effective 18px*
- em font-size scales according to the base font size
  - *modify base font-size, em sizes adjust*
- try different examples at
  - *W3 Schools - font-size*

Demo - CSS Fonts

- Demo - CSS Fonts
- JSFiddle - CSS Fonts

CSS Basics - fonts - part 3

- `rem` unit for font sizes
- size calculated against root of document

```css
body {
  font-size: 100%;
}
p {
  font-size: 1.5rem;
}
```

- element font-size will be `root size * rem size`
  - *e.g. body font-size is currently 16px*
  - *rem will be `16 * 1.5`*

CSS Basics - custom fonts

- using fonts and CSS has traditionally been a limiting experience
- reliant upon the installed fonts on a user's local machine
- JavaScript embedding was an old, slow option for custom fonts
- web fonts are a lot easier
- Google Fonts
  - *from the font options, select*
  - *required fonts*
  - *add a `<link>` reference for the font to our HTML document*
  - *then specify the fonts in our CSS*

```
font-family: 'Roboto';
```

Demo - CSS Custom Fonts

- Demo - CSS Custom Fonts
- JSFiddle - CSS Custom Fonts

CSS Basics - reset options

- to help us reduce browser defaults, we can use a CSS reset
- reset allows us to start from scratch
- customise aspects of the rendering of our HTML documents in browsers
- often considered a rather controversial option
- considered controversial for the following primary reasons
  - *accessibility*
  - *performance*
  - *redundancy*
- use resets with care
- notable example of resets is Eric Meyer
  - *discussed reset option in May 2007 blog post*
- resets often part of CSS frameworks...

Demo - CSS Reset - Before

Browser default styles are used for

- `<h1>`, `<h3>`, and `<p>`
- Demo - CSS Reset Before

Demo - CSS Reset - After

Browser resets are implemented using the Eric Meyer stylesheet.

- Demo - CSS Reset After

CSS - a return to inline styles

- *inline* styles are once more gaining in popularity
  - *helped by the rise of React &c.*
- for certain web applications they are now an option
  - *allow us to dynamically maintain and update our styles*
- their implementation is not the same as simply embedding styles in HTML
  - *dynamically generated*
  - *can be removed and updated*
  - *can form part of our maintenance of the underlying DOM*
- inherent benefits include
  - *no cascade*
  - *built using JavaScript*
  - *styles are dynamic*

CSS - against inline styles

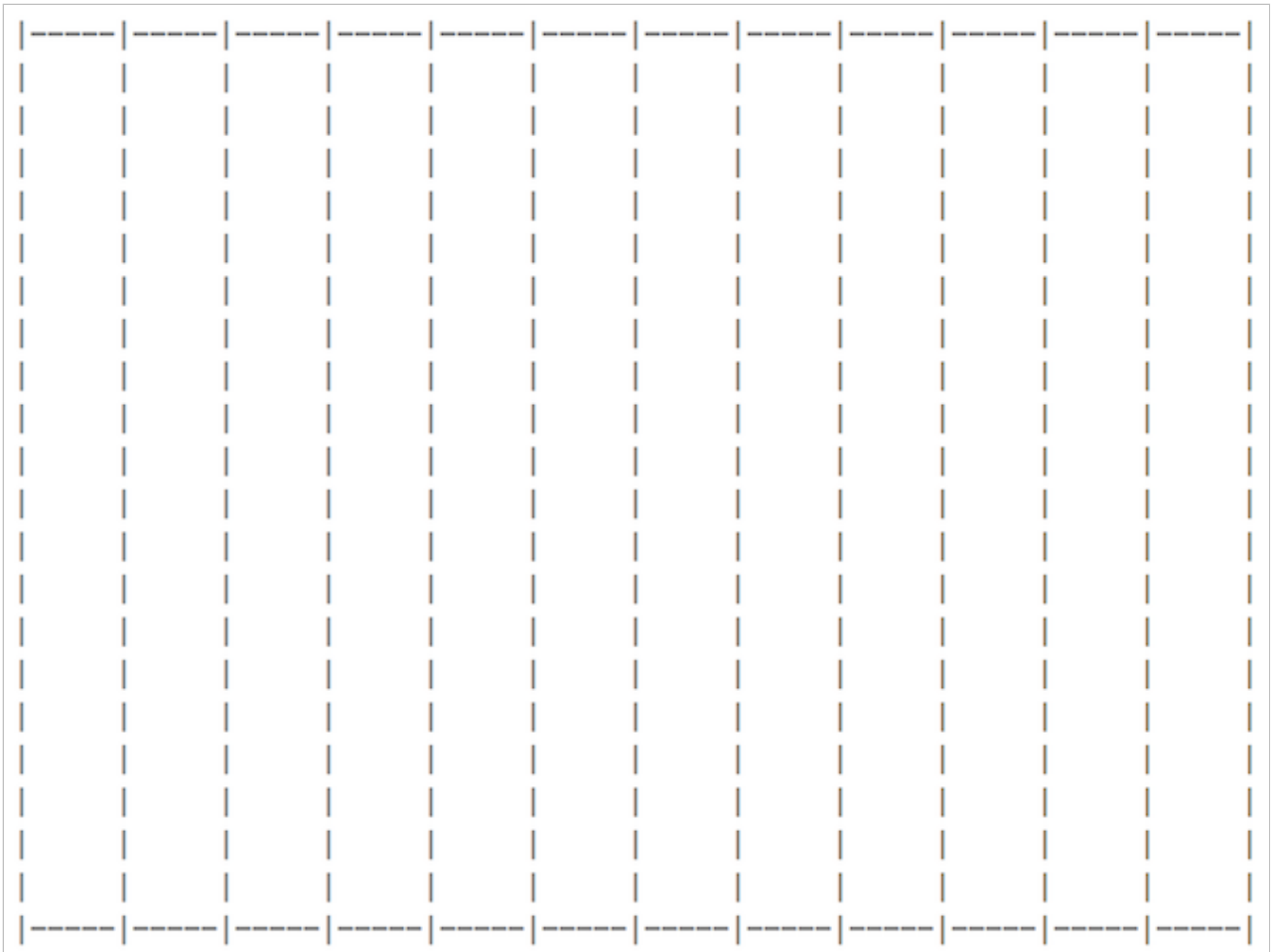
- CSS is designed for styling
  - *this is the extreme end of the scale - in effect, styling is only done with CSS*
- abstraction is a key part of CSS
  - *by separating out concerns, i.e. CSS for styling, our sites are easier to maintain*
- *inline* styles are too specific
  - *again, abstraction is the key here*
- some styling and states are easier to represent using CSS
  - *psuedoclasses etc, media queries...*
- CSS can add, remove, modify classes
  - *dynamically update selectors using classes*

CSS grid layout - part 1

intro

- grid designs for page layout, components...
  - *increasingly popular over the last few years*
  - *useful for creating responsive designs*
- quick and easy to layout a scaffolding framework for our structured content
- create boxes for our content
  - *then position them within our grid layout*
- content can be stacked in a horizontal and vertical manner
  - *creating most efficient layout for needs of a given application*
- another benefit of CSS grids is that they are framework and project agnostic
  - *thereby enabling easy transfer from one to another*
- concept is based upon a set number of columns per page with a width of 100%
- columns will increase and decrease relative to the size of the browser window
- also set break points in our styles
  - *helps to customise a layout relative to screen sizes, devices, aspect ratios...*
  - *helps us differentiate between desktop and mobile viewers*

Image - Grid Layout



Grid Layout - Columns and rows

CSS grid layout - part 2

grid.css

- build a grid based upon 12 columns
  - *other options with fewer columns as well*
- tend to keep our grid CSS separate from the rest of the site
  - *maintain a CSS file just for the grid layout*
- helps abstract the layout from the remaining styles
  - *makes it easier to reuse the grid styles with another site or application*
- add a link to this new stylesheet in the `head` element of our pages

```
<link rel="stylesheet" type="text/css" href="assets/styles/grid.css">
```

or

```
<link rel="stylesheet" href="assets/styles/grid.css">
```

- ensure padding and borders are included in total widths and heights for an element
  - *reset `box-sizing` property to include the `border-box`*
  - *resetting box model to ensure padding and borders are included*

```
* {
box-sizing: border-box;
}
```

CSS grid layout - example - part 3

grid.css

- set some widths for our columns, 12 in total
  - *each representing a proportion of the available width of a page*
  - *from a 12th to the full width of the page*

```
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

- classes allow us to set a column span for a given element
  - *from 1 to 12 in terms of the number of grid columns an element may span*

CSS grid layout - example - part 4

grid.css

- then set some further styling for each abstracted `col-` class

```
[class*="col-"] {
  position: relative;
  float:left;
  padding: 20px;
  border: 1px solid #333;
}
```

- create columns by wrapping our content elements into rows
- each row always needs 12 columns

```
<div class="row">
  <div class="col-6">left column</div>
  <div class="col-6">right column</div>
</div>
```

CSS grid layout - example - part 5

grid.css

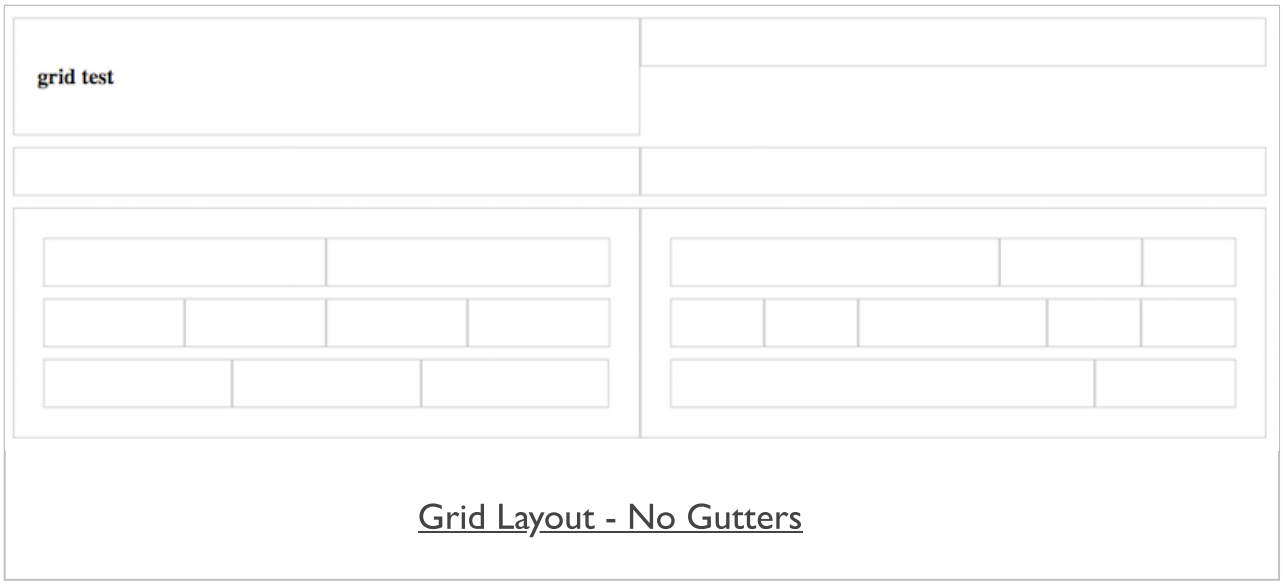
- due to the initial CSS of float left, each column is floated to the left
- columns are interpreted by subsequent elements in the hierarchy as non-existent
  - *initial placement will reflect this design*
- prevent this issue in layout, add the following CSS to grid stylesheet

```
.row:before, .row:after {
  content: "";
  clear: both;
  display: block;
}
```

- benefit of the clearfix, `clear: both`
  - *make row stretch to include columns it contains*
  - *without the need for additional markup*

DEMO - Grid Layout 1 - no gutters

Image - Grid Layout I



Grid Layout - No Gutters

CSS grid layout - example - part 6

grid.css

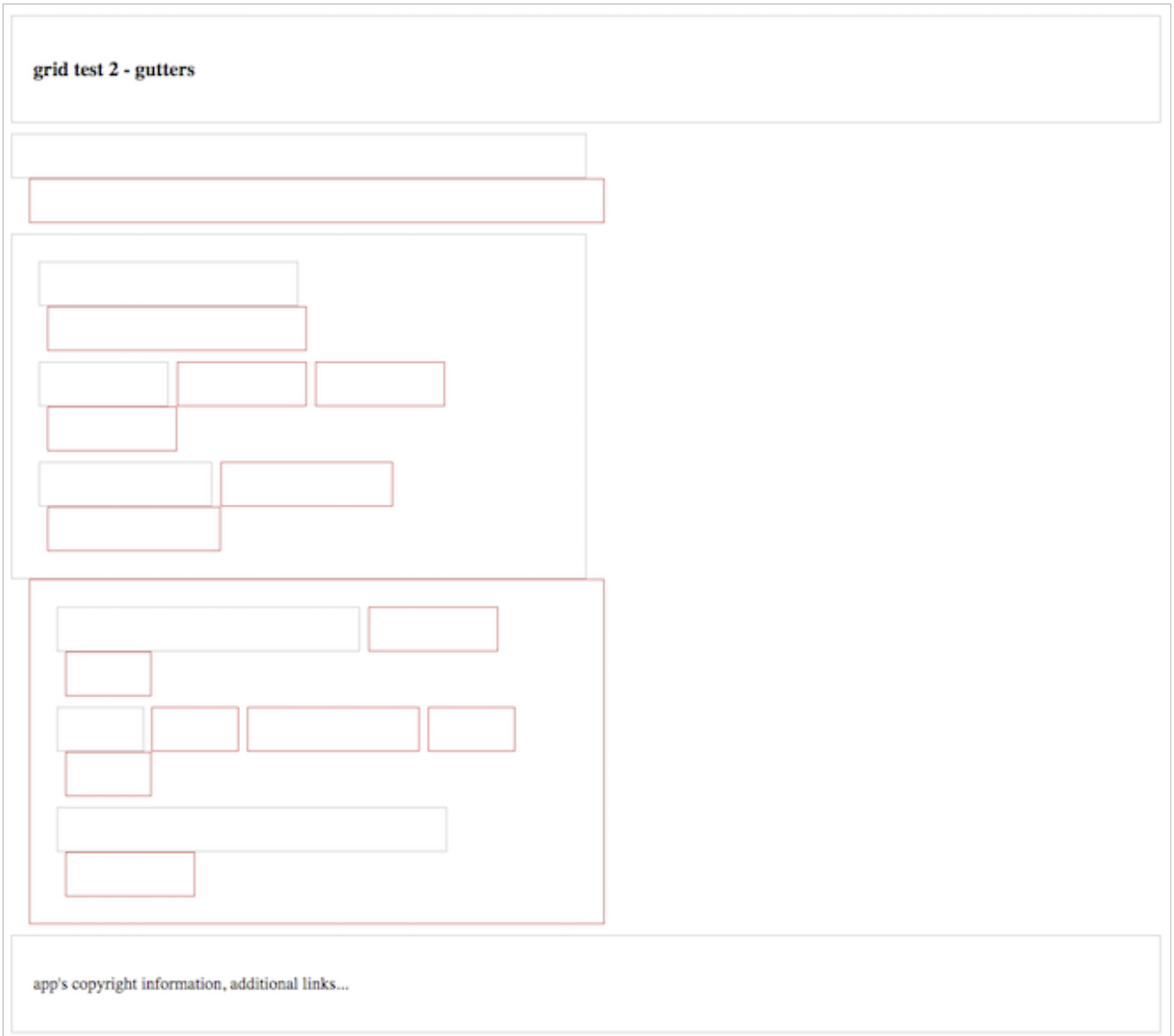
- add gutters to our grid to help create a sense of space and division in the content
- simplest way to add a gutter to the current grid css is to use padding
  - *rows can use padding, for example*

```
.row {
  padding: 5px;
}
```

- issue with simply adding padding to the columns
  - *margins are left in place, next to each other*
  - *column borders next to each with no external column gutter*
- fix this issue by targeting columns that are a sibling to a preceding column
- means we do not need to modify the first column, only subsequent siblings

```
[class*="col-"] + [class*="col-"] {
  margin-left: 1.6%;
}
```

Image - Grid Layout 2



Grid Layout - Gutters Overflow

CSS grid layout - part 7

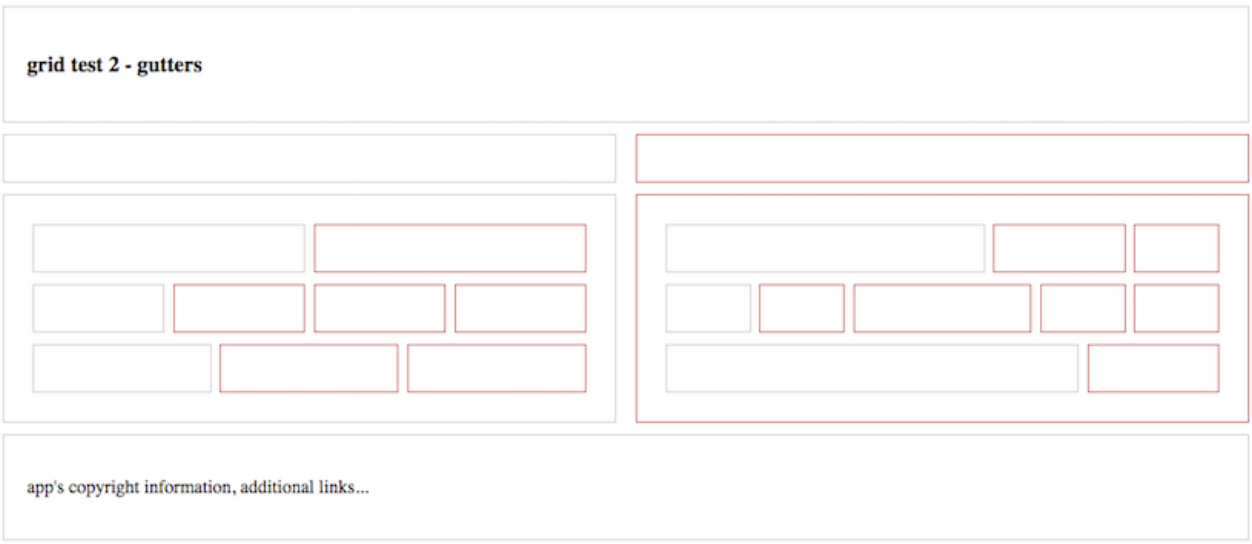
grid.css

- to fix this issue we recalculate permitted % widths for our columns in the CSS
  - *we now have % widths as follows*

```
.col-1 {width: 6.86%;}
.col-2 {width: 15.33%;}
.col-3 {width: 23.8%;}
.col-4 {width: 32.26%;}
.col-5 {width: 40.73%;}
.col-6 {width: 49.2%;}
.col-7 {width: 57.66%;}
.col-8 {width: 66.13%;}
.col-9 {width: 74.6%;}
.col-10 {width: 83.06%;}
.col-11 {width: 91.53%;}
.col-12 {width: 100%;}
```

- DEMO - Grid Layout 2 - gutters

Image - Grid Layout 3



Grid Layout - Gutters

CSS grid layout - part 8

media queries

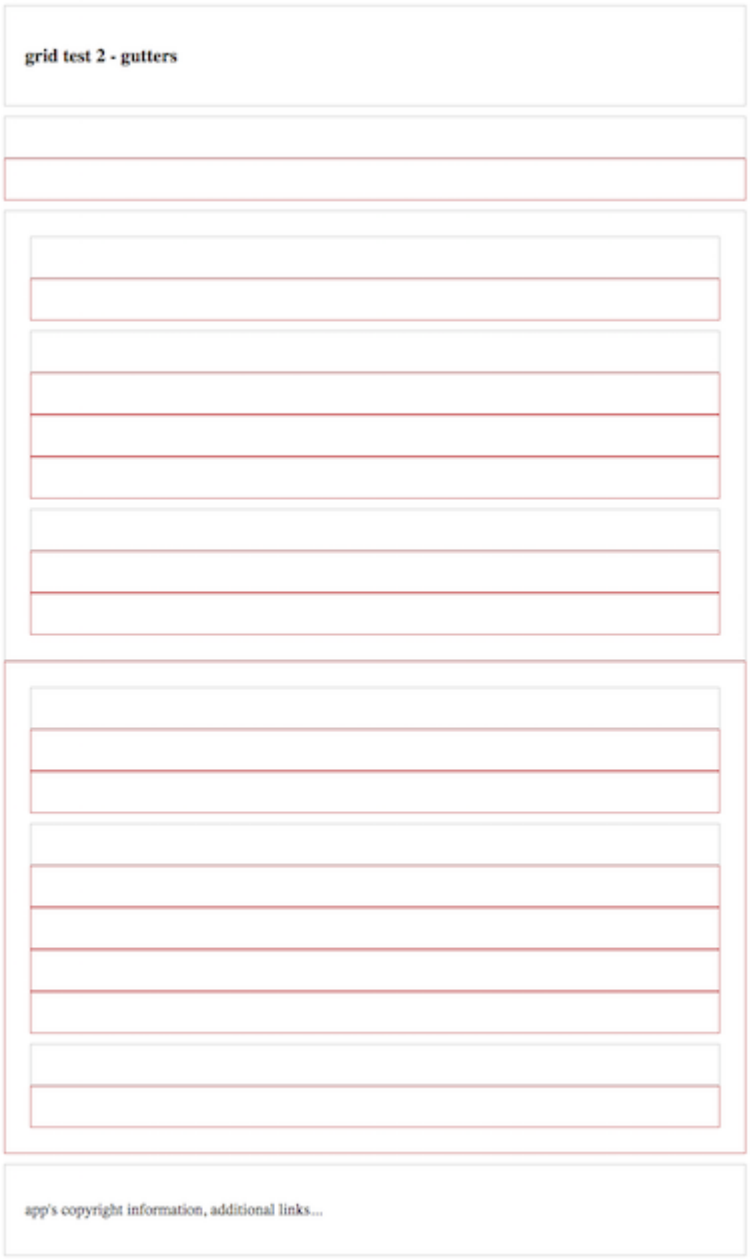
- often need to consider a mobile-first approach
- introduction of CSS3, we can now add **media queries**
- modify specified rulesets relative to a given condition
  - *eg: screen size for a desktop, tablet, and phone device*
- media queries allow us to specify a breakpoint in the width of the viewport
  - *will then trigger a different style for our application*
- could be a simple change in styles
  - *such as colour, font etc*
- could be a modification in the grid layout
  - *effective widths for our columns per screen size etc...*

```
@media only screen and (max-width: 900px) {
  [class*="col-"] {
  width: 100%;
  }
}
```

- gutters need to be removed
  - *specifying widths of 100% for our columns*

```
[class*="col-"] + [class*="col-"] {
  margin-left:0;
}
```

Image - Grid Layout 4



Grid Layout - Media Queries

CSS3 Grid - intro

- gid layout with CSS is useful for structure and organisation
  - *applied to HTML page*
- usage similar to table for structuring data
- in its basic form
  - *enables developers to add columns and rows to a page*
- grid layout also permits more complex, interesting layout options
  - *e.g. overlap and layers...*
- further information on MDN website,
  - *MDN - CSS Grid Layout*

CSS3 Grid - general concepts & usage

- grid may be composed of rows and columns
  - *thereby forming an intersecting set of horizontal and vertical lines*
- elements may be added to the grid with reference to this structured layout

Grid layout in CSS includes the following general features,

- additional tracks for content
  - *option to create more columns and rows as needed to fit dynamic content*
- control of alignment
  - *align a grid area or overall grid*
- control of overlapping content
  - *permit partial overlap of content*
  - *an item may overlap a grid cell or area*
- placement of items - explicit and implicit
  - *precise location of elements &c.*
  - *use line numbers, names, grid areas &c.*
- variable track sizes - fixed and flexible, e.g.
  - *specify pixel size for track sizes*
  - *or use flexible sizes with percentages or new `fr` unit*

CSS3 Grid - grid container

- define an element as a grid container using
  - *`display: grid` or `display: inline-grid`*
- any children of this element become *grid items*
  - *e.g.*

```
.wrapper {
  display: grid;
}
```

- we may also define other, child nodes as a `grid` container
  - *any direct child nodes to a grid container are now defined as grid items*

CSS3 Grid - what is a grid track?

- rows and columns defined with
  - *`grid-template-rows` and `grid-template-columns` properties*
- in effect, these define *grid tracks*
- as MDN notes,
  - *"a grid track is the space between any two lines on the grid.""*
  - *(https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout)*
- so, we may create both row and column tracks, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px 200px 200px;
}
```

- `wrapper` class now includes three defined columns of width 200px
  - *thereby creating three tracks*
- *n.b.* a track may be defined using any valid length unit, not just px...

CSS3 Grid - `fr` unit for tracks - part I

- CSS Grid now introduces an additional length unit for tracks, `fr`
- `fr` unit represents fractions of the space available in the current grid container
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

- we may also apportion various space to tracks, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 2fr 1fr 1fr;
}
```

- creates three tracks in the grid
  - *but overall space effectively now occupies four parts*
  - *two parts for `2fr`, and one part each for remaining two `1fr`*

CSS3 Grid - `fr` unit for tracks - part 2

- we may also be specific in this sub-division of parts in tracks, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px 1fr 1fr;
}
```

- first track will occupy a width of `200px`
  - *remaining two tracks will each occupy 1 fraction unit*

CSS3 Grid - `repeat()` notation for `fr` - part I

- for larger, repetitive grids, easier to use `repeat()`
  - *helps define multiple instances of the same track*
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
}
```

- this creates four separate tracks - each defined as `1fr` unit's width

CSS3 Grid - `repeat()` notation for `fr` - part 2

- `repeat()` notation may also be used as part of the track definition
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: 200px repeat(4, 1fr) 100px;
}
```

- this example will create
  - *one track of `200px` width*
  - *then four tracks of `1fr` width*
  - *and finally a single track of `100px` width*
- `repeat()` may also be used with multiple track definitions
  - *thereby repeating multiple times*
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr 2fr);
}
```

- this will now create eight tracks
  - *the first four of width `1fr`*
  - *and the remaining four of `2fr`*

CSS3 Grid - implicit and explicit grid creation

- in the above examples
  - *we simply define tracks for the columns*
  - *and CSS grid will then apportion content to required rows*
- we may also define an explicit grid of columns and rows
  - e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(2 1fr);
  grid-auto-rows: 150px;
}
```

- this slightly modifies an implicit grid to ensure each row is `200px` tall

CSS3 Grid - track sizing

- a grid may require tracks with a minimum size
  - *and the option to expand to fit dynamic content*
- e.g. ensuring a track does not collapse below a certain height or width
  - *and that it has the option to expand as necessary for the content...*
- CSS Grid provides a `minmax()` function, which we may use with rows
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(2 1fr);
  grid-auto-rows: minmax(150px, auto);
}
```

- ensures each row will occupy a minimum of `150px` in height
  - *still able to stretch to contain the tallest content*
  - *whole row will expand to meet the `auto` height requirements*
  - *thereby affecting each track in the row*

CSS3 Grid - grid lines

- a grid is defined using *tracks*
  - *and not lines in the grid*
- created grid also helps us with positioning by providing numbered lines
- e.g. in a three column, two row grid we have the following,
  - *four lines for the three vertical columns*
  - *three lines for the two horizontal rows*
- such lines start at the left for columns, and at the top for rows
- *n.b.* line numbers start relative to written script
  - *e.g left to right for western, right to left for arabic...*

CSS3 Grid - positioning against lines

- when we place an item in a grid
  - *we use these lines for positioning, and not the tracks*
- reflected in usage of
  - *`grid-column-start`, `grid-column-end`, `grid-row-start`, and `grid-row-end` properties.*
- items in the grid may be positioned from one line to another
  - *e.g. column line 1 to column line 3*
- *n.b.* default span for an item in a grid is one track,
  - *e.g. define column start and no end - default span will be one track...*
  - *e.g.*

```
.content1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}
```

CSS3 Grid - grid cell & grid area

grid cell

- a *cell* is the smallest unit on the defined grid layout
- it is conceptually the same as a cell in a standard table
- as content is added to the grid, it will be stored in one cell

grid area

- we may also store content in multiple cells
  - *thereby creating grid areas*
- grid areas must be rectangular in shape
- e.g. a grid area may span multiple row and column tracks for required content

CSS3 Grid - add some gutters

- gutters may be created using the *gap* property
  - *available for either column or row*
  - `column-gap` and `row-gap`
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr 2fr);
  column-gap: 5px;
  row-gap: 10px;
}
```

- *n.b.* any space used for gaps will be determined prior to assigned space for `fr` tracks

CSS3 Grid - structure and layout

fun exercise

Choose one of the following app examples,

- **sports** website for latest scores and updates
  - *e.g. scores for current matches, statistics, team data, player info &c.*
- **shopping** website
  - *product listings and adverts, cart, reviews, user account page &c.*
- **restaurant** website
  - *introductory info, menus, sample food images, user reviews &c.*

Then, consider the following

- use of a **grid** to layout your example pages
  - *where is it being used?*
  - *why is it being used for a given part of the UI?*
- how is the defined **grid** layout working with the **box model**?
- rendering of **box model** in the main content relative to **grid** usage
  - *i.e. box model updates due to changes in content*

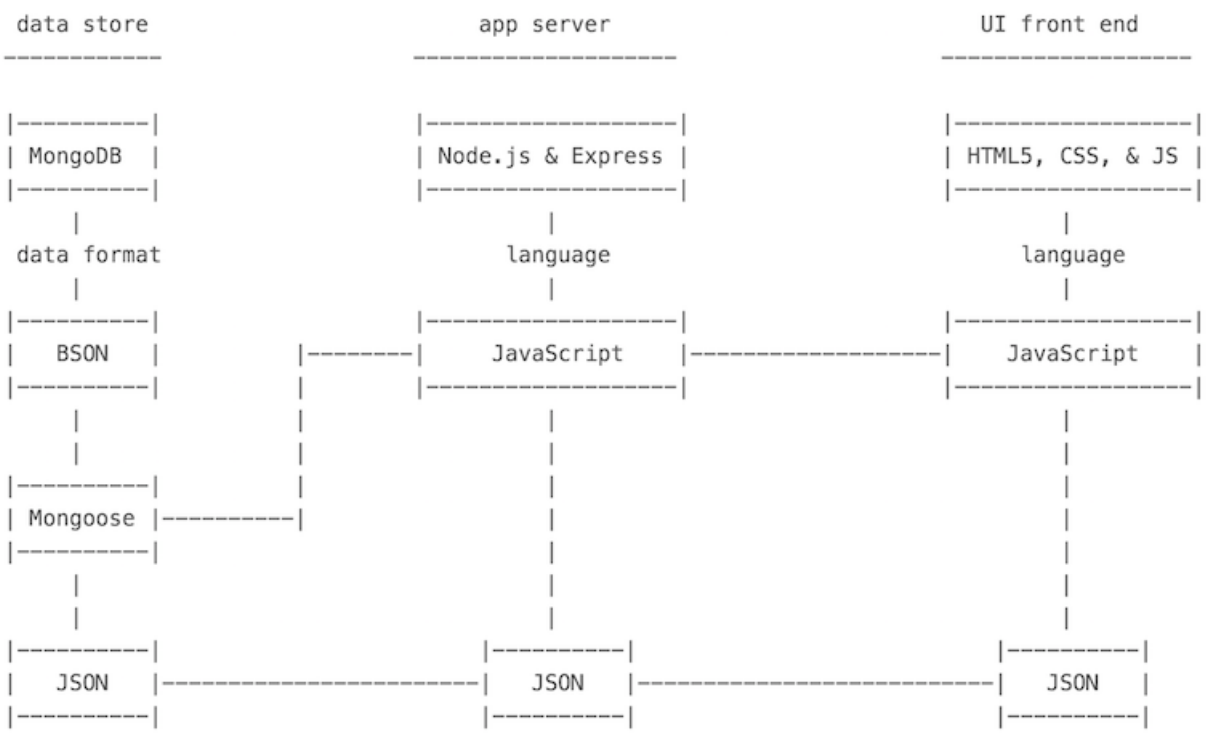
CSS3 Grid - working examples

- grid basic - page zones and groups
- grid basic - article style page
- grid layout - articles with scroll

building a web app - sample outline of underlying structure

- apps developed using a full JavaScript stack
- using and incorporating JS into each part of app's development
  - *UI front-end*
  - *app server and management*
  - *data store and management*
- Technologies will include
  - *front-end: HTML5, CSS, JS...*
  - *app server: Node.js, Express...*
  - *data store: MongoDB, Redis, Mongoose...*
- Data format is JSON

Image - building a web app - sample outline



JS full-stack outline

n.b. *I've explicitly omitted any arrows for flow within this diagram. This is something we'll return to as we start to work with Node.js, Mongoose, and MongoDB.*

Demos

- CSS - Fonts
  - *CSS Fonts*
  - *CSS Custom Fonts*
  - *CSS Reset Before*
  - *CSS Reset After*
- CSS - Grid
  - *grid basic - page zones and groups*
  - *grid basic - article style page*
  - *grid layout - articles with scroll*
- JSFiddle tests - CSS
  - *JSFiddle - CSS Box Model Padding*
  - *JSFiddle - CSS Fonts*
  - *JSFiddle - CSS Custom Fonts*

Resources

- Google Web Fonts
- MDN - CSS3 Grid
- W3 Schools - CSS Grid View