

# Flexbox, Grid, & Responsive Design

Web Development & Consulting Club  
Workshop One, Session Three



Auckland  
ICT Graduate School

# Flexbox



Auckland  
ICT Graduate School

# Flexbox

- **Flexbox**, is concerned primarily with laying out, aligning and distributing elements in a single row or column.
- Can be used for much more complex layouts than we will discuss, however it is intended to be used in conjunction with CSS Grid
  - A combination of the two technologies should be used for those more complex layouts
- Flexbox layouts utilize two components, the **container** in which the layout is described and where elements appear; and **items** which are laid out according to rules set by the container



# Flexbox containers - flex-direction

- Containers are any element into which we can place other elements that contain one key CSS property: `display: flex;`
  - With this property set, we gain access to the other flex-related properties that can control the layout for any items nested within the container
- The `flex-direction` property controls the direction child components are laid out, and can have the following values:
  - `row`: Horizontal alignment, left-to-right
  - `row-reverse`: Horizontal alignment, right-to-left
  - `column`: Vertical alignment, top-to-bottom
  - `column-reverse`: Vertical alignment, bottom-to-top

# Flexbox containers - align-items

- The `align-items` property determines how child elements are aligned with respect to each other. It can have the following values:
  - `center`: Aligns items by their centerlines
  - `flex-start`: Aligns items by their top / left margins
  - `flex-end`: Aligns items by their bottom/right margins
  - `stretch`: Causes items to stretch to become equal height
  - `baseline`: Aligns items by the baselines of text content (where an underline would appear)

# Flexbox containers - justify-content

- The `justify-content` property determines how children are distributed within the container, and can have the following values:
  - `flex-start` or `flex-end`: Arranges items much like floating left or right would do, with the items being stacked to either side of the container.
  - `center`: Places elements with their edges against each other with an equal amount of space to the left and right sides.
  - `space-between`: Places the first and last items to the start and end of the container, with the rest of the elements spaced with equal spaces between these.
  - `space-evenly`: Extends this so that there is an equal share of space between the edge of the container and the first/last items.
  - `space-around`: Divides the space available into  $2n$  pieces, then places one of these spaces on either side of each element

# More flexbox properties

Flexbox containers also have controls for what happens if there is not enough space in the row/column for all items - `flex-wrap`, and how to manage items aligned with `align-items` if wrapping does occur - `align-content`.

Items that are placed inside flexbox containers also have a number of properties they can set to change how they are laid out. The first is `order`; this determines where the element it is applied to will be placed in the layout. By default they will be placed as they are defined in the HTML, but `order` allows this to be changed.

The other of particular interest is `align-self`, which lets an item override the `align-items` property set in the container

# Grid



Auckland  
ICT Graduate School



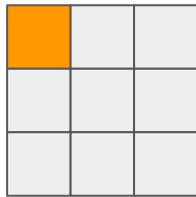
# Grid

Grid expands on the ideas given by flexbox into 2-dimensions, making it perfectly suited to layouts of a whole page rather than just a small part.

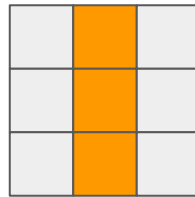
As with flexbox, Grid deals with **containers** and **items** for its layouts, but it also introduces **tracks** (rows and columns within the layout), **lines** (the 'borders' between tracks), and **areas** (rectangular groups of items within the layout)



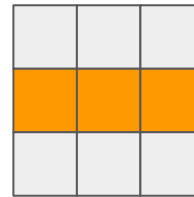
Container



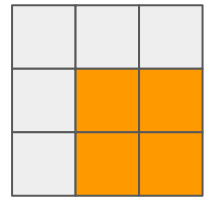
Item (cell)



Track (col)



Track (row)



Area

# Grid container

- To set a container as a grid, use `display: grid;`
- This is not useful on its own - two extra properties are required to control the number and sizes of columns and rows in the grid

```
grid-template-columns: 1fr 100px auto 100px 2fr;  
grid-template-rows: 1fr 2fr 3fr;
```

- This setup creates a 5 column, 3 row grid. The number of measurements provided determine the number of rows or columns, while each value determines the width or height of the column or row

# Grid container - Units

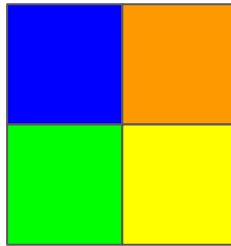
In the `grid-template-column` property from before, you will have noticed two new unit types that you haven't encountered yet:

- **auto**: indicates that the row or column should have its size determined automatically by the content contained within it
  - E.g. if the content is `100px` x `100px`, then that will be the width and height of that row/column
- **fr**: represents a **f**raction of available space. Used to create proportional layouts.
  - A `grid-template-column` definition of `1fr 1fr 1fr` will create 3 equally sized columns, while `5fr 1fr 1fr` will create a setup where the left-most column occupies 5/7<sup>th</sup>s of the page and the remainder is shared by the other 2 columns

# Grid container - Layout

- Once a grid container has been defined, items can be laid out inside it.
- In Grid, as we are dealing with two dimensions, we need to layout with that in mind:
  - Elements are laid out as they are encountered, left to right, top to bottom, with each item occupying a cell in the grid

```
<div class="container">  
  <div id="blue">  
  <div id="orange">  
  <div id="green">  
  <div id="yellow">  
</div>
```



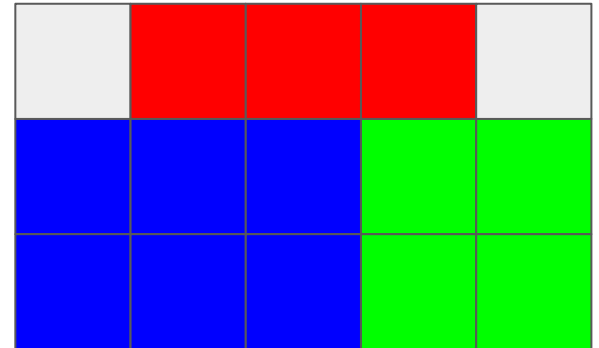
```
.container {  
  display: grid;  
  grid-template-columns: 1fr 1fr;  
  grid-template-rows: 1fr 1fr;  
}
```

# Grid container - Named areas

To make children span multiple grid cells, we can use `grid-template-areas`

- This property allows us to name areas within our grid
- Children can occupy specific named areas

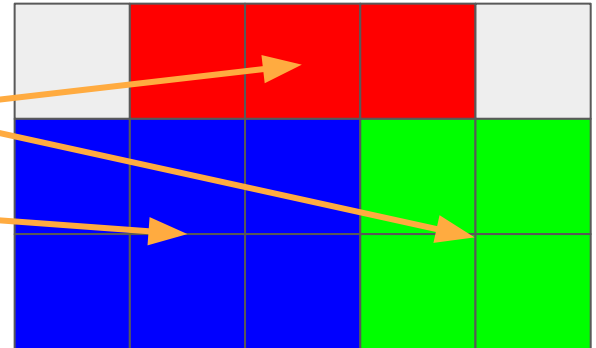
```
grid-template-columns: repeat(5, 1fr);  
grid-template-rows: repeat(3, 1fr);  
grid-template-areas:  
    ". r r r ."  
    "b b b g g"  
    "b b b g g";
```



# Grid Items - Named Areas

With named areas defined, items can place themselves into the appropriate area using the `grid-area` property. As they get to choose their position, the order in which they are placed in the HTML no longer matters

```
<div class="container">  
  <div style="grid-area: g">  
  <div style="grid-area: r">  
  <div style="grid-area: b">  
</div>
```



# More grid

- Grid is a very complex topic, and there are a great many properties and values that we have not discussed. For example:
  - Lines can be adjusted to space between tracks
  - Lines can be named so that their areas can be defined with respect to lines rather than laid out as a visual grid
  - Items in the grid cells can be aligned and justified as with flexbox.
- There is an excellent [Grid guide](#) created by css-tricks that explains each of these properties in detail, and gives visual aids to understanding each
- However, we can use what's covered in this lecture to begin designing complex layouts with ease!

# Responsive Design



Auckland  
ICT Graduate School



# The story so far

---

So far the web pages you have developed have been fairly simple. Elements have been placed on the page and mostly allowed to lay themselves out, and no consideration has been made for the size or type of display being used for viewing the document

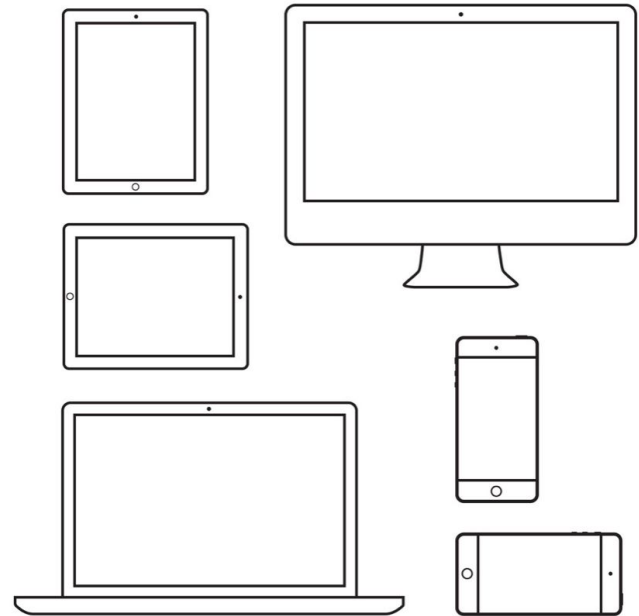
This style of web pages might have been fine in the 90's and early 00's, but not now. Modern computing devices include the cellphone, tablet, and smartwatch, and there are often more users on these than on 'traditional' desktops devices

We need to ensure that our pages work well on these devices

# Device considerations

The types of devices that we need to consider can be broadly split into 5 groups:

- Regular computer displays
  - Desktops, Laptops, Projectors
- Small displays
  - Phones, Tablets, Smartwatches
- Large displays
  - Digital signage, Televisions
- Accessibility tools
  - Screen readers, Braille
- Print



# Device considerations

---

Each of these device categories have their own layout requirements if we want to make our page usable and attractive, and these are often complex and contradictory

Regular displays favor a more open, spread out layout with plenty of space; whereas on a small screen device you want to make the best use of limited space, so the layout is compact and not wasteful

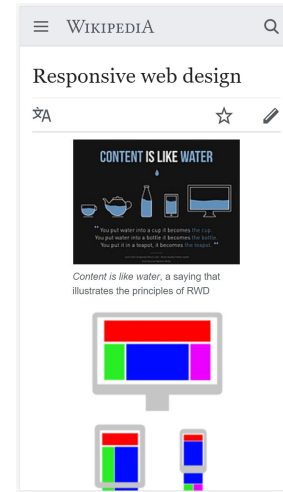
When printing pages you want dense, simple, black text with minimal wasteful images; but on large format screens you want bright colors that stand out with big clear text

# Device considerations

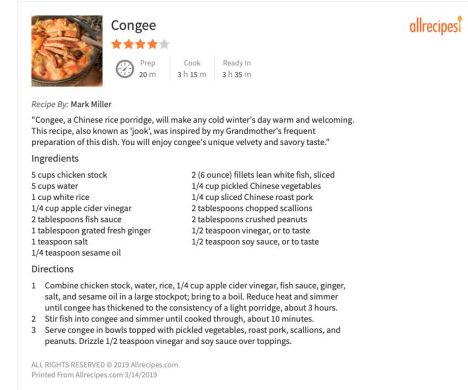
If we have to support these different devices, how do we write our web pages?

One approach that was popular throughout the 2000's and early-mid 2010's was to have separate pages for each of the supported devices. This leads to different URLs for each device, with layouts that need to be maintained separately - difficult and not scalable with the variety of devices

m.wikipedia.org



allrecipes.com/.../print/...



# Media Queries

# Media queries - Types

- A much better way to deal with supporting differing devices
- Supported on most devices beginning from 2012
- Allow us to write blocks of CSS that are conditionally applied, depending on what type of device the page is being viewed on

```
@media screen { /* Apply rules when viewed on a screen */  
  p { color: red; font-family: Impact; }  
}  
  
@media print { /* Apply rules when printing */  
  p { color: black; font-family: serif; }  
}
```

# Media queries - Types

- A media query's **type** filters for different types of devices on which we might render a page
- The following types are supported

`all, braille, embossed, handheld, print,  
projection, screen, speech, tty, TV`

- Useful to start - it allows us to detect accessibility devices, printing and screen categories
- But it doesn't give any granular control when we want to deal with different screen sizes (e.g. desktop monitors vs mobile device displays)

# Media queries - Features

- **Features** extend media queries, adding the ability to filter based on the capabilities of the device, which can be combined with the type query.
- There are a variety of features you can add into these queries to filter on:
  - width, height, aspect ratio, device orientation, color support, and a further [growing list of features](#).
- Lets us get much more granular with our queries to set different **breakpoints** where the layout can change in the document

```
@media screen and (max-width: 768px) {} /* small display */  
@media screen and (min-width: 768px) and (max-width: 1024px) {}  
@media screen and (min-width: 1024px) {} /* regular display */
```



# Eliminating duplication

- We do not need to rewrite the rules inside each media query block
  - We can write rules at the top-level of the stylesheet (outside of any media query blocks) that will apply to every device
  - We can nest media queries so that we can specialize the CSS for devices
- Rules will be applied as they are encountered in the file, reading top to bottom applying selectors at the top level of the file and within any matching media query blocks.
- We can overwrite earlier general rules inside media queries where necessary

# Eliminating duplication

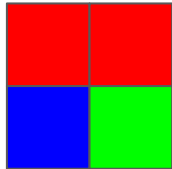
```
/* Applies to all devices */  
p { font-family: serif; }  
  
@media print { /* Applies only to print */ }  
  
@media screen {  
    /* Applies to all screens */  
    p { color: goldenrod; }  
  
    @media (max-width: 768px) { /* small displays only */ }  
    @media (min-width: 768px) { /* bigger displays only */ }  
}
```

# Grid and media queries

Using named areas for Grid layouts makes adapting our layouts for multiple devices very straightforward:

- We can simply modify the container `grid-template-*` properties inside the media query blocks

```
@media (min-width: 768px) {  
  .container {  
    grid-template-columns: 1fr 1fr;  
    grid-template-rows: 1fr auto;  
    grid-template-areas: "r r" "b g";  
  }  
}
```



```
@media (max-width: 768px) {  
  .container {  
    grid-template-columns: 1fr;  
    grid-template-rows: auto auto auto;  
    grid-template-areas: "r" "g" "b";  
  }  
}
```

