CSS Cheatsheet

Created	@July 10, 2021 5:11 PM
Created By	Luke Landau
Last Edited By	Luke Landau
Last Edited Time	@July 12, 2021 3:07 PM
Note	Created for and by a pack of fighting mongooses (take with mucho salt)
Stakeholders	
Status	
Type	

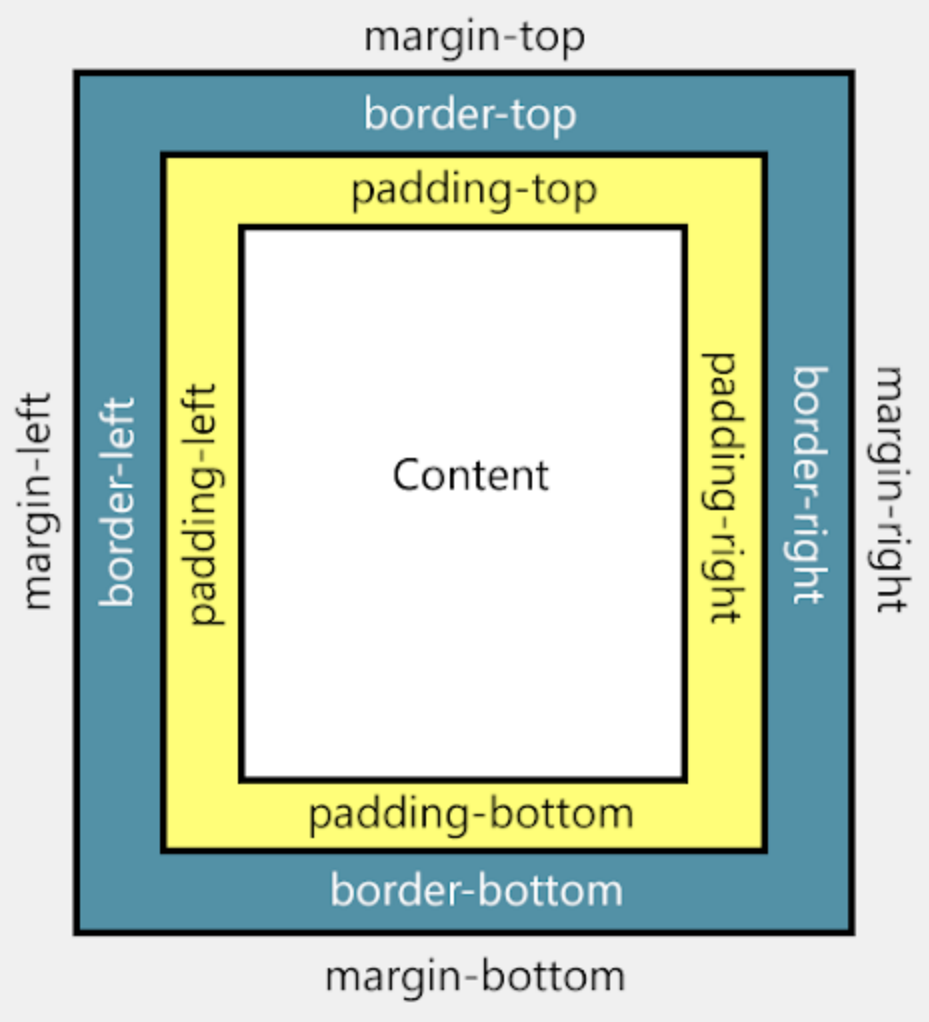
CSS Basics

- Cascading Style Sheets is the way to style HTML webpages. Best resource I've found so far
- achieved with `selector { property: value;}` syntax
- CSS allows a dev to implement almost any styling design

```
/* This is a comment and will be ignored */

/* Selectors can be an HTML <tag> */
p {
  color: black; /* This will apply to all <p> tags */
}

/* Or use some CSS reserve words, e.g select all */
* {
  margin: 0;
}

/* A selector can target elements by 'class' attribute */
.custom-class {
```

```
  color: black; /* This will apply to all elements with class="custom-class" */
}

/* A selector can target elements by 'id' attribute, but shouldn't */
#unique-id {
  color:black; /* This will apply to the element with id="unique-id" */
}

/* Multiple selectors can be used at once with commas to seperate them */
p, .customer-class, #unique-id {
  color: blue; /* This will apply to ALL of the above */
}

/* Selectors can target children of elements using a space to seperate them */
div a {
  text-decoration: none; /* This will apply to all <a> inside a <div> */
}

/* Selectors can target elements which have multiple properties with no space */
div.customer-class#unique-id {
  color: red; /* This will apply ONLY to elements that MATCH ALL of the above */
}

/* In combination, any type of element can be targeted for custom styling */
div a.custom-class {
  color: green; /* Only <a> inside <div> with class="custom-class" changed */
}
```

- an elements margin is the space around it, the border is self explanatory, and padding represents internal positioning of content



- by default the size of most elements are calculated by;
  - width + padding + border = actual width of an element
  - height + padding + border = actual height of an element
- to change this and ensure that padding and border do not effect the overall size of an element, the `box-sizing` property is often set to `border-box` for all elements

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

- there are multiple units of size in CSS, but the most common are;

```
/* 'pixels' are a common way to set a fixed value, used for largest elements */
div {
  width: 100px;
}

/* Relative size can be used by comparison to the size of the parent */
div p {
  width: 30%;
}

/* em stands for the width of the letter 'm' at a set font size. It can be
used to set relative sizes of text-properties for children of parent elements */
.parent-element {
  font-size: 10px;   /* set pixel value of parent to 10 */
}

.parent-element .child-element {
  font-size: 2em;   /* set child at 2x value of parent (20) */
}

.parent-element .child-element .grandchild-element {
  font-size: 2em;    /* set granchild at 2x value of child (40) */
}

/* rem stands for the size of the letter 'm' of the font size of the root element.
This allows text-properties to be based relatively to a single size, meaning the
size of text for the entire site can be changed with a single variable */
html {
  font-size: 10px; /* set font-size of root element */
}

h1 {
  font-size: 3.2rem; /* set font-size based on multiplication of rem (32)*/
}

h2 {
  font-size: 2.4rem; /* 24 */
}

p {
  font-size: 1.2rem; /* 12 */
}
```

- **colours** can be set in many ways, though some are better than others

```
.colours-code-example {

  /* Named colours are usually horrific,  only use them in testing */
  color: red;
```

```
  /* Hex colours are good, if a little confusing [ff, ff, ff] = [R, G, B] */
  color: #ffffff;

  /* RGB are good, easily understood and widely supported */
  color: rgb(255, 255, 255);
  color: rgb(100%, 100%, 100%);

  /* Both RGB and Hex have siblings, which include opacity */
  color: rgba(255, 255, 255, 0.8);
  color: #ffffff22;

  /* Or colours can be defined by Hue Saturation Luminosity (Opacity) */
  color: hsl(0, 0, 0);
  color: hsla(359,50%,40%,0.5);
}
```

CSS Useful Properties

```
.useful-code-examples {

  /* Element backgrounds can be set to images */
  background-image: url("/Library/Desktop Pictures/abstract.jpg");

  /* [text-decoration-line text-decoration-color text-decoration-style] */
  text-decoration: underline overline dotted red;

  /* Margin etc setting shorthand; [top, right, bottom, left] */
  margin: 20px, 10px, 20px, 10px;
  border: 1px, 0px, 1px, 0px;
  padding: 5%, 5%, 5%, 5%;

  /* Otherwise specify by property */
  margin-left: 10px;
  border-top: 1px;

  /* Horizontally align to center of parent container, 0 sets top/bottom */
  margin: 0 auto;

  /* Try to use a custom font firsts, then supply a number of backups */
  font-family: "My Crazy Font", "Times New Roman", serif;

  /* Border styling shorthand = [size, border-style, colour]; */
  border: 1px solid red;

  /* Element backgrounds can be set to images, set a colour as backup */
```

```
  background-image: url("img_tree.gif"), url("paper.gif");
  background-color: rgb(255, 255, 255);

  /* [text-decoration-line text-decoration-color text-decoration-style] */
  text-decoration: underline overline dotted red;

  /* Margin etc setting shorthand; [top, right, bottom, left] */
  margin: 20px, 10px, 20px, 10px;
  border: 1px, 0px, 1px, 0px;
  padding: 5%, 5%, 5%, 5%;

  /* Otherwise specify by property */
  margin-left: 10px;
  border-top: 1px;

  /* Horizontally align to center of parent container, 0 sets top/bottom */
  margin: 0 auto;

  /* Try to use a custom font firsts, then supply a number of backups */
  font-family: "My Crazy Font", "Times New Roman", serif;

  /* Border styling shorthand = [size, border-style, colour]; */
  border: 1px solid red;
}
```

- the `display` property specifies the type of rendering box an element will take

```
.display-code-examples {

  /* 'block's begin & end with a new line, takes up maximum width allowed by the
  parent by default but will respect width/height parameters set in the child */
  display: block;

  /* 'inline-blocks's respects height and width limits, have no new lines, and
  will ensure a block of content stays on the same line */
  display: inline-block;


  /* 'inline' ignores width and height parameters, as it is meant only for text,
  has no  new lines and will be just as large as the text it contains */
  display: inline;

  /* 'none' removes an element from the display altogether, used on elements that
  will be displayed when clicking something, achieved with JS. Removes element
  from the document flow, but it will still be read by search engines so can be
  used for SEO */
  display: none;
}
```

- `div` `p` `h1-h6` `form` etc are display-block by default
- `a` `em` `strong` `mark` `time` `span` etc are inline by default
- the `visibility` property hides elements without removing them from the document flow

```
.example-class {
  visibility: hidden;
}
```

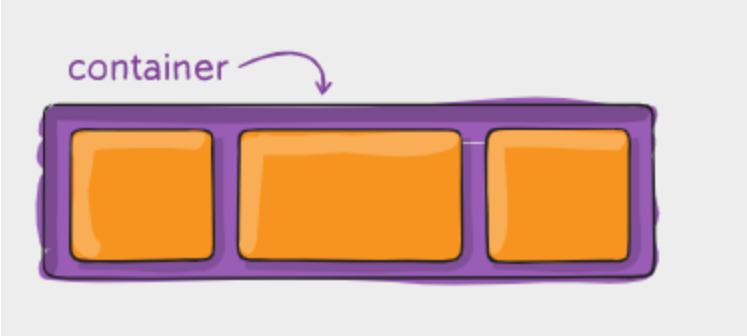
- the `float` property modifies the position of an element in the document flow, allowing it to be positioned almost anywhere. Behaves similar to a stack, e.g as the file is parsed each element floated to the right is added onto the left of the last element floated, until line fills

```
p.custom-text {
  float: right;
}
```

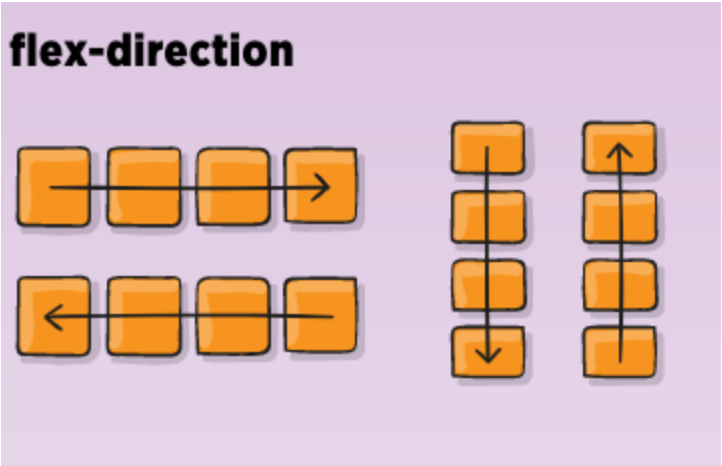
```
<p class="custom-text">This will be floated right first</p>\
<p class="custom-text">This will be floated to the left of the element above</p>
```

- `display` and `float` are enough to create any layout, however better ways exist to do so
- instead we can use Flexbox, a highly responsive rendering box that you should learn to love

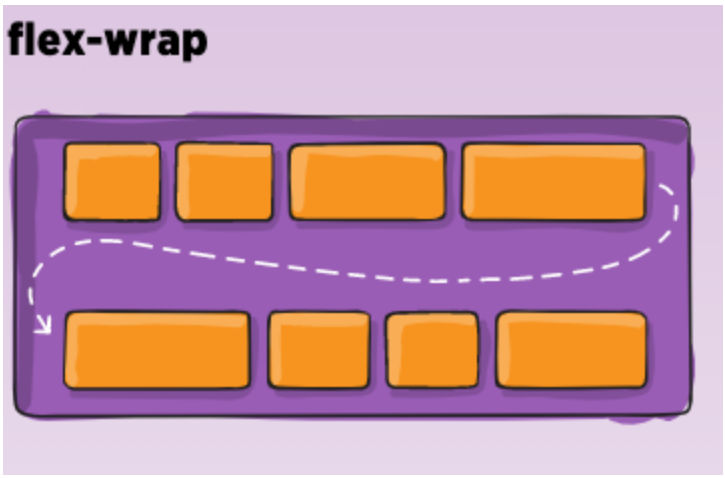
```
.container {
  display: flex; /* element acts as block, but inside has flex properties */
  display: inline-flex; /* element behaves as inline-block with flex inside */
}
```



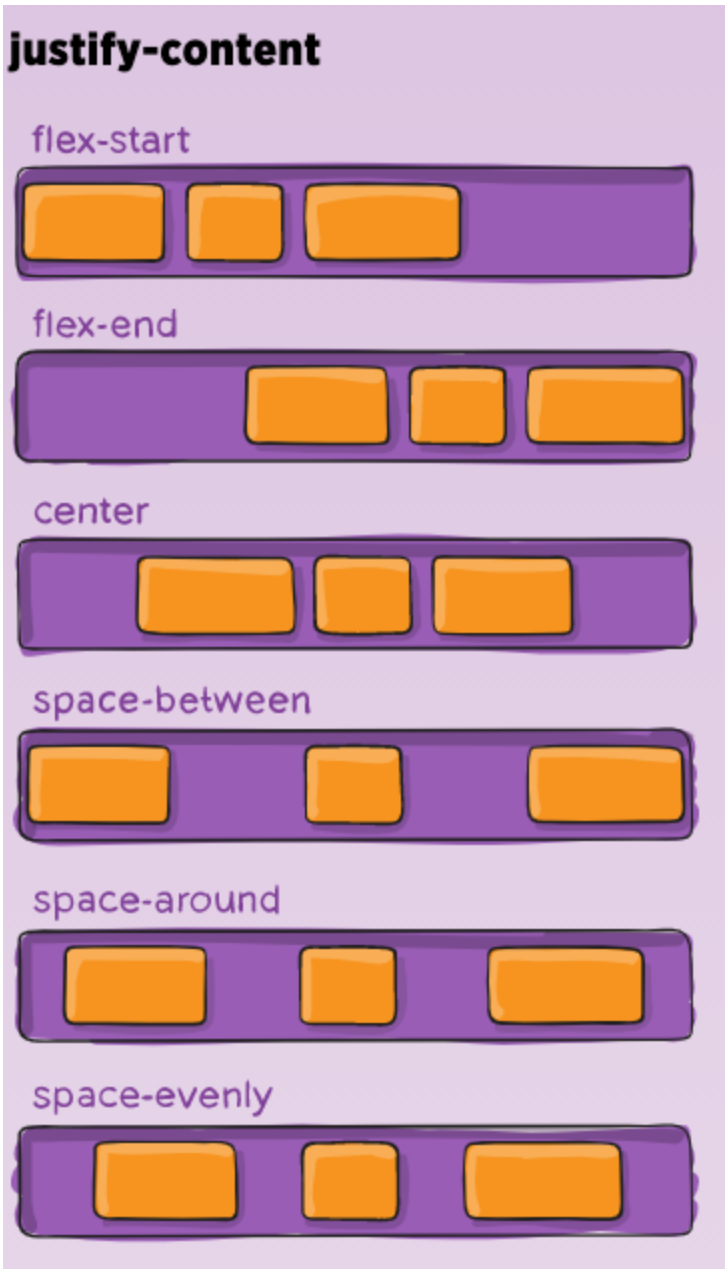

```
/* 'flex-direction' establishes the main-axis, thus defining the direction flex
items are placed in the flex container. Flexbox is (aside from optional wrapping)
a single-direction layout concept. Think of flex items as primarily laying out
either in horizontal rows or vertical columns */

.container {
  flex-direction: [row | row-reverse | column | column-reverse];
}
```



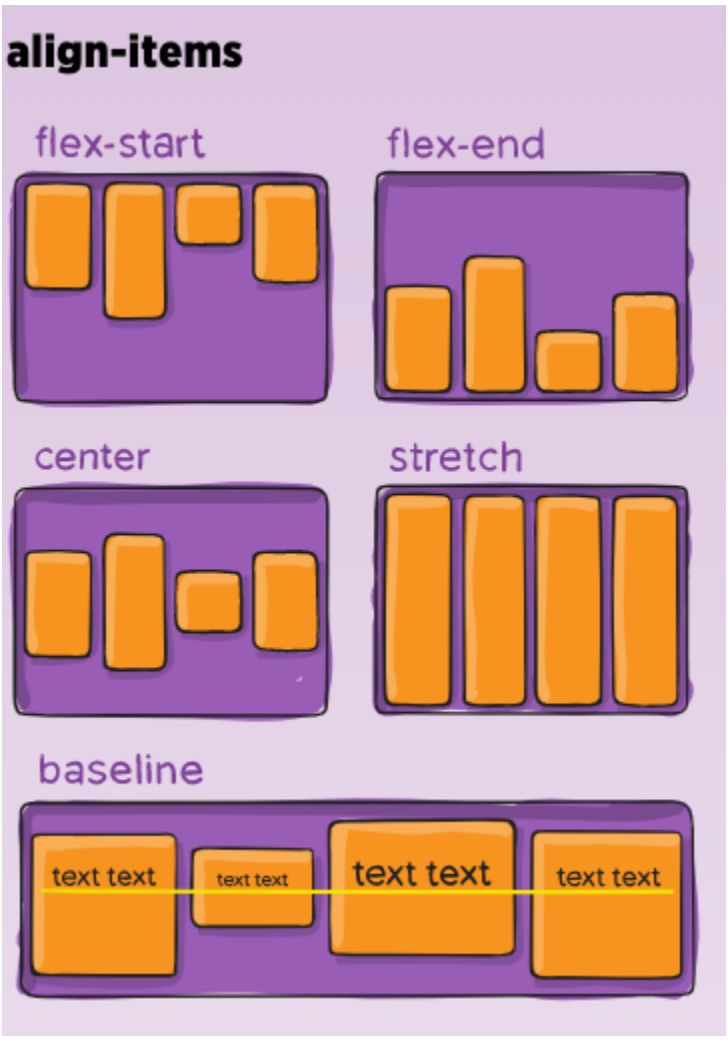

```
/* By default, flex items will all try to fit onto one line. You can change that
and allow the items to wrap as needed with 'flex-wrap' */

.container {
  flex-wrap: [nowrap | wrap | wrap-reverse];
}
```



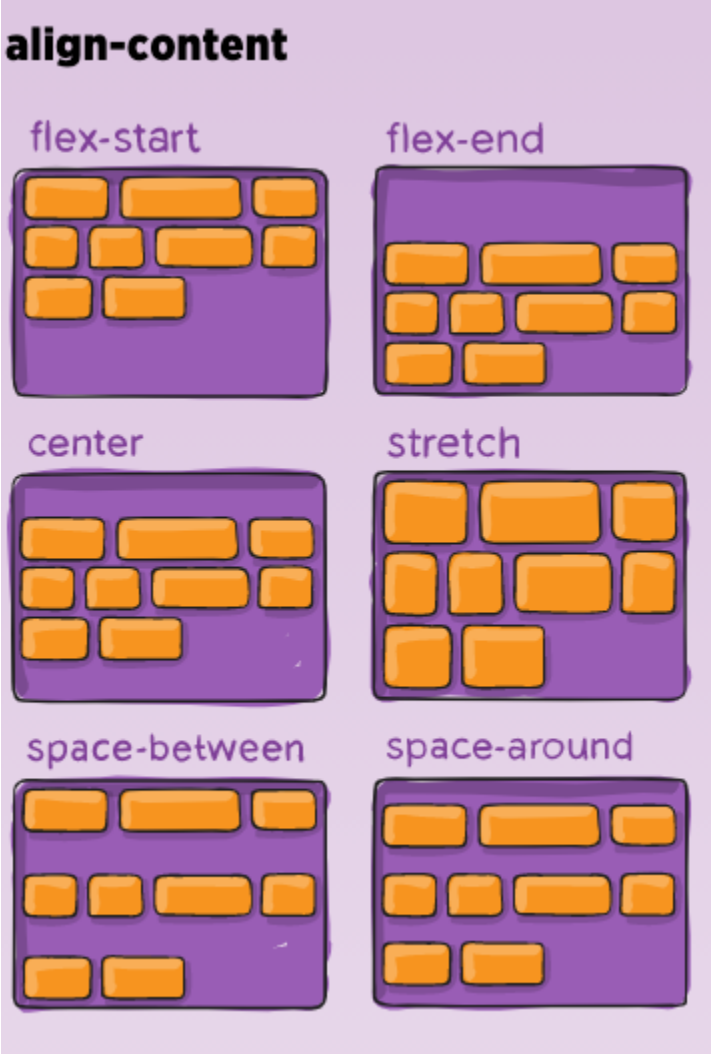

```
/* 'flex-flow' is a shorthand for the flex-direction and flex-wrap properties,
which together define the flex container’s main and cross axes. The default value
is flex-flox: row nowrap */

.container {
  flex-flow: flex-direction flex-wrap;
}

/* 'justify-content' defines the alignment along the main axis. It helps distribute
extra free space leftover when either all the flex items on a line are inflexible,
or are flexible but have reached their maximum size. It also exerts some control
over the alignment of items when they overflow the line. */

.container {
  justify-content: [flex-start | flex-end | center | space-between | space-around |
                   space-evenly | start | end | left | right ... + safe | unsafe];
}
```



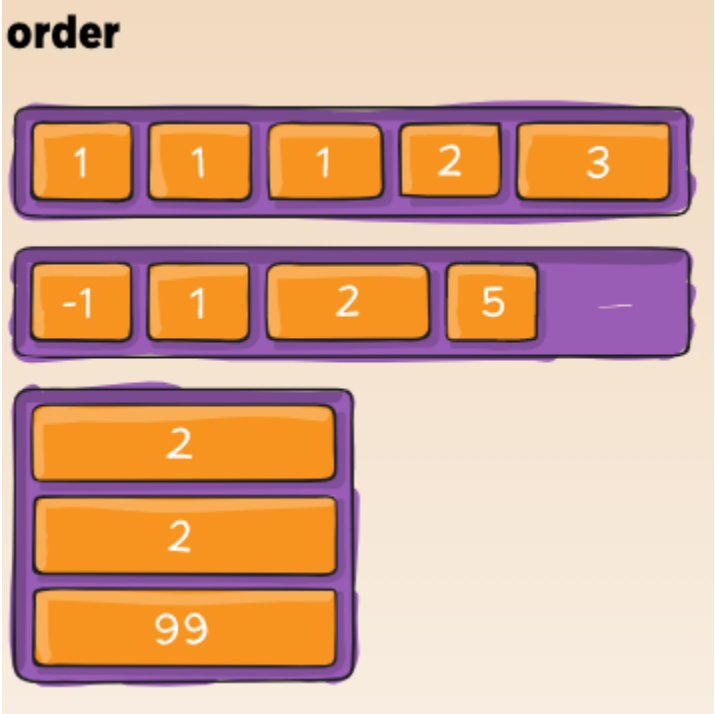

```
/* 'align-items' defines the default behavior for how flex items are laid out along
the cross axis on the current line. Think of it as the justify-content version for
the cross-axis (perpendicular to the main-axis) */

.container {
  align-items: [stretch | flex-start | flex-end | center | baseline |
                first baseline | last baseline | start | end | self-start |
                self-end + ... safe | unsafe];
}
```




```
/* 'align-content' aligns a flex container’s lines within when there is extra
space in the cross-axis, similar to how justify-content aligns individual items
within the main-axis. */

.container {
  align-content:[flex-start | flex-end | center | space-between | space-around |
                space-evenly | stretch | start | end | baseline | first baseline |
                last baseline + ... safe | unsafe;
}
```

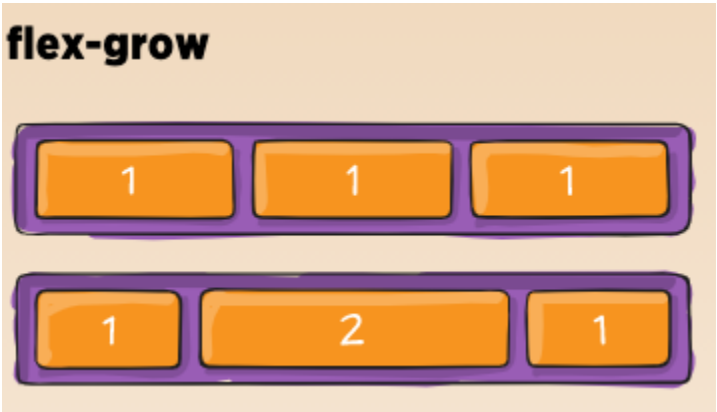



```
/* By default, flex items are laid out in the source order. However, the 'order'
property controls the order in which they appear in the flex container */

.item {
  order: 5; /* Default is 0 */
}
```

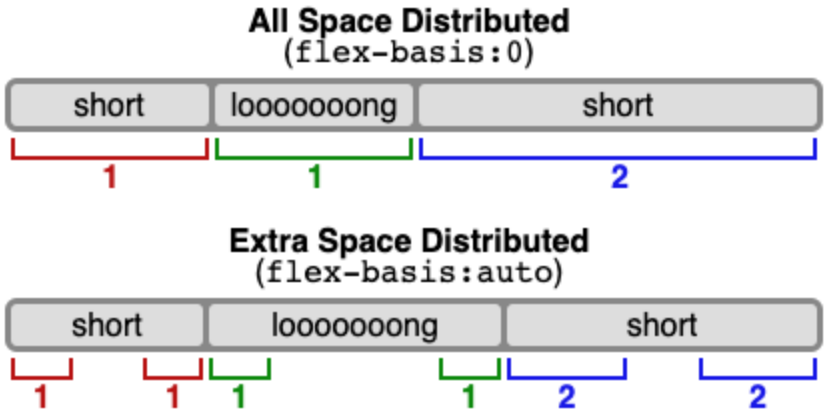


-

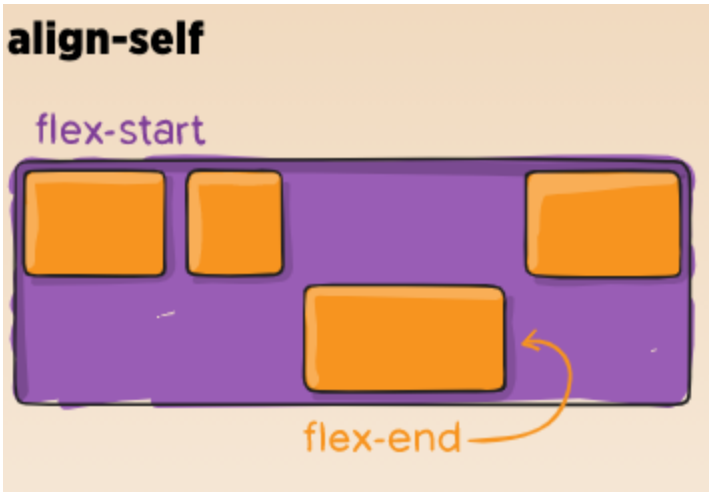
```
/* 'flex-grow' defines the ability for a flex item to grow if necessary. It accepts a
unitless value that serves as a proportion. It dictates what amount of the
available space inside the flex container the item should take up */

.item {
  flex-grow: 4; /* default 0 */
}
```



```
/* This defines the ability for a flex item to shrink if necessary. */

.item {
  flex-shrink: 3; /* default 1 */
}

/* This defines the default size of an element before the remaining space is
distributed. It can be a length (e.g. 20%, 5rem, etc.) or a keyword. The auto
keyword means “look at my width or height property” (which was temporarily done
by the main-size keyword until deprecated). The content keyword means “size it
based on the item’s content” – this keyword isn’t well supported yet, so it’s
hard to test and harder to know what its brethren max-content, min-content, and
fit-content do */

.item {
  flex-basis: [0px | 20% | 2rem | auto]; /* default auto */
}
```




```
/* 'flex' is the shorthand for flex-grow, flex-shrink and flex-basis combined. The
second and third parameters (flex-shrink and flex-basis) are optional. The default
is 0 1 auto, but if you set it with a single number value, it’s like 1 0 */

.item{
  flex: flex-grow flex-shrink flex-basis;
}

/* This allows the default alignment (or the one specified by align-items) to be
overridden for individual flex items. */

.item {
  align-self: [auto | flex-start | flex-end | center | baseline | stretch];
}
```



- Full credit for explanations and pics goes to css-tricks.com
- the CSS Grid allows a dev to define the page layout with columns and rows
  - rows do not require defining, unless a custom size is desired

```
.my-grid-container {
  display: [grid | inline-grid];
  grid-template-columns: 1fr 2fr 1fr; /* define columns by fractions of space */
  grid-template-rows: 10px 10px 20px 10px; /* can also define with other units */
}
```

- the CSS Grid is perhaps more powerful than Flexbox, so further reading is recommended
- in the normal document flow, the position of an element is determined by where it sits in the HTML and CSS properties such as flex, grid, float, margins etc
- the `position` property allows us to adjust an elements position in the document flow

```
.example-class {

  /* 'static' is the default for all elements. The element sits as expected in the
  document flow and ignores the properties of [top, bottom, left, right] */
```

```
  position: static;

  /* 'relative' allows the adjustment of position relative to the static position
  of the element before modification - will ignore (screw over) other elements */
  position: relative;

  /* [top | bottom | left | right] can be thought of as anchors, from which to
  define a relative movement away from e.g push away from the top by 100px */
  top: 100px;

  /* 'fixed' is similar to relative but instead of defining anchors by the static
  position, it is based on the viewport (i.e browser window) and will make an
  element move when scrolling down the page - removes element from normal document
  flow */
  position: fixed;

  /* 'absolute' completely removes an element from document flow and positions it
  based on the closest positioned parent e.g. parent-pos: [relative | fixed | abs]
  Though requiring the parent to have a non-static position set, the parent is not
  required to set [top, left, bottom, top] values */
  position: absolute;

  /*READ MORE ABOUT THIS*/
  position: sticky;

  /* 'z-index' allows a dev to reposition an element along the z-axis, equivilent
  to moving elements to the fore or the back of the screen. Use large numbers to
  ensure there is enough room for future layers - default value is auto */
  z-index: 30;
}
```

CSS Best Practices

- each browser provides a default stylesheet that sits underneath any custom one
  - e.g heading sizes, font, default font size, underlined blue links etc
  - headings and paragraphs have margins, sometimes the body tag has annoying styling
- best practice is to reset all styles with a CSS Reset and use verbose custom stylesheets
  - helps to ensure cross-browser compatibility

- be efficient, use a pre-made reset stylesheet, e.g. Normalise
  - download normalise.css and place in project folder
  - link normalise.css first to clear all preset styling
  - link custom stylesheet after to override normalise.css with preferred style
  - ensure any changes to CSS are made in a custom file, for clarity of authorship
  - **vendor code** - someone else's code, make sure to understand their licence
- **semantic class names** improve code maintainability and minimises the nightmare of troubleshooting and bug fixing
  - use intuitive and verbose class names such that anyone reading is clear on its purpose
  - class names should not conflict with HTML or CSS reserve words, e.g `div` or `flex`
  - name the class based on the purpose of the element, not aesthetics or syntax of code
- responsiveness is defined differently for mobile than for desktop
- mostly we are trying to make a website work on a variably sized screen
  - known as **responsive web development**
- multiple ways of doing this e.g flex, variable and or relative max-widths etc
- often useful to design for **mobile first**
  - build a separate wireframe for mobile and browser before writing any code
- another strategy is to use **media queries**
  - a feature of CSS that allows us to apply different styling based on the size of the viewport

```
.greeting {
  width: 800px;
}
```

```
@media screen and (max-width: 800px) {
  .greeting {
    width: 90%;
    margin: 0 auto;
  }
}
```

- CSS Media Queries certainly merits further reading