

CSS – Part 6

Typography: Getting Started

A typeface is how we define the overall design of lettering. First came the blackletter typeface then roman style then italics then Caslon came then Baskerville then sans-serif typeface came then future then gill sans then Helvetica.

Helvetica Usage



Signage



Advertising



Billboard



Branding



Caslon



Baskerville



Futura



Helvetica

Fonts of a typeface –

Anatomy *Anatomy* Anatomy **Anatomy**

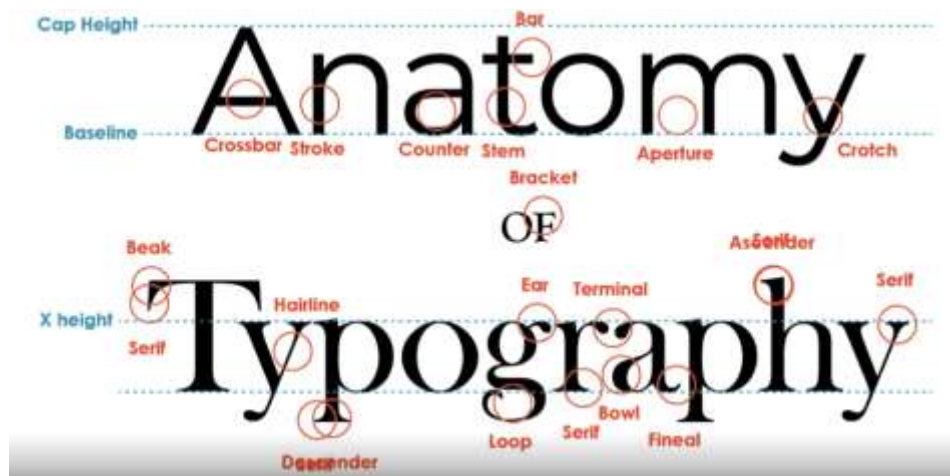
Bold

Italics

Narrow

Narrow Bold
Italics

Anatomy of typeface –



Typography classification – serif and sans-serif.

Justified alignment – just say no to justified alignment as it is hard to read.

Justified Alignment

Consectetur vel nisl ut amet quam peritiam
faucibus. Mauris vel justo odio. Incom
posuisti nulla a libero tristique porttitor in ut
lorem. Nullam ut suscipitque leo. Donec
bibendum, ipsum et porta ultrices, mauris
vel morbi et. Sed, nunc congueque itam
velit nec leo. Conditur a tristique orn
eget vestibulum et.

Incomposuisti tristique orn, vel ferment
lorem, imperdiet non. Vestibulum ante
ipsum porta in faucibus orn nunc et
ultrices posuere cubile – nunc
Suspendisse ultrices nunc frugiat vel
tristique, vel elementum vel fermentum,
bibendum et ipsum maximus, dictum
velit nunc, ultrices et.

Incomposuisti itam orn, vel suscipitque
mauris eget et.

Letter and word spacing kerned to make
flush on left and right

Known as fully justified or full justification

Align last line left or right

Overly stretched lines become loose

Tightly compacted lined become tight

Hyphenation – dividing words at the end of a line. Doesn't look clean. No clear advantage. On digital application do not hyphenate.

Hyphen- ation

Dividing words at the end of a line

Symbolized by hyphen mark

Manual and automatic methods

Don't put indent first paragraph in digital world –

→ Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Mauris eget tortor dictum,
faucibus ante quis, malesuada
neque. In hac habitasse platea
dictumst. Donec viverra urna a
pharetra tempor. Mauris in arcu
blandit felis porttitor consequat
eget non ligula. Mauris
bibendum est porta accumsan
ultrices.

Don't indent first paragraph

Don't indent when following larger
structures

Don't indent and also add space between
paragraphs, choose one or the other

Use all caps sparingly – instead of this try to use title cases



**MARTIANS
INVADE EARTH'S
HOLLOW CORE**

MASSIVE AMOUNTS OF
UNOBTAINABLE TO BLAME

Use sparingly

Reserve for title blocks, brief headlines, or single words

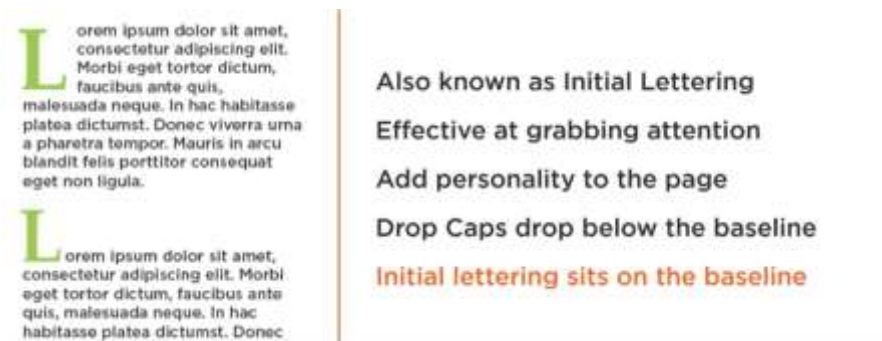
Use to really grab user's attention

When everything is caps, everything is ignored

Use camel case to combine multiple words begin with lowercase, no spaces between words, second word capitalized



Drop caps –



Also known as Initial Lettering

Effective at grabbing attention

Add personality to the page

Drop Caps drop below the baseline

Initial lettering sits on the baseline

Balance line height calculation –

Balanced Line Height Calculation

- White space between the lines equals 150% of the x-height.
- Or set the line height of 125% of text size.

Pull quotes – it should not be a copy of original text.

Pull Quotes



Without whitespace design is un-structured and difficult to consume –



Structural hierarchy –



Use maximum two different typefaces in the application like serif and sans-serif. Like one for header and another for content. We can use website fontjoy.com and fontinuse.com for font pairing.

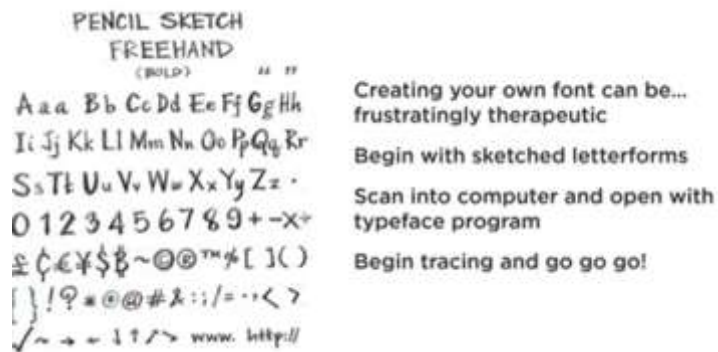
Font licenses for different media –



Web-safe fonts –

Arial, Times New Roman, Courier New, Palatino, Garamond, Bookman, Avante Garde, Verdana, Georgia, Comic Sans, Trebuchet and Impact

Creating your own custom fonts –



Font file types –



Applying Special Effects to a Site Using CSS

Basic rounded corner use border-radius as 5px. For rounded objects as circles use we can use icon instead of using button. To make it complete circle use border-radius as 50%.

To fit the picture on rounded use properties like object fit, object-position.



Border-radius can take 4 values based on the different corners to get rounded.

To add some depth to the website instead of flat use box shadows. By saying `box-shadow: 0px 3px`. Blurring the shadow to make more realistic look use `blur-radius`, it will be as a third parameter to the border radius. We need to use spread radius as the fourth parameter, it controls the size of the shadow. We can pass fifth parameter as `inset`, it tells the shadow to move inward from the edge as opposed to outward.



Transition between states use hover state as a pseudo class to change the state when user hover over the menu items and change its color. We can use `transition-duration: 3s`; to tell the browser take 3 seconds to take the transition from one color to another color. We can also pass the `transition-timing-function: linear`; to control the transformation of the animation. We can set ease, linear, ease-in, ease-out, ease-in-out, steps.

We can use below animation toolbar in chrome to control the transition speed –



Applying transition independently –

```

206 aside > ul > li:hover span {
207   opacity: 1;
208   transition-property: opacity, padding-right;
209   transition-duration: 2s, 5s;
210   font-weight: bold;

```


Waiting on transition to start another – we need to use transition-delay property –

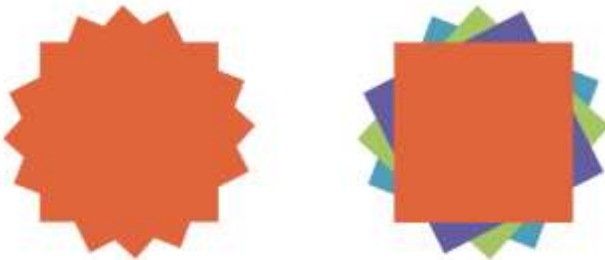
```
206 aside > ul > li:hover span {  
207   opacity: 1;  
208   transition-property: opacity, padding-right;  
209   transition-duration: 2s, 5s;  
210   transition-delay: 0s, 2s;  
211   font-weight: bold;
```

Transition shorthand –

```
206 aside > ul > li:hover span {  
207   opacity: 1;  
208   transition: opacity 2s 0s, padding-right 5s 2s;  
209   font-weight: bold;  
210   color: #a593c2;  
211   padding-right: 0%;
```

Not all properties are able to be transitioned like display for which we used opacity. Also the position can't be animated.

Transforming elements – rotating an object – we can use degree or radians.

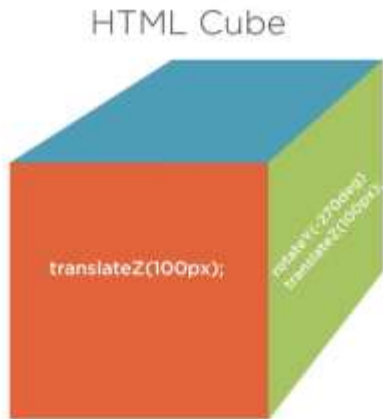


```
<div class="square one-  
sixteenth"></div>  
  
<div class="square one-eighth"></div>  
  
<div class="square three-  
sixteenth"></div>  
  
<div class="square"></div>  
  
.one-sixteenth {  
  transform: rotate(0.0625turn);  
  background: #2A9FBC;  
}
```

Translate function to move the object on X or Y axis. Scale function will grow the object in size, we can scale it either horizontally or vertically. Skew function, it will skew alter the points within the object. It will be skewing the horizontally and vertically. We can combine these function each other.

```
transform: skew(21deg, 25deg)  
translate(20px, 35px) rotate(15deg)  
scale(0.75);
```

3D transformation – using translateZ method.



Using animation module for loading indicator – creating custom animation by using @keyframes

```

32
33 .logo.load {
34   transform: translate(500px, 100px);
35   animation: 3s spin 1s;
36 }
37
38 @keyframes spin {
39   from {
40     transform: translate(500px, 100px) rotate(0deg);
41   }
42
43   to {
44     transform: translate(500px, 100px) rotate(360deg);
45   }

```

We can make animation keep looping by specify the another parameter by specify the infinite number –

```

32
33 .logo.load {
34   transform: translate(500px, 100px);
35   animation: 3s spin 1s infinite;
36 }

```

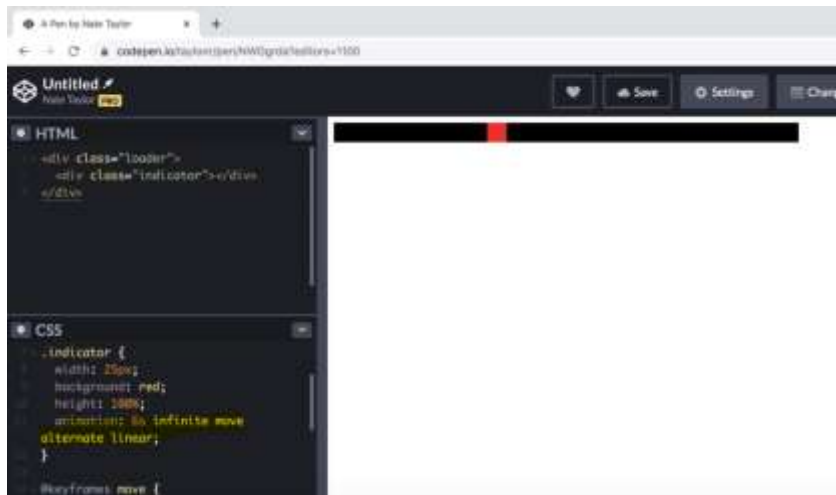
Keeping the last animated frame by using another keyword like none (default), forwards, reverse, both –

```

32
33 .logo.load {
34   transform: translate(500px, 100px);
35   animation: 1s spin 1s 1.25 forwards;
36 }

```

Controlling how animation restart like back and forth –



Multiple animations with different speed and different delay –

```

33 .logo.load {
34   transform: translate(500px, 100px);
35
36   animation-name: spin, fade;
37   animation-duration: 3s;
38   animation-delay: 1s;
39   animation-iteration-count: infinite;
40   animation-timing-function: linear, ease;
41 }

```

Styling Websites with CSS

For styling text we can use font-size, font-weight, font-style, line-height, letter-spacing, text-align, text-transform, font-family.

For externally hosted fonts, we need to provide its reference before style file –

```

11 <link
12 href="https://fonts.googleapis.com/css2?
13 family=Roboto&family=Lato:ital@0,1&display=swap"
14 rel="stylesheet">
15 <link rel="stylesheet" type="text/css"
16 href="styles/main.css">
17 </head>

```

Changing the color and size of an element – background-color, color, width, text-align, border-color.

Like border we can also use 'outline' property. A border adds to the content box, outline is just tacked on outside.

The shorthand follow clock pattern i.e. top-right-bottom-left.

Combinatory – descending, child, adjacent, general sibling.

Pseudo-classes - :hover, :nth-of-type, :first-of-type, :last-of-type, :nth-child, :first-child, :last-child

Creating Layouts with CSS Grid

Before CSS grids we have to use floats and lots of other CSS and different hacks. Hack in the sense of exploiting a tool for other than its original purpose. We just to add below CSS –

```
body {  
  display: grid;  
  grid: repeat(3, max-content) / 20em auto 15em;  
}  
  
header, footer {  
  grid-column: 1 / 4;  
}
```

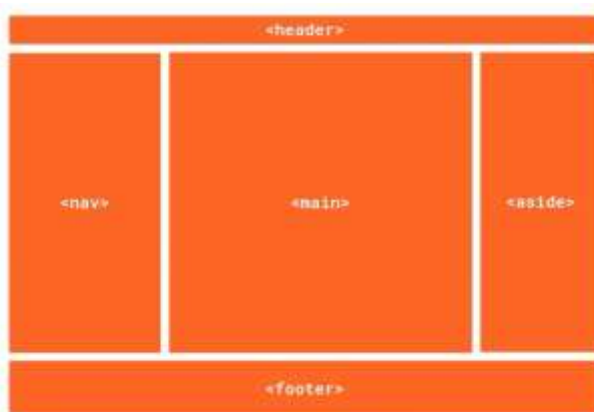
CSS grid properties –



Most common properties are – grid-column-end, grid-column-start, grid-template-rows, grid-template-columns

The two-step grid process – define a grid, position items

Major sections of a web-page – header, nav, main, aside, footer



Layout example with CSS grid rules -

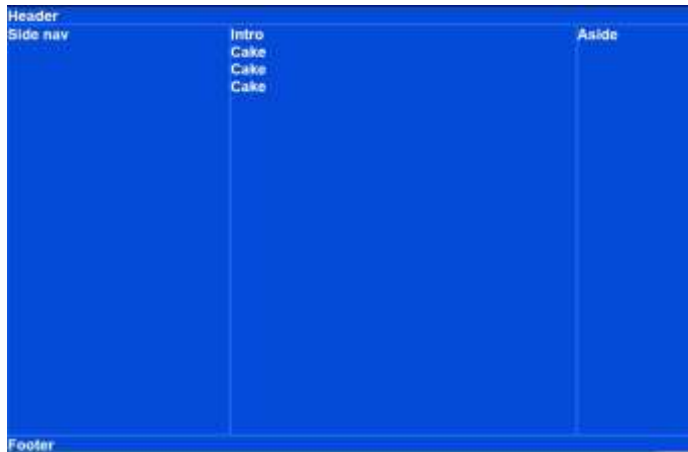
```

html, body {
  height: 100%;
}

body {
  display: grid;
  grid-template-columns: 15em auto 15em;
  grid-template-rows: min-content auto min-content;
}

header, footer {
  grid-column-start: 1;
  grid-column-end: 4;
}

```



Grid track sizes – static values (ems, rems), percentage, auto (let the browser decide), fr (fraction unit, divide up the remaining space), min-content (make track as small as possible to fit its content), max-content (make track as large as necessary to contain its content), minmax (function to specify minimum and maximum values for a track), fit-content (fit track size to the content, but don't go above specific value)

Grid shorthand's function repeat (number of repetitions, track sizes) – function to repeat a pattern of track sizes multiple times, grid-template (property to consolidate grid-template-rows and grid-template-columns), grid (property to consolidate grid-template-rows and grid-template-columns and some additional things)

The grid-auto-flow (whether the grid fills rows or columns first while assign grid items, default is row)

Implicit grid – the columns and rows CSS grid creates itself to accommodate your grid items. Explicit grid – the columns and rows you define yourself.

We can use grid-auto-columns and grid-auto-rows to size of any implicitly created columns or rows.

Space between the cells are called gutters or gaps. The grid-column-gap or grid-row-gap or grid-gap as shorthand.

Positioning alternatives – negative grid lines, span keyword.

We can align the grid or the grids content itself by properties like justify-content, align-content, justify-items, align-items, justify-self, align-self.

Justify is left to right and align is top to bottom which can be start, center, end, space-around, space-between, space-evenly. We can apply them on content, items and self.

We can use dense keyword to grid-auto-flow property which will flow items into the first empty grid area and won't create any hole in layout.

The grid-template-areas property – assigning a name to grid cells

```
body {  
  display: grid;  
  grid-template-areas: "header header header"  
                      "nav main aside"  
                      "footer footer footer";  
  grid-template-rows: min-content auto min-content;  
  grid-template-columns: 15em 1fr 1fr;  
}
```

Creating Responsive Pages with CSS FlexBox

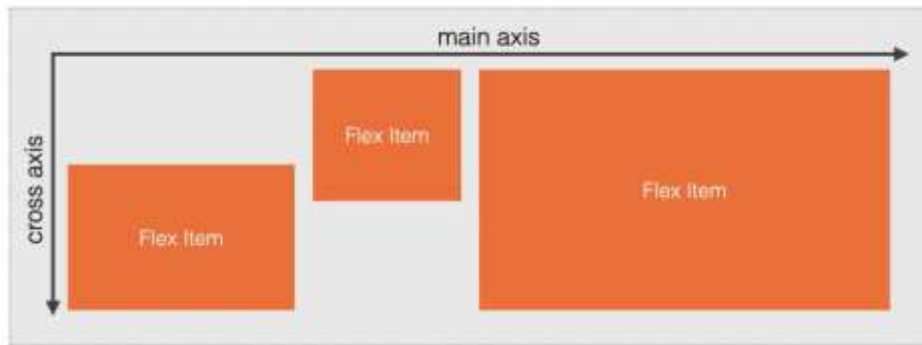
A flex container is the parent object and contains flex items inside of the parent container. Flex items are primary objects within the flex container like images, text, forms. We can set certain attributes on flex-container and properties on flex items.

Flexbox container properties – we don't need to use float or columns now as it requires complicated code to be written.

Flexbox stacking – rows, columns. We can also nest flexbox within flexboxes -



Main and cross axes –



Breaking down a page layout in-terms of row and column layout –



To display a page as a flexbox just use setting: `display: flex`; it will adjust it into default position as row

Apply `flex-direction: column`; on the footer to stack the content vertically instead of horizontal. If we use `column-reverse` or `row-reverse` settings it will reverse the content.

By default the content doesn't get wrap the content on next line on smaller screen, it will get squash on the same line, so need to use `flex-wrap`; `wrap`;; we can also use `wrap reverse`.

Create test.html file to try different flex-box styles to understand its working better.

We can justify the content by using `justify-content: flex-end` (i.e. main axis end), by default it was `flex-start`. We can also use `space-around`, `space-evenly`, `space-between`, `space-center`

Aligning content against cross-axis – `align-items: center`; to align the content as center. We can have `flex-start`, `flex-end`.

Aligning the containers along the axis – use `align-content: center`, we can use `flex-end`, `space-between`, `space-around`. It will set the complete content itself.

For margin or width use `rem` and percentage.

Adjusting the flexbox specific items – changing the order of items – `order: 1`; to change the order, default is 0. We can use media query to change the order on the smaller screen. With value as 1 it will start the content from second items and push the first on the last.

Adjusting the size of specific item – `flex-grow: 2`; to let it more space, default is 1.

Making flex items shrink – we can shrink the unimportant items we can apply `flex-shrink: 5`; default 0;

Defining the flex item default size – flex-grow will take up space automatically by using this property we can define the default size. Flex-basis: 60%; default is auto based on the content size. We can use media query to increase the size on mobile screen as 90%;

Aligning specific flex items without all of them – align-self: center; we can also use baseline, auto, self-start, flex-end, stretch

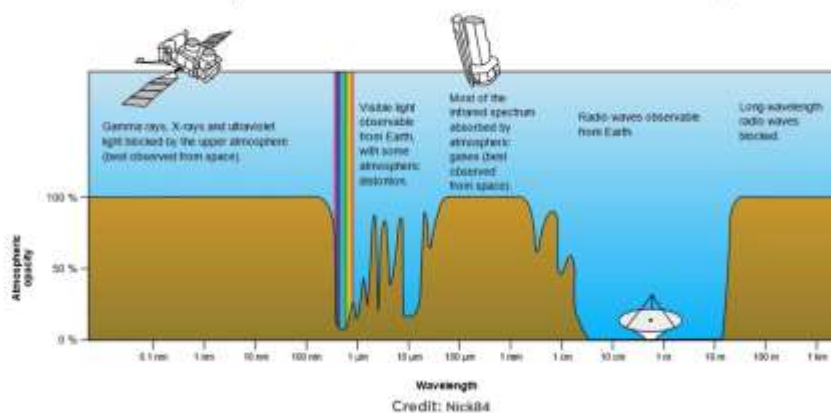
General CSS changes – we need to use general CSS changes to fine tune our website like on the images instead of using the hard-coded pixel use width: 100%; so that image can also get resize as per browser size as responsive.

Instead of float: left or right; use justify-content: flex-end or flex-start;

When not to use flexbox – they are good for simple layout where we can control over the components. But they are not meant for complex application grid systems. Better suited for individual components. So, should use more CSS grids instead of flexbox on entire page. Flexbox provide limited control over order. CSS grids provides better control over text layouts.

Working with Colors and Images in CSS

EM Absorption in the Earth's Atmosphere



How we perceive color – perceived primarily by the cones (long – red, medium – green, short - blue). Vision is trichromatic – ‘three-colored’. 8% of the male population is dichromatic (colorblindness).

Don’t represent information solely with colors, use text along with it.

Due to problem with color names, color names are lore, we use web-safe palette, a palette with 216 colors which could be represented untethered on older displays.



Dimmer switches allow intermediate values between off and on.

How many colors = $256 * 256 * 256$

There are two ways of making colors – adding light to black (RGB), subtracting light from white (CMYK which is used in printing)

Different ways to represent colors – RGB, CYMK, HSL, HSV and Pantone. RGB rules the world.

An image is two dimensional array of colors. There is no way to make them three dimensional.

Zip compression is lossless and great for precise digital information but have less effective when the data series is fuzzy and imprecise.

JPEGs have lossy compressions, we lose information that we can never recovered from resulting file.

Choosing an image format (bitmap, tiff, compressed TIFF, gif, jpeg) only gif and jpeg can be used over the internet. Use gif for illustrations, logs and backgrounds and jpeg for photographs. We don't use gif for photos because the compression would be lousy and if we use jpeg for illustration then its compression will be lousy.

Portable network graphics (PiNG) because GIF compression was patented. PiNG (PNG) provides lossless compression means we can save it over and over again without losing information. JPEG still have smaller size than PNG. But PNGs don't do animation. GIF and PNG can do transparency, JPEG cannot. Transparency allows the color of the parent layer to shine through.

So, use PNG unless we are sure you need JPEG, JPEG compression is lossy, don't use for working copies.

Rater vs. vector based images – for vector we use .svg files -



A scene rendered with vector instructions



A raster graphic

Don't ever use inline styles. Don't use the <blink> tag

CSS sprites –



CSS Sprites



We're going to take this image of the cupcakes...

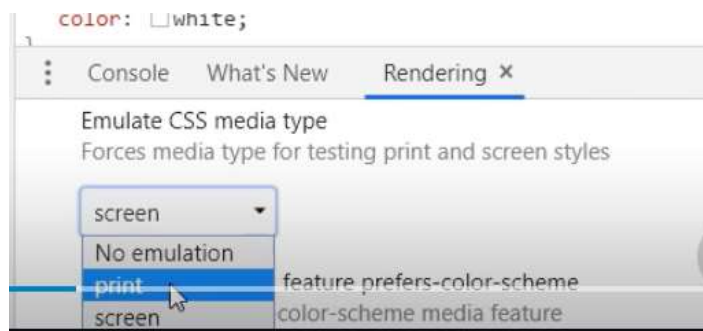


And slice it into individual cupcakes

CSS filters – we can apply effects on images or other elements. Supported filters –

blur	brightness	contrast	drop-shadow
grayscale	hue-rotate	invert	opacity
	saturate	sepia	

We can emulate the rendering of media type in chrome dev tool like below like screen to print –



```
<link rel="stylesheet" href="main.css" media="screen" />
<link rel="stylesheet" href="print.css" media="print" />
```

What is missing in CSS – no CSS constant and no cross browser supports, for this we should use SASS. SASS is the technology platform and SCSS is syntax. It has more features like nested style as below second one is more cleaner with SASS –

```
.cupcakeDiv p{
  color: #23cfa7;
}

.cupcakeDiv{
  // non-p css properties
  p{
    color: $cupcakeColor;
  }
}
```

SASS modules – color, list, map, math, meta, selector, string

In version control we should store the SASS files not their output CSS files.

Creating functional styles – showing color code based on the order late days and urgency

```
function getUrgencyLevel(value){
  if (value < 4){
    return 1;
  }
  if (value < 8){
    return 2;
  }
  return 3;
}

function colorUrgencies(){
  var lateOrders = document.querySelectorAll('#lateOrders tr-late');
  for(var i = 0; i < lateOrders.length; i++){
    var id = lateOrders[i].attributes["id"].value;
    var daysLateId = "#" + id + " td.daysLate";
    var daysLate = parseInt(document.querySelector(daysLateId).innerText);
    var urgencyLevel = getUrgencyLevel(daysLate);
    var lateCell = document.querySelector(daysLateId);
    lateCell.classList.add("urgency" + urgencyLevel);
  }
}

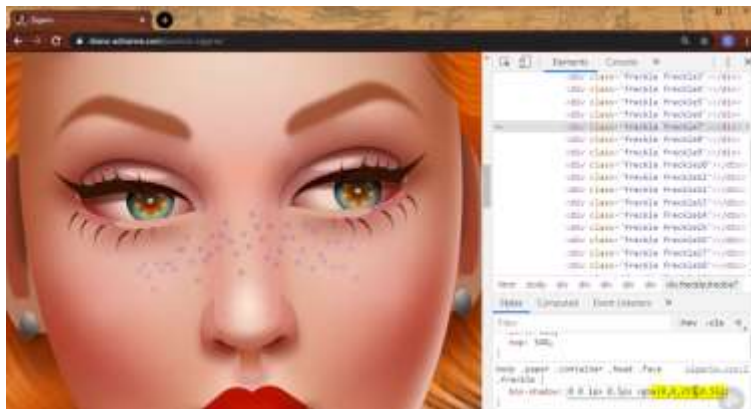
$urgencyColor: #FF0000;
.urgency1{
  background-color: color.adjust($urgencyColor, $saturation: -66%);
}
.urgency2{
  background-color: color.adjust($urgencyColor, $saturation: -33%);
}
.urgency3{
  background-color: color.adjust($urgencyColor, $saturation: -0%);
}
```

Late Orders

Customer	Order Number	Days Late
Francois	34	2
Marie	17	8
Jean-Claude	2	14

Below image is created using purely CSS –

<https://diana-adrienne.com/purecss-zigario/>



Adding Styling with Bootstrap Typography and Utilities

We can use different websites like fontjoy.com to find the proper pairs of fonts.

Bootstrap classes – lead, list-unstyled, list-inline, text-justify, text-center, text-right, text-left, text-wrap, text-unwrap, text-break, text-uppercase, text-lowercase, text-weight-bold, text-weight-bolder, font-weight-normal, font-weight-light, font-weight-lighter, text-reset, text-decoration-none

By wrapping a text in <abbr> tag, it will show text in underline and on hover it will show the helper text –



To enable the font responsiveness we just need to enable the \$enable-responsive-font-size variable to true.

Bootstrap Components: Playbook

Button can be created using tags like ``, `<a>`, `<p>`, `<button>`, `<input>`, `<div>`, also we can use classes like `active` or `disabled` to give enhanced look to that button.

Jumbotron has been removed from bootstrap 5 version, we can use some other utility class like `'bg-light'`, `'p-5'`, `'rounded'` in place of it.

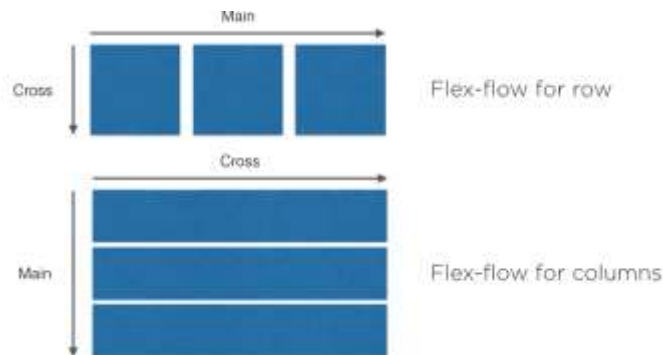
The utility class `'container'` provides needed padding and space for the child element.

Web Layouts with Flexbox and Bootstrap 4

Flex layout has been added in Bootstrap 4 version.

Two types of flex containers – row and vertical. Default is row level.

Flex axis –



Bootstrap 5: Fundamentals

Building blocks – layout, components, interactivity, style utilities.

Check <https://www.getbootstrap.com/docs/5.0/examples> for website templates and <https://www.themes.getbootstrap.com> for themes.

Available breakpoints –

Name	Prefix	Width
Extra small	None	< 576px
Small	sm	>= 576px
Medium	md	>= 768px
Large	lg	>= 992px
Extra large	xl	>= 1200px
Extra extra large	xxl	>= 1400px

Difference between without and with fluid container – The grid will take all unused horizontal space.

```
<div class="container">
  ...
</div>
```

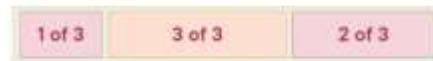


```
<div class="container-fluid">
  ...
</div>
```



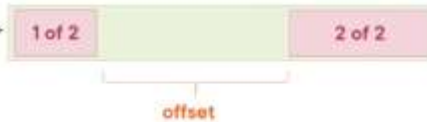
Using order number – the lower the order, it will move to the left.

```
<div class="row">
  <div class="col-md-2">
    1 of 3
  </div>
  <div class="col-md-4 order-5">
    2 of 3
  </div>
  <div class="col-md-6 order-1">
    3 of 3
  </div>
</div>
```



We can use offset to leave some gap –

```
<div class="row">
  <div class="col-md-2">
    1 of 2
  </div>
  <div class="col-md-4 offset-md-6">
    2 of 2
  </div>
</div>
```



Use gutters to add space into the different columns. We can also use g0 to remove all gaps between columns –

```
<div class="row gx-4">
  <div class="col-md-2">
    <div>1 of 2</div>
  </div>
  <div class="col-md-4">
    <div>2 of 2</div>
  </div>
</div>
```



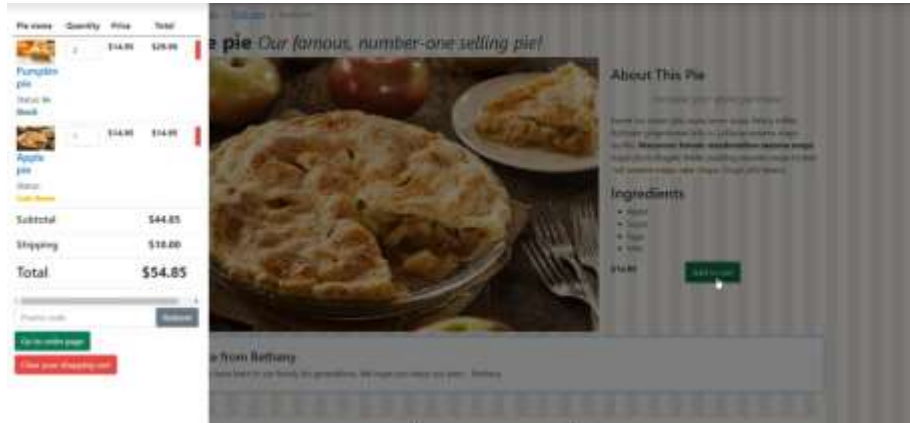
Style classes – display-1 to display6, mark, small, u, em, strong, s, text-center, text-start, text-end, text-wrap, text-break, text-primary, text-white, lead, text-capitalize, table-responsive (horizontal scroll bar will appear), table, table-dark, , table-striped, table-bordered, table-hover, table-light, table-success, table-warning, img-fluid, rounded, rounded-pill, img-fluid, border, border-bottom, border-primary.

Shortcuts – mt-1 (margin-top: 1px), pb-3, p-3, me-0 (margin on right of 0), py-5 (padding vertical of 5)

Form classes – form-label, form-control, form-check (checkbox and radio buttons), form-check-label, form-check-input, form-text (inline text like hint or help support text), form-floating, form-control-lg,

form-control-sm (change the height of form control), form-range, form-select, form-switch, input-group-text, needs-validation, invalid-feedback.

Interactive components offcanvas – carousel slide, carousel-item, offcanvas (sidebar, more like a dialog which blocks other page controls), offcanvas-start



Theme colors –



New feature interactions – to provide support for select or disable selection of the text. We can use classes like user-select-all, user-select-none

CSS: Specificity, the Box Model, and Best Practices (Interactive)

CSS specificity –



If we set element positioning as relative then it renders in normal flow like static but by this we have an ability to set the element as top/left/right/bottom. By absolute positioning, it takes an element out of the normal flow for manual positioning, it will be scoped to the window unless it falls within element that is

positioned. The fixed positioning affixes an element to a specific place in the window, where it will stay regardless of scrolling.

We can use selector combination to reuse styles for different components –



Pseudo classes allow us to conditionally select an element based on the state or position like hover, selected, even, nth-child, etc.

Pseudo elements allows us to add the content via CSS property for the decorative use like 'after', 'before', 'first-letter', 'first-line'.

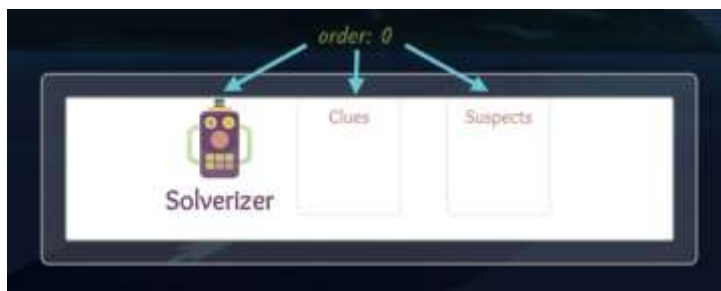
CSS: Using Flexbox for Layout (Interactive)

Flexbox is a collection of CSS properties used to align content and distribute space. It includes concepts such as flex containers, flex items and flex lines.

Flex containers control layout of child items.

Some important properties – display: 'flex', display: 'flex-inline', flex-wrap: 'nowrap | wrap | wrap-reverse', flex-direction: 'row | row-reverse | column | column-reverse', flex-direction: 'flex-start | flex-end | center | space-between | space-around'.

We can use 'order' property to determine the order of flex items along the main axis. It defaults to 0 and accepts positive and negative numbers.





Flex items are aligned along a cross axis. The cross axis is perpendicular to the main axis. In the column direction, the cross axis is horizontal, in the row direction, the cross axis is vertical.

We can use align-items property to align the content on the cross axis, the default value is stretch and its accepts: stretch | flex-start | flex-end | center | baseline.

The flex-grow can be specified the ratio of the space an item should fill in the main axis. It accepts numbers and the default is 0. Assigning a value of 1 makes an item fill as much space as possible.

The flex-shrink will specify the 'shrink factor' of a flex item. It accepts numbers and default is 1. Also, with flex-basis, the items that is an absolute unit will not grow beyond the value by default like {flex-basis: 250px;}



The align-self can align individual flex items by overriding the align-items value. The default value is stretch and it accepts: stretch | flex-start | flex-end | center | baseline.

The align-content can be used to align wrapped flex items. The default value is stretch and it accepts: stretch | flex-start | flex-end | center | space-between | space-around.

The 'flex' property is shortcut for flex-grow, flex-shrink and flex-basis properties and flex-flow is shortcut for flex-direction and flex-wrap.

Single number and percentage assigns the **grow** and **basis** values:

site.css

CSS

```
.flex-item {  
  flex-grow: 1;  
  flex-shrink: 1; /* Default */  
  flex-basis: 47.5%; }  
}
```

site.css

```
.flex {  
  flex: 1 47.5%; }  
}
```

None sets **shrink** value, removing flex behavior

site.css

CSS

```
.flex-item {  
  flex-grow: 0; /* Default */  
  flex-shrink: 0;  
  flex-basis: auto; } /* Default */  
}
```

site.css

```
.flex {  
  flex: none; }  
}
```