# LAYOUT

## Terms

Absolute positioning
Absolute units
Box model
Breakpoints
Collapsing parent
Fixed positioning
FlexBox (Flexible box layout)
Floating elements
Grid layout
Margin collapsing
Media queries
Mobile-first approach
Overflowing
Relative positioning
Relative units
Responsive web design

## Summary

- When rendering an HTML document, the browser puts each element inside a box. The box contains four areas: the content area, the padding area, the border area and the margin area.

- Padding is the space between the border and the content area. Margin is the space outside of an element and should be used to separate elements from each other.

- Margin collapsing happens when the top and bottom margins of elements are combined into a single margin. The size of the margin is equal to the largest of the two margins.

- There are two types of HTML elements: block-level and inline.
- Block-level elements always start on a new line and take up the entire available horizontal space. The `<p>` and `<div>` elements are examples of block-level elements.
- Inline elements don't start on a new line. They take up as much width as necessary. The `<span>`, `<a>` and `<img>` are a few examples of inline elements.
- We can size elements by setting their `width` and `height` properties. These properties have no effect on inline elements. To size an inline element, we need to set its `display` property to `inline-block`.
- By default, the width and height properties are applied to the content box. So paddings and borders increase the size of the visible box. This behavior can be changed by setting the `box-sizing` property to `border-box`.
- Overflow occurs when an element's content is too large to fit. Using the `overflow` property we can specify what should happen when overflow occurs.
- Measurement units in CSS fall into two categories: absolute and relative units. Examples of absolute units are px, pt, in, cm, etc. Examples of relative units are %, vw, vh, em and rem.
- Using the `position` property we can precisely position an element. The default value of this property is `static`. If we change the value of this property, the element is considered *positioned*.
- By setting the `position` to `relative`, we can position an element relative to its normal position. By setting it to `absolute`, we can position it relative to its positioned parent. That means, the parent (or ancestor) should be a positioned element. By setting the position to `fixed`, we can position the element relative to the viewport.
- By setting the `float` property, we can push an element to the left or right side of its container. Other elements will flow around the floated element and fill the available space.

- Floated elements are invisible to their parent. This behavior is called collapsing parent and often causes layout issues. To fix this, we have to clear the floated elements.

- The Flexible Box Layout (or FlexBox or just Flex) is used for laying out elements in one direction (in a row or column). A common application of Flex is in building navigation menus.

- The Grid Layout is a two-dimensional grid system. It's often used to lay out major page areas, photo galleries, etc.

- With media queries we can provide different styles for different devices depending on their features such as screen size, orientation, etc. The most common application of media queries is in providing different styles based on the viewport width.

- By using media queries and relative measurement units we can build responsive web sites that adjust smoothly to various screen sizes.

# CSS Cheat Sheet

## Box Model

```
padding: 10px 20px;
padding-top: 30px;
margin: 1px 2px 3px 4px;
margin-top: 5px;
border: 1px solid black;
border-top: 1px solid black;
```

## Sizing Elements

```
width: 5rem;
height: 20%;
box-sizing: border-box;
```
To prevent paddings/borders from increasing the size of the visible box.

## Overflowing

| | |
|---|---|
| `overflow: hidden;` | Hides the overflown content |
| `overflow: scroll;` | Always shows scroll bars |
| `overflow: auto;` | Shows scroll bars only if content overflows |

## Positioning

| | |
|---|---|
| `position: static;` | The default value |
| `position: relative;` | To position relative to the element's normal position |
| `position: absolute;` | To position relative to the element's positioned parent |
| `position: fixed;` | To position relative to the viewport |
| `z-index: 1;` | To change the stacking order of an element |

## Floating

```
float: left;
float: right;
clear: both;
```

## FlexBox

### Container properties

| | |
|---|---|
| `display: flex;` | To enable the flex layout on a container |
| `flex-direction: column;` | Direction (row, column) |
| `justify-content: center;` | To align items along the main axis |
| `align-items: center;` | To align items along the cross axis |
| `flex-wrap: wrap;` | To enable wrapping |
| `align-content: center;` | To align flex lines along the cross axis |

### Item properties

| | |
|---|---|
| `align-self: center;` | To overwrite the alignment |
| `flex-basis: 10rem;` | The initial size of an item |
| `flex-grow: 1;` | The growth factor |
| `flex-shrink: 0;` | The shrink factor |
| `flex: 0 1 10rem;` | Shorthand (grow shrink basis) |

# Grid

## Defining a grid

```
display: grid;
grid-template-rows: repeat(3, 100px);
grid-template-columns: repeat(2, 100px);
grid-template: repeat(3, 100px) / repeat(2, 100px);
grid-template-areas:
   “header   header”
   “sidebar  main”
   “footer   footer”;
```

## Gaps

```
row-gap: 10px;
column-gap: 20px;
gap: 10px 20px;
```

Shorthand (row column)

## Alignment

```
justify-items: center;
```

Align the items horizontally within their cell

```
align-items: center;
```

Align the items vertically within their cell

```
justify-content: center;
```

Align the grid horizontally within its container

```
align-content: center;
```

Align the grid vertically within its container

## Placing items

```
grid-column: 2;
grid-column: 1 / 3;
grid-column: 1 / -1;
grid-column: 1 / span 2;

grid-row: 2 / 4;

grid-area: header;
```

## Hiding elements

| | |
|---|---|
| `display: none;` | Hides the element |
| `visibility: hidden;` | Hides the element but keeps the reserved space |

## Media queries

```
@media screen and (min-width: 500px) {
}

@media screen and (min-width: 500px) and (max-width: 700px) {
}

@media print {
}
```

# TYPOGRAPHY

## Terms

Browser cache
Flash of invisible text (FOIT)
Flash of unstyled text (FOUT)
Font services
Font stack
Law of proximity
Monospace fonts
Query string parameters
Retina display
Sans-serif Fonts
Serif Fonts
System fonts
Throttling
Web safe fonts

## Summary

- Typography is the art of creating beautiful and easy-to-read text. Given that 95% of the content on the web is text, as a front-end developer, you must ensure that the text on your web pages is easy to read and visually appealing on various screen sizes.

- Fonts fall into three main categories: Serif, Sans-serif and Monospace. Serif fonts have a line/stroke at the edges of their characters. They're more professional and serious. Sans-serif fonts don't have those edges. They're more modern, warm and friendly. Monospace fonts have equal-width characters. They're often used in displaying code.

- The default color for the body text is pure black (#000). It's best to change it to dark grey.

- We can use the `font-family` property to set the font for an element. We should set this property to a font stack which contains multiple fonts as fallbacks.

- In the past, we used web safe fonts because they're available on almost all computers. These days, however, we can easily embed custom fonts.

- Font files come in a variety of different formats: TTF, OTF, EOT, WOFF and WOFF 2.0. Out of these, WOFF and WOFF 2.0 are recommended for the web because they're more compressed and can be downloaded in less time.

- We can convert any font file to a WOFF file on fontsquirrel.com.

- To embed a custom font, we should first register it using the `@font-face` rule.

- When using a custom font, the user may experience a flash of unstyled text (FOUT). Some browsers display text using a fallback font while downloading the custom font and swap it once the custom font is available. This may cause a layout shift depending on how the content is structured. Some browsers hide the text initially while downloading the custom font. This causes a flash of invisible text (FOIT). Using the `font-display` property we can tell the browser how to handle this situation.

- Using font services we can get access to thousands of beautiful fonts with zero or minimal cost. Google Web Fonts is the most popular and free font service. When using these services, fonts and `@font-face` rules are served from the provider's servers.

- A common practice for content-heavy websites is to use the system font stack which represents the default font used by an operating system. With the system font stack, we achieve a better performance because no fonts need to be downloaded and the FOUT / FOIT doesn't happen either. Plus, the page looks more familiar to the user because they see the same default font used by their device. On the flip side, the default font varies from one device to another.

- It's best to size fonts using the rem unit. This will set the font size relative to the font size of the root (html) element. Using media queries, we can resize the base font size, and as a result, the font size for all elements will be re-calculated with no extra code.

- It's best to use the rem unit for vertical margins. For headings, the top margin should be noticeably greater than the bottom margin so the heading gets separated from the text before and gets connected to the text after.

- The law of proximity describes how humans perceive the connection between objects. Objects that are closer are perceived to be related.

- Using the `line-height` property we can specify the height of lines. It's best to set this property to a unitless value around 1.5. This value will be multiplied by the font size of the current element so we don't need to remember to change the line height if we modify the font size.

- The three properties used for horizontal spacing are: `letter-spacing`, `word-spacing`, and `width`. It's often better to apply a negative letter spacing to headings so they look more compact.

- The ideal line length is about 60-70 characters. We can achieve that by applying a width of 50ch. The ch unit represents the width of the 0. 50 zeroes roughly represents 60-70 characters because some characters (like i and 1) are more narrow than 0.

- Using the Network tab in Chrome DevTools, we can simulate a slow connection. This is called Network Throttling.

- Browsers store some assets behind web pages in a permanent storage called cache. It's essentially somewhere on the disk. The cache can always be cleared.

# CSS Cheat Sheet

## Styling Fonts

```
font-family: Arial, Helvetica, sans-serif;
font-size: 1rem;
font-weight: bold;
font-style: italic;
```

## Vertical Spacing

```
margin: 3rem 0 1rem;
line-height: 1.5;
```

## Horizontal Spacing

```
letter-spacing: -1px;
word-spacing: 2px;
width: 50ch;
```

## Formatting Text

```
text-align: center;
text-indent: 1rem;
text-decoration: underline;
text-transform: uppercase;
white-space: nowrap;
direction: rtl;
```

## Multi-column Text

```
column-count: 2;
column-gap: 2rem;
column-rule: 3px dotted #999;
```

# IMAGES

## Terms

Art direction
Clipping
CSS sprites
Data URLs
Device Pixel Ratio (DPR)
Filters
High-density screens
Icon fonts
Logical resolution
Physical resolution
Raster images
Resolution switching
Retina displays
Scalable Vector Graphics (SVG)
Vector images

## Summary

- Images fall into two categories: raster and vector. Raster images are made up of pixels. Vector graphics are defined by a set of mathematical vectors (eg lines and curves).

- Raster images often come from cameras and scanners but they can also be produced in software. Any file with the extension of JPG, PNG, GIF is a raster image. We use these images for displaying photos.

- Vector images are exported from drawing tools like Adobe Illustrator. Files with the SVG extension represent vector graphics.

- We use the `img` element to display content images. Content images can represent meaningful content or be used for decorative purposes. If used for decoration, we should set the `alt` attribute to an empty string; otherwise, screen readers will read out the name of the file which may be distracting to the user.

- Using CSS sprites we can combine multiple images into a single image (sprite) and reduce the number of HTTP requests. The problem with CSS sprites is that every time we need to change one of the images in the sprite, we have to re-generate the sprite. So, use this technique for small images that don't change often. You can generate a sprite using https://csspritestool.com.

- Data URLs allow us to embed an image data directly in an HTML document or a stylesheet. The embedded code is always greater than the size of the original resource and makes the document convoluted and hard to maintain. Use this technique only if you know what you're doing!

- We can clip an image using the `clip-path` property in CSS. To generate a clip path from basic templates, visit https://bennettfeely.com/clippy

- Using the `filter` property in CSS, we can apply filters such as grayscale, blur, saturate, brightness and so on.

- High-density screens like Apple's Retina displays contain more pixels than standard-density screens. The pixels on these screens are smaller than the pixels on standard-density screens. So when displaying an image, the screen uses a scale factor (1.5 or greater) to scale up the image. As a result, raster images may look a bit blurry when shown on these screens. To solve this problem, we can supply 2x or 3x versions of an image using the `srcset` attribute of the `img` element.

- For flexible-sized images, we need to supply the image in various sizes for different devices like mobiles, tablets and desktop computers. If we supply a single image, the browser on each device has to resize the image which can be a costly operation. The larger the image, the more memory is needed and the more costly the resizing operation will be. Plus, the extra bytes used to download the image will be wasted. This is the *resolution switching* problem. To address this, we should give the `img` element a few image sources and the size of the image for various viewports. The browser will take the screen resolution and pixel density into account and download the image that best fits the final size.

- We can use https://responsivebreakpoints.com to generate our image assets for various screen sizes.

- WebP is a modern image format created by Google and is widely supported except in Internet Explorer. To support modern image formats, we can use the `picture` element with multiple sources. The `picture` element should always contain an `img` element otherwise the image is not shown.

- Sometimes we need to show a zoomed in or a cropped version of an image for certain viewport sizes. This is the *art direction* problem. To handle this, we use the `picture` element with multiple sources. Each source should contain a media condition and a `srcset`. The browser will pick the first source whose media condition matches.

- Scalable Vector Graphics (SVG) files are great for logos, icons, simple graphics and backgrounds with patterns. They are often very small and can scale without losing quality. You can get find plenty of beautiful SVG backgrounds on https://svgbackgrounds.com

- We can also use icon fonts for displaying icons. The most popular icon fonts are Font Awesome, Ionicons and Material Design Icons.

# CSS Cheat Sheet

## Background Images

```
background: url(../images/bg.jpg);
background-repeat: no-repeat;
background-position: 100px 100px;
background-size: cover;
background-attachment: fixed;
```

## Clipping

```
clip-path: polygon(50% 0%, …);
```

## Filters

```
filter: grayscale(70%);
filter: blur(3px);
filter: brightness(0.5);
filter: contrast(200%);
filter: drop-shadow(10px 10px 10px grey);
filter: hue-rotate(90deg);
filter: invert(50%);
filter: saturate(25%);
filter: sepia(50%);
filter: opacity(50%);
filter: grayscale(70%) blur(3px);
```

## Supporting High-density Screens

```
<img
   srcset="
      images/meal.jpg     1x,
      images/meal@2x.jpg  2x,
      images/meal@3x.jpg  3x
   "
   src="images/meal.jpg"
/>
```

Use this for fixed-size images. The browser will download the best image based on the pixel ratio of the device.

## Resolution Switching

```
<img
   srcset="
      images/meal.jpg     400w,
      images/meal@2x.jpg  800w,
      images/meal@3x.jpg  1200w
   "
   sizes="
      (max-width: 500px) 100vw,
      (max-width: 700px) 50vw,
      33vw
   "
   src="images/meal.jpg"
/>
```

Use this for flexible-size images. The browser will download the best image that requires less resizing based on the pixel ratio and screen resolution of the device.

## Supporting Modern Image Formats

```
<picture>
    <source type="image/webp" srcset="…" />
    <source type="image/jpeg" srcset="…" />
    <img src="…" alt="…" />
</picture>
```

## Art Direction

```
<picture>
    <source media="(max-width: 500px)" srcset="…" />
    <source media="(min-width: 501px)" srcset="…" />
    <img src="…" alt="…" />
</picture>
```

# FORMS

## Terms

Check boxes
CSS frameworks
Data validation
Data lists
Fieldsets
Hidden fields
HTTP method
Input fields
Input controls
Radio buttons
Sliders
Text areas

## Summary

- Label elements make our forms more accessible. When the user clicks on a label, the associated input field gets focus.
- We use the **`input`** element to accept data from the user. The **`type`** attribute of the input element determines the type of the input field and can be **`text`**, **`password`**, **`email`**, **`checkbox`**, **`radio`**, **`range`**, **`date`**, **`file`**, etc.
- Using data lists we can provide a suggestion list (autocompletion).
- To display a drop-down list, we use the **`select`** element. This element can contain one or more **`option`** elements.
- The **`fieldset`** element (along with a legend) is used to group related input fields. Alternatively, we can use the **`section`** element to group related fields.
- Hidden fields are not visible to the user and are often used to specify the ID of the content being edited. Don't use them to store sensitive information because they can easily be accessed via the source code of the page.

- Always validate your form data to prevent security attacks and data corruption.

- HTML5 comes with built-in validation. The most common validation attributes are **`required`**, **`minlength`**, **`maxlength`**, **`min`**, and **`max`**.

- To submit a form, you should set the **`action`** and the **`method`** of the form. The action attribute represents where data is sent. The method specifies how the data is sent and can be either **GET** or **POST**.

- With the POST method, form data is included in the body of the request. With the GET method, form data is included in the URL as query string parameters. That's why the GET method is often used when we need to enable bookmarking pages. In contrast, the POST method is used when we need to update the data.

- Use https://formspree.io to create a backend for your forms.

# CSS Cheat Sheet

## Forms

```
<form action="https://formspree.io/…" method="POST">
   <label for="name" />
   <input type="text" id="name" />
   <button type="submit">Register</button>
</form>
```

## Inputs

```
<input type="email" />
<input type="password" />
<input type="checkbox" />
<input type="radio" />
<input type="range" />
<input type="date" />
<input type="file" />
```

## Grouping fields

```
<fieldset>
    <legend>Contact</legend>
    <input type="text" />
    <input type="text" />
</fieldset>
```

## Data Validation

```
<input type="number" min="0" max="5" required />
<input type="text" minlength="3" maxlength="50" required />
```

# Animations

## Summary

- Using the **`transform`** property, we can apply one or more transformations to an element.
- The most common transformation functions are **`rotate()`, `skew()`, `translate()`** and **`scale()`.**
- The **`transition`** property is used to animate one or more properties.
- To create a custom animation, first, we need to define the keyframes. Each keyframe includes the list of styles to be applied at a given moment in time. Once we define the keyframes, we can use the **`animation`** property to animate an element.

# CSS Cheat Sheet

## Transformations

```
transform: rotate(15deg);
transform: rotate(-15deg);
transform: scale(1.3);
transform: skew(15deg);
transform: translate(10px, 20px);
transform: translateX(10px);
transform: translateY(20px);
transform: rotate(15deg) scale(1.3);
```

## Transitions

```
transition: transform .5s;
transition: transform .5s ease-out;
transition: transform .5s ease-out 1s;    /* 1s delay */
transition: transform .5s, color .3s;
```

## Animations

```
@keyframes pop {
   0% { transform: scale(1); }
   50% { transform: scale(1.5); }
   75% { transform: rotate(45deg); background: tomato; }
   100% { transform: rotate(0); }
}

.box {
   animation: pop 3s ease-out;
}
```

# Clean CSS

## Summary

- Follow a naming convention for naming IDs and classes. The most common naming conventions are **PascalCase**, **camelCase** and **kabob-case**.

- For a small project, you can write all of your CSS rules in one stylesheet. Use CSS comments to create logical sections in your stylesheet. For a more complex project, you need to separate your stylesheet into multiple files and combine them together using build tools like Webpack, Rollup or Parcel.

- Avoid over-specific selectors. Limit nesting to two or maximum three selectors.

- Avoid the **!important** keyword as much as possible.

- Sort CSS properties. This makes it easier to read your code.

- Take advantage of style inheritance and reduce duplication in your styles.

- Use *CSS variables*, also called custom properties, to keep your code DRY.

- We often declare variables using the `:root` selector that targets the `html` element. We can then access these variables using the `var()` function.

- *Object-oriented CSS* is a set of principles for creating reusable components. The two principles in object-oriented CSS are: 1- Separate container and content. 2- Separate structure and skin.

- *BEM (Block Element Modifier)* is a popular naming convention for CSS classes.