Frontend Crash Course: CSS

Chen Hui Jing / @hj_chen
Cascading Style Sheets
Structure of a CSS rule

```css
selector {
    property1: value;
    property2: value;
    property3: value;
}
```

- The **selector** identifies which HTML elements the rule will be applied to.
- The **curly braces** contain the property-value pairs, separated with semi-colons.
- The **properties** define the style of the selected element.
- The **values** are dependent on the property, and indicate the value of the properties to be set.
CSS Selectors

- Type selector
- Attribute selectors
- Pseudo-classes
- Pseudo-elements
- Class selectors
- ID selectors
- Combinators
Type selector
Matches all the elements of that type on the page

main {
    /* some properties */
}

div {
    /* some properties */
}
p {
    /* some properties */
}
Attribute selectors

<attr>

Targets elements that contain the <attr> attribute, regardless of its value.

```html
<div data-colour="green"></div>
<div data-colour="blue"></div>
<div data-colour="yellow"></div>
```

```javascript
[data-colour] {
  /* some properties */
}
```
**<attr>=<val>**

Targets elements where the `<attr>`’s value is exactly `<val>`.

```html
<div data-colour="green"></div>
```

```javascript
[data-colour="green"] { /* some properties */ }
```

---

**<attr>~=<val>**

Targets elements with the `<attr>` attribute whose value is a list of white space separated words, one of which must be `<val>`. `<val>` itself cannot contain white space, and neither can it be an empty string.

```html
<div data-colour="green yellow blue"></div>
```

```javascript
[data-colour~="green"] { /* some properties */ }
```
<attr>|=<val>

Targets elements with the `<attr>` attribute whose value is exactly `<val>` or starts with `<val>` immediately followed by a “-”.
Primary use-case being for language subcode matching, like “en”, “en-US” and “en-UK”.

```
<div data-colour="green-table"></div>
<div data-colour="green-chair"></div>
<div data-colour="green-bottle"></div>
```

[data-colour|="green"] {
  /* some properties */
}

<attr>^=<val>

A substring matching selector. Targets elements with the `<attr>` attribute whose value starts with `<val>`.
`<val>` cannot be an empty string.

```
<div data-colour="greenish-yellow"></div>
<div data-colour="greengoblin"></div>
```

```
[data-colour^="green"] {
  /* some properties */
}
```
**<attr>$=<val>**

A substring matching selector. Targets elements with the `<attr>` attribute whose value ends with `<val>`. `<val>` cannot be an empty string.

```html
<div data-colour="yellowish-green"></div>
<div data-colour="seagreen"></div>
```

```javascript
[data-colour="$=""green"] { /* some properties */ }
```

**<attr>*=<val>**

A substring matching selector. Targets elements with the `<attr>` attribute whose value contains an instance of `<val>`. `<val>` cannot be an empty string.

```html
<div data-colour="goblingreenish"></div>
```

```javascript
[data-colour="=""green"] { /* some properties */ }
```
Pseudo-classes

Structural pseudo-classes
- :root
- :nth-child(n)
- :nth-last-child(n)
- :nth-of-type(n)
- :nth-last-of-type(n)
- :first-child
- :last-child
- :first-of-type
- :last-of-type
- :only-child
- :only-of-type
- :empty

Link pseudo-classes
- :link
- :visited

User action pseudo-classes
- :active
- :hover
- :focus

UI element pseudo-classes
- :enabled
- :disabled
- :checked

Negation pseudo-class
- :not(selector)

Language pseudo-class
- :lang(zh)

Target pseudo-class
- :target
Pseudo-elements

::first-line, ::first-letter, ::before, ::after

DO SOMETHING TO THE FIRST LETTER, OR FIRST line. Just the first though. Not the others. Leave them lines alone. 🦓

```css
.pseudoelem p::first-line {
  text-transform: uppercase
}

.pseudoelem p::first-letter {
  font-size: 2em;
  color: seagreen;
}

.pseudoelem p::before {
  content: 'Dasterxml'
}
```
Class selector

Matches all the elements with the specified class attribute

```html
<div class="example">
    // some random markup
</div>
```

```css
.example {
    /* some properties */
}
```
ID selector

Matches matches the element with the specified id attribute

```html
<div id="example">
  // some random markup
</div>
```

```css
#example {
  /* some properties */
}
```
Combinators

Descendant
Can be any level of descendant, multiple nesting

Child
Immediate child element only

```
<div>
  <p>This is a <p> element in a <div> element</p>
</div>
```

```
div span {
  /* some properties */
}
```

```
<div>
  <p>This is <p> element is a direct child</p>
</div>
```

```
div > p {
  /* some properties */
}
```
Next sibling

Same parent, immediately following

```html
<p>This is a <p> element.</p>
<ul>
  <li>This is a list item</li>
  <li>This is a list item</li>
</ul>
```

```css
p + ul {
  /* some properties */
}
```

Subsequent sibling

Same parent, can be anywhere following

```html
<p>This is a <p> element.</p>
<p>This is another <p> element.</p>
<ul>
  <li>This is a list item</li>
  <li>This is a list item</li>
</ul>
```

```css
p ~ ul {
  /* some properties */
}
```
Who wins?

1. Importance
2. Specificity
3. Source order
!important

Don’t use it if you can help it.

This paragraph has a class of cookie and an id of butterscotch.

This paragraph has a class of cookie.
CSS Specificity

- Inline styles: 0-∞
- IDs: 0-∞
- Classes, attributes and pseudo-classes: 0-∞
- Elements and pseudo-elements: 0-∞

```
ul {
  // CSS properties
}
```

```
.class-1 .class-2 p {
  // CSS properties
}
```

```
#id-1 .class-3 div {
  // CSS properties
}
```
Source order

The tie-breaker for equal specificity values

This paragraph has a class of cookie and an id of butterscotch.

This paragraph has a class of cookie.

```
.sourceorder .cookie.cookie {
    background-color: crimson;
    color: whitesmoke;
    border: 0.25em inset teal;
}

.sourceorder .cookie:first-child {
    background-color: mediumslateblue;
    color: lavender;
    border: none;
```
General guidelines for writing CSS

- Declare your styles from lowest specificity then move up
- Keep your specificity as low as possible
- Name your classes sensibly
- Never style IDs
- Don’t write inline styles
Layout

Box Alignment

Flexbox

Grid
Flexbox support

98.68% of internet users are using a browser that supports Flexbox.
Grid support

93.26% of internet users are using a browser that supports Grid.
Flexbox
For single dimension components

Grid
For two-dimensional layouts
Container–child relationship
<table>
<thead>
<tr>
<th>flex: initial</th>
<th>flex: 0 1 auto, cannot grow but can shrink when there isn’t enough space</th>
</tr>
</thead>
<tbody>
<tr>
<td>flex: auto</td>
<td>flex: 1 1 auto, can grow and shrink to fit available space</td>
</tr>
<tr>
<td>flex: none</td>
<td>flex: 0 0 auto, cannot grow or shrink, AKA inflexible</td>
</tr>
<tr>
<td>flex: &lt;positive-number&gt;</td>
<td>flex: &lt;positive-number&gt; 1 0, can grow and shrink, extent of growth depends on flex factor</td>
</tr>
</tbody>
</table>
Flexbox use-cases

Centre stuff in their parent container

```css
.automargin {
    display: flex;
}
.automargin div {
    margin: auto;
}
```
Flexbox use-cases

When you need 1 thing by itself on the opposite side

```
.navbar ul {
    display: flex;
}

.navbar li:last-child {
    margin-left: auto;
}
```
Flexbox use-cases

To stick the header and footer at the top and bottom

```
.flex1 {
  display: flex;
  flex-direction: column
}

.flex1 .main {
  flex: 1;
}
```
CSS grid basics

Define your grid.

Place items in the grid.
Defining a grid

```
<div class="grid1">
  <div class="grid1__item">
    <p>Item A</p>
  </div>
  <div class="grid1__item">
    <p>Item B</p>
  </div>
  <div class="grid1__item">
    <p>Item C</p>
  </div>
  <div class="grid1__item">
    <p>Item D</p>
  </div>
  <div class="grid1__item">
    <p>Item E</p>
  </div>
  <div class="grid1__item">
    <p>Item F</p>
  </div>
</div>
```

```
.grid1 {
  display: grid;
  grid-template-columns: 200px 200px 200px;
  grid-template-rows: 150px 150px;
  gap: 1em;
}
```
The \texttt{fr} unit

Represents a \textit{fraction} of the \textbf{free space} in the grid container.

```
.grid4 {
    display: grid;
    grid-template-columns: 150px 1fr 2fr;
}
```
Fluid CSS grid

```css
.container {
    display: grid;
    grid-template-columns: repeat(3, 3fr 2fr);
}
```
The `minmax()` function

Defines a size range for columns or rows in the grid.

```css
.grid5 {
    display: grid;
    grid-template-columns: minmax(200px, 1fr) 300px 300px;
}
```
The `repeat()` function

To specify a large number of columns or rows that follow a similar pattern

<table>
<thead>
<tr>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
</tr>
</thead>
</table>

```css
.grid6 {
  display: grid;
  grid-template-columns: repeat(4, 90px 140px);
}
```
**auto-fill vs. auto-fit**

Allow browser to determine how many tracks to create depending on track size.

```
repeat(auto-fill, 100px);
```

```
repeat(auto-fit, 100px);
```

auto-fit collapses empty tracks.
auto-fill *versus* auto-fit

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
</table>

```css
.keyword {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
}
```
Responsive grid without media queries

.container {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(10em, 1fr));
}
Explicit Grid and Implicit Grid

The browser will create an implicit grid to hold items placed outside the grid.
The **grid-auto-flow** property

Adjusting the direction and density of grid items

```css
.autoflow {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(120px, 1fr));
    grid-auto-rows: 120px;
    grid-auto-flow: row;
}
.autoflow__item:nth-child(3n) {
    background-color: #ff8200;
}
```
Box Alignment

To create a cohesive and common box alignment model to share among all of CSS
<table>
<thead>
<tr>
<th>Property</th>
<th>Axis</th>
<th>Aligns</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>justify-content</td>
<td>main/inline</td>
<td>content within element (effectively adjusts padding)</td>
<td>block containers, flex containers and grid containers</td>
</tr>
<tr>
<td>align-content</td>
<td>cross/block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>justify-self</td>
<td>inline</td>
<td>element within parent (effectively adjusts margins)</td>
<td>block-level boxes, absolutely-positioned boxes and grid items</td>
</tr>
<tr>
<td>align-self</td>
<td>cross/block</td>
<td></td>
<td>absolutely-positioned boxes, flex items and grid items</td>
</tr>
<tr>
<td>justify-items</td>
<td>inline</td>
<td>items inside box (controls child items)</td>
<td>block containers and grid containers</td>
</tr>
<tr>
<td>align-items</td>
<td>cross/block</td>
<td></td>
<td>flex-containers and grid-containers</td>
</tr>
</tbody>
</table>

Source: **CSS Box Alignment Module Level 3**
Flexbox

- align-content
- justify-content
- align-items
- align-self

The justify-items/justify-self properties do not apply to flex items

Grid

- align-content
- justify-content
- align-items
- align-self
- justify-items
- justify-self
Flexbox

- align-content
- justify-content
- align-items
- align-self

The justify-items/justify-self properties do not apply to flex items

Grid

- align-content
- justify-content
- align-items
- align-self
- justify-items
- justify-self
justify-content & align-content

```
.gridcontent {
  justify-content: end;
  align-content: center;

  display: grid;
  grid-template-columns: repeat(3, 25%);
  grid-template-rows: repeat(3, 20%);
  grid-gap: 1em;
  grid-template-areas:
    "a b b"
}
```
justify-content & align-content

<table>
<thead>
<tr>
<th>一</th>
<th>二</th>
<th>三</th>
<th>四</th>
<th>五</th>
<th>六</th>
<th>七</th>
<th>八</th>
</tr>
</thead>
<tbody>
<tr>
<td>一</td>
<td>二</td>
<td>三</td>
<td>四</td>
<td>五</td>
<td>六</td>
<td>七</td>
<td>八</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
.flexcontent .wrapper {
  display: flex;
  flex-wrap: wrap;
  justify-content: flex-start;
  align-content: end;
  height: 70vh;
}
.flexcontent .box {
  height: 3.5em;
  width: 4.5em;
  border: 1px solid;
```
justify-self & align-self

```css
.gridself {
  display: grid;
}

.gridself .box.d {
  grid-area: d;
  align-self: center;
  justify-self: center;
}

.gridself .box.c {
  grid-area: c;
  align-self: end;
  justify-self: end;
}
```

A  B

C  D
### justify-items & align-items

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

```css
.griditems {
  justify-items: normal;
  align-items: end;

  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-gap: 1em;
  grid-auto-rows: calc(25% - 1em);
  grid-template-areas: "a a b b"
}
```
align-items & align-self

```css
.flexitems .wrapper {
  display: flex;
  flex-wrap: wrap;
  align-items: stretch;
}

.flexitems .box:nth-child(2n+1) {
  padding: 0.5em 1.5em 3em;
  align-self: baseline;
}
```
Let’s build a thing

Demo
Cool stuff & pretty things

If time permits...
CSS transforms

Allows us to change the **shape** and **position** of the affected content without disrupting the normal document flow
## 2D transform functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rotate(&lt;angle&gt;)</code></td>
<td>Performs a 2D rotation by the angle specified around the element’s origin</td>
</tr>
<tr>
<td><code>translate(&lt;translation-value&gt;[, &lt;translation-value&gt;]?)</code></td>
<td>Performs a 2D translation in the specified X and Y directions</td>
</tr>
<tr>
<td><code>skew(&lt;angle&gt;[, &lt;angle&gt;]?)</code></td>
<td>Performs a 2D skew by the angles specified</td>
</tr>
<tr>
<td><code>scale(&lt;number&gt;[, &lt;number&gt;]?)</code></td>
<td>Performs a 2D scale operation by the scaling vector specified</td>
</tr>
</tbody>
</table>
Diagonal text with rotate()
CSS transitions

Lets you create gradual transitions between the values of specific CSS properties

CSS animations

Lets you animate the values of CSS properties over time, using keyframes

Note: not all CSS properties are animatable
Transition properties

transition is a shorthand property, covers the following (values are initial default values):

- transition-delay: 0s
- transition-duration: 0s
- transition-property: all
- transition-timing-function: ease
Animation properties

animation is a shorthand property, covers the following (values are initial default values):

- animation-name: none
- animation-duration: 0s
- animation-timing-function: ease
- animation-delay: 0s
- animation-iteration-count: 1
- animation-direction: normal
- animation-fill-mode: none
- animation-play-state: running

Specified as one or more single animations, separated by commas.
Fool’s Mate

In chess, Fool’s Mate, also known as the Two-Move Checkmate, is the checkmate in the fewest possible number of moves from the start of the game. This can be achieved only by Black, who can deliver checkmate on move 2 with the queen. Fool’s Mate received its name because it can only occur if White commits an extraordinary blunder. Even among rank beginners, the mate almost never occurs in practice.
Magical kittencorn
Specifications

Read them first, tutorials later.

CSS specifications: https://www.w3.org/Style/CSS/specs.en.html
MDN web docs

Go-to resource for all things web development.

Thank you!

https://www.chenhuijing.com

@hj_chen

@hj_chen

@huijing

Font is Magnetic Pro by Olivier Gourvat