CSS

Generell

Das CSS-Vokabular:

```
Selektor {
    Eigenschaft: Wert;
}
```

Eigenschaft und Wert ergeben zusammen die Deklaration.

Validierung

Das CSS ist valide.

Struktur

Grundsätzlich ist die Reihenfolge der Regeln im CSS aufgrund der Vererbung mit Bedacht zu wählen. Dabei soll die natürliche Wertigkeit (Spezifität) der Selektoren beachtet und genutzt werden. `!important`-Anweisungen sind möglichst nur für Debugging-Zwecke zu verwenden.

1. Die CSS-Regeln sind modular, gemäss Reihenfolge in der HTML-Struktur gruppiert. Innerhalb der Gruppe gilt invertierte Spezifität: Erst die allgemeinen Regeln, dann die spezifischeren. Zusammengehörige Deklarationen (zb. Module, Komponenten, Layoutelemente) werden einheitlich mit einem Kommentar eingeleitet:

```
/* =footer
------------------------------------------------------------ */
```

2. Bei Layout unabhängigen Elementen gilt das DOM und dann die invertierte Spezifität:

```
html, body, main, h1, h2, h2.warning, p, img, img > a, img a.lead {

}
```

Formatierung

1. Jede Deklaration ist auf einer eigenen Zeile.
2. Jede Deklaration ist eingerückt.
3. Nach dem Selektor und zwischen Eigenschaft und Wert ist immer ein Leerzeichen.
4. Jede Deklaration – auch die Letzte – endet mit einem Semikolon.
5. Vor einem modularen Block ist ein Kommentar, nach der letzten Regel des Blocks eine Leerzeilen einzufügen.
   Ergo: *Kommentar Block* > *Regeln* > *Leerzeile* > *Kommentar nächster Block* > ...
6. Werte sind kleingeschrieben (`#fff` anstatt `#FFF` , `arial` anstatt `Arial`).
7. Deklarationen und Werte sind – wo möglich und sinnvoll – in der Kurznotation zu schreiben: `#fff` anstatt `#ffffff`
8. Leere Deklarationen oder Media Queries, auskommentierte Zeilen und alles, was sonst nicht gebraucht wird, ist spätestens in der Produktivumgebung zu entfernen.

Falsch:

```
p{
margin-bottom:0;
margin-top:2em; margin-right:0;
margin-left:1.5em;
color:#FFFFFF
}
```

Richtig:

```
p {
    margin: 2em 0 0 1.5em;
    color: #fff;
}
```

Sortierung der Deklarationen

Deklarationen werden möglichst von aussen nach innen sortiert. Erst das Verhalten der Box (Position, Box-Model), dann deren Aussehen (Typo, Visuelles, Animationen). Bei Elementen mit vielen Deklarationen können diese Gruppen bei Bedarf auch visuell gruppiert werden.

```
.element {
    /* Custom Properties  */
    --_color: #fff;

    /* Positioning */
    position, top, right, z-index, …

    /* Box Model */
    display, flex-wrap, float, clear, …
    width, min-width, max-width, height, …
    margin, border, padding, …

    /* Typography */
    font, line-height, color, …

    /* Visual */
    background, box-shadow, border-radius, opacity, overflow, transform

    /* Animation */
    transition, animation, …

    /* Misc */
    user-select, …
}
```

CSS Variablen / Custom Properties

Die im Framework vordefinierten Variablen sind zu adaptieren. Für Farbdefinitionen und Schriftgrössen stehen entsprechende Vorgaben bereit.

```
:root {
    --shadow-xs: 0 0.1px 0.3px rgba(0, 0, 0, 0.06), 0 1px 2px rgba(0, 0
    --shadow-sm: 0 0.3px 0.4px rgba(0, 0, 0, 0.025), 0 0.9px 1.5px rgba
    --shadow-md: 0 0.9px 1.5px rgba(0, 0, 0, 0.03), 0 3.1px 5.5px rgba(0
    --shadow-lg: 0 1.2px 1.9px -1px rgba(0, 0, 0, 0.014), 0 3.3px 5.3px
    --shadow-xl: 0 1.5px 2.1px -6px rgba(0, 0, 0, 0.012), 0 3.6px 5.2px

    --clr-primary: hotpink;

    --clr-neutral-100: hsl(0deg 0% 100%);
    --clr-neutral-900: hsl(0deg 0% 0%);

    --fontsize-200: 1.2rem;
    --fontsize-300: 1.4rem;
    --fontsize-400: 1.6rem; /* Basis / Grundschrift */
    --fontsize-600: 1.8rem;
    --fontsize-700: 2.4rem;
    --fontsize-800: 3.0rem;
    --fontsize-900: 3.6rem;

    --radius-sm: .1875em;

    accent-color: var(--clr-primary);
}
```

Abstufungen in Variablen-Namen

Für Gruppen mit potenziellem Bedarf an künftig zusätzlichen Definitionen, sind Namenkonventionen mit numerischen Suffixen (`-100` , … `-400` , … `-900`) sinnvoll, ähnlich wie wir es von Schriftstärken kennen. Der 400er-Wert ist dabei – wenn sinnvoll – der Basiswert.

Für Gruppen mit wenigen Abstufungen können die Suffixe mit den Kleidergrössen eingesetzt werden (`-xs` , `-sm` , `-md` , `-lg` , `-xl`).

Bei beiden Varianten sollen sinnvolle und realistische Abstufungen gewählt werden, die bei Bedarf ergänzt werden können.

Auch kundenspezifische Farbwerte können im Bedarfsfall noch ergänzt werden (`--clr-primary` , `--clr-secondary` , `--clr-tertiary` …). Sind hier noch Farbabstufungen nötig, reichen meist die Suffixe `-dark` und `-bright` :

```
:root {
    --clr-primary-bright: hsl(330deg 100% 61%);
    --clr-primary:        hsl(330deg 100% 71%); /* brandcolor hotpink */
    --clr-primary-dark:   hsl(330deg 100% 81%);
}
```

Gerade bei Farbabstufungen sollte die Wahl auf den HSL-Farbraum fallen, da die Werte so les- und vergleichbar bleiben.

Komponenten-Variablen mit lokalem Scope

Werden Variablen nicht global zur Verfügung gestellt, sondern haben einen lokalen Scope auf Komponenten-Level, so sollten diese mit einer vorangestellten Underscore versehen werden. Damit sind sie auf einen Blick als solche erkennbar.

```
.form-builder {
    --_form-clr-hint: #585858;
    --_form-clr-error: #cc0000;
}
.form-builder .form-hint {
    …
    color: var(--_form-clr-hint);
}
```

Kommentare

Kommentare gliedern das CSS-Dokument in sinnvolle Blocks, um die Übersicht zu vereinfachen. Der Name des Blocks ist dabei mit einem vorangestellten = zu versehen, um die schnelle Suche zu ermöglichen.

```
/* =header
------------------------------------------------------------- */
header {
    position: relative;
    height: 5em;
}
header > h2 {
    position: absolute;
    top: 1em;
    right: 2.5em;
}
```

Kommentare hinter einzelnen Deklarationen geben Auskunft über Lösungswege (beispielsweise Grössenberechnungen) oder weisen auf Hacks hin:

```
.mainnav__link--os a {
    color: var(--clr-neutral-400);
    font-weight: normal;
    border-radius: 0.222222222222222em; /* 4/18 */
}
```

Während der Entwicklung oder zu Testzwecken auskommentierte Codeblöcke oder Deklarationen sind spätestens in der Produktivumgebung zu entfernen.

Hacks

Hacks sind zu vermeiden. Lässt sich ihr Gebrauch nicht gänzlich vermeiden, sind diese auf ein Minimum reduziert. Es sind Lösungen anzustreben, die in allen zu unterstützenden Browsern ohne Hacks gleichermassen funktionieren.

1. Hacks werden kommentiert
2. Unvermeidbare Hacks für ältere IE's (sofern diese noch zu unterstützen sind) sollten in einer eigenen CSS-Datei abgelegt, die mit CC's (Conditional Comments) eingebunden wird. Diese wird nur den IE-Versionen ausgeliefert, die sie auch benötigen. IE ab Version 10 unterstützt keine CC's mehr.

Sind Hacks/Filter oder browserspezifische Regeln unvermeidlich, sind diese wo nötig zu beschreiben:

```
@supports not (-ms-ime-align: auto) { /* no ie11 & legacy edge ≤18 */
    summary {
        display: flex;
        align-items: baseline;
        cursor: pointer;
        margin: -1.5em;
        padding: 1.5em;
        overflow: hidden;
    }
}
```

Progressive Enhancement

Progressive Enhancement ist grundsätzlich zulässig, solange auch für andere unterstütze Browsern eine zugängliche Alternative geboten wird.

Beim Einsatz von neueren CSS-Eigenschaften ist eine möglichst abwärtskompatible Browserunterstützung zu berücksichtigen.

```
.card p {
    display: -webkit-box;
    -webkit-line-clamp: 3;
    -webkit-box-orient: vertical;
    overflow: hidden;
}
```

Autoprefixer und CSS Feature Queries sind weitere Möglichkeiten.

externe Quellen

Dokumentationen

- Idiomatic-CSS – Grundlagen zum Schreiben von einheitlichem, idiomatischem CSS
- Cascading Style Sheets Homepage
- Übersicht aller CSS-Eigenschaften

CSS

- Can I use …
- CSS Selector Specificity
- The Official Source to Follow the Development of CSS 3
- Better Ways to Organise CSS Properties
- Harry Potter and the Order of CSS